Resource Allocation for Sequential Decision Making under Uncertainty: Studies in Vehicular Traffic Control, Service Systems, Sensor Networks and Mechanism Design

A THESIS

SUBMITTED FOR THE DEGREE OF

Doctor of Philosophy

IN THE

Faculty of Engineering

BY

PRASHANTH L.A.

UNDER THE GUIDANCE OF

Prof. Shalabh Bhatnagar

Department of Computer Science and Automation

Indian Institute of Science

Bangalore - 560 012

March 2013

“Its not where you take things from – it’s where you take them to.”
–Jean-Luc Godard

To
Samudyata

Acknowledgements

There are many people who have assisted me in this journey and I would like to express my gratitude to each one of them.

First, I would like to thank my supervising professor Dr. Shalabh Bhatnagar, who has taught me much more than stochastic optimization and reinforcement learning. One quality that I would like to imbibe is his professionalism and work ethics, which in simple terms translates to 'work hard and let it speak for itself' or equivalently 'what matters is what you do and not what you say'. While the extensive interactions with him served to sharpen the understanding of the subject, it also helped in improving my writing skills as well.

Second, I would like to thank to H.L.Prasad, collaboration with whom has been most valuable. Further, my stint as a project associate with DIT-ASTec as well as an intern at IBM Research presented many opportunities for developing as well as applying stochastic optimization algorithms in various practical contexts and I would like to thank Prof. Anurag Kumar, Prof. K.V.S. Hari of ECE department, IISc and Nirmit Desai and Gargi Dasgupta of IBM Research for several useful interactions on the subject matter. I would also like to thank my thesis committee members, Dr. Remi Munos and Dr. Vivek Borkar, for their feedback.

Third, to my course instructors, from whom I have 'measurably' benefited. Dr. Manjunath Krishnapur deserves a special mention here for patiently taking us (Prasad and I) through much of probability theory and related topics.

Fourth, I am also grateful to Computer Science department in particular and Indian Institute of Science in general for providing an excellent environment conducive for conducting research. The cheerful assistance I received from my pals - Manu, Vikram, Amrish, Harshan, Bharath, Naveen, Manjesh to name a few - made my stay pleasant as ever.

Finally, I would like to thank my family members - particularly my parents, my wife and my brother, who provided the necessary emotional support almost surely during this work. Ever since I enrolled for a PhD, there have many upheavals in my life that included moving out of a IT job, arrival of life partner, change of address, baby being born and so on. One question that has remained throughout from my well-wishers has been “Is your thesis done yet?”. I am sure that they will be happy to see my thesis completed.

Contents

List of Tables

List of Figures

Abstract

A fundamental question in a sequential decision making setting under uncertainty is "how to allocate resources amongst competing entities so as to maximize the rewards accumulated in the long run?". The resources allocated may be either abstract quantities such as time or concrete quantities such as manpower. The sequential decision making setting involves one or more agents interacting with an environment to procure rewards at every time instant and the goal is to find an optimal policy for choosing actions. Most of these problems involve multiple (infinite) stages and the objective function is usually a long-run performance objective. The problem is further complicated by the uncertainties in the system, for instance, the stochastic noise and partial observability in a single-agent setting or private information of the agents in a multi-agent setting. The dimensionality of the problem also plays an important role in the solution methodology adopted. Most of the real-world problems involve high-dimensional state and action spaces and an important design aspect of the solution is the choice of knowledge representation.

The aim of this thesis is to answer important resource allocation related questions in different real-world application contexts and in the process contribute novel algorithms to the theory as well. The resource allocation algorithms considered include those from stochastic optimization, stochastic control and reinforcement learning. A number of new algorithms are developed as well. The application contexts selected encompass both single and multi-agent systems, abstract and concrete resources and contain high-dimensional state and control spaces. The empirical results from the various studies performed indicate that the algorithms presented here perform significantly better than those previously proposed in the literature. Further, the algorithms presented here are also shown to theoretically converge, hence guaranteeing optimal performance.

We now briefly describe the various studies conducted here to investigate problems of resource allocation under uncertainties of different kinds:

Vehicular Traffic Control The aim here is to optimize the 'green time' resource of the individual lanes in road networks that maximizes a certain long-term performance objective. We develop several reinforcement learning based algorithms for solving this problem. In the infinite horizon discounted Markov decision process setting, a Q-learning based traffic light control (TLC) algorithm that incorporates feature based representations and function approximation to handle large road networks is proposed, see Prashanth and Bhatnagar [2011b]. This TLC algorithm works with coarse information, obtained via graded thresholds, about the congestion level on the lanes of the road network. However, the graded threshold values used in the above Q-learning based TLC algorithm as well as several other graded threshold-based TLC algorithms that we propose, may not be optimal for all traffic conditions. We therefore also develop a new algorithm based on SPSA to tune the associated thresholds to the 'optimal' values (Prashanth and Bhatnagar [2012]). Our threshold tuning algorithm is online, incremental with proven convergence to the optimal values of thresholds. Further, we also study average cost traffic signal control and develop two novel reinforcement learning based TLC algorithms with function approximation (Prashanth and Bhatnagar [2011c]). Lastly, we also develop a feature adaptation method for 'optimal' feature selection (Bhatnagar et al. [2012a]). This algorithm adapts the features in a way as to converge to an optimal set of features, which can then be used in the algorithm.

Service Systems The aim here is to optimize the 'workforce', the critical resource of any service system. However, adapting the staffing levels to the workloads in such systems is nontrivial as the queue stability and aggregate service level agreement (SLA) constraints have to be complied with. We formulate this problem as a constrained hidden Markov process with a (discrete) worker parameter and propose simultaneous perturbation based simulation optimization algorithms for this purpose. The algorithms include both first order as well as second order methods and incorporate SPSA based gradient estimates in the primal, with dual ascent for the Lagrange multipliers. All the algorithms that we propose are online, incremental and are easy to implement. Further, they involve a certain generalized smooth projection operator, which is essential to project the continuous-valued worker parameter updates obtained from the SASOC algorithms onto the discrete set. We validate our algorithms on five real-life service systems and compare their performance with a state-of-the-art optimization tool-kit OptQuest. Being 25 times faster than OptQuest, our scheme is particularly suitable for adaptive labor staffing. Also, we observe that it guarantees convergence and finds better solutions than OptQuest in many cases.

Wireless Sensor Networks The aim here is to allocate the 'sleep time' (resource) of the individual sensors in an intrusion detection application such that the energy consumption from the sensors is reduced, while keeping the tracking error to a minimum. We model this sleep–wake scheduling problem as a partially-observed Markov decision process (POMDP) and propose novel RL-based algorithms - with both long-run discounted and average cost objectives - for solving this problem. All our algorithms incorporate function approximation and feature-based representations to handle the curse of dimensionality. Further, the feature selection scheme used in each of the proposed algorithms intelligently manages the energy cost and tracking cost factors, which in turn, assists the search for the optimal sleeping policy. The results from the simulation experiments suggest that our proposed algorithms perform better than a recently proposed algorithm from Fuemmeler and Veeravalli [2008], Fuemmeler et al. [2011].

Mechanism Design The setting here is of multiple self-interested agents with limited capacities, attempting to maximize their individual utilities, which often comes at the expense of the group's utility. The aim of the resource allocator here then is to efficiently allocate the resource (which is being contended for, by the agents) and also maximize the social welfare via the 'right' transfer of payments. In other words, the problem is to find an incentive compatible transfer scheme following a socially efficient allocation. We present two novel mechanisms with progressively realistic assumptions about agent types aimed at economic scenarios where agents have limited capacities. For the simplest case where agent types consist of a unit cost of production and a capacity that does not change with time, we provide an enhancement to the static mechanism of Dash et al. [2007] that effectively deters misreport of the capacity type element by an agent to receive an allocation beyond its capacity, which thereby damages other agents. Our model incorporates an agent's preference to harm other agents through a additive factor in the utility function of an agent and the mechanism we propose achieves strategyproofness by means of a novel penalty scheme. Next, we consider a dynamic setting where agent types evolve and the individual agents here again have a preference to harm others via capacity misreports. We show via a counterexample that the dynamic pivot mechanism of Bergemann and Välimäki [2010] cannot be directly applied in our setting with capacity-limited

agents. We propose an enhancement to the mechanism of Bergemann and Välimäki [2010] that ensures truthtelling w.r.t. capacity type element through a variable penalty scheme (in the spirit of the static mechanism). We show that each of our mechanisms is ex-post incentive compatible, ex-post individually rational, and socially efficient.

Publications

1. **Prashanth L. A.** and S. Bhatnagar, "Reinforcement Learning With Function Approximation for Traffic Signal Control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 2, pp. 412 – 421, 2011.

2. **Prashanth L. A.** and S. Bhatnagar, "Threshold Tuning using Stochastic Optimization for Graded Signal Control," *IEEE Transactions on Vehicular Technology*, vol. 61, no. 9, pp. 3865–3880, 2012.

3. **Prashanth L. A.** and S. Bhatnagar, "Reinforcement learning with average cost for adaptive control of traffic lights at intersections," in *Proceedings of the 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, 2011, pp. 1640–1645.

4. S. Bhatnagar, V. Borkar, and **Prashanth L. A.**, "Adaptive feature pursuit: Online adaptation of features in reinforcement learning," *Reinforcement Learning and Approximate Dynamic Programming for Feedback Control (Ed. F. Lewis and D. Liu), IEEE Press Computational Intelligence Series*, pp. 517–534, 2012.

5. **Prashanth L. A.**, H. L. Prasad, N. Desai, S. Bhatnagar and G. Dasgupta. "Stochastic optimization for adaptive labor staffing in service systems," in *Proceedings of 9th International Conference on Service Oriented Computing (ICSOC)*, 2011.

6. **Prashanth L. A.**, H. L. Prasad, N. Desai and S. Bhatnagar. "Mechanisms for Hostile Agents with Capacity Constraints," in *Proceedings of the Twelfth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2013 (Accepted).

7. H. L. Prasad, **Prashanth L. A.**, N. Desai and S. Bhatnagar. "Adaptive Smoothed Functional based Algorithms for Labor Cost Optimization in Service Systems," *Service Science (INFORMS)*, 2012.

8. **Prashanth L. A.**, H. L. Prasad, N. Desai, S. Bhatnagar and G. Dasgupta. “Simultaneous Perturbation Methods for Adaptive Labor Staffing in Service Systems,”, *Journal of Service Research*, 2013 (Under Review).

9. **Prashanth L.A.**, A.Chatterjee and S.Bhatnagar, “Reinforcement learning algorithms for adaptive sleep–wake control in sensor networks”, *IEEE Transactions on Communications*, 2012 (Under Revision).

10. **Prashanth L.A.** and S.Bhatnagar, “Control of traffic lights at junctions using reinforcement learning”, Proceedings of *Workshop on Computer Aided Transportation Planning and Traffic Engineering*, pp.129-138, Dec.7-11, 2009.

Chapter 1

Introduction

1.1 Motivation and Overview

A fundamental question in a sequential decision making setting under uncertainty is “how to allocate resources amongst competing entities so as to maximize the rewards accumulated in the long run?”. The resources allocated may be either abstract quantities such as time or concrete quantities such as manpower. The sequential decision making setting involves one or more agents interacting with an environment to procure rewards at every time instant and the goal is to find an optimal policy for choosing actions by each agent. Most of these problems involve multiple (often infinite) stages and the objective function is usually a long-run performance metric. The problem is further complicated by the uncertainties in the system, for instance, the stochastic noise and partial observability in a single-agent setting or private information of the agents in a multi-agent setting. The dimensionality of the problem also plays an important role in the solution methodology adopted. Most of the real-world problems involve high-dimensional state and action spaces and an important design aspect of the solution is the choice of knowledge representation.

The field of reinforcement learning, with its most significant algorithms (such as TD-learning and Q-learning) using full state representations, effectively addresses problems of the above kind, in settings where the state-space is manageable. However, several problems in practice possess high-dimensional state spaces and the above RL algorithms do not naturally extend here and one requires feature-based representations and function approximation to handle the curse of dimensionality. But, the function approximation based approaches fall short in two very important respects – first, the choice of features is based on the designer’s understanding of the problem and second, establishing convergence is non-trivial.

The aim of this thesis is to answer important resource allocation related questions in different real-world application contexts and in the process contribute novel algorithms to the theory as well. The resource allocation algorithms considered include those from stochastic optimization, stochastic control and reinforcement learning. A number of new algorithms are also developed. The application contexts selected encompass both single and multi-agent systems, abstract and concrete resources and contain high-dimensional state-action spaces. The empirical results from the various studies performed indicate that the algorithms presented here perform significantly better than those previously proposed in the literature. Further, the algorithms presented here are also shown to theoretically converge, hence guaranteeing optimal performance.

From a sequential decision-making viewpoint, the contributions of this thesis lie in the realm of stochastic optimization, learning in Markov decision processes (MDP) and equilibrium achievement in dynamic mechanism design problems. For instance, the first three studies involve problems that are modeled as MDPs, with either fully-observable or partially observable state, and the aim is to learn the optimal policy in a model-free setting. Further, there are several sub-problems in the first three studies where the goal is to optimize the long-run average of a noisy objective function, with the function values being only observable via simulation. The last study involves multiple agents with limited capacity in a mechanism design setting and the aim here is to design a suitable transfer scheme that ensures Nash equilibrium is attained.

We now provide an overview of the various resource allocation problems considered and describe our contributions area-wise below.

1.2 Study 1: Vehicular Traffic Control

We study the problem of maximizing traffic flow through the adaptive control of traffic lights at traffic intersections. From a resource allocation standpoint, green time is the resource and a traffic light control algorithm attempts to balance the green times allocated to the individual lanes of the road networks so as to optimize a long run performance objective. The traffic light control (TLC) algorithms use as input - coarse estimates of the congestion levels on the individual lanes of the road network as well as the times elapsed (for each lane) since the last signal light switch over. The congestion level input is considered to minimize the average junction waiting times of the road users, while the elapsed time input ensures fairness i.e., no lane is allowed to stay green for a long time at the cost of other lanes.

We develop several reinforcement learning based algorithms for solving the problem of adaptive traffic light control (TLC) in the infinite horizon Markov decision process setting. RL has the advantage of being model-free (it does not assume any model of the system) and provides online, incremental, easy to implement algorithms. However, since (quite clearly) this problem involves high dimensional states and actions (this will be even more so when we consider larger network scenarios), we develop feature based methods to tackle the accompanying problem of curse of dimensionality (i.e., the computational complexity with solving the associated Markov decision process increases exponentially with the dimension and cardinality of the state and action spaces). Feature-based methods serve to alleviate this problem by making computational complexity manageable but at the cost of accuracy. An advantage with an RL based implementation is that unlike previous works, we are able to prove that the given algorithm converges to an optimal policy (i.e., provides an optimal action in a given state). This is the case when feature-based approximations are not used. However, when they are used, we shall show that the policy to which the given algorithm converges is good enough.

The contributions in this context are summarized as follows:

- In the infinite horizon discounted Markov decision process setting, we develop a Q-learning based TLC algorithm that incorporates feature based representation and function approximation to handle larger road networks (Prashanth and Bhatnagar [2011b]). This TLC algorithm works with only coarse information, obtained via graded thresholds, about the congestion level on the lanes of the road network.

- Considering the fact that the graded threshold values used in the above Q-learning based TLC algorithm as well as several other graded threshold-based TLC algorithms that we propose, are in general not optimal for all traffic conditions, we develop a new algorithm based on SPSA to tune the associated thresholds to their 'optimal' values (Prashanth and Bhatnagar [2012]). Our threshold tuning algorithm is online, incremental with proven convergence to the optimal values of the thresholds.

- We also study average cost traffic signal control and develop two novel reinforcement learning based TLC algorithms with function approximation (Prashanth and Bhatnagar [2011c]) for solving this problem.

- Finally, we have also developed a feature adaptation method for 'optimal' feature selection (Bhatnagar et al. [2012a]). This algorithm adapts the features in a way as to converge to an optimal set of features, which can then be used in the algorithm. A significant advantage of our algorithm is that we keep the number of columns in the feature matrix fixed.

1.3 Study 2: Service Systems

Service systems (SS) are labor intensive systems and have time varying workload. Adapting the staffing levels to the workloads in such systems is crucial for business objectives such as obtaining minimal labor inventory. However, it is a nontrivial task because of a large number of parameters and operational variations. In this context, the resource allocation problem that we address is of optimizing the staffing levels while maintaining the system in steady-state and compliant to aggregate service level agreement (SLA) constraints.

The staffing levels constitute the worker parameter that we optimize and specify the number of workers in each shift and of each skill level. The SLA constraints specify the target resolution time and the aggregate percentage for an service request (SR) originating from a particular customer and with a specified priority level. For instance, a sample SLA constraint could specify that 95% of all SRs from customer 1 with 'urgent' priority must be resolved within 4 hours. While the need for SLA constraints to be met is obvious, the requirement for having queues holding unresolved SRs bounded is also necessary because SLA attainments are calculated only for the work completed.

We formulate this problem as a constrained hidden Markov cost process that depends on the worker parameter. To have a sense of the size of the search space, an SS consisting of 30 service workers (SWs) who work in 6 shifts and 3 distinct skill levels corresponds to more than 2 trillion configurations. Apart from the high cardinality of the discrete parameter set, the constrained Markov cost process involves a hidden or un-observed state component. We design a novel single stage cost function for the constrained Markov cost process that balances the conflicting objectives of worker under-utilization and SLA under/over-achievement. An SLA under-achievement implies violation of the SLA constraint. Whereas worker under-utilization clearly points to suboptimal staffing, SLA over-achievement points to over-delivery and hence is also suboptimal. The performance objective is a long-run average of this single stage cost function and the goal is to find the optimum steady state worker parameter (i.e., the one that minimizes this objective) from a discrete high-dimensional parameter set. However, our problem setting also involves constraints relating to queue stability and SLA compliance. Thus, the optimum worker parameter is in fact a constrained minimum. Another difficulty with finding the optimum (constrained) worker parameter is that the single stage cost and constraint functions can be estimated only via simulation. Hence, the need is for a simulation-optimization algorithm that incrementally updates the worker parameter along a descent direction, while adhering to a set of queue stability and SLA constraints.

We develop three novel stochastic optimization algorithms for solving the staffing optimization problem in a service system. The core of each of the algorithms is a multi-timescale stochastic approximation scheme that incorporates a random perturbation based algorithm for ‘primal descent’ and couples it with a ‘dual ascent’ scheme for the Lagrange multipliers. The first order algorithm (Prashanth et al. [2011b]) proposes the simultaneous perturbation stochastic approximation (SPSA) based technique for gradient estimation in the primal. We also develop two second order (Newton) methods (Prashanth et al. [2012b]). The first algorithm here estimates the Hessian of the objective function using SPSA and involves finding an explicit inverse, while the second is a novel algorithm that leverages Woodbury’s identity to directly estimate the inverse of the Hessian and thereby achieves compututational efficiency. These algorithms are found to perform significantly better than the state-of-the-art OptQuest optimization toolkit, which is being used in IBM’s pool simulation project.

1.4 Study 3: Sensor Networks

We study the problem of tracking a single object moving randomly through a dense network of wireless sensors such that the entire sensing region has no discontinuities. However, we wish to conserve energy by optimizing the sleep times of the sensors in a way that does not compromise on the tracking quality. We remove the waking channel assumption, i.e., a sleeping sensor can be communicated with. In this context, the resource considered is the ‘sleep time’ of the individual sensors and the objective is to manage the sleep times of the sensors efficiently, while keeping the tracking error to a minimum.

We model the sleep–wake scheduling problem as a partially-observed Markov decision process (POMDP). We propose novel RL-based algorithms - with both long-run discounted and average cost objectives - for solving this problem. All our algorithms incorporate function approximation and feature-based representations to handle the curse of dimensionality. Further, the feature selection scheme used in each of the proposed algorithms intelligently manages the energy cost and tracking cost factors, which in turn, assists the search for the optimal sleeping policy. The results from the simulation experiments suggest that our proposed algorithms perform better than a recently proposed algorithm from Fuemmeler and Veeravalli [2008], Atia et al. [2010].

1.5 Study 4: Mechanism Design

Mechanism design has been a key technique for resource allocation in various economic scenarios. The setting involves self-interested agents, with private information, competing for resources and a central controller is entrusted with the responsibility of efficiently allocating resources based on the reported types of the agents and also, more importantly, achieve truth-telling on the agents' part via a suitably incentivized transfer scheme. Now, if we repeat this process, i.e., agents' reports followed by allocation and transfer, over the infinite horizon, then design of incentive-compatible transfer strategies comes under the purview of dynamic mechanism design. Note that, in the dynamic case, the agents' types change with time. This is quite realistic as in many of the economic scenarios of interest, the agents' types evolve, e.g., the cost of production of a good. Also, in most real economic scenarios, agents have limited capacities that change with time, for instance, the supply capacity of a power plant.

Unlike the earlier studies where the uncertainty involved was either due to the stochasticity or partial observability of the system, here the uncertainty is due to the private information of the agents. The central controller has to efficiently allocate resources while being unaware of agents' true types. Or equivalently, the decision making problem here is of achieving a desirable equilibrium point. To solve this problem, we develop novel mechanisms for the domains that lie at the intersection of these two scenarios, i.e., agents having limited capacities and unit costs of production that change with time. Unfortunately, capacity constraints are nontrivial to accommodate in the design of truthful and socially efficient dynamic mechanisms. The main challenge is that in a given time step, the payoff to an agent cannot be determined until later when the actual number of units produced by an agent is known. The question then is what should the payoff to an agent be once he completes the allocated task such that the resulting mechanism is truthful in terms of unit costs as well as capacities and is socially efficient? We establish via a counterexample that providing the marginal contribution alone is not sufficient to ensure incentive compatibility of the mechanism. For this purpose, a penalty component is necessary in the transfer to the agents. Our contributions can be summarized as follows:

- We first present a novel static mechanism that, unlike Dash et al. [2007], incorporates a variable penalty component. Our model incorporates an agent's preference to harm other agents through a additive factor in the utility function of an agent and the mechanism we propose achieves strategyproofness by means of a novel penalty scheme. Here, the penalty imposed on agents misreporting their capacities is the same as the loss of allocation to other agents due to misreported capacities.
- Next, we consider a dynamic setting where agent types evolve and the individual agents here again have a preference to harm others via capacity misreports. We show via a counterexample that the dynamic pivot mechanism of Bergemann and Välimäki [2010] cannot be directly applied in our setting with capacity-limited agents. We propose an enhancement to the mechanism of Bergemann and Välimäki [2010] that ensures truthtelling w.r.t. capacity type element through a variable penalty scheme (in the spirit of the static mechanism).
- We establish within period ex-post incentive compatibility, allocative efficiency, and individual rationality of all the above mechanisms.

Our proposed mechanisms, esp. the dynamic variant is applicable to several real-world settings involving capacity constraints. To name a few, work dispatch in service systems, purchase of electricity from power plants, resource allocation for computational grids and sensor networks are some application scenarios where our proposed dynamic mechanism can be effective, unlike the dynamic pivot mechanism Bergemann and Välimäki [2010] which does not guarantee truthful capacity reports in the limited capacity setting.

1.6 Outline of chapters

Chapter 2

Chapter 2 provides the background material on stochastic optimization and reinforcement learning. An introduction to the MDP framework and the well-known dynamic programming (DP) algorithms are presented first. Thereafter, the classic reinforcement learning algorithms, which include temporal-difference (TD) learning and Q-learning, are described. The last topic covered in this context includes the important idea of function approximation to handle the curse of dimensionality associated with high-dimensional state spaces.

In the area of stochastic optimization, we briefly cover the well-known SPSA algorithm. The topics covered in this chapter essentially provide the foundation for the entire thesis, as we develop several resource allocation algorithms in a variety of application contexts using stochastic optimization and reinforcement learning algorithms.

Chapter 3

Chapter 3 motivates the need for adaptive control of traffic lights at intersections to maximize traffic flow in the long-term. This chapter also surveys relevant literature on traffic signal control strategies as well as the important developments in stochastic optimization and reinforcement learning areas. The subsequent chapters then develop novel reinforcement learning based algorithms for adaptive traffic signal control.

Chapter 4

Chapter 4 provides an introduction to the problem of maximizing the traffic flow in the long term through adaptive control of traffic lights at intersections and models it as a Markov decision process (MDP). Then, we present a Q-learning based traffic light control (TLC) algorithm that uses full-state representations and study its performance on a single junction and a two-junction corridor, respectively. Empirically, we observe that the Q-learning based TLC algorithm outperforms several existing TLC algorithms, demonstrating the usefulness of RL in this context. However, the full-state Q-learning algorithm does not scale well to larger network scenarios. A particularly useful technique to handle the curse of dimensionality is function approximation, a topic handled in the next chapter.

Chapter 5

Chapter 5 presents a Q-learning based TLC algorithm that incorporates feature-based representations and function approximation to handle the curse of dimensionality associated with larger road networks. An advantage of this algorithm is that unlike prior work based on RL, it does not require precise information on queue lengths and elapsed times at each lane, but instead works with the coarse information about the congestion levels in the road network. From the simulation experiments, we observe that our algorithm outperforms other TLC algorithms previously proposed in the literature on all the road network settings considered.

Chapter 6

Chapter 6 addresses the case when an infinite horizon average cost setting is considered, unlike the discounted cost setting of Chapter 5. We develop two novel reinforcement learning algorithms with function approximation for average cost adaptive control of traffic lights. One of these algorithms is a version of Q-learning with function approximation while the other is a policy gradient actor-critic algorithm that incorporates multi-timescale stochastic approximation. We show performance comparisons on various network settings of these algorithms with a range of fixed timing algorithms, as well as a Q-learning algorithm with full state representation that we also implement. We observe that whereas (as expected) on a two-junction corridor, the full state representation algorithm shows the best results, this algorithm is not implementable on larger road networks. The policy gradient actor-critic TLC algorithm that we propose is seen to show the best overall performance.

Chapter 7

Chapter 7 presents an algorithm based on stochastic optimization to tune the thresholds associated with a TLC algorithm for optimal performance. The thresholds associated with a TLC algorithm are on the queue lengths on the lanes of the road network considered, as precise information is hard to obtain in practice. Many TLC algorithms proposed previously in the literature incorporate fixed values for the thresholds, which in general are not optimal for all traffic conditions. Our tuning algorithm is an incremental update, online scheme with proven convergence to the optimal values of thresholds and the additional computational effort required because of the integration of the tuning scheme in any of the graded threshold based TLC algorithms is minimal. We also present three novel TLC algorithms - a full state Q-learning algorithm with state aggregation, a Q-learning algorithm with function approximation involving an enhanced feature selection scheme and a priority based TLC scheme, respectively. All these algorithms are threshold-based. Next, we combine the threshold tuning algorithm with the three aforementioned algorithms and discuss the interesting consequences that arise from this combination. We demonstrate empirically the usefulness of our tuning algorithm by comparing the performance of the various TLC algorithms with their counterparts without tuning, i.e., with fixed thresholds.

Chapter 8

Chapter 8 presents a novel feature adaptation scheme based on temporal difference learning for the problem of prediction. The scheme suitably combines aspects of *exploitation* and *exploration* by (a) finding the worst basis vector in the feature matrix at each stage and replacing it with the current best estimate of the normalized value function, and (b) replacing the second worst basis vector with another vector chosen randomly that would result in a new subspace of basis vectors getting picked. We demonstrate the empirical usefulness of our algorithm by applying it to a problem of prediction in traffic signal control.

Chapter 9

Chapter 9 provides the background on service systems and also formalizes the adaptive labor staffing problem. We define the service system framework and introduce the important problem of optimizing the staffing levels while maintaining the system in steady-state and compliant to aggregate SLA constraints. The problem is nontrivial because of a large number of parameters and operational variations leading to a huge search space. Further, because these parameters change on a weekly basis, the optimization should not take longer than a few hours. We formulate this problem as a constrained hidden Markov cost process parameterized by the (discrete) staffing levels.

Chapter 10

Chapter 10 presents three novel discrete optimization methods SASOC-G, SASOC-H, and SASOC-W for solving the staffing optimization problem described in Chapter 9. The core of each of the algorithms is a multi-timescale scheme that incorporates a SPSA based algorithm for primal descent and couples it with a dual ascent scheme for the Lagrange multipliers. Further, all SASOC algorithms incorporate a generalized projection operator that helps imitate a continuous parameter system, thus facilitating the use of the simultaneous perturbation technique. The SASOC algorithms differ in the gradient estimation and optimization methods used. Whereas SASOC-G is a first-order gradient estimation method, SASOC-H and SASOC-W are second-order Newton methods. Convergence proofs of all the SASOC algorithms are also presented in this chapter.

Chapter 11

Chapter 11 presents the results from empirical evaluation of the SASOC algorithms described in Chapter 10 on data from five real-life service systems. These results also include comparisons with a state-of-the-art optimization tool-kit OptQuest. We observe that our algorithms are more than an order of magnitude faster than OptQuest and thus are particularly suitable for adaptive labor staffing. Also, we observe that our schemes guarantee convergence and find better solutions than OptQuest in many cases. Amongst our algorithms, SASOC-H and SASOC-W find better solutions than SASOC-G in many cases, with SASOC-W being marginally better.

Chapter 12

Chapter 12 provides the background on the intrusion detection application using wireless sensor networks. The sensors are deployed to cover a given area and the objective is to track an intruder moving through this network. The movement of the intruder is assumed random with a given distribution. In this setting, we consider the problem of optimizing the energy of each sensor through sleep-wake cycling while detecting the intruder to a reasonable accuracy. The problem is formulated as an MDP in this chapter and the associated state, actions and single-stage cost functions are presented.

Chapter 13

Chapter 13 presents novel RL-based algorithms - with both long-run discounted and average cost objectives - for solving the sleep-wake scheduling problem described in Chapter 12. All our algorithms incorporate function approximation and feature-based representations to handle the curse of dimensionality. Further, the feature selection scheme used in each of the proposed algorithms intelligently manages the energy cost and tracking cost factors, which in turn, assists the search for the optimal sleeping policy.

Chapter 14

Chapter 14 presents the results from the empirical evaluation of the sleep-wake scheduling algorithms presented in Chapter 13. From the simulation experiments, we observe that our algorithms perform significantly better than a recently proposed algorithm (Fuemmeler and Veeravalli [2008]). Amongst our algorithms, we observe that the two-timescale Q-learning variant performs marginally better than Q-learning with function approximation, with the former algorithm being provably convergent unlike the latter.

Chapter 15

Chapter 15 introduces a static mechanism setting where the individual agents have limited capacity. A few mechanisms have been proposed to account for limited capacities in static settings. We present a novel static mechanism that involves a variable penalty scheme. Here, the penalty imposed on agents misreporting their capacities is the same as the loss of allocation to other agents due to misreported capacities. We formally establish that the proposed mechanism is strategyproof and individually rational.

Chapter 16

Chapter 16 considers a dynamic setting for a procurement scenario, where the agent types include the unit price as well as capacity elements and the agent types evolve with time. This setting allows us more realistic assumptions about agent types aimed at economic scenarios where agents have limited capacities that change with time. With an example, we show how a well-known VCG like dynamic pivot mechanism (Bergemann and Välimäki [2010]) would not ensure incentive compatibility in this setting. We propose an enhancement to the dynamic pivot mechanism Bergemann and Välimäki [2010] by incorporating a delayed penalty based transfer scheme and establish that this results in truthful reporting of capacities. The proposed mechanism, $\mathcal{DMC}$, compensates for the delay in transfer by adjusting the payoffs upwards based on a discount factor and penalizes an agent only to the extent of the damage caused by his capacity misreport (in the spirit of the static mechanism $\mathcal{MC}$ described in Chapter 15).

Chapter 17

Chapter 17 concludes the thesis with a brief summary of results from the studies on resource allocation algorithms in various application contexts and also discusses some important future research directions.

Chapter 2

Background

This chapter provides background on stochastic optimization and reinforcement learning. An introduction to the MDP framework and the well-known dynamic programming (DP) algorithms are presented first. Thereafter, the classic reinforcement learning algorithms, which include temporal-difference (TD) learning and Q-learning, are described. The last topic covered here includes the important idea of function approximation to handle the curse of dimensionality associated with high-dimensional state spaces. For an excellent introduction to reinforcement learning methods, the reader is referred to Bertsekas and Tsitsiklis [1996b], Sutton and Barto [1998b]. In the area of stochastic optimization, we briefly discuss the well-known SPSA algorithm.

The topics covered in this chapter essentially provide the foundation for the entire thesis, as we develop several resource allocation algorithms in a variety of application contexts using stochastic optimization and reinforcement learning techniques.

2.1 Markov decision processes

2.1.1 The framework

A stochastic process $\{X_n\}$ taking values in a set S is called a Markov decision process (MDP) if its evolution is governed by a control-valued sequence $\{Z_n\}$ and satisfies the controlled Markov property:

$$Pr(X(n+1)=j \mid X(n)=i, Z(n)=a, X(n-1)=i_{n-1}, Z(n-1)=a_{n-1},\ldots,\\ X_0=i_0, Z_0=a_0) \triangleq p(i,j,a). \tag{2.1}$$

for any $i_0,\ldots,i_{n-1},i,j,a_0,\ldots,a_{n-1},a$, in appropriate sets. We assume here that if $X_n=i$ for any n, the set of feasible actions or controls is $A(i)$. Thus, in (2.1), $a \in A(i)$, $a_{n-1} \in A(i_{n-1})$ etc. Let $A = \cup_{i\in S} A(i)$ denote the control space (i.e., the set of all controls). We assume that both S and A are finite sets. Depending on the current system state, the decision maker or controller picks a control in a way as to minimize a long-term cost. We consider the infinite horizon discounted cost criterion for this purpose. Let $k(i,a)$ denote the single-stage cost incurred when the system is in state i and action $a \in A(i)$ is chosen. Further, when the state is i and action a is chosen, the next state is j with a probability of $p(i,j,a)$. Let $\gamma \in (0,1)$ denote the discount factor.

A sequence of functions $\pi = \{\mu_1, \mu_2, \ldots\}$ with each $\mu_n : S \to A$, $n \geq 1$, is said to be an admissible policy if $\mu_n(i) \in A(i)$, $\forall i \in S$. This corresponds to the control choice $Z_n = \mu_n(X_n)$ $\forall n$. Let Π denote the set of all admissible policies. An admissible policy $\pi = \{\mu_1, \mu_2, \ldots\}$ with each $\mu_n = \mu$, $n \geq 1$, is said to be a stationary deterministic policy (SDP). By a common abuse of notation, we simply refer to μ itself as the SDP.

For a given admissible policy $\pi \in \Pi$, the value function $V^\pi : S \to \mathcal{R}$ is defined by

$$V^\pi(i) = E\left[\sum_{m=0}^{\infty} \gamma^m k(X_m, \mu_m(X_m)) \mid X_0 = i\right], \tag{2.2}$$

for all $i \in S$. The aim then is to find an optimal value function $V^* : S \to \mathcal{R}$, i.e.,

$$V^*(i) = \min_{\pi\in\Pi} V^\pi(i). \tag{2.3}$$

It is also well known, see Puterman [1994], that an SDP achieves the optimal policy, i.e., the one that corresponds to the optimal value $V^*(i)$ $\forall i \in S$. Further, the optimal value function $V^*(\cdot)$ satisfies the Bellman equation of optimality:

$$V^*(i) = \min_{a\in A(i)} \left(k(i,a) + \gamma \sum_{j\in S} p(i,j,a) V^*(j)\right), \tag{2.4}$$

for all $i \in S$.

2.1.2 DP algorithms

Dynamic programming (DP) is used to obtain the optimal policy given an MDP. While in practice, a model of the system such as MDP is not available, a theoretical understanding of DP algorithms is still necessary. RL methods attempt to obtain the optimal policy at a significantly lower computation cost as compared to DP and also do not assume a perfect model of the system.

Policy Iteration The *policy iteration* algorithm works by performing alternately a sequence of policy evaluation and policy improvement steps, until an optimal policy is achieved. In other words, the idea here is to start with a policy π_0 and estimate V^{π_0}. Following this step, we generate a new policy π_1 which is an improved policy as compared to π_0. This is procedure is repeated until the optimal policy is obtained, which happens in a finite number of steps since a finite MDP has only a finite number of policies.

The policy evaluation step computes V^{π_n}, where π_n is the policy that is evaluated according to $V_T^{\pi_n} = (I - \alpha P^{\pi_n})^{-1} k$, where $k = (k(1, \pi_n(1)), \ldots, k(|S|, \pi_n(|S|)))^T$ is the column vector of single-stage costs under policy π_n and P^{π_n} is the transition probability matrix of the Markov chain under policy π_n. In the case when the above inversion of the matrix is computationally difficult, we follow a value iteration type procedure (explained below for the general case) to estimate the value of π_n as follows:

$$V_{k+1}^{\pi_n}(i) = \left(k(i, \pi_n(i)) + \alpha \sum_{j \in S} p(i, j, \pi(i)) V_k^{\pi_n}(j) \right), \text{ for } k = 0, 1, 2, \ldots. \quad (2.5)$$

In the above, $V_0^{\pi_n}$ can be initialized arbitrarily. As $k \to \infty$, it can be shown that $V_k^{\pi_n}(i) \to V^{\pi_n}(i)$.

The policy improvement step improves the policy by selecting at each state the action that appears best, i.e.

$$\pi_{n+1}(i) \in \arg\min \left(r(i, \cdot) + \alpha \sum_{j \in S} p(i, j, \cdot) V^{\pi_n}(j) \right). \quad (2.6)$$

Value Iteration The *value iteration* algorithm is similar to policy iteration except that the policy evaluation is stopped after one iteration i.e.

$$V_{n+1}(i) = \min_{a \in A(i)} \left(r(i,a) + \alpha \sum_{j \in S} p(i,j,a) V_n(j) \right). \tag{2.7}$$

Essentially, the Bellman optimality condition (2.4) is turned into an update rule in value iteration. In the above, one can start with an arbitrary initial estimate $V_0(\cdot)$ of the value function.

2.2 Reinforcement Learning

Reinforcement learning can be applied as a tool on any problem modeled as a MDP, i.e., a setting where the system evolves probabilistically over states, incurring state/action-dependent costs at each instant and the goal is to find a policy for choosing actions that minimize a long-run cost objective. While this constitutes the setting for DP algorithms as well, the main difference with RL algorithms is that the latter are model-free,i.e., the transition probabilities of the underlying MDP are unknown. RL combines numerical procedures to solve the DP equations together with stochastic approximation and works only with simulated observations, or alternately real data.

However, such an approach would again suffer from the curse of dimensionality of DP methods, i.e., the exponential blow up of the computation with the increase in the state space. RL tackles this problem by combining ideas from dynamic programming and supervised learning. DP provides the algorithms for optimization and control, but at a high computational expense, whereas supervised learning provides ways of training a parametrized function approximator. RL algorithms are stochastic-approximation variants of well-known DP methods such as value and policy iteration. Further, they employ feature based representations and function approximation to effectively handle the curse of dimensionality.

2.2.1 Temporal-difference learning

A key algorithm in RL is *TD learning*. The problem here is to estimate the value function V_π for a given policy π. Traditional Monte-Carlo methods for estimating the value of a state i do so by averaging sample returns. In other words, the idea in Monte-Carlo methods is to perform a number of simulation runs and obtain the empirical average of the cost samples from the generated trajectories.

$$V^\pi_{n+1}(i) = V^\pi(i) + \alpha(m)\left(c(i,m) - V^\pi_n(i)\right), \tag{2.8}$$

where $c(i,m)$ is the discounted cumulative cost obtained from a trajectory and $\alpha(m) = \frac{1}{m}$ is the step-size parameter. On the other hand, unlike Monte-Carlo methods, TD methods do not wait until the end of the trajectory to update their estimate $V^\pi(i)$. For example, in TD(0) at time instant $n+1$ an update happens using the observed cost $g(i,a)$ and the estimate of the next state $V^\pi(j)$ i.e.

$$V^\pi(i) = V^\pi(i) + \alpha(m)\left(g(i,a) + \gamma V^\pi(j) - V^\pi(i)\right). \tag{2.9}$$

In essence,TD methods learn their estimates on the basis of other estimates i.e. they bootstrap. The advantage here is that they do not need a model of the environment (as DP does) and they are on-line incremental algorithms as compared to Monte Carlo methods where one needs to wait till the end of a trajectory to know the return. TD methods are shown to converge to V^π in the mean for a constant sufficiently small step-size and with probability 1 if the step-size decreases according to the standard stochastic approximation conditions.

2.2.2 Q-learning with full state representations

In practice the quantities $p(i,j,a)$ are usually not known. Reinforcement learning (RL) techniques are useful in such scenarios. The idea there is that in order to solve (2.4), one runs a stochastic iterative algorithm using observations obtained from online samples. It is then shown that the algorithm asymptotically converges to an optimal value function – policy tuple. An important RL algorithm goes by the name Q-learning (Watkins and Dayan [1992a]). Here one defines Q-values $Q(i,a)$, $i \in S$, $a \in A(i)$ as follows:

$$Q(i,a) = \left(k(i,a) + \gamma \sum_{j\in S} p(i,j,a) V^*(j)\right). \tag{2.10}$$

From (2.4), it is easy to see that

$$V^*(i) = \min_{a\in A(i)} Q(i,a). \tag{2.11}$$

From (2.11) and (2.10), one obtains the following form of the Bellman equation of optimality that is also many times called the Q-Bellman equation:

$$Q(i,a) = \left(k(i,a) + \gamma \sum_{j \in S} p(i,j,a) \min_{b \in A(j)} Q(j,b) \right). \tag{2.12}$$

Whereas in (2.4), the minimization is immediately to the right of the equality, in (2.12), the same gets pushed inside the summation on the right. The reason for this is that we are no longer looking at just state-values but at values of state-action tuples. The stochastic approximation version of (2.12) is the following online incremental algorithm that is called the Q-learning algorithm:

$$Q_{n+1}(i,a) = Q_n(i,a) + a(n) \left(k(i,a) + \gamma \min_{b \in A(\eta_n(i,a))} Q_n(\eta_n(i,a), b). - Q_n(i,a) \right). \tag{2.13}$$

One can start this algorithm by initializing values of all $Q_0(i,a)$ arbitrarily, a simple choice is to simply set them all to zero. In the above, $\eta_n(i,a)$, $n \geq 0$ are independent and identically distributed (i.i.d.) random variables that have the distribution $p(i,\cdot,a)$, i.e., $\eta_n(i,a) = j$ with probability $p(i,j,a)$. Also, $a(n)$, $n \geq 0$ are (positive) step-sizes that satisfy the conditions

$$\sum_n a(n) = \infty, \quad \sum_n a(n)^2 < \infty. \tag{2.14}$$

The first condition above ensures that the algorithm does not converge prematurely while the second ensures that the noise in the algorithm vanishes asymptotically. Most often, the step-sizes are simply chosen to be $a(n) = \frac{c_0}{c_1 + n}$ for some positive constants c_0 and c_1. The convergence of this algorithm has been analyzed in Watkins and Dayan [1992a], Tsitsiklis [1994a] and Borkar and Meyn [2000]. Upon convergence, one obtains the optimal Q-value function and hence also the optimal policy.

2.2.3 Q-learning with function approximation

Note that the Bellman equation for optimality (2.4) requires solving a system of equations in $|S|$ variables. Similarly, a solution to the Q-Bellman equation (2.12) requires solving a system of equations in $|S \times A(S)|$ unknowns. Here $S \times A(S)$ denotes the set of all feasible state-action tuples.

In the setting of Q-learning with function based approximation, the idea is to approximate the Q-value function $Q(s, a)$ as

$$Q(s, a) \approx \theta^T \sigma_{s,a}, \tag{2.15}$$

where $\sigma_{s,a}$ is a d-dimensional feature (column) vector corresponding to the state-action tuple (s, a), with $s \in S$ and $a \in A(s)$. The dimension d is significantly less in comparison to the cardinality of the set of feasible state-action tuples (s, a). Here θ is a tunable parameter whose dimension is the same as that of $\sigma_{s,a}$.

Let Φ denote a matrix with rows $\sigma^T_{s,a},\ s \in S, a \in A(s)$. The number of rows of this matrix is thus $|S \times A(S)|$, while the number of columns is d. Let

$$\sigma_{s,a} = (\sigma_{s,a}(1), \ldots, \sigma_{s,a}(d))^T.$$

Then $\Phi = (\Phi(i), i = 1, \ldots, d)$ where $\Phi(i)$ is the column vector

$$\Phi(i) = (\sigma_{s,a}(i), s \in S, a \in A(s))^T,\ i = 1, \ldots, d.$$

Let $\theta = (\theta_1, \ldots, \theta_d)^T$. Then,

$$Q \approx \sum_{i=1}^{d} \Phi(i)\theta_i,\ \text{or alternatively, } Q \approx \Phi\theta,$$

where $Q = [Q(s, a), s \in S, a \in A(s)]^T$ is the vector of the Q-values $Q(s, a)$ over all feasible (s, a) tuples. In other words, Q is approximated using $\Phi\theta$. Note that the gradient of the approximate Q-value function w.r.t. θ is

$$\nabla_\theta Q(s, a) \approx \sigma_{s,a}.$$

The update rule of the Q-learning algorithm with function approximation is given by

$$\theta_{n+1} = \theta_n + \alpha(n)\sigma_{s_n,a_n}(k(s_n, a_n) + \gamma \min_{v \in A(s_{n+1})} \theta_n^T \sigma_{s_{n+1},v} - \theta_n^T \sigma_{s_n,a_n}), \tag{2.16}$$

where θ_0 is set arbitrarily. In (2.16), the action a_n is chosen in state s_n according to $a_n = arg\min_{v \in A(s_n)} \theta_n^T \sigma_{s_n,v}$. A routine requirement used to prove convergence of function approximation based algorithms is that the columns $\Phi(i), i = 1, \ldots, d$ of the feature matrix Φ be linearly independent. It is in general difficult to prove the convergence of Q-learning with function approximation because of the nonlinearity in its update rule resulting from the explicit minimization, as a result of which, Q-learning with function approximation suffers from the off-policy problem, see however Melo and Ribeiro [2007].

2.3 Simultaneous perturbation stochastic approximation (SPSA)

SPSA, is very effective in addressing high-dimensional stochastic optimization problems. Standard SPSA requires only two system simulations for estimating the gradient of the objective function by perturbing all parameter components along random directions. Commonly used perturbation random variables are independent, symmetric, ± 1-valued and Bernoulli distributed.

The standard stochastic approximation algorithm is of the following type:

$$\hat{\theta}_{k+1} = \hat{\theta}_k + a_k Y_k, \tag{2.17}$$

where $\hat{\theta}_k \equiv (\hat{\theta}_{k,1}, \ldots, \hat{\theta}_{k,N})^T, k \geq 0$ are tunable parameter vectors, a_k correspond to step-sizes and $Y_k \equiv (Y_{k,1}, \ldots, Y_{k,N})^T$ are certain 'criterion functions'. The finite-difference Kiefer-Wolfowitz algorithm, on the other hand, tries to minimize the objective function $F(\theta)$ by finding zeroes of $\nabla F(\theta)$ using a gradient descent algorithm where the gradient estimate has the form:

$$Y_{k,i} = -\left(\frac{f(\hat{\theta}_k + \delta_k e_i, \xi^+_{k,i}) - f(\hat{\theta}_k - \delta_k e_i, \xi^-_{k,i})}{2\delta_k}\right), i = 1, \ldots, N, \tag{2.18}$$

where $\delta_k \to 0$ as $k \to \infty$ 'slowly enough', $\xi^+_{k,i}$, $\xi^-_{k,i}$ are i.i.d., and e_i is the unit vector with 1 in the ith place and 0's elsewhere. The disadvantage with this approach is that at any update epoch k, one requires $2N$ samples of the objective function $f(\cdot, \cdot)$.

On the other hand, SPSA requires only two samples of the objective function at each update epoch, as follows:

$$Y_{k,i} = -\left(\frac{f(\hat{\theta}_k + \delta_k \triangle(k), \xi_k^+) - f(\hat{\theta}_k - \delta_k \triangle(k), \xi_k^-)}{2\delta_k \triangle_i(k)}\right), \tag{2.19}$$

where ξ_k^+, ξ_k^- are i.i.d. random vectors in a suitable finite-dimensional Euclidean space, $\triangle(k) \equiv (\triangle_1(k), \ldots, \triangle_N(k))^T$, where $\triangle_i(k)$, $i = 1, \ldots, N$, $k \geq 0$, are i.i.d., zero-mean random variables with $P(\triangle_i(k) = 0) = 0$. The convergence analysis of this algorithm can be seen, for instance, in Spall [1992],[Bhatnagar et al., 2012b, Theorem 5.1].

Remark 1 *In Bhatnagar et al. [2003], variants of the SPSA algorithm, where the averaging out of undesirable gradient directions is achieved in a less "noisy" fashion using certain deterministic perturbation sequences are described and analyzed. Here the perturbation sequences are obtained using certain deterministic constructions instead of randomized.*

Remark 2 *While the above SPSA algorithm is for an expected cost objective $J(\theta(n)) = E_\xi[h(\theta(n), \xi)]$, one could also obtain variants of the algorithm for a long-run average cost objective $J(\theta)$ given by*

$$J(\theta) = \lim_{l \to \infty} \frac{1}{l} \sum_{j=0}^{l-1} h(X_j). \tag{2.20}$$

The reader is referred to [Bhatnagar et al., 2012b, Section 5.6] for a detailed introduction to this subject matter.

Part I

Vehicular Traffic Control

This part discusses stochastic optimization and reinforcement learning methods developed to solve the problem of traffic signal control. In the infinite horizon discounted Markov decision process setting, a Q-learning based TLC algorithm that incorporates feature based representations and function approximation to handle larger road networks was proposed (Prashanth and Bhatnagar [2011b]). This TLC algorithm works with only coarse information, obtained via certain graded thresholds on the congestion levels on the various lanes of the road network. However, the graded threshold values used in the above Q-learning based TLC algorithm as well as several other graded threshold-based TLC algorithms that we propose, are in general not optimal for all traffic conditions. We develop a new algorithm based on SPSA to tune the associated thresholds to their 'optimal' values (Prashanth and Bhatnagar [2012]). Our threshold tuning algorithm is online, incremental with proven convergence to the optimal values of thresholds. Further, we also studied average cost traffic signal control and developed two novel reinforcement learning based TLC algorithms with function approximation (Prashanth and Bhatnagar [2011c]). Lastly, we have also developed a feature adaptation method for 'optimal' feature selection (Bhatnagar et al. [2012a]). This algorithm adapts the features in a way as to converge to an optimal set of features, which can then be used in the algorithm.

Chapter 3

Introduction

With increasing traffic in urban areas and limitations of the road infrastructure, any attempt to improve the traffic flow of the system would involve an intelligent design of the traffic signal timings for the junctions. The traffic junctions play a very important role in determining congestion state of the road network. Many traffic junctions worldwide currently use fixed signal timings, i.e., they periodically cycle through the sign configurations in a round-robin manner. Even though such a strategy is easy to implement, it does not take into consideration the actual traffic conditions and may result in more congestion. In this chapter, we study the use of a reinforcement learning (RL) based traffic control system. The objective is to maximize traffic flow by performing adaptive control of traffic lights at intersections. Sensors, placed along the lanes leading to a junction, periodically convey the traffic information to a controller or agent, which then decides on the signal timings.

3.1 Literature Survey

We now review literature in two different areas of related work: (1) techniques pertaining to traffic signal control and (2) developments in stochastic optimization approaches.

3.1.1 Traffic Signal Control

We briefly review some traffic light control strategies that have been proposed previously in the literature. The reader is referred to Papageorgiou et al. [2003, 2007] for a detailed discussion of relevant traffic signal control literature. In Robertson [1969], a commercial software package (TRANSYT) that uses a static optimization technique is designed to generate the signal timings off-line, based on the traffic conditions measured at different periods of the day. This technique is however not adaptive. In Robertson and Bretherton [1991], the authors propose the SCOOT method, where inductive sensors are used to collect "Cyclic Flow Profiles" (CFPs) and relay the same to the central SCOOT optimizer. The CFPs are then used to estimate the queues especially to calculate the effect of alterations in the predicted signal timings. The SCOOT optimizer then makes many split and offset alterations to co-ordinate the traffic flows. SCATS (Sims and Dobinson [1980]) is another well-deployed scheme that picks one of the pre-calculated controls based on the traffic state.

Several on-line TLC algorithms that adapt in real-time have also been proposed. For instance, genetic algorithms (Girianna and Benekohal [2004], Sanchez-Medina et al. [2010]), neural network based algorithms (Spall and Chin [1997], Srinivasan et al. [2006]), algorithms based on cellular automata (Wei et al. [2005]), stochastic control (Yu and Recker [2006]) and dynamic optimization (Gartner [1983], Henry et al. [1984], Boillot et al. [1992], Sen and Head [1997]) as well as reinforcement learning (Abdulhai et al. [2003], Prashanth and Bhatnagar [2011a]) are all online schemes.

In Gartner [1983], Henry et al. [1984], Boillot et al. [1992], Sen and Head [1997], a model of the system is assumed and the optimal signal timings are obtained by solving a dynamic optimization problem in real-time. Lin and Wang [2004] discuss a 0-1 mixed-integer linear programming formulation of the traffic signal control problem. The adaptive control based TLC algorithm proposed in Yu and Recker [2006] incorporates a Markov decision process (MDP) formulation. However, this requires a precise model of the system, which in general is hard to obtain in realistic settings. In Li et al. [2008], an adaptive dynamic programming technique for traffic signal control is presented. A controller at each intersection adjusts its signal timing based on traffic information as well as the performance in the neighboring intersections. In de Oliveira et al. [2006], an RL algorithm for the case of non-stationary traffic conditions in a decentralized framework is presented. In Abdulhai et al. [2003], a Q-learning algorithm with full state representation has been applied to traffic light control on a single junction. The case of a road network with multiple junctions has not been considered there. Further, the algorithm in Abdulhai et al. [2003] has been developed for the case when all the lanes are given equal priority for the purpose of switching lights. Full state representations cannot be directly used in the case of road networks with multiple traffic junctions because of the exponential blow up in the computational requirements as the number of junctions and thereby the cardinalities of the state and action spaces increase. In Cools et al. [2008], a traffic light control method, SOTL (self organizing traffic lights), is presented that switches a lane to green if the elapsed time since the signal turned red on that lane crosses a certain threshold, provided the number of vehicles on the lane is above another threshold. Thus, while queue lengths on the signalled lanes of the network are not directly considered there in deciding the sign configuration, an estimate of the congestion level is used. A distributed multi-agent based approach for traffic signal control is presented in Gokulan and Srinivasan [2010]. Further, traffic signal scheduling in the context of a multi-agent system using a reinforcement learning framework has been explored, for instance, in De Oliveira and Camponogara [2010], Arel et al. [2010], Salkham et al. [2008] and Bakker et al. [2005].

In some body of work on adaptive traffic control, the usage of neural network based controllers is recommended. For instance, in Spall and Chin [1997], SPSA based gradient estimates are used in a neural network (NN) feedback controller to optimize system performance. The idea is to develop a function that takes in current traffic information and outputs the signal timings. This function is approximated via a NN. In Smith and Chin [1995], Chin et al. [1999], the above mentioned NN-SPSA algorithm was studied via simulations on the mid-Manhattan-New York network and it was found to give a 10% reduction in the vehicle waiting times as compared to the previously used strategy employed there. In Srinivasan et al. [2006], a NN-based traffic signal control approach in a multi-agent system is presented and compared against an existing traffic control algorithm as well as an SPSA-NN multi-agent traffic control method developed there. In Kosmatopoulos et al. [2007], a SPSA-like algorithm that incorporates linear function approximation, in addition, has been proposed for adaptive optimization of control systems and applied to tune the split module parameters in a traffic-responsive control strategy. Random perturbations are used in the algorithm proposed in Kosmatopoulos et al. [2007], where the authors note that the basic SPSA algorithm with random perturbations did not perform well. However, in our work, we use one-simulation SPSA with deterministic perturbations based on certain Hadamard matrices and this choice ensures convergence and good empirical performance of the scheme. In Kosmatopoulos and Kouvelas [2009], the algorithm from Kosmatopoulos et al. [2007] has been combined with neural networks. The resulting algorithm has been studied in the context of optimizing the split and cycle times of traffic control strategy via simulation in Kouvelas et al. [2011].

3.1.2 Stochastic optimization

A popular approach for simulation based parameter optimization is SPSA proposed in Spall [1992]. SPSA is an efficient gradient search technique that aims at finding a local minimum of a performance objective. The regular SPSA algorithm perturbs the parameter vector randomly using i.i.d., symmetric, zero-mean random variables and has the critical advantage that it needs only two samples of the objective function for any N-dimensional parameter. A variant of the SPSA algorithm that uses one simulation was proposed in Spall [1997]. Unlike its two-simulation counterpart, the algorithm in Spall [1997] does not work well in practice. In Bhatnagar et al. [2003], several variants of the SPSA algorithm that work with deterministic perturbations (in place of randomized) were developed that show performance improvements over their randomized perturbation counterparts. The perturbation variables there are based on either lexicographic or Hadamard matrix based sequences. In particular, the one-simulation variant of the SPSA algorithm that is based on Hadamard matrices has been found to perform significantly better than the one-simulation random-perturbation algorithm presented in Spall [1997]. A Newton based SPSA algorithm that requires four system simulations with Bernoulli random perturbations was proposed in Spall [2000]. In Bhatnagar [2005], three SPSA based estimates of the Hessian that require three, two and one system simulation(s), respectively, were proposed. In Bhatnagar [2007], certain smoothed functional Newton algorithms that incorporate Gaussian-based perturbations were proposed. It is however the case that Newton based algorithms, even though more accurate than gradient based schemes, require significantly higher computational effort resulting from (a) projecting the Hessian update at each iteration to the set of positive definite and symmetric matrices and (b) inverting the projected Hessian after each update. Hence, we present in this chapter, the application of one-simulation deterministic perturbation SPSA for tuning the threshold parameters.

3.1.3 Markov decision processes and reinforcement learning

Markov decision process (MDP) (Bertsekas [2005, 2007], Puterman [1994]) is a general framework for solving stochastic control problems. Classical solution approaches for MDP such as policy and value iteration solve the associated control problem precisely by identifying an optimal action to pick in each state. These approaches typically suffer from two major problems: (a) they require precise knowledge of the transition probabilities, i.e., the system model, and (b) the amount of computation required to obtain a solution using these approaches grows exponentially with the size of the state and/or action space. Reinforcement learning (RL) (Bertsekas and Tsitsiklis [1996a], Bertsekas [2011], Sutton and Barto [1998a]) provides efficient solutions for both problems above. Many RL algorithms are incremental update stochastic approximation algorithms that work directly with real or simulated data. These algorithms make use of the averaging property of stochastic approximation as a result of which they work efficiently even when the system transition model is not known. For problems where the cardinality of the state/action space is so large that precise solutions cannot be obtained easily, one resorts to certain approximation methods. Often the value function is approximated as a function of certain parameter(s). The functional form of the approximator is also called an architecture. Architectures based on linear function approximators have been well studied in the literature because algorithms such as temporal difference (TD) learning (Sutton [1988]) have been shown convergent when linear architectures are used (Tsitsiklis and Van Roy [1997], Tsitsikis and Van Roy [1999]). Here the value of a given state is approximated using the scalar product of the parameter vector with the feature associated with the state. The feature usually quantifies important state attributes. On the other hand, TD with nonlinear function approximators has been seen to diverge in some cases. An important problem that we address in this chapter is to design an efficient scheme to adaptively select the 'best features' when linear function approximation is used.

The problem of feature adaptation has been studied recently for problems of prediction as well as control. In Menache and Shimkin [2005], features are assumed parameterized and the problem considered is one of finding the optimum feature parameter when TD learning is used. Two algorithms, one based on a gradient approach and the other based on a cross entropy method are then presented for tuning the feature parameter. Certain generalizations of the parameterized basis adaptation approach of Menache and Shimkin [2005] have been presented in Yu and Bertsekas [2009]. The basis adaptation scheme there involves low order calculations. In Keller and Precup [2006], a procedure based on state aggregation and neighborhood component analysis is used for constructing basis functions assuming a linear approximation architecture. In Bertsekas and Yu [2009], Krylov subspace basis functions, involving powers of the transition probability matrix of the associated Markov chain are used. Noisy samples of the basis functions are obtained as they cannot be computed exactly. In Mahadevan and Liu [2010], a Laurent series expansion of the value function is used for basis construction. Another interesting work is Sun et al. [2011] where through a recursive procedure, an 'ideal' basis function that is a representative of the value function is obtained. Whereas the above references deal with basis adaptation for the problem of prediction, in Di Castro and Mannor [2010], the problem of control with adaptive bases is considered. Multiscale actor-critic algorithms are developed where on the 'slowest' timescale, the feature parameters are updated. Algorithms based on TD error, mean square Bellman error and mean square projected Bellman error are presented there. All of the above works consider features to be parameterized and the goal in these references is to find optimal parameters and thus the optimal features within the specific parameterized feature classes. In Parr et al. [2007], feature adaptation for a problem of prediction is considered. Unlike in the above references, the basis functions there are not assumed parameterized. The Bellman error is included as an additional basis in each iteration, thereby increasing the dimension of the subspace at each iteration. In Huang et al. [2011], an a priori approximation of the value function based on a simplified model as one of the features has been considered. In Baras and Borkar [2000], an approach that makes use of both state aggregation and linear function approximation is presented. While the feature matrix is kept fixed there, state aggregation is done adaptively on a slower timescale using estimates of the approximate value function. In this case, one obtains locally optimal state clusters asymptotically.

Chapter 4

Q-learning for traffic signal control

We consider in this chapter the problem of efficient traffic flow control at signal junctions. With increasing traffic in urban areas and limitations of the road infrastructure, any attempt to improve the traffic flow of the system would involve an intelligent design of the traffic signal timings for the junctions. The traffic junctions play a very important role in determining congestion state of the road network. Most of the traffic junctions currently use fixed signal timings, i.e., they periodically cycle through the sign configurations in a round-robin manner. Even though such a strategy is easy to implement, it does not take into consideration the actual traffic conditions and may result in more congestion. In this chapter, we study the use of a sensor network based traffic control system. The objective is to maximize traffic flow by performing adaptive control of traffic lights at intersections. Sensors, placed along the lanes leading to a junction, periodically convey the traffic information to a controller or agent at the junction, which then decides on the signal timings.

In this work, we use a reinforcement learning (Sutton and Barto [1998b], Bertsekas and Tsitsiklis [1996b]) based algorithm to solve the traffic signal control problem. The reason for using this approach is that reinforcement learning (RL) allows for learning the optimal strategy for signal timing without assuming any model of the system. The benefits of using RL are two-fold. First, it is model-free, i.e., it learns and adapts the policy through interaction with the environment. RL algorithms are online, incremental and easy to implement. Second, on high dimensional state spaces, function approximation techniques in RL can be used to achieve computational efficiency. Q-learning (Watkins and Dayan [1992a]) is an important and well-studied RL algorithm that is efficient and known to converge to the optimal policy. In this chapter, we consider two scenarios for traffic control: (i) a single junction scenario, i.e., involving just a single traffic junction and (ii) a scenario involving a corridor type of road network with multiple junctions. Through performance simulations, we show that our Q-learning based algorithm outperforms some of the existing TLC algorithms.

Our Contributions

- To the best of our knowledge, we are the first to explore the application of Q-learning based TLC, in the context of a corridor type of road network.
- An open source java based software, Green Light District (GLD) (Wiering et al. [2004]), is used for traffic control simulations. We implement the Q-learning TLC algorithm and perform a comparative analysis of it against various fixed timing TLC algorithms as well as a TLC algorithm that serves the lane with the longest traffic queue.
- We perform our experiments for two different settings - one where all lanes are given equal priority and another for which the main road is given a higher priority than the side roads. For Q-learning, we implement this prioritization by means of the cost structure. We observe empirically that Q-learning TLC outperforms the fixed timing TLC for various timing intervals as well as the longest queue TLC, in both settings.

In Abdulhai et al. [2003], a case study involving a Q-learning algorithm has been shown for a single junction controller. The cost function there does not incorporate elapsed times. This can lead to lack of fairness between different lanes, i.e., a lane can continue to be green for a long time at the cost of other lanes. Also, performance in terms of the average delay metric alone is shown via a plot. On the other hand, we not only consider a corridor-type of road network in addition to a single junction scenario, but also propose TLC algorithms - QTLC and QTLC-pri - that consider fairness in our cost function by incorporating elapsed times. Further, we implement various fixed timing algorithms and show performance comparisons with these over a range of performance metrics (not just delays). In addition, our QTLC-pri algorithm has been developed for the case when main road traffic is given a higher priority in comparison to traffic on side roads, a scenario not considered in Abdulhai et al. [2003].

The rest of this chapter is organized as follows: In Section 4.1, we describe in detail the problem formulation using Markov Decision Processes (MDP) and discuss the Q-learning method to solve the traffic light control problem. In Section 4.3, we discuss the implementation of the TLC algorithms and present the performance simulation results. Finally, Section 4.4 provides the concluding remarks.

4.1 Traffic Signal Control Problem as an MDP

We consider the problem of finding an optimal schedule for the sign configurations of traffic junctions, with the aim of maximizing the flow of traffic. We consider two scenarios - (i) involving a single junction and (ii) involving a corridor-type of road network with multiple junctions. For a corridor-type of road network, centralized control has been proposed for instance in Ravikumar [2009]. However, reinforcement learning algorithms such as Q-learning have not been proposed or implemented in the past, for such a scenario. We assume for a corridor network a centralized controller that periodically receives information through an array of sensors that are embedded in the roads. This information includes queue lengths on the individual lanes and the time elapsed since the last signal light switch over. We formulate this problem in the setting of Markov Decision Processes (MDP) and apply the Q-learning algorithm for its solution.

We consider a road network with m junctions, $m > 1$. Each junction has multiple cross-roads with each road having j lanes. Our algorithms require a description of states, actions and costs. The state is the vector of queue lengths and the elapsed times since the signal turned red on those lanes that have a traffic signal at the various junctions in the network. Control decisions are made by a centralized controller that receives the state information from the various lanes and makes decision on which traffic lights to switch green during a cycle. This decision is then relayed back to the individual junctions. We assume no propagation and feedback delays for simplicity. The elapsed time counter for a lane with green signal stays at zero till the time the signal turns red. For a network with a total of N signalled lanes, the state at time n is

$$s_n = (q_1(n), \ldots, q_N(n), t_1(n), \ldots, t_N(n))^T,$$

where $q_i(n)$ is the queue length on lane i at time n and $t_i(n)$ is the elapsed time for the red signal on lane i at time n.

The actions a_n comprise of the sign configuration (i.e.,which feasible combination of traffic lights to switch) in the m junctions of the road network and have the form: $a_n = (a_1(n), \ldots, a_m(n))^T$, where $a_i(n)$ is the sign configuration at junction i in the time slot n.

We consider only sign configurations that are feasible in the action set and not all possible red-green combinations of traffic lights (which would grow exponentially with the number of traffic lights). Thus, the action set $A(s_n) = \{\text{feasible sign configurations in state } s_n\}$.

The cost function here has two components. The first component is the sum of the queue lengths of the individual lanes and the second component is the sum of the elapsed times since the signal turned red on the lanes on which the signal is red. The elapsed time on lanes for which the signal is green is zero. The idea here is to regulate the flow of traffic so as to minimize the queue lengths, while at the same time ensure fairness so that no lane suffers from being red for a long duration. Lanes on the main road are given higher priority over others. We achieve prioritization of main road traffic as follows: Let I_p denote the set of indices of lanes whose traffic should be given higher priority. Then the single-stage cost $k(s_n, a_n)$ has the form

$$\begin{aligned} k(s_n, a_n) = \quad & r_1 * (\textstyle\sum_{i \in I_p} r_2 * q_i(n) + \sum_{i \notin I_p} s_2 * q_i(n)) \\ + \quad & s_1 * (\textstyle\sum_{i \in I_p} r_2 * t_i(n) + \sum_{i \notin I_p} s_2 * t_i(n)), \end{aligned} \tag{4.1}$$

where $r_i, s_i \geq 0$ and $r_i + s_i = 1, i = 1, 2$. Further, $r_2 > s_2$. Thus, lanes in I_p are assigned a higher cost and hence a cost optimizing strategy must assign a higher priority to these lanes in order to minimize the overall cost.

As stated before, we consider the infinite horizon discounted cost framework. The discount factor γ plays a crucial role as a lower γ serves to discount the future costs more, thereby putting less emphasis on these as opposed to a higher value of γ. We let $\gamma = 0.9$ in our experiments.

4.2 Q-Learning based TLC with Full State Representations

This algorithm requires a full state representation as it updates over all feasible state-action tuples. It addresses the case when the system model is not known, however, the state and action spaces are manageable. The underlying MDP for the traffic signal control problem, with its state, action spaces and the cost function have been described in Section 4.1. Q-learning finds the optimal policy even without knowledge of the transition probabilities of the underlying MDP. It does so by iteratively updating the $Q(s, a)$ using (2.13) in order to obtain the optimal sign configuration policy.

The algorithm estimates the 'Q-values' $Q(i,a)$ of all feasible state-action tuples (i,a) i.e., $i \in S$ and $a \in A(i)$. Let $s_{n+1}(i,a)$ denote the state of the system at instant $(n+1)$ when the state at instant n is i and action chosen is a. Let $Q_n(i,a)$ denote the Q-value estimate at instant n associated with the tuple (i,a). Then the algorithm updates according to

$$\boxed{Q_{n+1}(i,a) = Q_n(i,a) + \alpha(n)\left(c(i,a) + \gamma \min_{b \in A(s_{n+1}(i,a))} Q_n(s_{n+1}(i,a),b) - Q_n(i,a)\right), \quad (4.2)}$$

for all feasible (i,a) tuples. Upon convergence, one obtains the optimal Q-values $Q^*(i,a)$ that are seen to satisfy the Q-Bellman equation (2.12). The optimal action in state i then corresponds to $arg \min_{b \in A(i)} Q^*(i,b)$. A convergence proof of this algorithm can be found in Watkins and Dayan [1992a] and Tsitsiklis [1994a].

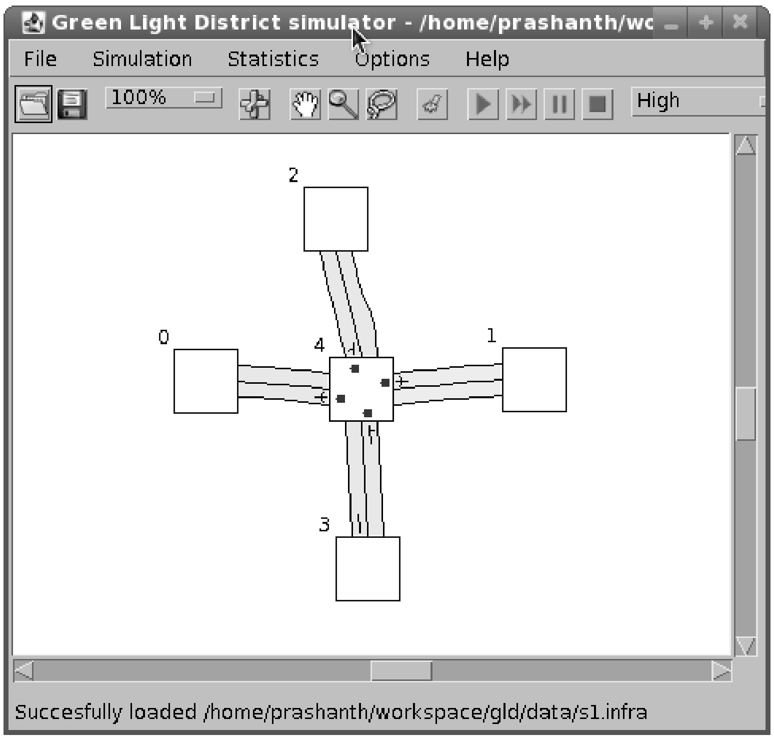
We refer to the Q-learning algorithm with full state representation or the lookup table case when applied to our setting as the QTLC algorithm.

4.3 Simulation Experiments

We use the Green Light District (GLD) Simulator (Wiering et al. [2004]) for implementation and evaluation of our TLC algorithms. GLD allows users to build road networks (involving lanes, junctions and road users), simulate traffic from various road users and obtain performance statistics. The crucial part is that it allows implementation and evaluation of traffic light algorithms. It consists of an interface for constructing the road layouts and also a traffic simulator for conducting the experiments with existing and new traffic light algorithms.

We refer to QTLC (QTLC-pri) as the algorithm that assigns equal priority to all lanes (higher priority to main road traffic). In addition, we implement the fixed timing TLC algorithm for various signal timings as well as the Longest Queue TLC (LTLC) algorithm. In this (last) algorithm, the number of road users waiting for a traffic light to turn green is counted, and used to decide the combination of traffic lights to be turned green in the next time slot. In essence, LTLC attempts to switch the lane with the highest number of waiting road users, to green.

The Fixed timing TLC, on the other hand, periodically cycles through the list of feasible sign configurations, while not considering the traffic load on the lanes of the road network. The cycling period in this algorithm is a tunable parameter and we show the results of its



Succesfully loaded /home/prashanth/workspace/gld/data/s1.infra

Figure 4.1: A Screen Shot of the Single Junction Road Traffic Network Obtained from GLD - Controller is Junction 4

performance for various cycling periods. In the case when traffic on all lanes has the same priority, we let the on-period (time for which the green light is on) to be the same for all lanes. However, in the case when the traffic on the main road is given a higher priority over that on side roads, we give a lower on-period for traffic on the side lanes as compared to the main road traffic.

We perform simulation experiments for two road networks - one where there is single junction (Fig. 4.1) and another where there is a corridor with two junctions (Fig. 4.3).

4.3.1 Single junction

The simulations are conducted for 5000 cycles for all algorithms. Each road user's destination is fixed randomly, using a discrete uniform distribution to choose one of the edge nodes. For the QTLC and QTLC-pri algorithms, $n = 4$ in the state description, thereby indicating four signalled lanes. For the QTLC algorithm, we set the weights in the single stage cost function as $r = 4/5$ and $s = 1/5$ respectively. For QTLC-pri, we let $r_i = 4/5, i = 1, 2$ and $s_i = 1/5, i = 1, 2$. This assignment gives a higher priority to the lanes on the main road that connect node 0 with node 1 in Fig. 4.1. The simulation

Table 4.1: Performance comparison of QTLC with Fixed timing algorithms in Scenario I

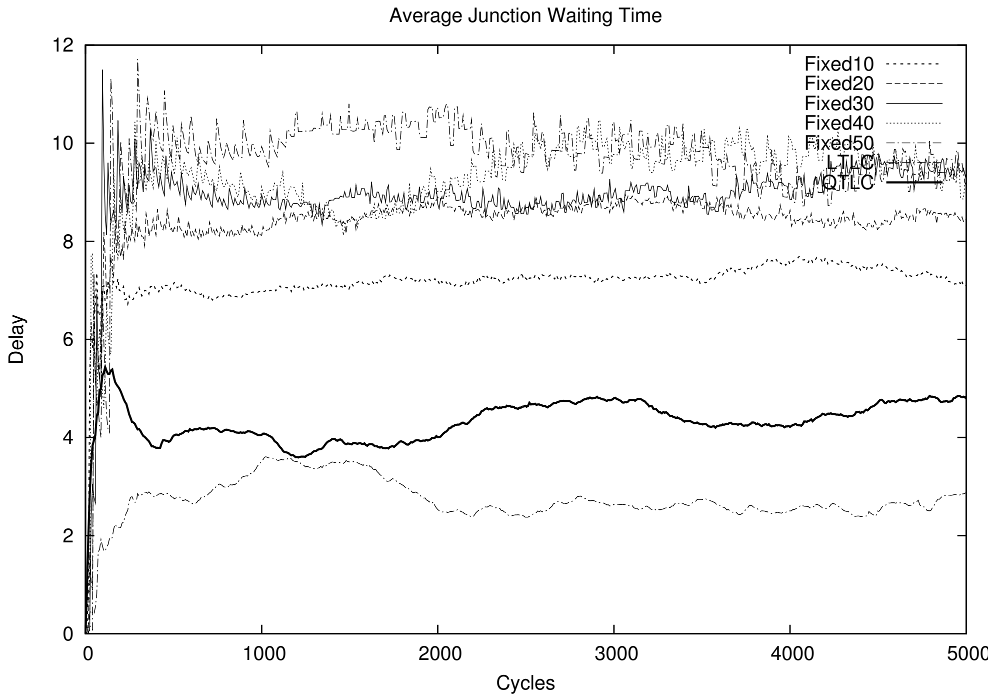
	Average Waiting Time (Main Road	Average Waiting Time (Side Roads)	Total Arrived Road Users	Total Waiting Queue Length	Average Trip Waiting Time	Elapsed time
Fixed-10	14.26 ± 0.63	13.78 ± 0.75	6008.45 ± 78.94	6.77 ± 0.61	14.02 ± 0.67	30.69 ± 0.00
Fixed-20	22.06 ± 1.10	21.47 ± 0.62	5986.82 ± 56.32	13.23 ± 0.96	21.76 ± 0.79	57.27 ± 0.00
Fixed-30	29.86 ± 0.76	29.66 ± 0.38	5957.66 ± 37.60	20.55 ± 1.33	29.76 ± 0.49	84.38 ± 0.00
Fixed-40	38.89 ± 1.04	38.29 ± 0.99	5966.83 ± 41.25	28.73 ± 1.18	38.59 ± 0.96	110.50 ± 0.00
Fixed-50	47.41 ± 0.59	46.63 ± 0.93	5947.80 ± 51.02	38.08 ± 1.73	47.02 ± 0.48	137.12 ± 0.00
LTLC	3.83 ± 0.23	4.85 ± 0.35	6673.49 ± 46.19	1.59 ± 0.33	4.34 ± 0.27	11.26 ± 0.47
QTLC	8.94 ± 2.12	8.33 ± 1.64	6387.51 ± 32.45	4.53 ± 1.36	8.55 ± 1.97	10.64 ± 0.90

Table 4.2: Performance comparison of QTLC-pri with Fixed timing algorithms in Scenario II

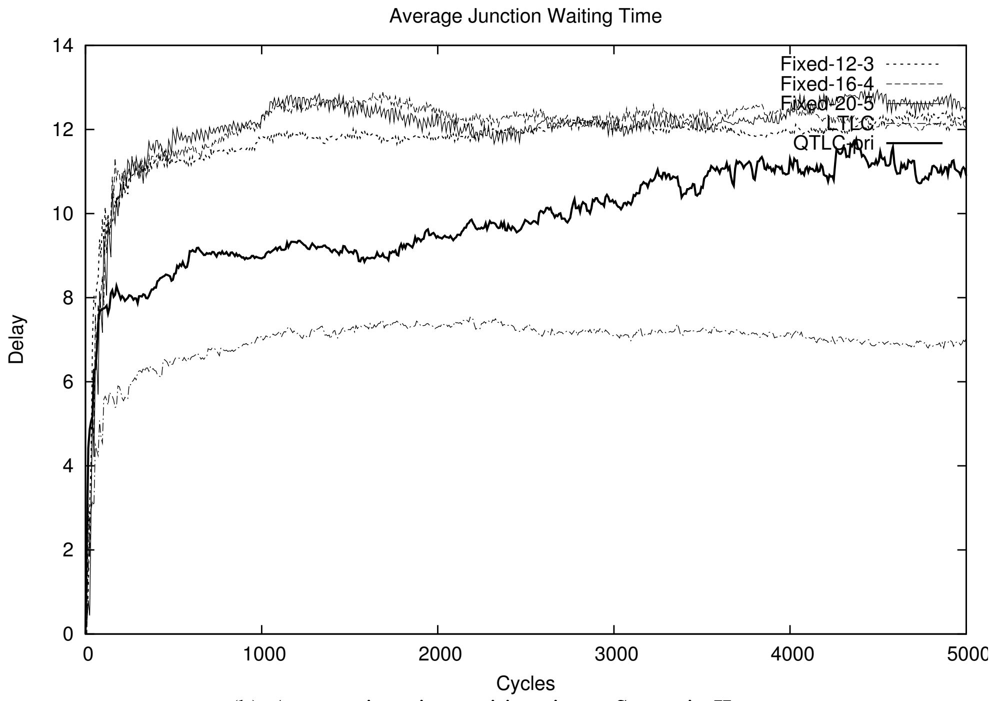
	Average Waiting Time (Main Road)	Average Waiting Time (Side Roads)	Total Arrived Road Users	Total Waiting Queue Length	Average Trip Waiting Time	Elapsed time
Fixed-12-3	1264.53 ± 22.79	1093.42 ± 9.82	6212.63 ± 29.89	1491.66 ± 10.13	1178.98 ± 15.32	39.56 ± 0.00
Fixed-16-4	1251.82 ± 15.43	1068.79 ± 13.66	6141.67 ± 15.64	1491.35 ± 14.11	1160.30 ± 11.52	49.53 ± 0.00
Fixed-20-5	1198.50 ± 14.48	1038.18 ± 12.46	6089.66 ± 13.01	1491.22 ± 8.35	1118.35 ± 8.02	59.30 ± 0.00
LTLC	347.75 ± 91.80	276.45 ± 68.26	6561.59 ± 44.86	806.55 ± 46.04	298.90 ± 23.57	58.79 ± 12.44
QTLC-pri	548.66 ± 31.49	702.81 ± 14.04	6318.56 ± 17.69	1098.79 ± 10.63	617.98 ± 17.00	27.93 ± 1.14

experiments are conducted for two different scenarios: In Scenario I (equal priority to all lanes), the spawn frequencies (rate at which new vehicles are added to a lane) are set to 0.25/cycle for all edge nodes. In Scenario II (main road given a higher priority than the side roads), the spawn frequencies for the edge nodes on the main road (nodes 0 and 1 in Fig. 4.1) is 0.75/cycle, while for the edge nodes on the side roads (nodes 2 and 3), it is 0.25/cycle.

We compare the performance of the TLC algorithms using various performance metrics involving the average waiting times, queue lengths, total number of road users arrived, elapsed time, etc. The performance comparisons are shown in Fig. 4.2, and Tables 4.1–4.2. The tables show the mean and standard error results from ten independent simulation runs for various performance metrics for the different TLC algorithms. Fig. 4.2 shows the results for the average junction waiting time in the two scenarios. We observe from the performance plots in Fig. 4.2 as well as the numerical comparisons of the various performance metrics in Tables 4.1–4.2 that both QTLC and QTLC-pri perform better than the fixed timing TLC algorithms. In Fig. 4.2(b) and Table 4.2, Fixed-12-3 corresponds to the fixed timing TLC that assigns 12 cycles to the main road, while each side road is given 3 cycles. Fixed-16-4 and Fixed-20-5 have a similar interpretation. It may however be noted that LTLC performs the best in this scenario with QTLC and QTLC-pri being close in their performance. In the case of a network corridor (considered next), however, this trend is reversed as QTLC and QTLC-pri show the best results.

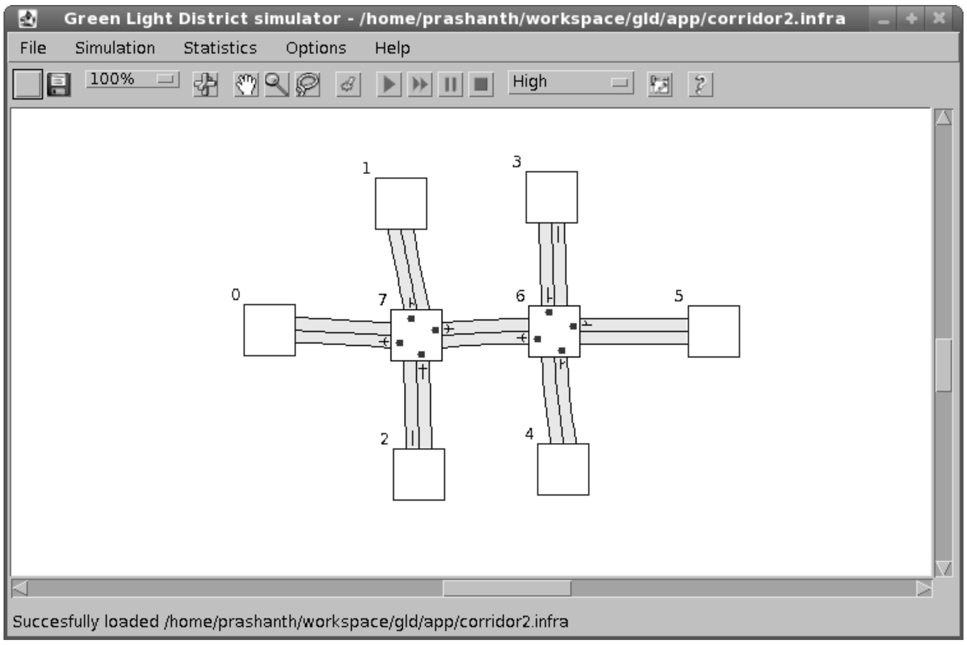


(a) Average junction waiting time - Scenario I



(b) Average junction waiting time - Scenario II

Figure 4.2: Performance Comparison of TLC algorithms - Single Junction


Green Light District simulator - /home/prashanth/workspace/gld/app/corridor2.infra
File
Simulation
Statistics
Options
Help
100%
High
0
1
2
3
4
5
6
7

Succesfully loaded /home/prashanth/workspace/gld/app/corridor2.infra

Figure 4.3: A Screen Shot of the Corridor-type Road Traffic Network Obtained from GLD - Controllers are in Junctions 7 and 6

4.3.2 Corridor network

The simulations are conducted for 5000 cycles for all algorithms. Each road user's destination is fixed randomly, using a discrete uniform distribution to choose one of the edge nodes. For the QTLC and QTLC-pri algorithms, $n = 8$, thereby indicating eight signalled lanes. For the QTLC algorithm, we set the weights in the single stage cost function in (4.1) as $r = 4/5$ and $s = 1/5$ respectively. For QTLC-pri, we use $r_i = 4/5, i = 1, 2$ and $s_i = 1/5, i = 1, 2$. This assignment gives a higher priority to the lanes of the main road that connect node 0 with node 5 in Fig. 4.3.

The simulation experiments are conducted for two different scenarios:

- Scenario I: All lanes are given equal priority. The spawn frequencies (rate at which new vehicles are added to a lane) are set to 0.25/cycle for all edge nodes here.

- Scenario II: The main road is given a higher priority than the side roads. The spawn frequencies for the edge nodes on the main road (nodes 0 and 5 in Fig. 4.3) in this case is 0.75/cycle, while for the edge nodes on the side roads (nodes 1,2,3 and 4) it is 0.25/cycle.

We compare the performance of the TLC algorithms using various performance metrics involving the average waiting times, queue lengths, total number of road users arrived and

Table 4.3: Performance comparison of QTLC with Fixed timing and LTLC algorithms in Scenario I

	Total Arrived Road Users	**Average Waiting Time - Junction 7**	**Average Waiting Time - Junction 6**	**Total Waiting Queue Length**	**Average Trip Waiting Time**	**Elapsed time**
Fixed-5	3929.70 ± 29.28	18.06 ± 0.28	40.46 ± 0.71	1651.50 ± 28.69	832.85 ± 19.64	144.00 ± 0.00
Fixed-10	3681.40 ± 18.38	19.79 ± 0.43	46.25 ± 1.00	1682.80 ± 36.82	886.23 ± 28.03	294.00 ± 0.00
Fixed-20	3508.00 ± 18.69	20.65 ± 0.50	52.69 ± 0.80	1717.50 ± 53.80	909.89 ± 44.68	474.00 ± 0.00
Fixed-30	3403.40 ± 17.48	22.84 ± 0.67	51.96 ± 0.92	1779.80 ± 33.39	962.57 ± 19.87	564.00 ± 0.00
Fixed-40	3352.60 ± 14.03	22.06 ± 1.31	53.75 ± 1.07	1810.20 ± 20.41	980.10 ± 21.40	654.00 ± 0.00
Fixed-50	3342.10 ± 23.04	21.68 ± 0.82	55.31 ± 1.40	1809.00 ± 54.43	994.68 ± 38.05	1104.00 ± 0.00
LTLC	70.33 ± 42.58	4.81 ± 1.74	7.39 ± 3.33	2244.00 ± 0	8.65 ± 4.02	17595.22 ± 198.93
QTLC	5926.00 ± 64.32	10.05 ± 0.42	15.92 ± 1.38	1253.75 ± 60.42	557.69 ± 44.00	32.25 ± 4.32

Table 4.4: Performance comparison of QTLC-pri with Fixed timing and LTLC algorithms in Scenario II

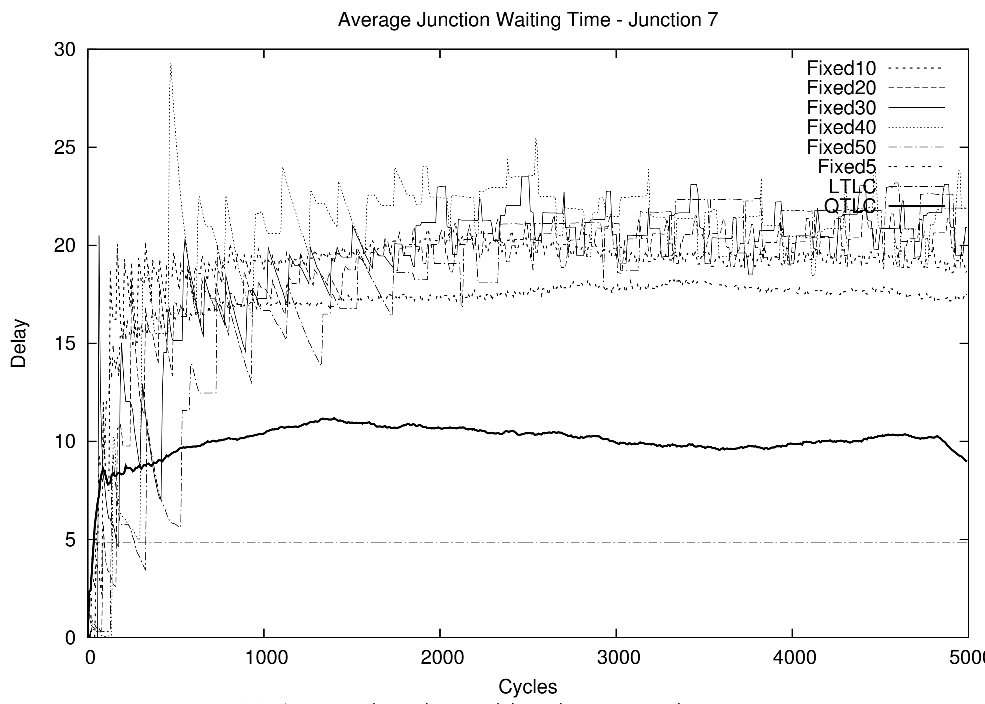
	Total Arrived Road Users	**Average Waiting Time (Main Road)**	**Average Waiting Time (Side Roads)**	**Total Waiting Queue Length**	**Average Trip Waiting Time**	**Elapsed time**
Fixed-12-3	5352.31 ± 40.20	801.14 ± 13.18	822.65 ± 12.53	1867.72 ± 0.28	914.63 ± 12.41	79.14 ± 0.00
Fixed-16-4	5147.20 ± 34.75	785.89 ± 20.64	816.26 ± 17.56	1867.70 ± 0.20	895.06 ± 17.38	98.94 ± 0.00
Fixed-20-5	4996.99 ± 55.17	746.76 ± 14.18	779.97 ± 13.30	1867.90 ± 0.15	842.70 ± 13.41	118.68 ± 0.00
LTLC	71.77 ± 43.30	18.91 ± 19.19	14.51 ± 12.54	2244.00 ± 0.00	29.90 ± 41.60	35502.53 ± 282.44
QTLC-pri	6843.96 ± 88.58	590.69 ± 50.54	865.89 ± 40.36	1238.65 ± 44.16	773.31 ± 37.26	30.24 ± 0.15

elapsed times. The performance comparisons are shown in Figs 4.4–4.5 and Tables 4.3–4.4 respectively.

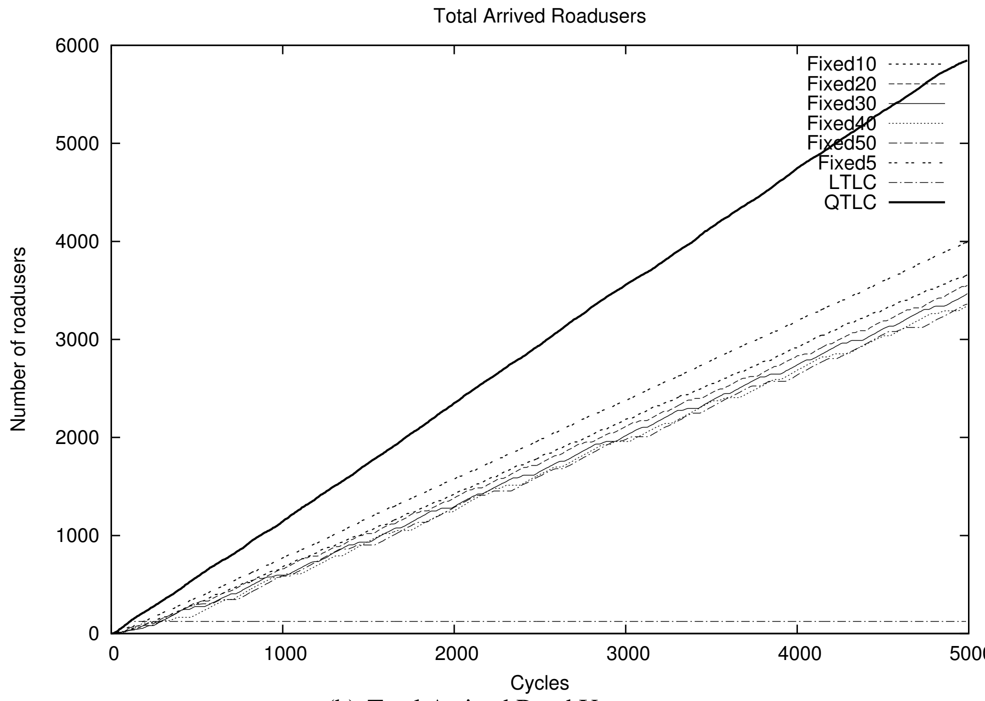
Tables 4.3 and 4.4 provide the mean and standard error results from ten independent simulation runs for various performance metrics for the different TLC algorithms that we studied.

Fig. 4.4 and 4.5 show the results for the average junction waiting time on junction 7, the total arrived road users and the total waiting queue length. We observe that QTLC and QTLC-pri perform better than the fixed timing TLC for all cycling periods considered.

An important observation here is that both QTLC and QTLC-pri outperform LTLC. In fact, LTLC performed very poorly when compared to Q-learning in this scenario of a corridor network as traffic came to a standstill using this algorithm. This situation of deadlock or impasse was due to the fact that LTLC, invariably in all cases studied, arrived at a sign configuration on junctions 6 and 7 which did not allow traffic to pass across junctions. The longest queue status of the lanes remained the same because of the gridlock, leaving the sign configuration unchanged. On hitting a standstill traffic situation, the junction waiting times in GLD of all the road users froze, which resulted in a misrepresentation as even though from Figs 4.4(a) and 4.5(a), it would appear that LTLC shows the lowest average junction waiting time, this only happened because the waiting time counters in GLD froze upon reaching a gridlock. This can also be seen from the plots in Figs 4.4(b) and 4.5(b), where the total arrived road users (i.e., those that have completed their journey) is the

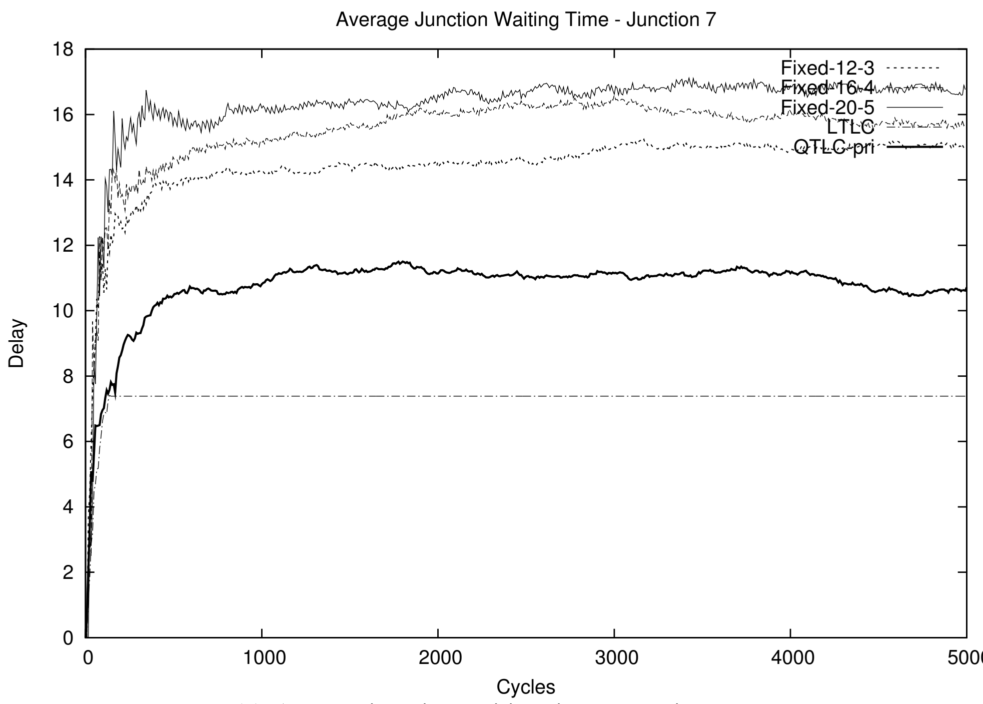


(a) Average junction waiting time - Junction 7

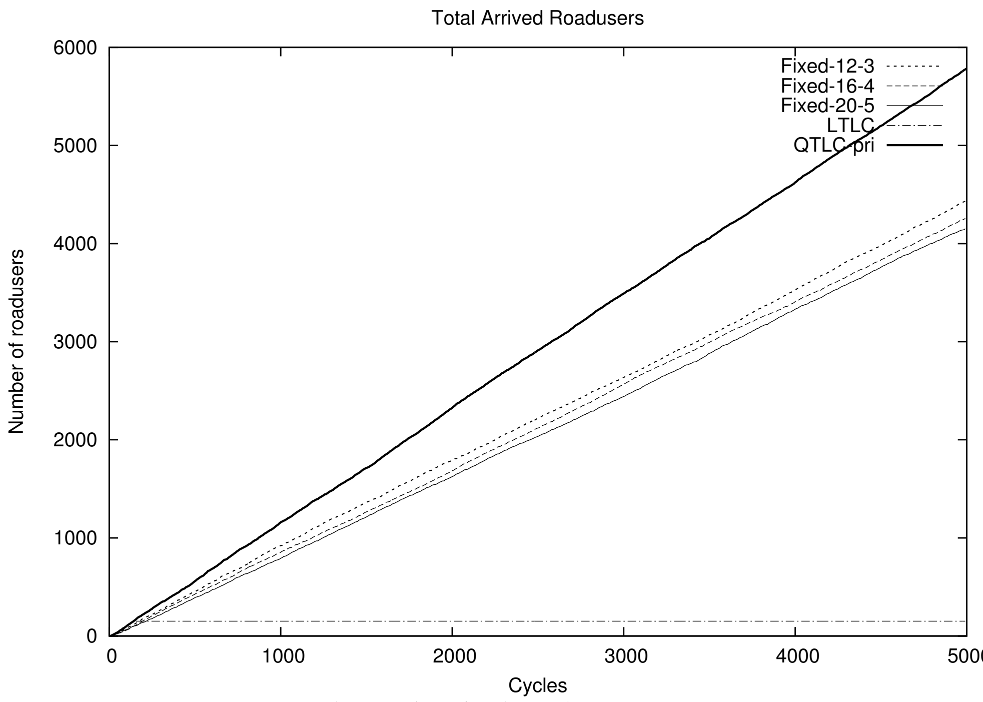


(b) Total Arrived Road Users

Figure 4.4: Performance Comparison of TLC algorithms in Scenario I - Corridor-type road network



(a) Average junction waiting time - Junction 7



(b) Total Arrived Road Users

Figure 4.5: Performance Comparison of TLC algorithms in Scenario II - Corridor-type road network

lowest when LTLC is used. it can be seen that LTLC shows the worst performance in terms of total waiting queue length. Further, from Tables 4.3 and 4.4, the elapsed times are seen to be more than an order of magnitude higher when LTLC is used. On the other hand, as mentioned previously, QTLC and QTLC-pri show the best results overall. Note from Table 4.4 that QTLC-pri shows the best results for main road traffic as expected. For traffic on side roads, Fixed-20-5 shows lower average junction waiting time on the side roads than QTLC-pri. This is because QTLC-pri gives a higher priority to main road traffic.

4.4 Summary

Designing a road traffic management system based on wireless sensor networks that achieves high traffic flow rates with minimum congestion is a challenging task. Reinforcement learning presents an interesting paradigm for solving such problems. We designed and evaluated a full-state Q-learning based traffic light control algorithm for a corridor type of road network under the scenarios when traffic on all roads gets the same priority and when the main road traffic gets a higher priority as compared to traffic on the side roads. This algorithm has the advantages of model-free learning algorithms that adapt in real-time to the traffic conditions. From the performance simulations, it is observed that Q-learning TLC algorithms outperform the fixed timing and longest queue strategies in both scenarios.

Considering the usefulness of RL for solving the traffic signal control problem, in the next chapter we develop enhancements to the Q-learning based TLC algorithm described here by incorporating feature-based representations and function approximation. Feature-based approaches will be needed when bigger road networks are considered as in such a case, the traffic control problem will become high-dimensional. Further, we also develop RL algorithms for average cost traffic signal control in Chapter 6.

Chapter 5

Qlearning with function approximation for traffic signal control

In this chapter, we use an RL (Sutton and Barto [1998b], Bertsekas and Tsitsiklis [1996b]) based algorithm with function approximation to solve the traffic signal control problem. As shown in Chapter 4, RL allows for learning the optimal strategy for signal timing without assuming any model of the system. However, the full state Q-learning algorithm described in Chapter 4 cannot be implemented on larger networks owing to the curse of dimensionality. In other words, the high-dimensional state-action spaces associated with larger networks render the full state Q-learning practically infeasible on them. To alleviate this problem, function approximation techniques in RL, which achieve computational efficiency, can be used.

In this chapter, we develop a Q-learning based traffic light control (TLC) algorithm that incorporates function approximation. Such an algorithm has been applied for the first time in the area of traffic signal control. Amongst its many advantages, it is seen to be easily implementable on a range of high-dimensional network settings and gives much superior performance as compared to other related algorithms in the literature. We compare the performance of our algorithm with a range of algorithms that assign a certain (fixed) traffic light duration to the various sign configurations. In addition, we compare the performance of our algorithm with an algorithm that switches traffic lights to green for those lanes that have the longest queue. We also show performance comparisons of our algorithm with the algorithms of Cools et al. [2008] and Abdulhai et al. [2003]. Further, we also compare the performance of our algorithm with the full state Q-learning TLC algorithm described in Chapter 4.

We now describe comparisons of our work with prior work in the literature. The algorithm of Abdulhai et al. [2003], while being a Q-learning algorithm, requires full-state representations and cannot be implemented even on road networks of moderate size. Indeed, the implementation of this algorithm has been shown in Abdulhai et al. [2003] only for the case of a single junction road network. On the other hand, we use function approximation and hence our algorithm is found to be easily implementable on larger road networks such as a 3x3-grid and an eight-junction corridor. We observe that on a 3x3-grid, the cardinality of the state-action space is nearly 10^{101} while the number of features that our algorithm requires is only about 200. Thus, while algorithms such as in Abdulhai et al. [2003] are not implementable on such road networks, our algorithm is found to be easily implementable and gives fast convergence. While the algorithm in Cools et al. [2008] also requires information on the level of congestion and elapsed times on the various lanes, it is not based on RL, unlike our algorithm. Hence, the algorithm of Cools et al. [2008] does not have the advantage of an RL-based algorithm as it does not update its policy adaptively using information from interactions with the environment. The state-action features that we incorporate require information on whether the level of congestion on any given lane is low, medium or high, and also whether the elapsed time is below or above a threshold.

Note that the full state Q-learning algorithm from Chapter 4, unlike Abdulhai et al. [2003], incorporates prioritization in its cost objective. We implemented this algorithm mentioned above and the one proposed in Abdulhai et al. [2003] only on a two-junction corridor setting as these could not be implemented on larger road networks because of the exponential increase in the computational complexity mentioned above. However, we found that even on a two-junction corridor, our algorithm with function approximation outperformed both of these algorithms.

Next, unlike Abu-Lebdeh and Benekohal [2003], Yun and Park [2006] and Girianna and Benekohal [2004], Q-learning in the case of full state representation can be shown to converge to an optimal policy. Convergence under function approximation is not straightforward, except under certain conditions, see Melo and Ribeiro [2007]. Whereas we also assume the underlying process to be an MDP, unlike Yu and Recker [2006], we do not require information on the transition probabilities of the system, which are hard to obtain in any (real) system. On the other hand, our algorithm works directly with observed data in terms of the level of congestion and elapsed times on the signalled lanes of the road network. Further, unlike Abdulhai et al. [2003] that considers only a single junction scenario, we develop and apply our algorithm on road networks with multiple junctions. As stated before, we propose and apply Q-learning with function approximation in the case of larger road networks where full state representations cannot be used because of the curse of dimensionality.

Our Contributions

- We consider the problem of adaptive signal control of traffic lights at junctions and develop a Q-learning algorithm with feature based state-action representations and function approximation. To the best of our knowledge, reinforcement learning with function approximation for traffic signal control has been proposed by us for the first time in the literature.
- We study the performance of our function approximation based RL algorithm on various road network settings - a two-junction corridor, a 2x2-grid network, a 3x3-grid network and an eight-junction corridor, respectively. We compare the performance of our algorithm with various existing TLC algorithms - Fixed timing, Longest queue and SOTL from Cools et al. [2008] as well as the Q-learning based TLC algorithms that use full state representations - the algorithm that we develop for prioritized traffic and the algorithm from Abdulhai et al. [2003]. Our algorithm is seen to be easily implementable over all network settings, converges fast and consistently outperforms all the other TLC algorithms considered.

We implement our algorithms on an open source Java based software, Green Light District (GLD) (Wiering et al. [2004]). The other TLC algorithms with which we compare the performance of our algorithm were also implemented in GLD.

The rest of the chapter is organized as follows: In Section 5.1, we briefly recall the traffic signal control problem formulation, which was described in detail in Section 4.1. Note that a full-state Q-learning based TLC algorithm for this problem was presented in Chapter 4. In Section 5.2, we present our Q-learning algorithm with function approximation. In Section 5.3, we discuss the implementation of the various TLC algorithms and present the performance simulation results. Finally, in Section 5.4 we provide the concluding remarks.

5.1 The traffic control problem

We consider the problem of finding an optimal schedule for the sign configurations at traffic junctions with the aim of maximizing traffic flow. The signals associated with a phase i.e., those that can be switched to green simultaneously form a sign configuration.

We formulated this traffic signal control problem as an MDP in Section 4.1 and presented a Q-learning algorithm based on full state representations for solving this problem. We refer to the Q-learning algorithm with full state representation i.e., the look-up table case when applied to our setting as the QTLC-FS algorithm. We now develop a Q-learning based TLC algorithm that incorporates function approximation. This algorithm is seen to be easily implementable on high-dimensional settings and also outperforms other well known TLC algorithms.

5.2 Q-Learning with Function Approximation

The QTLC-FS algorithm obtains the optimal sign configuration policy running the incremental stochastic algorithm (4.2). However, this requires a look-up table to store the Q-values for every possible (s, a)-tuple. While this is useful in small state and action spaces, it becomes computationally expensive for larger road networks involving multiple junctions. For instance, in the case of a small road network (e.g. a two-junction corridor) say with 10 signalled lanes, with each lane accommodating 20 vehicles, the number of state-action tuples (and hence the size of the $Q(s, a)$ lookup table) is of the order of 10^{14}. This leads to an extraordinary computation time and space as lookup table representation requires a lot of memory and secondly, the lookup and update operation of $Q(s, a)$ for any (s, a) tuple is expensive because of the number of (s, a)-tuples. For instance, in the case of the ten-lane example above, (4.2) would correspond to a system of 10^{14} equations needed to update $Q(s, a)$ for each feasible (s, a)-tuple once. The situation is aggravated when we consider larger road networks such as a grid or a corridor with several junctions, as the sizes of the state and action spaces blow up exponentially. It is precisely for this reason that the Q-learning algorithm proposed in Abdulhai et al. [2003] is not even implementable on medium and large-sized road networks. To alleviate this problem of curse of dimensionality, we incorporate feature based methods. These methods handle the above problem by making computational complexity manageable. An introduction to feature based methods is given in Chapter 2.

We now describe QTLC-FA, a Q-Learning based TLC algorithm that uses function approximation. While the QTLC-FS algorithm as such requires complete state information and so is computationally less efficient, its function approximation based variant parametrizes the value function and requires significantly less computation in terms of space and time requirements, while giving good performance. The algorithm QTLC-FA which is a variant of the algorithm QTLC-FS updates the parameter θ, a d-dimensional quantity. Thus, instead of solving a system in $|S \times A(S)|$ variables, we solve here a system in only d variables. For instance, in the case of the 3x3-grid road network that we consider in our experiments, it can be seen that while $|S \times A(S)| \sim 10^{101}$, d is only about 200. This results in significant speed up in the computation time when feature-based representations are used.

Recall that in a function approximation setting, we associate with each state-action tuple (i, a), a state-action feature denoted by $\phi_{i,a}$. The Q-function is then approximated as

$$Q^*(i,a) \approx \theta^T \phi_{i,a}. \tag{5.1}$$

Let s_n, s_{n+1} denote the state at instants $n, n+1$, respectively. Let θ_n be the nth update of the parameter θ. The algorithm QTLC-FA uses the following update rule:

$$\theta_{n+1} = \theta_n + \alpha(n)\sigma_{s_n,a_n}(k(s_n,a_n) + \gamma \min_{v \in A(s_{n+1})} \theta_n^T \sigma_{s_{n+1},v} - \theta_n^T \sigma_{s_n,a_n}), \tag{5.2}$$

where θ_0 is set arbitrarily. In (5.2), the action a_n is chosen in state s_n according to $a_n = arg\min_{v \in A(s_n)} \theta_n^T \sigma_{s_n,v}$.

While the algorithm (5.2) updates the parameter $\theta \in \mathbf{R}^d$, this results in updating the projected Q-value functions i.e., those that are obtained according to $Q \approx \Phi\theta$. Note that the set $\{\Phi\theta | \theta \in \mathbf{R}^d\}$ forms a subspace of the set of all functions on $|S \times A(S)|$ (i.e., the original Q-value functions).

The features are chosen based on the queue lengths and elapsed times of each signalled lane of the road network. In particular, we select features σ_{s_n,a_n} to have the following form:

$$\sigma_{s_n,a_n} = (\sigma_{q_1(n)}, \ldots, \sigma_{q_N(n)}, \sigma_{t_1(n)}, \ldots, \sigma_{t_N(n)}, \ \sigma_{a_1(n)}, \ldots, \sigma_{a_m(n)})^T, \tag{5.3}$$

where

$$\sigma_{q_i(n)} = \begin{cases} 0 & \text{if } q_i(n) < L_1 \\ 0.5 & \text{if } L_1 \leq q_i(n) \leq L_2 \\ 1 & \text{if } q_i(n) > L_2 \end{cases} \tag{5.4}$$

$$\sigma_{t_i(n)} = \begin{cases} 0 & \text{if } t_i(n) \leq T_1 \\ 1 & \text{if } t_i(n) > T_1 \end{cases}$$

Further $\sigma_{a_1(n)}, \ldots, \sigma_{a_m(n)}$ corresponds to the actions or sign configurations chosen at each of the m junctions. As before, N is the total number of lanes (inclusive of all junctions) in the network. L_1 and L_2 are thresholds on the queue lengths and T_1 is a threshold on the elapsed time. Note that the parameter θ_n has dimension the same as that of σ_{s_n,a_n}. Again the advantage here is that instead of updating the Q-values for each feasible (s, a)-tuple as before, one estimates these according to the parametrization (5.1). In the case of a fixed policy, the algorithm (5.2) is analogous to the well studied temporal difference learning algorithm. A proof of convergence of the algorithm (5.2) under some conditions is provided in Melo and Ribeiro [2007].

An advantage in using the above features is that one does not require full information on the queue lengths or the elapsed times. Thresholds L_1 and L_2 can be marked on the lanes and used to estimate low (less than L_1), medium (between L_1 and L_2) or high (above L_2) traffic. Likewise the elapsed time can be categorized as being below the threshold (T_1) or above it. More gradations of the queue lengths and elapsed times can also be considered. Thus another advantage of our algorithm QTLC-FA over the algorithm in Abdulhai et al. [2003] or the QTLC-FS algorithm is that it does not require precise queue length information. Such information is often hard to obtain whereas a characterization of traffic at any time as low, medium or high is easier.

Practical implementation of QTLC-FA would require placement of sensors along the lanes of the road network. Since we require only information on whether or not the traffic congestion is below threshold L_1, in between thresholds L_1 and L_2 or above threshold L_2, one can use two loops of sensors - one placed along L_1 and the other along L_2. If sensors at L_1 do not detect congestion, it can be inferred that congestion level on that lane is 'low' i.e., below L_1. If, on the other hand, sensors at L_1 detect congestion but those at L_2 do not, then congestion can be inferred to be in the 'medium' range (i.e., above L_1 but below L_2) and similarly if sensors at L_2 also detect congestion, then the congestion level can be inferred to be in the 'high' range (i.e., at the level of L_2 or more). Elapsed times are usually measured by time counters placed at signal intersections. Again, we only need information on whether or not the elapsed time on a lane is above or below a threshold T_1. Information on congestion levels as well as elapsed times being below or above a threshold will then have to be communicated to the central controller which would then run the QTLC-FA algorithm to obtain the sign configuration policy. As we observe from our experiments, QTLC-FA has a very short transient phase and is computationally very efficient (both in time and space complexities) and hence can be easily implemented to obtain the sign configurations online in a real system. Note that while we do not require precise information on vehicle count, the problem of getting precise estimates of vehicle count is an interesting problem in itself and has been independently addressed for instance in Kwong et al. [2010].

There is another advantage in placing sensors along fixed distances (say L_1 and L_2) from the traffic junction and using distance of "detected" congestion (from the junction) as a proxy for queue length thresholds. For instance, in the case of the traffic situation prevalent in India i.e., highly congested traffic with a high diversity of vehicles, getting precise queue length information is extremely difficult. Using passenger car units (PCUs) as a measure of queue length, see Ravikumar [2009] (for instance, three motor bikes could correspond to one PCU), one can estimate the number of PCUs (could be a fraction) that can be accommodated in a unit distance (say one metre) of the lane. That number multiplied by L_1 or L_2 would roughly correspond to the number of PCUs that can be packed in L_1 or L_2 metres of the lane. Thus L_1 and L_2 could themselves be used as proxies for queue thresholds except for a multiplication factor.

5.3 Simulation Experiments

We use the Green Light District (GLD) Simulator (Wiering et al. [2004]) for implementation and evaluation of our TLC algorithms. GLD allows users to build road networks (involving lanes, junctions and road users), simulate traffic from various road users and obtain performance statistics. The crucial part is that it allows implementation and evaluation of traffic light algorithms. It consists of an interface for constructing the road layouts and a traffic simulator for conducting the experiments with existing and new TLC algorithms.

5.3.1 Implementation

We implement our Q-learning based TLC algorithm with function approximation - QTLC-FA. For the sake of comparison, we also implement the following algorithms:

- **Fixed Timing TLC**: This algorithm periodically cycles through the list of feasible sign configurations, while not considering the traffic load on the lanes of the road network. The cycling period in this algorithm is a tunable parameter and we show the results of its performance for various cycling periods.

- **Self Organizing TLC** (SOTL) Cools et al. [2008]: This algorithm uses the elapsed times to decide the sign configuration i.e., traffic light switch to green happens when the elapsed time crosses a threshold, provided the number of vehicles crosses another threshold L. We set $L = 5$ in our experiments, as has been done in Cools et al. [2008].

- **Longest Queue TLC** (LTLC): Here the number of road users waiting for a traffic light to turn green is counted, and used to decide the combination of traffic lights to be turned green in the next time slot. In essence, LTLC attempts to switch the lane with the highest number of waiting road users to green.

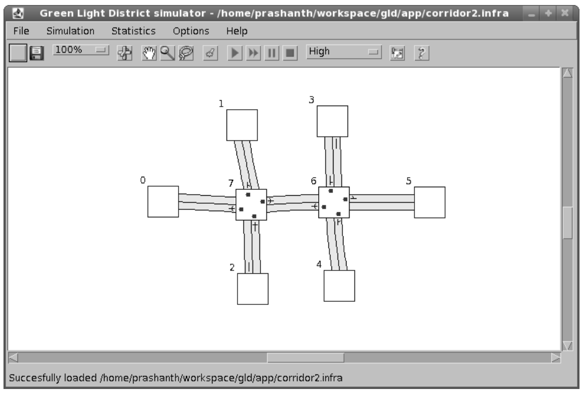
- **Q-learning with Full State Representation** (QTLC-FS): This algorithm has been described in Section 4.2.

- **Q-learning with No Priority** (QTLC-NP) Abdulhai et al. [2003]: This algorithm has a similar update rule as QTLC-FS, however, the cost function here is

$$k(s_n, a_n) = \sum_{i=1}^{N} q_i(n) + \sum_{i=1}^{N} t_i(n). \tag{5.5}$$

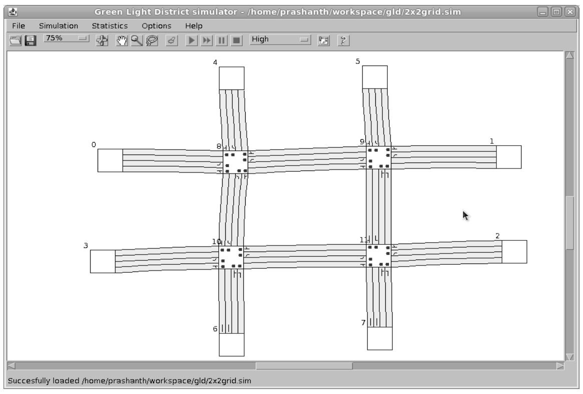
  Thus, in particular, unlike QTLC-FS, this algorithm does not assign a higher priority to main road traffic.

We consider four different network scenarios: a two-junction corridor, a 2x2-grid network, a 3x3-grid network and a eight-junction corridor. The road networks are shown in Fig. 5.1 – these are snapshots obtained from the GLD software. While we consider all roads to be of two lanes in the two-junction corridor, we consider all roads to be of four lanes in all the other settings. This was because we could implement the QTLC-FS and QTLC-NP algorithms only on the two-junction corridor setting when all roads had two lanes. When


Green Light District simulator - /home/prashanth/workspace/gld/app/corridor2.infra

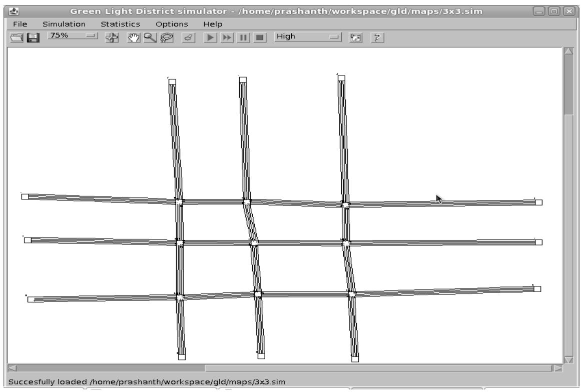
Succesfully loaded /home/prashanth/workspace/gld/app/corridor2.infra

(a) Two-Junction Corridor


Green Light District simulator - /home/prashanth/workspace/gld/2x2grid.sim

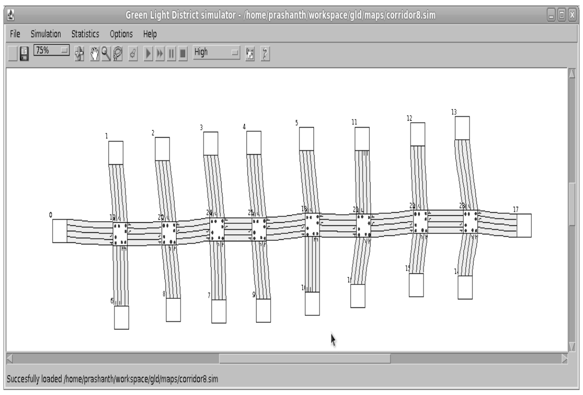
Succesfully loaded /home/prashanth/workspace/gld/2x2grid.sim

(b) 2x2-Grid Network


Green Light District simulator - /home/prashanth/workspace/gld/maps/3x3.sim

Succesfully loaded /home/prashanth/workspace/gld/maps/3x3.sim

(c) 3x3-Grid Network


Green Light District simulator - /home/prashanth/workspace/gld/maps/corridor8.sim

Succesfully loaded /home/prashanth/workspace/gld/maps/corridor8.sim

(d) Eight-Junction Corridor

Figure 5.1: Road Networks used for our Experiments

the number of lanes is increased, both of these algorithms could not be implemented because of the curse of dimensionality effect discussed previously. On the other hand, our algorithm with function approximation, QTLC-FA, was found to be easily implementable on all the settings that we considered. The simulations were conducted for 5000 cycles for all algorithms. Each road user's destination was fixed randomly using a discrete uniform distribution to choose one of the edge nodes.

In all the road networks, we set the spawn frequencies (the average rate at which traffic is generated randomly in GLD) so that the proportion of cars flowing on the main road to those on the side roads is in the ratio $100 : 5$. This setting is close to real life traffic scenarios on many busy corridor and grid networks and has also been used for instance in Cools et al. [2008].

For both QTLC-FS and QTLC-FA, we set the weights in the single-stage cost function $k(s, a)$ in (4.1) as $r_1 = s_1 = 0.5$. We thus give equal weightage to both the queue length and elapsed time components. Further, we set $r_2 = 0.6$ and $s_2 = 0.4$. This assignment gives a higher priority to the lanes on the main road than those on the side roads. The thresholds L_1 and L_2 were set to 6 and 14 respectively, taking into consideration the fact that the length of the roads in all the road networks that we study is 20. The threshold T_1 was set to 90.

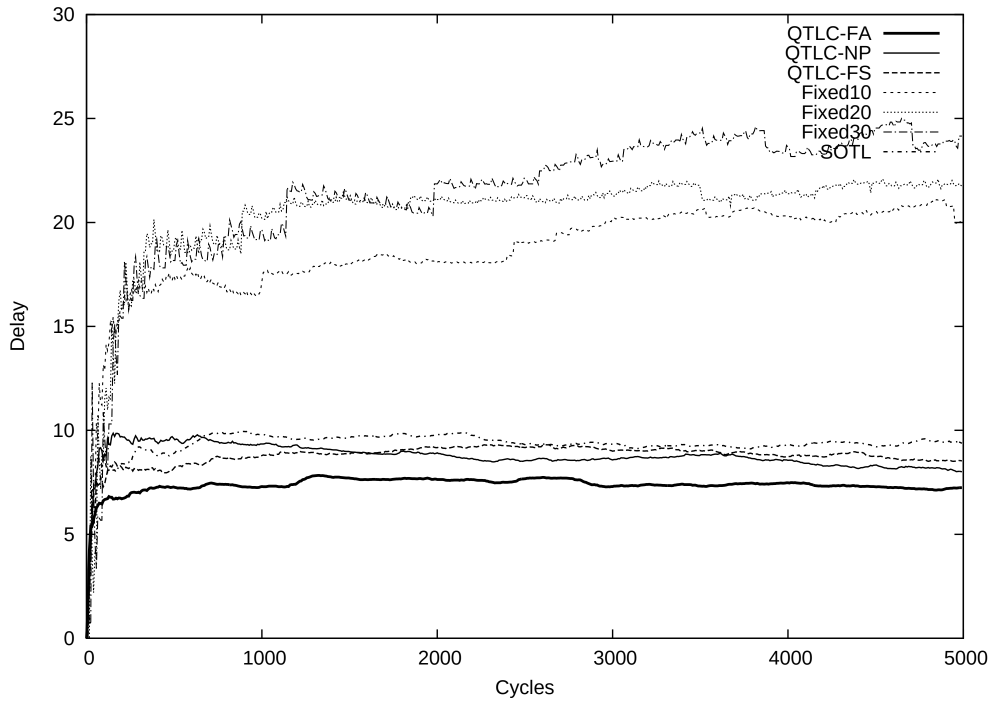
5.3.2 Results

We compare the performance of the TLC algorithms using the average junction waiting times (AJWT) and total arrived road users (TAR) i.e., the number of road users who have reached their destination. The performance plots of AJWT and TAR vs. the number of cycles in all four road networks studied are shown in Figs 5.2, 5.3, 5.4 and 5.5.

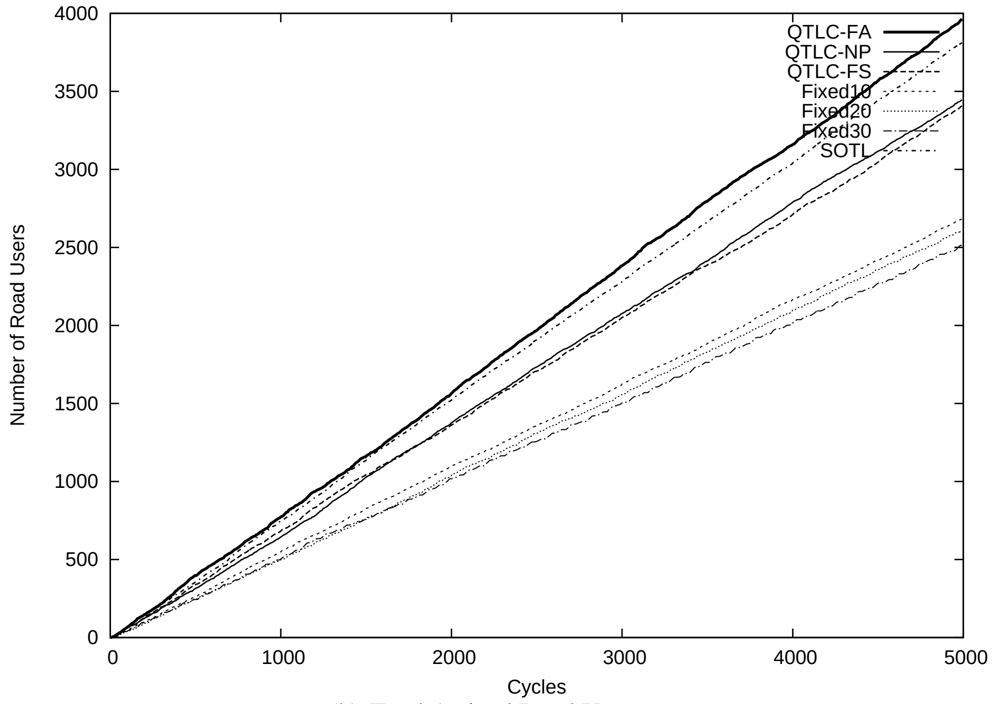
From these plots, we observe that QTLC-FA consistently shows the best results in all the four road networks studied. We now discuss the performance results in more detail.

From the AJWT and TAR plots, we observe that QTLC-FA performs better than the fixed timing TLC algorithm for all cycling periods considered in the latter. Using a broad estimate of queue lengths and elapsed times of the signalled lanes enables the QTLC-FA algorithm to adapt the sign configuration policy to the traffic situation, while fixed timing algorithms being unmindful of the current traffic situation lead to longer waiting times.

We do not show the plots of the LTLC algorithm as it performed very poorly in comparison to our algorithms. In fact, when LTLC was used, the traffic invariably entered a deadlock situation as the algorithm always arrived at a sign configuration that did not allow

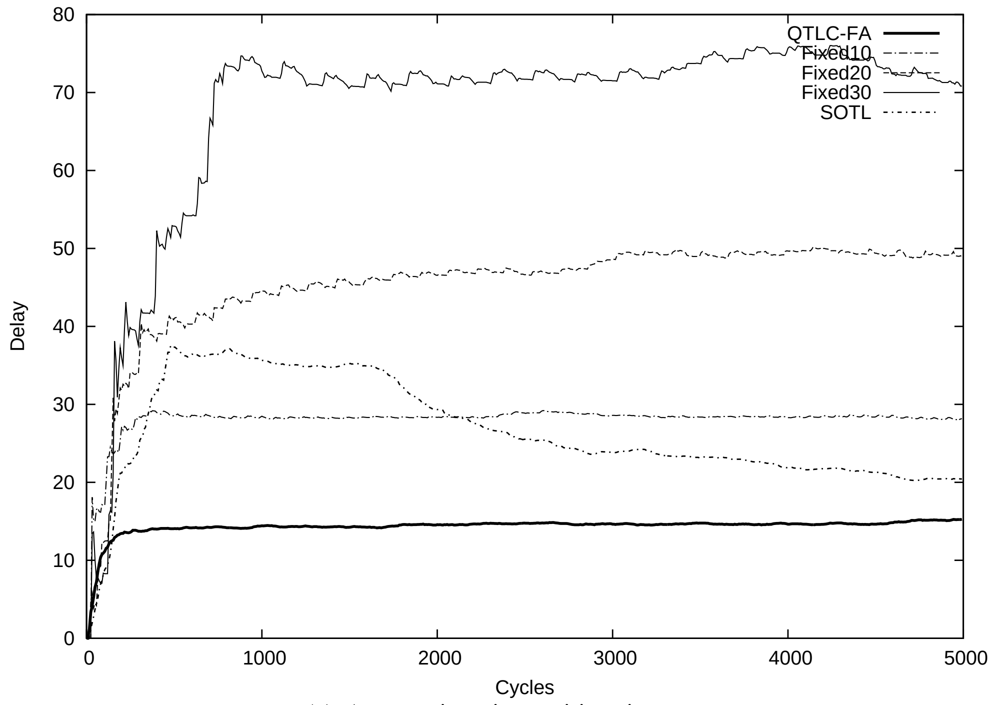


(a) Average junction waiting time

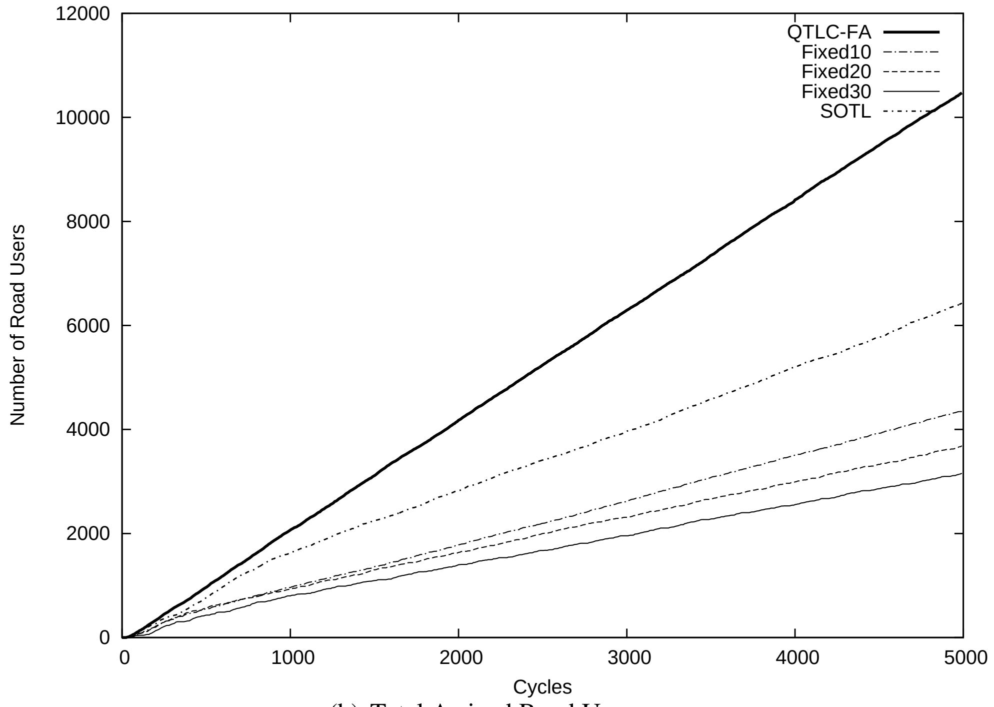


(b) Total Arrived Road Users

Figure 5.2: Performance Comparison of TLC Algorithms - Two-Junction Corridor Case

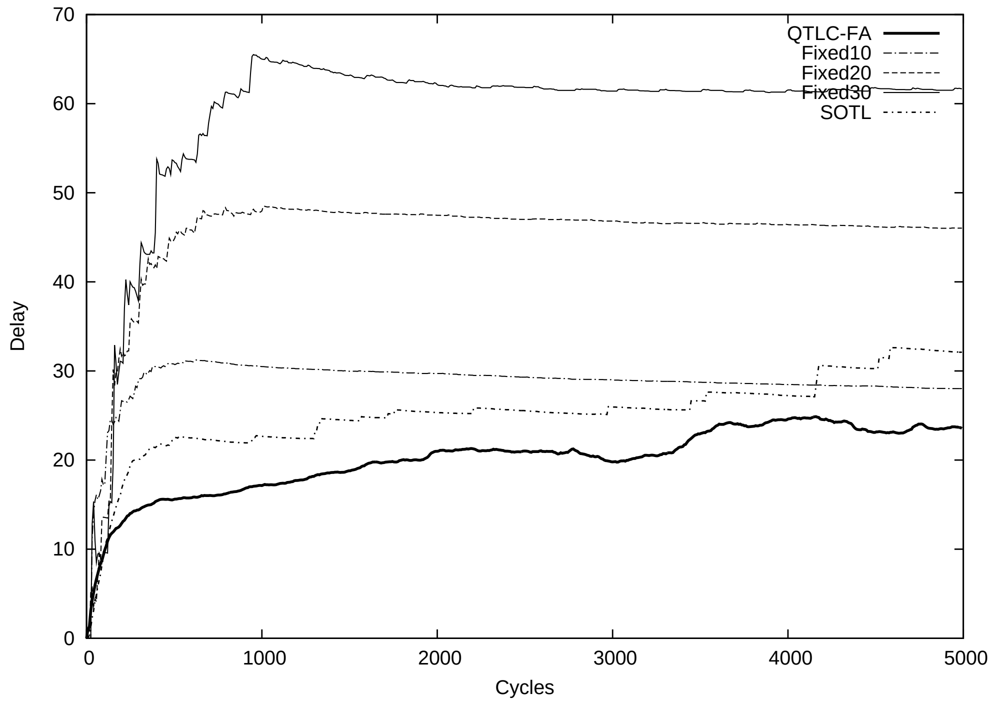


(a) Average junction waiting time

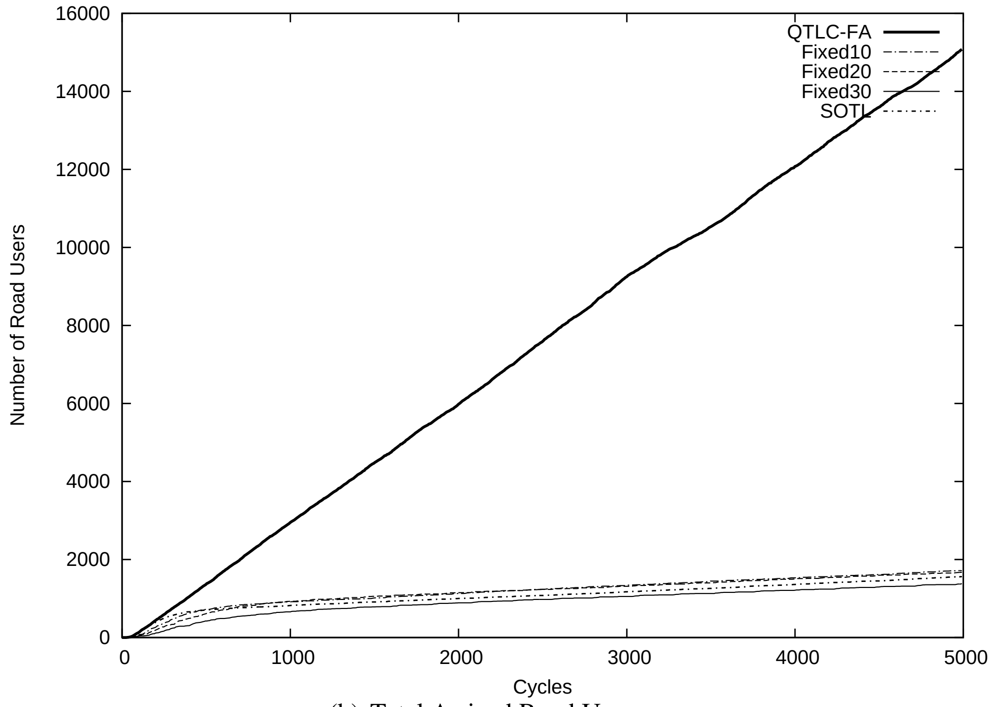


(b) Total Arrived Road Users

Figure 5.3: Performance Comparison of TLC Algorithms - 2x2-Grid Network Case

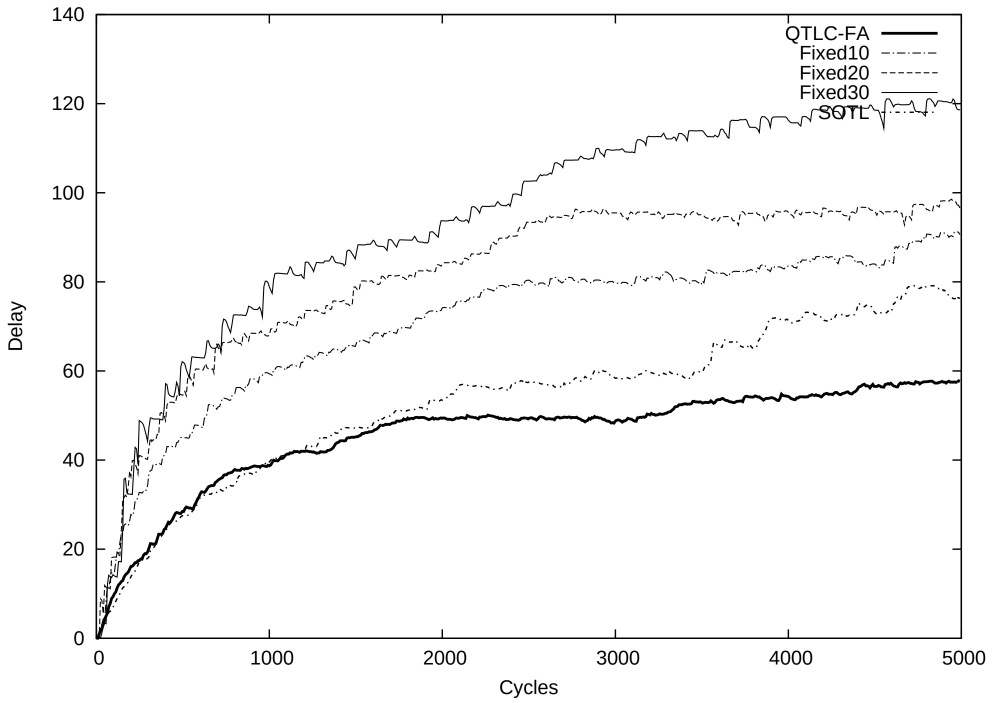


(a) Average junction waiting time

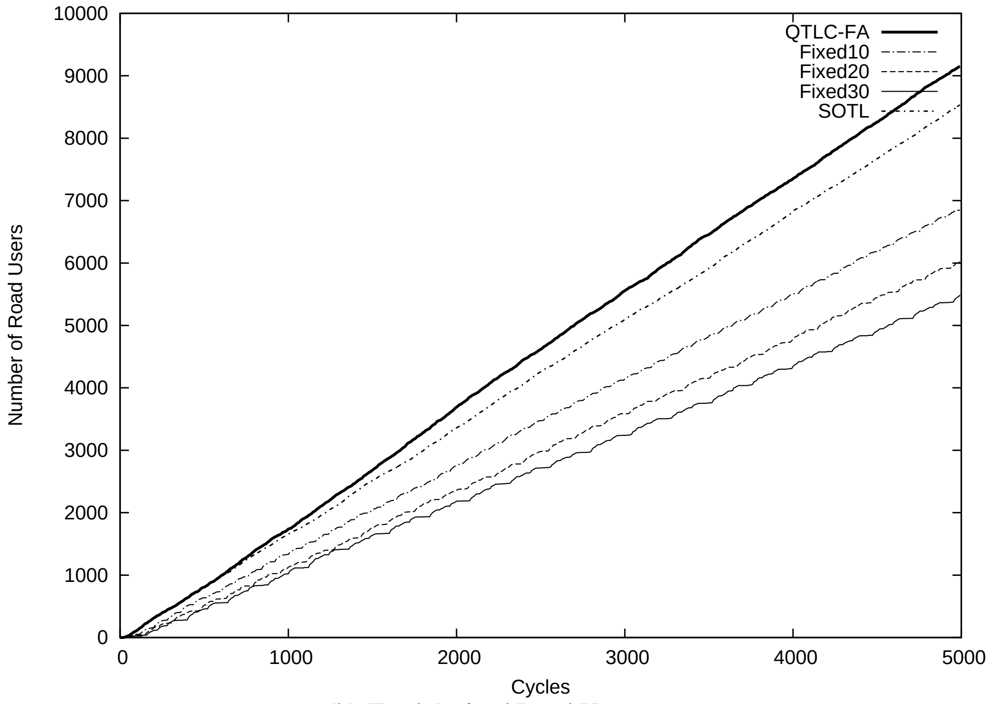


(b) Total Arrived Road Users

Figure 5.4: Performance Comparison of TLC Algorithms - 3x3-Grid Network Case



(a) Average junction waiting time



(b) Total Arrived Road Users

Figure 5.5: Performance Comparison of TLC Algorithms - Eight-Junction Corridor Case

traffic to pass across junctions. The longest queue status of the lanes then remained the same because of the gridlock leaving the sign configuration unchanged.

QTLC-FA also outperformed the SOTL algorithm (Cools et al. [2008]) in all the four road networks. Using a broad estimate of congestion on the signalled lanes apart from the elapsed times, the QTLC-FA algorithm was able to find an approximately optimal sign configuration policy that minimized the long-term discounted cost, which, in essence, ensured smooth traffic flow. While both SOTL and QTLC-FA used a rough measure of the level of congestion based on certain thresholds, QTLC-FA outperformed SOTL as it tunes the feedback policy adaptively unlike SOTL.

As explained before, we implemented the QTLC-FS and QTLC-NP (Abdulhai et al. [2003]) algorithms only on the two-junction corridor with each road having two lanes and not on the other networks because of the exponential increase in computational complexity in the case of networks with more lanes and junctions. On the other hand, QTLC-FA is easily implementable on larger network scenarios, and requires much less computation. From the performance plots in Fig. 5.2, we observe that QTLC-FA did better than both QTLC-FS and QTLC-NP, apart from the other TLC algorithms that it was compared against. A judicious choice of features that take into account both the congestion levels on the lanes of the road network as well as the elapsed times to ensure that no lane waits for a long duration for its signal to turn green, resulted in an improved performance for QTLC-FA as compared to both QTLC-FS and QTLC-NP.

From the AJWT plots, we observe that the transient phase, i.e., the initial period when QTLC-FA is tuning its parameters before stabilizing on a policy, is only a few cycles and hence, QTLC-FA converges rapidly to a near optimal sign configuration policy. Also, QTLC-FA has the advantages of any RL algorithm i.e., it adapts well to traffic conditions on different types of road networks. This is evident in the superior performance of QTLC-FA on all road networks considered above, with no specific tuning of the QTLC-FA algorithm being done for a particular road network.

5.4 Summary

Designing a road traffic management system based on wireless sensor networks that achieves high traffic flow rates with minimum congestion is a challenging task. Reinforcement learning presents an interesting paradigm for solving such problems. We designed and evaluated two Q-learning based algorithms for road traffic control on a network of junctions.

Q-learning TLCs have the advantages of model-free learning algorithms that adapt in real-time to the traffic conditions. Our Q-learning algorithm with function approximation is proposed for the first time in the literature for traffic signal control. From the performance simulations, it is observed that our QTLC-FA algorithm consistently outperforms all the other algorithms with which we showed performance comparisons over all the network settings considered.

Chapter 6

RL algorithms for average cost traffic signal control

Traffic signal control forms a crucial component of any intelligent transportation system. As we saw in the earlier chapters, designing an adaptive traffic signal control algorithm that maximizes the long-term traffic flow is an extremely challenging problem, considering the fact that a model of the system is not available in most of the real traffic environments. A second handicap to the traffic light control (TLC) algorithm is that the critical inputs - queue lengths and/or elapsed times (since signal turned red) on the lanes of the road network - are hard to obtain precisely in realistic settings and the TLC algorithm has to work with coarse estimates of these inputs. It is thus necessary to develop a TLC algorithm that minimizes a long-term cost objective using the coarse inputs of queue lengths and/or elapsed times and without assuming a system model. The TLC algorithm should be online, computationally efficient and possess the necessary convergence properties.

The problem that we consider in this chapter is one of designing TLC algorithms that minimize a long term average cost objective, while possessing the properties outlined above. Reinforcement learning (RL) is an efficient technique for developing model-free algorithms that minimize a long-term cost objective based on sample observations from simulations. RL based TLC algorithms that minimize a long run discounted cost have been proposed in literature, for instance, in Abdulhai et al. [2003] and Prashanth and Bhatnagar [2011a]. However, to the best of our knowledge, we are the first to design RL-based TLC algorithms that minimize a long-run "average cost" criterion. The motivation behind using an infinite horizon average cost framework is to understand the steady state behaviour of the traffic control system. We develop two new TLC algorithms, one based on Q-learning and the other a policy gradient actor critic algorithm for solving the traffic signal control problem in an average cost setting. While the Q-learning based TLC algorithms (Abdulhai et al. [2003], Prashanth and Bhatnagar [2011a]) proposed for the discounted cost problem are stochastic approximation analogues of the value iteration algorithm, they do not extend easily to the average cost setting. The Q-learning based TLC for average cost that we develop here is based on relative Q-value iteration scheme and has been adapted from Abounadi et al. [2002]. The second TLC algorithm that we develop is a two-timescale actor critic algorithm that incorporates policy gradient for the actor recursion and temporal difference learning for the critic.

We develop in this chapter, the first RL based TLC algorithms with function approximation that minimize a long term average cost objective. Whereas a discounted cost objective is more suitable for optimizing short term performance, the long run-average cost objective assigns equal weightage to all stages and is concerned with the steady-state system behaviour.

Our Contributions

- We develop, for the first time, two reinforcement learning algorithms with function approximation for the problem of average cost traffic signal control.

- Our first algorithm is a Q-learning based TLC algorithm and is the function approximation analogue of the Q-learning with average cost algorithm proposed in Abounadi et al. [2002].

- The second algorithm is a policy gradient actor critic based TLC algorithm. This algorithm is also shown to converge to the optimal sign configuration policy that minimizes the long run average cost.

- Both our algorithms require only coarse information on the level of congestion (for instance, low, medium or high) and do not require precise queue length information. This is unlike the algorithm of Abdulhai et al. [2003] that requires precise queue length estimates.

- We study the performance of our TLC algorithms in the context of a two-junction corridor, a five-junction corridor and a 2x2-grid network. From the performance comparisons of our algorithms between themselves and with a range of fixed timing TLC algorithms, we observe that the policy gradient actor critic based TLC algorithm performs the best, while also converging rapidly to the optimal sign configuration policy.

The rest of the chapter is organized as follows: In Section 6.1, we present our learning algorithms with average cost for traffic light control. In Section 6.2, we discuss the implementation of the various TLC algorithms and present the performance simulation results. Finally, in Section 6.3 we provide the concluding remarks.

The traffic signal control MDP used for average cost analysis here is the same as the ones used in the previous chapters. In other words, we use the same states, actions and the cost function of the traffic signal control problem described in Section 4.1 here. However, the objective here is to develop algorithms that minimize a certain long run average cost objective, unlike earlier where a discounted cost objective was used. Specifically, we now present two traffic light control algorithms - both using function approximation in an average cost framework. The first is a Q-learning with average cost based TLC scheme, while the second is a policy gradient actor-critic TLC algorithm.

6.1 Our TLC Algorithms

The MDP for the problem of traffic signal control corresponds to the sequence $\{s_n\}$, with s_n as defined in Section 4.1, with which is associated the control sequence $\{a_n\}$. Let $p(i, j, a)$ denote the transition probability of the MDP for transiting from state i to j under action a. We consider here an infinite horizon average cost framework, where our aim is to find a sequence $\{a_n\}$ of actions that minimize the "average cost"

$$\boxed{\hat{\lambda} = \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} c(s_n, a_n) \triangleq \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} c_{n+1}, \qquad (6.1)}$$

starting from any given state i (i.e., $s_0 = i$).

Let $h(i), i \in \mathcal{S}$ be the differential cost function corresponding to state i. Then $h(.)$ satisfies

$$\lambda + h(i) = \min_a \sum_j p(i, j, a)(c(i, a) + h(j)), \forall i \in \mathcal{S}, \tag{6.2}$$

where λ is the optimal cost.

Define the Q-factors $Q(i,a), i \in \mathcal{S}, a \in \mathcal{A}(i)$ as

$$Q(i,a) = \sum_j p(i,j,a)(c(i,a) + h(j)). \tag{6.3}$$

The Q-factors then satisfy the Bellman equation of optimality

$$\boxed{\lambda + Q(i,a) = \sum_j p(i,j,a)(c(i,a) + \min_{b \in \mathcal{A}(j)} Q(j,b)), \tag{6.4}}$$

for all $i \in \mathcal{S}, a \in \mathcal{A}(i)$.

The Q-learning algorithm with full state representation (that we present next) addresses the case when the system model is not known, however, state and action spaces are manageable. Further, the next two algorithms viz., Q-learning with function approximation and policy gradient actor-critic in addition to considering the case of 'lack of system model', also effectively handle large state and action spaces.

6.1.1 Q-learning with full state representations (QTLC-FS-AC)

Our QTLC-FS-AC algorithm estimates the 'Q-factors' $Q(i,a)$ of all feasible state-action tuples (i,a) i.e., those with $i \in \mathcal{S}$ and $a \in \mathcal{A}(i)$ using the relative value iteration of Q-factors and has been adapted from Abounadi et al. [2002]. Let $s_{n+1}(i,a)$ denote the state of the system at instant $(n+1)$ when the state at instant n is i and action chosen is a. Let $Q_n(i,a)$ denote the Q-value estimate at instant n associated with the tuple (i,a). The relative Q-value iteration (RQVI) scheme is

$$\boxed{Q_{n+1}(i,a) = \sum_j p(i,a,j)(c_{n+1} + \min_{b \in \mathcal{A}(j)} Q_n(j,b)) - \min_{r \in \mathcal{A}(s)} Q_n(s,r). \tag{6.5}}$$

The QTLC-FS-AC algorithm is a stochastic approximation analogue of the RQVI and updates according to

$$\boxed{Q_{n+1}(i,a) = Q_n(i,a) + \alpha(n)(c_{n+1} + \min_{b \in \mathcal{A}(j)} Q_n(j,b) - \min_{r \in \mathcal{A}(s)} Q_n(s,r)), \tag{6.6}}$$

for all feasible (i,a) tuples. Here $\alpha(n), n \geq 0$ are the step-sizes that satisfy $\sum_n \alpha(n) = \infty$ and $\sum_n \alpha^2(n) < \infty$. Upon convergence, one obtains the optimal Q-values $Q^*(i,a)$ that are seen to satisfy the Bellman equation (6.4) and $\min_{a\in\mathcal{A}(i)} Q_n(i,a)$ gives the optimal differential cost $h^*(i)$. The optimal action in state i then corresponds to $arg\min_{b\in\mathcal{A}(i)} Q^*(i,b)$. A convergence proof of this algorithm can be found in Abounadi et al. [2002].

6.1.2 Q-learning with function approximation (QTLC-FA-AC)

While the QTLC-FS-AC algorithm is useful in small state and action spaces, it becomes computationally expensive for larger road networks involving multiple junctions. This is because of the exponential increase in the sizes of the state and action spaces with the number of junctions. To alleviate this problem of curse of dimensionality, we incorporate feature based methods. These methods handle this problem by making computational complexity manageable. While the QTLC-FS-AC algorithm as such requires complete state information and so is less efficient, its function approximation based variant parameterizes the value function.

We now present QTLC-FA-AC, a Q-Learning based TLC that uses function approximation. Here we associate with each state-action tuple (i,a), a state-action feature denoted by $\phi_{i,a}$. The Q-function is then approximated as

$$Q^*(i,a) \approx \theta^T \phi_{i,a}. \tag{6.7}$$

Let s_n, s_{n+1} denote the state at instants $n, n+1$, respectively, measured online. Let θ_n be the estimate of the parameter θ at instant n. Let s be any fixed state in $\mathcal{S}$.

The algorithm QTLC-FA-AC uses the following update rule:

$$\theta_{n+1} = \theta_n + \alpha(n)\phi_{s_n,a_n}(c_{n+1} + \min_{v\in\mathcal{A}(s_{n+1})} \theta_n^T\phi_{s_{n+1},v} - \min_{r\in\mathcal{A}(s)} \theta_n^T\phi_{s,r}), \tag{6.8}$$

where θ_0 is set arbitrarily. In (6.8), the action a_n is chosen in state s_n according to $a_n = arg\min_{v\in A(s_n)} \theta_n^T\phi_{s_n,v}$. For our experiments, we chose the following features (as in Prashanth and Bhatnagar [2011a]):

$$\phi_{s_n,a_n} = (\phi_{q_1(n)},\ldots,\phi_{q_N(n)},\phi_{t_1(n)},\ldots,\phi_{t_N(n)},\phi_{a_1(n)},\ldots,\phi_{a_m(n)})^T, \tag{6.9}$$

where for some given thresholds $L1, L2$ (on queue lengths) and $T1$ (on elapsed times),

$$\phi_{q_i(n)} = \begin{cases} 0 & \text{if } q_i(n) < L1 \\ 0.5 & \text{if } L1 \le q_i(n) \le L2 \\ 1 & \text{if } q_i(n) > L2 \end{cases}$$

$$\phi_{t_i(n)} = \begin{cases} 0 & \text{if } t_i(n) \le T1 \\ 1 & \text{if } t_i(n) > T1 \end{cases}$$

Note that the parameter θ_n has dimension the same as that of ϕ_{s_n,a_n}. Again the advantage here is that instead of updating the Q-values for each feasible (s,a)-tuple as before, one estimates these according to the parameterization (6.7).

6.1.3 Policy Gradient Actor-Critic TLC (PG-AC-TLC)

Actor critic algorithms are reinforcement learning algorithms that are based on the policy iteration (PI) method for MDP. The classical PI algorithm proceeds via two loops - the inner loop performs policy evaluation for a given policy while the outer loop performs policy improvement. Actor-critic algorithms use two-timescale stochastic approximation in order to perform both updates (evaluation and improvement) simultaneously at each instant. The difference in step-size schedules results in an appropriate algorithmic behaviour.

Our PG-AC-TLC algorithm incorporates policy gradients for the actor and temporal difference learning in the critic and has been adapted from Bhatnagar et al. [2009b]. The idea here is that the policy is considered to be parametrized, in addition to the value function. We consider a class of parametrized randomized policies and linear function approximation for the value function. Specifically, we assume that policies $\pi_\theta(i,a)$, parameterized by $\theta \in \Re^d$ have the form

$$\boxed{\pi_\theta(i,a) = \frac{e^{\theta^\top \phi_{i,a}}}{\sum_{a' \in \mathcal{A}(i)} e^{\theta^\top \phi_{i,a'}}}, \quad \forall s \in \mathcal{S}\ ,\ \forall a \in \mathcal{A}, \qquad (6.10)}$$

where each $\phi_{i,a}$ is a d-dimensional feature vector as before.

Further, we let $V^\pi(i) \approx v^\top f_i, i \in \mathcal{S}$, to be the approximation to the differential cost function, where $f_i, i \in \mathcal{S}$ are $\hat{d}$-dimensional state features and v the corresponding parameter vector. Let $\psi_{ia} = \nabla \log \pi_\theta(i,a)$ denote the compatible state-action features. When $\pi_\theta(i,a)$ are selected as in (6.10), it can be seen that $\psi_{ia} = \phi_{i,a} - \sum_{a' \in \mathcal{A}(i)} \pi_\theta(i,a')\phi_{i,a'}$.

The PG-AC-TLC algorithm is as follows:

$$\hat{J}_{n+1} = (1-\alpha_n)\hat{J}_n + \alpha_n c_{n+1}, \tag{6.11}$$
$$\delta_n = c_{n+1} - \hat{J}_{n+1} + v_n^\top f_{s_{n+1}} - v_n^\top f_{s_n} \tag{6.12}$$
$$v_{n+1} = v_n + \alpha_n \delta_n f_{s_n}, \tag{6.13}$$
$$\theta_{n+1} = \Gamma(\theta_n + \beta_n \delta_n \psi_{s_n a_n}). \tag{6.14}$$

Here, $\{s_n\}$ is the sequence of states visited by the MDP, i.e., we consider a single trajectory of states for the MDP. Further $\{a_n\}$ is the sequence of actions obtained upon following the randomized policy π. Here α_n and $\beta_n, n \geq 0$ are two step-size sequences that satisfy

$$\sum_n \alpha_n = \sum_n \beta_n = \infty; \sum_n (\alpha_n^2 + \beta_n^2) < \infty, \lim_{n\to\infty} \frac{\beta_n}{\alpha_n} = 0.$$

It has been shown in Bhatnagar et al. [2009b] that if one replaces δ_n by $\delta_n^\pi \triangleq c_{n+1} - \hat{J}_{n+1} + v_n^{\pi\top} f_{s_{n+1}} - v_n^{\pi\top} f_{s_n}$, where $\lim_{n\to\infty} v_n = v^\pi$ (assuming fixed π), then

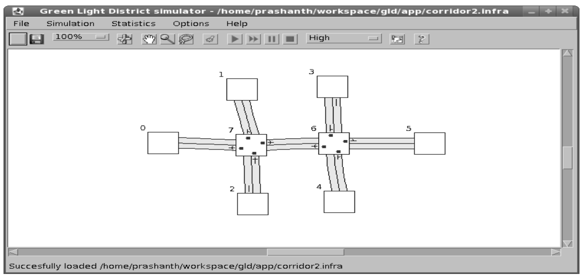
$$\mathbf{E}[\delta_n^\pi \psi_{s_n a_n} | \theta] = \nabla J(\theta) + \varepsilon(\theta),$$

where $\varepsilon(\theta)$ is an error term that arises from the use of linear function approximation. Further, it has been shown in Bhatnagar et al. [2009b] that if $||\varepsilon(\theta)||$ is "small", then $\theta_n, n \geq 0$ given by the recursions (6.11) - (6.14) converge asymptotically to a "small" neighbourhood of the local minima of $J(.)$.

6.2 Simulation Experiments

6.2.1 Implementation

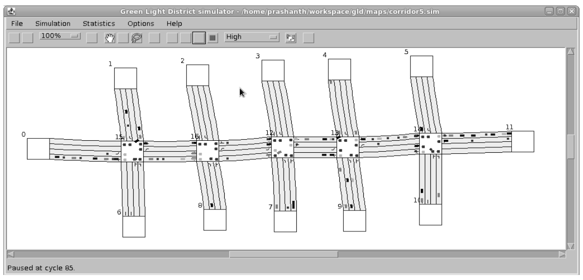
We use the Green Light District (GLD) simulator (Wiering et al. [2004]) for implementation and evaluation of our TLC algorithms - QTLC-FS-AC, QTLC-FA-AC and PG-AC-TLC, respectively. Recall that whereas QTLC-FS-AC incorporates full state representations, the other two algorithms viz., QTLC-FA-AC and PG-AC-TLC incorporate function approximation. For the sake of comparison, we also implement a range of fixed timing TLC algorithms that periodically cycle through the list of feasible sign configurations irrespective of the traffic conditions. The cycling period here is a tunable parameter and we show the results of the performance of these algorithms for various cycling periods.



Green Light District simulator - /home/prashanth/workspace/gld/app/corridor2.infra

Succesfully loaded /home/prashanth/workspace/gld/app/corridor2.infra

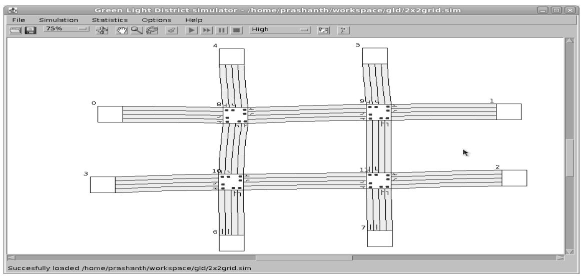
(a) Two-Junction Corridor



Green Light District simulator - /home/prashanth/workspace/gld/maps/corridor5.sim

Paused at cycle 65.

(b) Five-Junction Corridor



Green Light District simulator - /home/prashanth/workspace/gld/2x2grid.sim

Succesfully loaded /home/prashanth/workspace/gld/2x2grid.sim

(c) 2x2-Grid Network

Figure 6.1: Road Networks used for our Experiments

We consider three different network scenarios: a two-junction corridor, a five-junction corridor and a 2x2-grid network. We show snapshots of these networks obtained from the GLD software in Fig. 6.1. The simulations are conducted for 5000 cycles in all algorithms. Each road user's destination is fixed randomly, using a discrete uniform distribution, to choose one of the edge nodes. The spawn frequency i.e., the rate at which traffic is generated randomly in GLD, was set in a way that ensures that the proportion of cars flowing on the main road to those on the side roads is in the ratio 100:5.

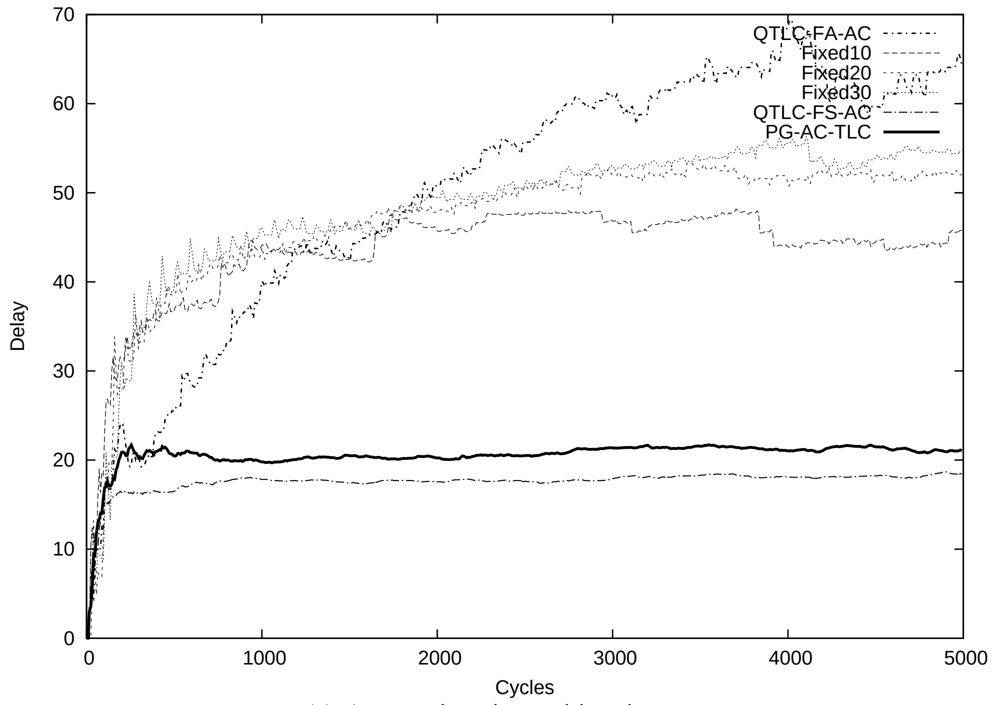
The performance of QTLC-FS-AC is tested only on the two-junction corridor while that of QTLC-FA-AC and PG-AC-TLC is tested on all the settings in Fig. 6.1. QTLC-FS-AC could not be implemented on larger road networks because of the exponential blow up in computational complexity (described previously) with the numbers of lanes and junctions (when full state representations are used). On the other hand, QTLC-FA-AC and PG-AC-TLC are easily implementable even on larger road networks, and require much less computation.

For all the three TLC algorithms, we set the weights in the single stage cost function $c(s, a)$ in (4.1) as $r_1 = s_1 = 0.5$ and $r_2 = 0.6, s_2 = 0.4$. This assignment gives a higher priority to the lanes on the main road than those on the side roads, while according equal weightage to both queue length and elapsed time components of the cost function.

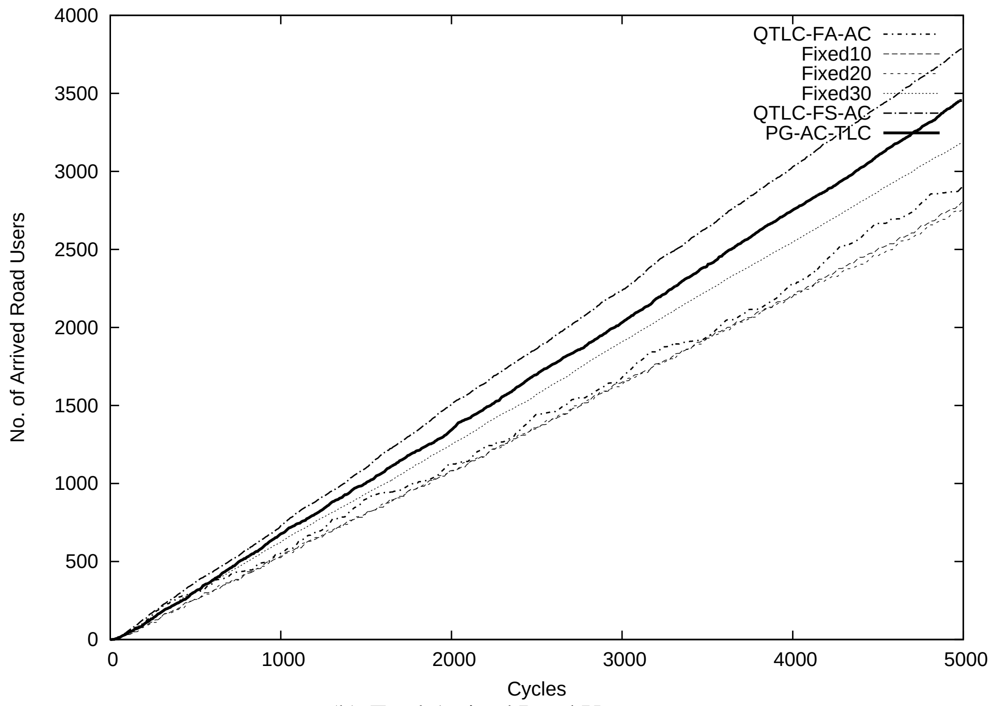
6.2.2 Results

Figs. 6.2, 6.3 and 6.4 show the plots of average junction waiting time and total arrived road users using the various algorithms on the three road network settings. From the above plots, we observe that the policy gradient actor critic based TLC algorithm (PG-AC-TLC) shows the best overall performance as compared to the other TLC algorithms considered. In the case of the two-junction corridor, QTLC-FS-AC shows the best results. This is however expected because QTLC-FS-AC uses the knowledge of the full state whereas the PG-AC-TLC and QTLC-FA-AC use only coarse information on whether the level of congestion is in the low, medium or high range, and also whether the elapsed time is below or above a threshold.

We also observe that the parameter θ converges in the case of PG-AC-TLC algorithm, while the same is not true of QTLC-FA-AC algorithm. This is evident in Fig. 6.5 where the convergence of θ for PG-AC-TLC and oscillation of θ for QTLC-FA-AC are illustrated using one of the co-ordinates of θ for the case of a 2x2-grid network. This is because QTLC-FA-AC suffers from the off-policy problem (Bertsekas and Tsitsiklis [1996b])

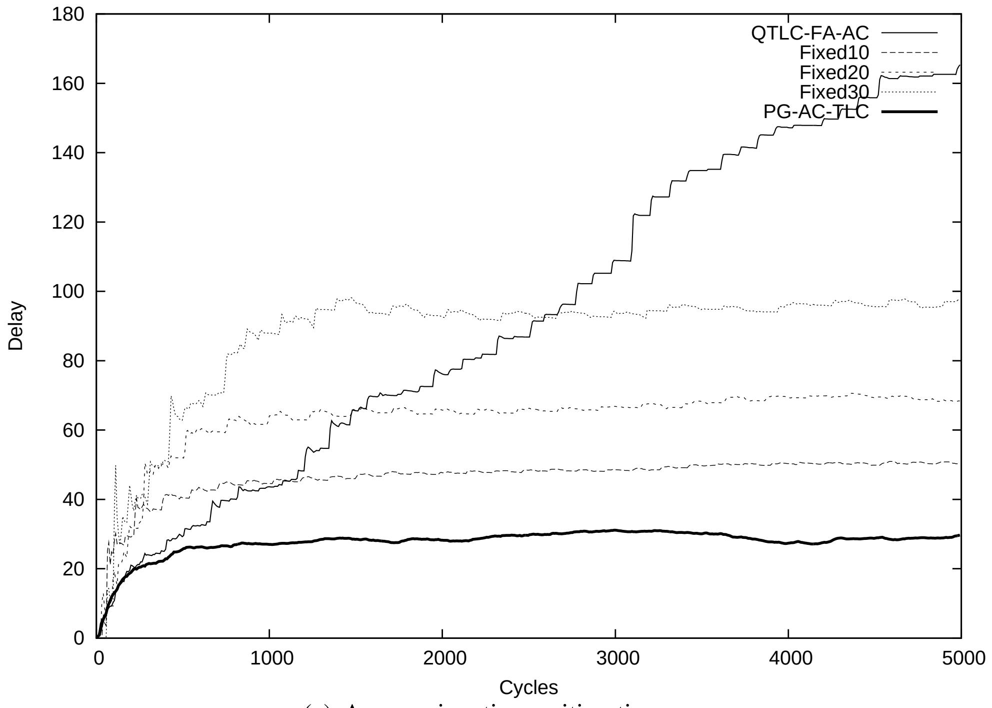


(a) Average junction waiting time

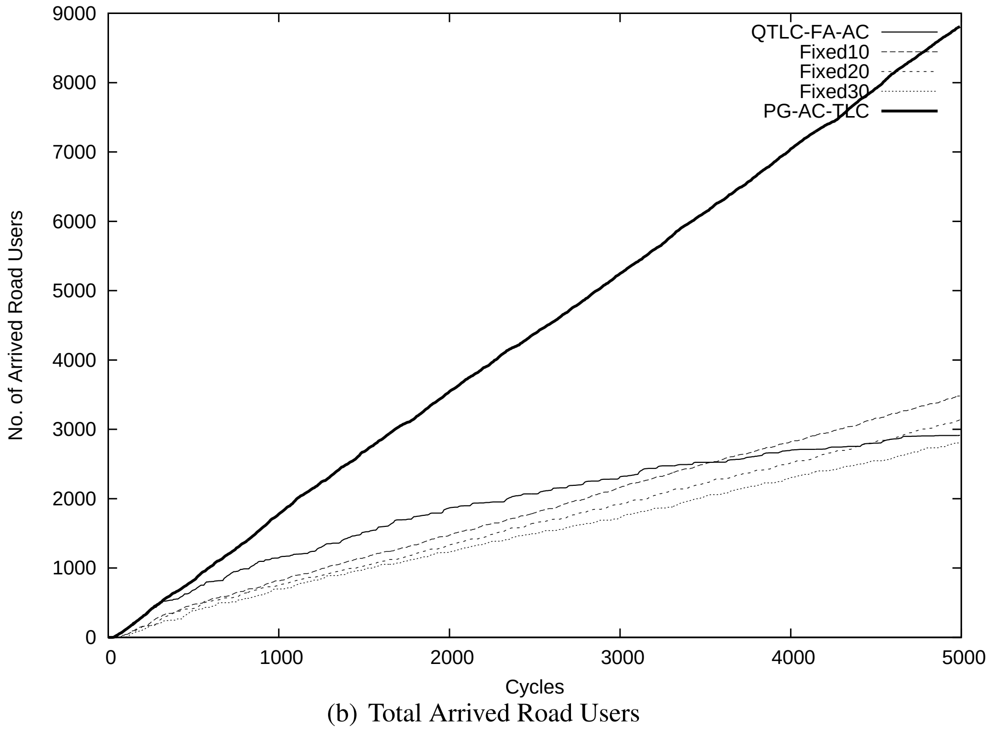


(b) Total Arrived Road Users

Figure 6.2: Performance Comparison of TLC Algorithms - Two-Junction Corridor Case

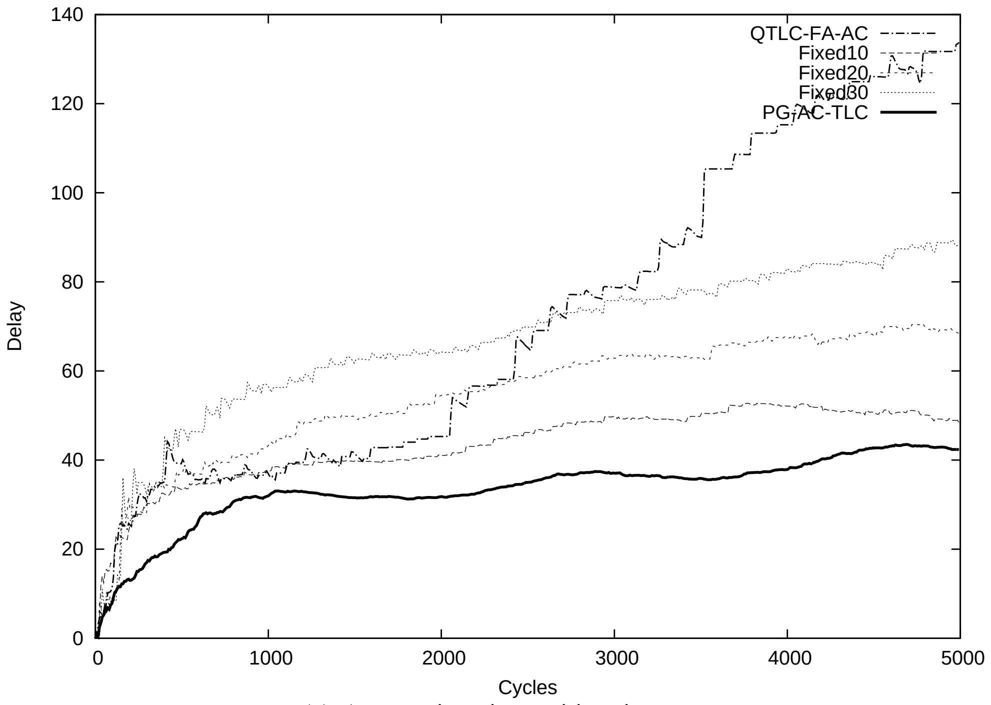


(a) Average junction waiting time

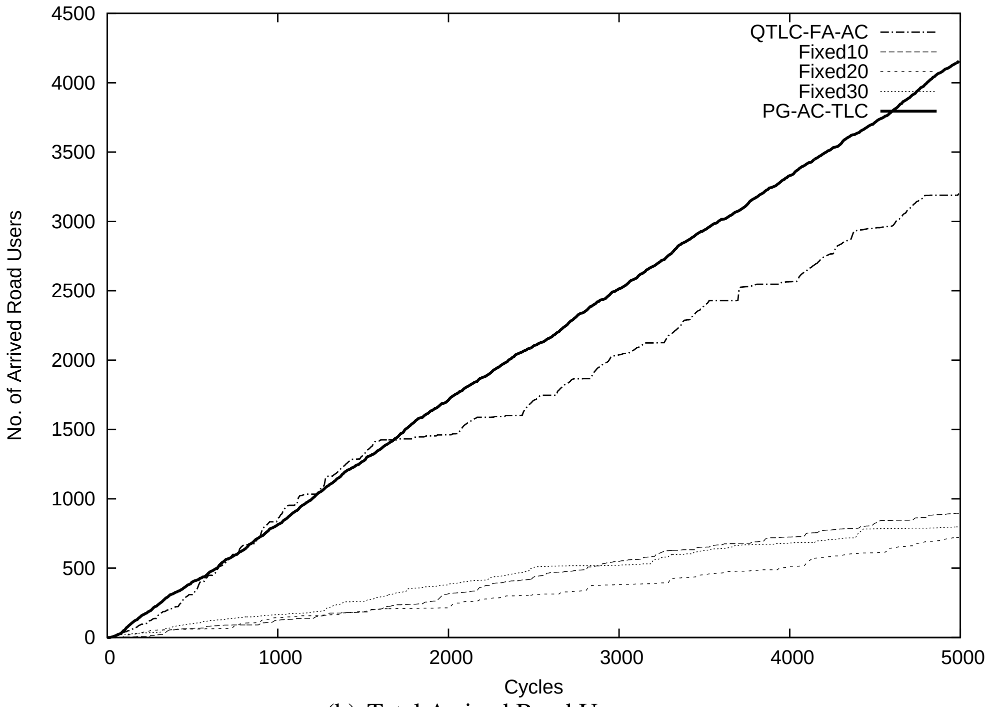


(b) Total Arrived Road Users

Figure 6.3: Performance Comparison of TLC Algorithms - 2x2-Grid Network Case

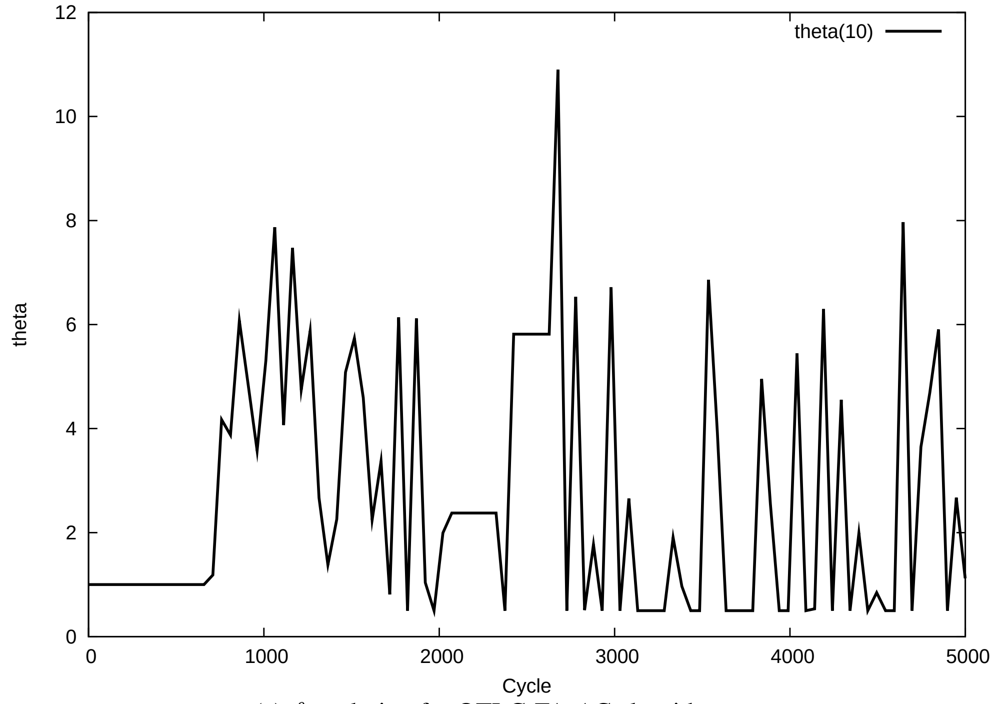


(a) Average junction waiting time

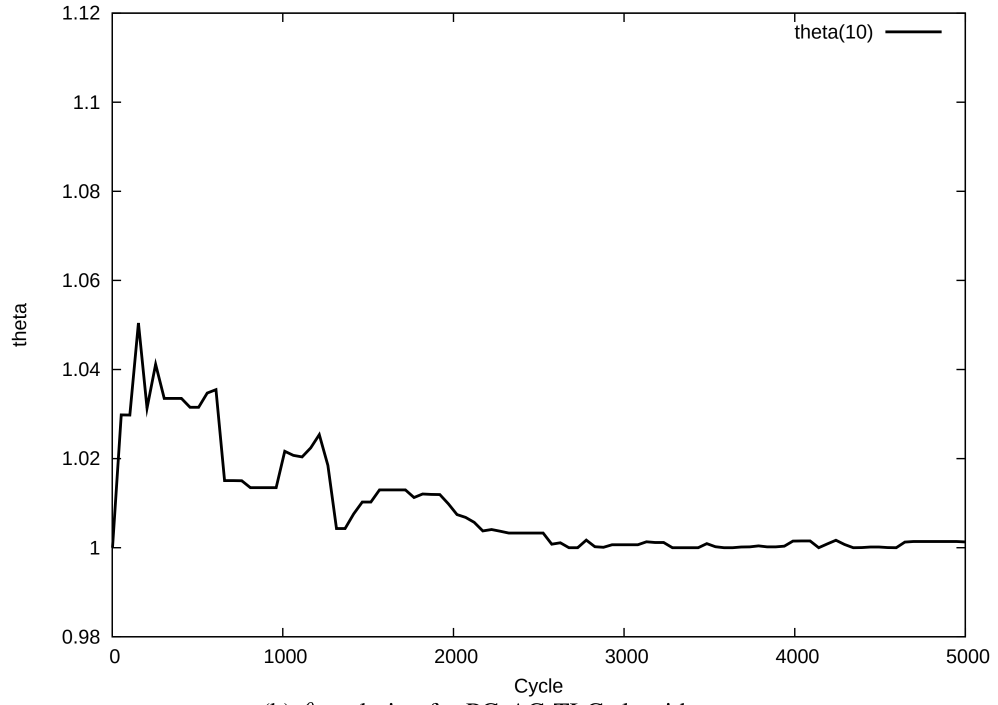


(b) Total Arrived Road Users

Figure 6.4: Performance Comparison of TLC Algorithms - Five-Junction Corridor Case



(a) θ evolution for QTLC-FA-AC algorithm



(b) θ evolution for PG-AC-TLC algorithm

Figure 6.5: Convergence of θ - illustration using one of the co-ordinates in the case of a 2x2-Grid Network

unlike PG-AC-TLC that does not suffer from this problem because of the use of multi-timescale stochastic approximation.

6.3 Summary

Our goal in this chapter was to design an algorithm for average cost traffic signal control in order to optimize steady state system performance. We developed two reinforcement learning algorithms with average cost - one an analogue of Q-learning in a function approximation setting and another a policy gradient actor critic algorithm. From the performance comparisons, we observe that the policy gradient actor critic algorithm showed the best overall performance. On a two-junction corridor, the full state representation algorithm was better, however, it suffers from the limitation of not being implementable on larger road networks unlike algorithms that use function approximation.

In conclusion, we mention two important future research directions below:

- Apart from the policy gradient based actor-critic algorithms, there are other algorithms based on natural gradients proposed in Bhatnagar et al. [2009b]. It would be interesting to develop TLC algorithms that combine natural gradients and function approximation for the problem of average cost traffic control, as algorithms based on natural gradients have been found to yield better performance.

- We used arbitrarily set, fixed values for the thresholds $L1, L2, T1$ in our TLC algorithms. An interesting problem is to develop a threshold tuning algorithm that combines with policy optimization in order to find an optimal policy via an optimal choice of thresholds. This is precisely the contribution of the next chapter, where an SPSA based threshold tuning algorithm that works in conjunction with any graded threshold-based TLC to find the optimal threshold values, is presented.

Chapter 7

Threshold tuning using stochastic optimization for graded signal control

Adaptive control of traffic lights forms an integral part of the design of any intelligent transportation system. Information about the queue lengths on the lanes of the road network is a necessary input for many adaptive traffic light control (TLC) algorithms. However, in practice, this information is hard to obtain with precision, but instead certain thresholds on the queue lengths can be used to classify the traffic condition. Thresholds may also be used on the elapsed time, i.e., the amount of time since the signal turned red on any lane. In essence, the TLC algorithm could consider switching a lane to green if either the queue length or the elapsed time exceed certain prescribed thresholds. Designing a TLC algorithm that maximizes the traffic flow in the long term across various junctions in a road network, is a challenging problem. This is because of the variability in traffic patterns and the lack of precise state information. Many times, one only has a coarse knowledge of the level of congestion on a lane, such as *low*, *medium* or *high*. Such information can be better obtained using graded feedback policies. For instance, for some suitably chosen threshold levels L_1 and L_2 for each lane with $L_1 < L_2$, one could mark congestion as being in the low range if the queue length on that lane is below L_1. On the other hand, if the queue length is between L_1 and L_2, congestion could be said to be in the medium range, while if it is above L_2, it could be inferred to be high. In such (graded threshold based) policies, however, the choice of the thresholds (such as L_1 and L_2) plays a critical role and a problem is to select such thresholds optimally. Graded threshold policies have been considered for instance in Prashanth and Bhatnagar [2011a]. These policies however consider fixed values for the thresholds. A problem of interest is to find optimal thresholds for such classes of feedback policies.

We consider the problem of designing new TLC algorithms with graded thresholds that are however set optimally. For obtaining optimal thresholds, we develop an algorithm that works in conjunction with the given TLC algorithm and tunes the thresholds in order to optimize a given cost objective. The problem can be seen to be equivalent to one of finding an optimal feedback policy within a given class of parameterized feedback policies with the underlying parameter in general being a vector of the various thresholds on which the policies depend. In a stochastic dynamic setting as we consider, it is necessary to design an online algorithm to tune the thresholds on queue lengths and/or elapsed times and thereby tune the parameter of the associated feedback policy. The threshold tuning algorithm should be easily implementable, have the necessary convergence properties and most importantly, work for any graded threshold based TLC algorithm and in general for any parameterized class of policies.

In this chapter, (a) we design an efficient Simultaneous Perturbation Stochastic Approximation (SPSA) based online threshold tuning algorithm that works with any graded threshold based TLC algorithm, and (b) we propose three new TLC algorithms - a variant of Q-learning with full state representation that incorporates a novel state aggregation scheme, a function approximation based Q-learning TLC algorithm that enhances the scheme proposed in Prashanth and Bhatnagar [2011a] by incorporating a new feature selection procedure, and a novel priority based algorithm. We study our threshold tuning algorithm in conjunction with these TLC algorithms.

The SPSA based online threshold tuning algorithm that we design here falls in the general category of simulation-based optimization - a collection of methods that do not require a priori knowledge or assumption on the traffic system dynamics. Further, our algorithm has the added advantage that it is easily implementable, possesses the necessary convergence properties and works with any graded threshold based TLC algorithm regardless of the network and traffic conditions. Specifically, we incorporate the one-simulation variant of the SPSA algorithm that uses Hadamard matrix based deterministic perturbations from Bhatnagar et al. [2003] to tune the graded thresholds used in the sign configuration policy. To the best of our knowledge, we are the first to explore threshold tuning for different classes of graded TLC algorithms. We study the empirical performance of our threshold tuning algorithm when combined with the three new threshold-based TLC algorithms mentioned above. From the simulation experiments, we observe that our tuning algorithm is easily implementable over all road network settings and converges rapidly to the optimal thresholds in any graded threshold based TLC algorithm. Further, the additional computational effort required in finding the optimal thresholds over algorithms with fixed thresholds is observed to be small.

Amongst the three TLC algorithms that we propose, the Q-learning based TLC algorithm (in the full state case) incorporates a novel state aggregation technique to handle large state spaces than regular Q-learning (with full state representations). The combination of Q-learning based TLC with state aggregation is seen to result in a significant reduction in the cardinality of the state space, which results in a computational advantage over regular Q-learning (i.e., without state aggregation). Further, we also develop a Q-learning based TLC algorithm with function approximation along the lines of Prashanth and Bhatnagar [2011a], however, with an important difference in the feature selection procedure. The feature selection procedure that we propose intelligently combines the queue lengths and elapsed times with the feasible sign configurations to arrive at the state-action features. This is unlike the features used in Prashanth and Bhatnagar [2011a] in which the state and the action components of the features are almost independent. By obtaining features in a novel manner whereby the state and action components cannot be separated, we observed that the resulting Q-learning algorithm with function approximation significantly outperforms the algorithm proposed in Prashanth and Bhatnagar [2011a]. The priority based TLC that we propose assigns priorities to the various sign configurations based on graded thresholds.

The combination of our threshold tuning algorithm with the other TLC algorithms mentioned above results in several interesting consequences. For instance, in the case of the Q-learning with full state representation, our threshold tuning algorithm results in finding an optimal way of clustering states so as to reduce the cardinality of the state space and in the case of the Q-learning algorithm with function approximation, our threshold tuning algorithm results in tuning online the associated state representation features, and thereby obtains the optimal features within a parameterized class of features (that are parameterized by the threshold parameters). In the context of reinforcement learning (RL), developing algorithms for feature adaptation is currently a hot area of research in itself.

The rest of the chapter is organized as follows: In Section 7.1, we describe in detail the problem framework. In Section 7.2, we present our threshold tuning algorithm. In Section 7.3, we present our algorithms for traffic signal control and combine them with the threshold tuning algorithm. In Section 7.5, we discuss the implementation of the various TLC schemes where our algorithm was used, in order to find the optimal thresholds in each scheme and present the performance simulation results. Finally, in Section 7.6, we provide the concluding remarks.

7.1 Problem Formulation

We study in this chapter the problem of maximizing traffic flow through the adaptive control of traffic lights at intersections. Our solution methodology consists of two important components - a threshold-based sign configuration policy obtained from a TLC algorithm for a fixed set of thresholds and another algorithm that operates on top of the TLC algorithm itself for tuning the thresholds. A sign configuration here refers to all the signals associated with a phase, i.e., those that can be switched to green simultaneously. The TLC algorithms, that we consider, indicate when to switch a sign configuration and are all based on graded thresholds. The threshold tuning algorithm, that is common to all the schemes, tunes the graded feedback policy to find the optimal parameters (thresholds) and does not indicate how to switch the sign configurations - a task that is performed by the particular TLC algorithm used. Apart from designing efficient TLC algorithms, an important problem that we address is one of finding the "optimal" set of parameters to use for a given TLC algorithm and which give the optimal feedback policies within the given parameterized class of policies. To the best of our knowledge, this problem has not been addressed in the literature previously.

Each TLC algorithm that we consider uses as input - the queue lengths along the individual lanes leading to the intersection and the time elapsed since the last signal light switch over on each lane. The queue length input is used to minimize (depending on the objective) the average junction waiting times of the road users, while the elapsed time input is used to ensure fairness, i.e., no lane is allowed to stay green for a long time at the cost of other lanes.

We consider a centralized control setting, where control decisions are made by a centralized controller that receives the state information from the various lanes and makes decision on which traffic lights to switch green during a cycle. This decision is then relayed back to the individual junctions. We assume no propagation and feedback delays for simplicity. The elapsed time counter for a lane with green signal stays at zero until the time the signal turns red. For a road network with m junctions and a total of K signalled lanes across junctions, the state at time n is the vector

$$\boxed{s_n = (q_1(n), \ldots, q_K(n), t_1(n), \ldots, t_K(n)), \qquad (7.1)}$$

where $q_i(n)$ and $t_i(n)$ are respectively the queue length and elapsed time on lane i at time n. The above state formulation is valid for all the TLC algorithms except the Q-learning based TLC that incorporates state aggregation. The details of the state aggregation and the formulation of the state parameterized by thresholds for this algorithm is given in Section 7.3.1.

The action a_n chosen by the central controller at time n is the sign configuration at each of the m junctions of the road network and has the form: $a_n = (a_1(n), \dots, a_m(n))$, where $a_i(n)$ is the sign configuration at junction i in time slot n. We assume here that there are a total of m junctions in the road network.

The cost function is designed to maximize traffic flow and at the same time, ensure fairness so that no lane suffers from being red for a long duration. This is achieved by letting the cost function be the weighted sum of queue lengths and elapsed times on the lanes of the road network with the weights chosen suitably to incorporate prioritization of traffic. As before, let I_p denote the set of indices of prioritized lanes, i.e., whose traffic is given a higher priority. We let the cost $k(s_n, a_n)$ have the form

$$\begin{aligned} k(s_n, a_n) = \quad & \alpha_1 * (\textstyle\sum_{i \in I_p} \alpha_2 * q_i(n) + \sum_{i \notin I_p} \beta_2 * q_i(n)) \\ + \quad & \beta_1 * (\textstyle\sum_{i \in I_p} \alpha_2 * t_i(n) + \sum_{i \notin I_p} \beta_2 * t_i(n)), \end{aligned} \tag{7.2}$$

where $\alpha_i, \beta_i \geq 0$ and $\alpha_i + \beta_i = 1, i = 1, 2$. Further, $\alpha_2 > \beta_2$. Thus, lanes in I_p are assigned a higher cost and hence a cost optimizing strategy must assign a higher priority to these lanes in order to minimize the overall cost. Note from the way it is defined (cf. (7.2)), $k(s_n, a_n)$ depends explicitly on s_n and not a_n. However, it depends on a_n indirectly as a change in the sign configuration has an impact on the queue lengths and elapsed times on the various lanes.

The sign configuration policy governing the state evolution is based on given queue-length thresholds L_1 and L_2 and elapsed time threshold T_1. We want to find an optimal value for the parameter vector θ that minimizes the long-run average cost (cf. (7.3)), where θ corresponds to the vector $(L_1, L_2, T_1)^T$ for all the TLC algorithms that we propose in Section 7.3.

We let the parameter vector θ take values in a compact set

$$C \stackrel{\triangle}{=} [L_{\min}, L_{\max}] \times [L_{\min}, L_{\max}] \times [T_{\min}, T_{\max}] \subset \mathcal{R}^3$$

by making use of the projection operator π defined later, where $L_{\min}, L_{\max}, T_{\min}, T_{\max}$ are certain prescribed thresholds. The objective is to find a θ that minimizes

$$J(\theta) = \lim_{l \to \infty} \frac{1}{l} \sum_{j=0}^{l-1} k(s_j, a_j). \tag{7.3}$$

The actions a_j are assumed to be governed by one of the policies that we present below, that in turn will be parameterized by the threshold parameter θ. While it is desirable to find a $\theta^* \in C$ that minimizes $J(\theta)$, it is in general difficult to achieve a global minimum. We use therefore a local optimization method for which one needs to evaluate $\nabla J(\theta) \equiv (\nabla_1 J(\theta), \nabla_2 J(\theta), \nabla_3 J(\theta))^T$, for all the algorithms.

We make the following assumption on the function $J(\cdot)$:

Assumption (A1)

The long run average cost $J(\theta)$ is continuously differentiable with bounded second derivative.

Note that (A1) is a technical requirement used to push through a Taylor's argument in the convergence analysis (see Section 7.4). In the case of finite state parameterized Markov chains (as here), one can show using a similar argument as Schweitzer [1968] that the parameterized stationary distribution is continuously differentiable if the parameterized transition probabilities are.

The threshold tuning algorithm that we present in the next section finds the optimum parameter θ using only one simulation trajectory, while the TLC schemes that we present estimate the optimal policy for a given set of thresholds. Hence, in combination with the threshold tuning algorithm, the TLC algorithms that we propose are seen to exhibit significant performance improvements.

7.2 The threshold tuning algorithm

A stochastic iterative algorithm to find the optimal thresholds in the case of a long-run average cost objective would require two nested loops as follows:

1. The inner loop estimates the long-run average cost and also picks actions from the underlying TLC algorithm.

2. The outer loop updates θ along a negative descent direction using an estimate obtained using the outcome of the inner loop procedure.

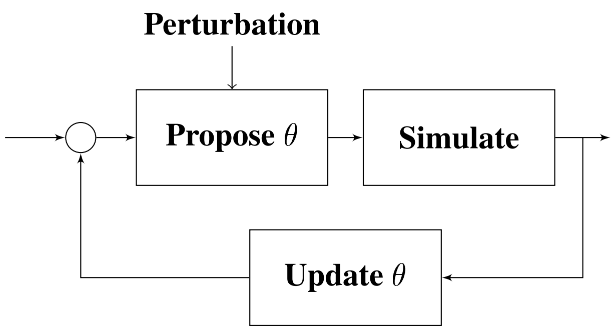


Figure 7.1: Illustration of the operation of the threshold tuning algorithm

The above procedure involving two loops has to be performed iteratively until the parameter θ converges to a local minimum. However, such a procedure would typically be computationally expensive considering the fact that one step of the outer loop, i.e., updating θ in the direction of $-\nabla_\theta J(\theta)$, happens only after the convergence of the corresponding inner loop procedure. Using a multi-timescale stochastic approximation procedure [Borkar, 2008, Chapter 6], we circumvent this problem as both the inner and outer loops can run in tandem. The resulting scheme is shown to converge to the optimal solution for the long-run average cost objective (7.3).

The threshold tuning algorithm estimates the gradient of the objective function $\nabla_\theta J(\theta)$ using a one-sided SPSA based estimate, i.e.,

$$\boxed{\nabla_\theta J(\theta) \approx \left(\frac{J(\theta + \delta\triangle)}{\delta}\right) \triangle^{-1}, \qquad (7.4)}$$

that incorporates a Hadamard matrix based deterministic construction for the perturbations $\triangle$ (see Section 7.2.1). While one-simulation SPSA gradient estimates based on randomized perturbations were proposed in Spall [1997], these are seen to suffer from a large bias and hence do not give good performance. In Bhatnagar et al. [2003], a one-simulation SPSA algorithm that incorporates certain Hadamard matrix based deterministic perturbations was proposed. This algorithm from Bhatnagar et al. [2003] was seen to yield good performance and has a significantly lower bias in comparison to the algorithm in Spall [1997].

Fig 7.1 illustrates the operation of the threshold tuning algorithm. In essence, it is a closed-loop procedure where the system is simulated for a perturbed parameter value $(\theta + \delta\triangle)$ and the average cost estimates obtained via simulation are used to update θ in the negative gradient descent direction using the estimate (7.4).

In what follows, we use $\hat{s}_l, l \geq 0$ to denote the state-valued process governed by the perturbed parameter sequence $\hat{\theta}_l, l \geq 0$. This process and the corresponding sequence of actions $\hat{a}_l, l \geq 0$ help in performing the parameter updates (7.5). The threshold tuning algorithm is as follows:

$$\begin{aligned} L_1(n+1) =& \pi_1\left(L_1(n) - a(n)\left(\frac{\tilde{Z}(nL)}{\delta\triangle_1(n)}\right)\right), \\ L_2(n+1) =& \pi_1\left(L_2(n) - a(n)\left(\frac{\tilde{Z}(nL)}{\delta\triangle_2(n)}\right)\right), \\ T_1(n+1) =& \pi_2\left(T_1(n) - a(n)\left(\frac{\tilde{Z}(nL)}{\delta\triangle_3(n)}\right)\right). \end{aligned} \tag{7.5}$$

In the above,

- $L_1(n), L_2(n), T_1(n)$ denote the n-th updates of the thresholds L_1, L_2 and T_1, respectively.
- $\tilde{Z}(nL)$ represents the cost function averaging term obtained by accumulating the single stage cost over L cycles and is specific to the TLC algorithm being used to obtain the sign configuration policy on the faster timescale. These updates will be explained in the TLC algorithms in the next section.
- $L \geq 1$ is a fixed parameter which controls the rate of update of θ in relation to that of $\tilde{Z}$. This parameter allows for accumulation of updates to $\tilde{Z}$ for L iterations in between two successive θ updates. It is usually observed that allowing L to be greater than 1 improves the algorithm's performance, even though convergence can be proven for any value of L including 1.
- $\delta > 0$ is a given small constant and $\triangle(n) = (\triangle_1(n), \triangle_2(n), \triangle_3(n))^T$ is a vector of ± 1-valued random variables $\triangle_i(n), i = 1, 2, 3$. The $\triangle_i(n)$ themselves correspond to perturbation directions for the three parameter components. For any n, $\triangle(n)$ is obtained from a Hadamard matrix construction described in Section 7.2.1.
- $\pi : \mathbb{R}^3 \rightarrow C$ is the projection operator defined by $\pi(\theta) \stackrel{\triangle}{=} (\pi_1(\theta_1), \pi_1(\theta_2), \pi_2(\theta_3))^T, \theta \in \mathbb{R}^3$. Here for any $x \in \mathbb{R}$, $\pi_1(x) \stackrel{\triangle}{=} \min(\max(L_{\min}, x), L_{\max})$ and $\pi_2(x) \stackrel{\triangle}{=} \min(\max(T_{\min}, x), T_{\max})$, respectively.

Remark 3 *Our threshold tuning algorithm is different from the algorithms of Kosmatopoulos et al. [2007] and Kosmatopoulos and Kouvelas [2009] in many ways. An expected cost objective is considered in the latter and the objective function is parameterized, in the spirit of reinforcement learning (RL), using a linear function approximation architecture, with the parameter being tuned in SPSA-like fashion. Whereas in our case, we consider the long-run average cost objective (7.3). Further, we are able to directly consider the objective function without resorting to parameterized approximations of the cost and using SPSA with deterministic perturbations, we are able to ensure convergence to a locally optimal point of the objective. Note that our tuning algorithm requires only one simulation for estimating the objective function, unlike the algorithms proposed in Kosmatopoulos et al. [2007] as well as Kosmatopoulos and Kouvelas [2009]. In addition, we provide three new TLC algorithms, all based on graded thresholds and combine our threshold tuning algorithm with these.*

The complete algorithm is described as under.

Algorithm 1 The threshold tuning algorithm

Input:

- R, a large positive integer; θ_0, initial parameter vector; $\delta > 0$; $\triangle$;
- UpdateTheta(), the stochastic update rule discussed in (7.5)
- Simulate(θ) $\rightarrow X$: the function that performs one time-step of the road traffic simulation and output the single-stage cost value $k(\hat{s}_n, \cdot)$ (cf. (7.2))
- UpdateAverageCost(): the function that updates the average cost estimate $\tilde{Z}(\cdot)$ used in (7.5) and is specific to the TLC-algorithm.
- UpdateTheta(): the function that updates the threshold parameter θ according to (7.5).

Output: $\theta^* \triangleq \theta_R$.

```
$\theta \leftarrow \theta_0$, $n \leftarrow 1$
begin loop
  $\hat{X} \leftarrow$ Simulate($\theta + \delta\Delta$)
  UpdateAverageCost()
  if $n \% L = 0$ then
    UpdateTheta()
    end if
    $n \leftarrow n + 1$
    if $n = R$ then
        Terminate with $\theta$.
    end if
end loop
```

We make the following assumption on the step-sizes $a(n)$ and $b(n)$:

Assumption (A2)

$$\sum_n a(n) = \sum_n b(n) = \infty; \sum_n (a^2(n) + b^2(n)) < \infty, \text{ and } \lim_{n \to \infty} \frac{a(n)}{b(n)} = 0.$$

The first two requirements in (A2) are standard in stochastic approximation algorithms. In particular, the first requirement ensures that the recursions do not converge prematurely while the second requirement aids in canceling the effect of stochastic noise. As described previously, the third requirement in (A2) essentially gives rise to the desired separation of timescales between the recursion (7.5) that is common to all algorithms and the recursions (7.8), (7.11) and (7.12) of the various TLC algorithms that we describe below. As we do in our experiments for all algorithms, one can select the following step-sizes $a(n)$ and $b(n), n \geq 0$ that satisfy the requirements in (A2):

$$a(0) = \hat{a},\ b(0) = \hat{b},\ a(n) = \hat{a}/n,\ b(n) = \hat{b}/n^{\alpha},\ n \geq 1, \text{ with } 1/2 < \alpha < 1, 0 < \hat{a}, \hat{b} < \infty.$$

7.2.1 The Hadamard matrix based construction for $\{\triangle(n)\}$

An $m \times m$ $(m \geq 2)$ matrix H is called a **Hadamard matrix** of order m if its entries belong to $\{1, -1\}$ and $H^T H = m I_m$, where I_m denotes the $m \times m$ identity matrix. Further, a Hadamard matrix is said to be normalized if all the elements of its first row and column are 1. A simple and systematic way of constructing normalized Hadamard matrices of order $m = 2^k$ is as follows:

For $k = 1$,

$$H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix},$$

and for general $k > 1$,

$$H_{2^k} = \left[\begin{array}{cc} H_{2^{k-1}} & H_{2^{k-1}} \\ H_{2^{k-1}} & -H_{2^{k-1}} \end{array} \right].$$

Let $P = 2^{\lceil log_2(N+1) \rceil}$, where, as mentioned before, N is the parameter dimension. This implies $P \geq N+1$. Now construct a normalized Hadamard matrix H_P of order P using the above procedure. Let $h(1), \dots, h(N)$ be any N columns other than the first column of H_P. The first column is not considered because all elements in the first column are 1, while all the other columns have an equal number of $+1$ and -1 elements. The latter property aids in canceling the bias terms. Form a new matrix $\widetilde{H}_P$ of order $P \times N$ with $h(1), \dots, h(N)$ as its columns. Let $\widetilde{\triangle}(k), k = 1, \dots, P$ denote the rows of $\widetilde{H}_P$. The perturbation sequence $\{\triangle(m)\}$ is now generated by cycling through the rows of $\widetilde{H}_P$, i.e.,

$$\triangle(n) = \widetilde{\triangle}(n \bmod P + 1), \forall n \geq 0.$$

Having described the problem framework and the threshold tuning algorithm, we now present three TLC algorithms - each corresponding to a given class of graded threshold based sign configuration policies with given thresholds. The thresholds L_1, L_2 and T_1 are held fixed and used in various ways (as we explain below) in the three algorithms. When combined with the threshold tuning algorithm (7.5), these algorithms represent the most effective traffic light control strategies in their respective classes. The first two algorithms that we present below are based on Q-learning and incorporate state aggregation and function approximation, respectively, while the last algorithm is a priority based scheme.

7.3 Traffic Light Control algorithms

7.3.1 Q-learning TLC with State Aggregation (QTLC-SA)

Q-learning is an important RL algorithm that has the following incremental update rule, as discussed earlier:

$$\boxed{Q_{n+1}(i,a) = \quad Q_n(i,a) + a(n) \left(k(i,a) + \gamma \min_{b \in \mathcal{A}(j)} Q_n(j,b) - Q_n(i,a) \right), \quad (7.6)}$$

for all feasible tuples (i, a) of states and actions. Here j is the next state after state i that is observed via simulation and follows the distribution $p(i, ., a)$. Also, a (resp. b) is a feasible action in state i (resp. j) and $0 < \gamma < 1$ is a given discount factor. While traditional methods to solve the MDP (see Bertsekas and Tsitsiklis [1996b]) formulated in the previous section would require the specification of $p(i,.,a)$, the Q-learning solution (7.6) works with only simulation based sample estimates of the single stage cost function (7.2) to obtain the optimal sign configuration policy, and explicit knowledge of the transition probabilities $p(i,\cdot,a)$ is not required. Quantities $Q_n(i,a)$ are estimates of the Q-value when the feasible state-action tuple is (i,a), i.e., $a \in \mathcal{A}(i), i \in \mathcal{S}$. $Q_0(i,a)$ can be initialized arbitrarily, a simple choice is to set them all to zero. Note that recursion (7.6) is run for all feasible state-action tuples (i,a).

An MDP framework (Bertsekas and Tsitsiklis [1996b]) requires the identification of states, actions and costs. While the actions and single-stage cost function were identified in Section 7.1, the state formulation in our algorithm below is different and uses state aggregation to tackle the curse of dimensionality. Consider the process $\{s'_n(\theta), n \geq 0\}$, parameterized by $\theta = (L_1, L_2, T_1)^T$, defined as follows:

$$s'_n(\theta) = (q'_1(n), \dots, q'_K(n), t'_1(n), \dots, t'_K(n)), \text{ where} \tag{7.7}$$

$$q'_i(n) = \begin{cases} 0 & \text{if } q_i(n) < L_1 \\ 0.5 & \text{if } L_1 \leq q_i(n) \leq L_2 \\ 1 & \text{if } q_i(n) > L_2 \end{cases} \quad \text{and} \quad t'_i(n) = \begin{cases} 0 & \text{if } t_i(n) \leq T_1 \\ 1 & \text{if } t_i(n) > T_1. \end{cases}$$

Note that, as before, $q_i(n)$ and $t_i(n)$ denote the waiting queue lengths and elapsed times corresponding to the lane i at time n. Thus, $q'_i(n)$ and $t'_i(n)$ serve as proxies for $q_i(n)$ and $t_i(n), i = 1, \dots, K$ respectively. By only allowing $q'_i(n)$ to take three possible values and $t'_i(n)$ to take two values depending on $q_i(n), t_i(n)$ and θ, we cluster together states s_n into $s'_n(\theta), n \geq 0$ in order to make computation more manageable. The problem of the curse of dimensionality associated with large state spaces is the primary reason why regular Q-learning with full state representations (cf. recursion (7.6)) is not implementable in the case of large networks, see Prashanth and Bhatnagar [2011a]. The Q-learning algorithm (7.6) when applied to our setting with state aggregation will be referred to as the QTLC-SA algorithm.

QTLC-SA with threshold tuning (QTLC-SA-TT)

The QTLC-SA-TT algorithm is a two timescale stochastic approximation algorithm that updates θ on the slower timescale and learns the optimal sign configuration policy (parameterized by θ) on the faster timescale. The update of the threshold parameters is done using the threshold tuning algorithm (7.5), whereas the cost function averaging (required for the threshold tuning algorithm) and update of the Q-learning algorithm are performed as follows: For $m = nL, \dots, (n+1)L - 1$,

$$\tilde{Z}(m+1) = \tilde{Z}(m) + b(n)(k(\hat{s}_m(\theta), \tilde{a}_m) - \tilde{Z}(m)), \tag{7.8a}$$

$$Q_{m+1}(\hat{s}(\theta), \hat{a}) = Q_m(\hat{s}(\theta), \hat{a}) + b(n)(k(\hat{s}, \hat{a}) + \gamma \min_{b \in \mathcal{A}(j(\theta))} Q_m(j(\theta), b) - Q_m(\hat{s}(\theta), \hat{a})) \tag{7.8b}$$

- Here, $\{\hat{s}_l(\theta)\}$ denotes the aggregated state-valued process that is governed by $\{\hat{\theta}_l\}$, where $\hat{\theta}_l = \theta_n + \delta\triangle(n)$ for $n = \left[\frac{l}{L}\right]$, with a given $L \geq 1$.
- Whereas the Q-function update (cf. (7.8b)) is performed for all feasible state action tuples (where states are now aggregated states), the action $\tilde{a}_m$ in the argument of the single-stage cost $k(\hat{s}_m(\theta), \cdot)$ in (7.8a) is chosen as per the policy suggested by the Q-value update.
- The perturbations $\triangle(n)$ are generated using Hadamard matrices as outlined in Section 7.2.1.

We make the following assumption on the underlying process, when the actions are obtained from the QTLC-SA algorithm:

Assumption (A3)

(A3a) The basic underlying process $\{\hat{s}_n(\theta), n \geq 1\}$ is an MDP that is parameterized by θ

(A3b) The Markov chain $\{\hat{s}_n(\theta), n \geq 1\}$ under a given parameter $\theta \in C$ and any fixed stationary sign configuration policy is ergodic.

Assumption (A3) ensures in particular that the long-run average cost in (7.3) is well-defined for any given $\theta \in C$ under any stationary sign configuration policy.

7.3.2 Q-learning with Function Approximation with a Novel Feature Selection scheme (QTLC-FA-NFS)

It turns out that with QTLC-SA, the curse of dimensionality using state aggregation cannot be fully controlled for large networks with high-dimensional states, as the cardinality of the aggregated state space increases significantly with the dimension of the states. To alleviate the curse of dimensionality problem, the use of function approximation has been advocated in Prashanth and Bhatnagar [2011a] and the QTLC-FA algorithm developed therein is a computationally efficient TLC algorithm that is based on Q-learning. However, the state-action features used in Prashanth and Bhatnagar [2011a] do not incorporate a clear dependence between states and actions. We further improve the performance of the QTLC-FA algorithm proposed in Prashanth and Bhatnagar [2011a] by incorporating a novel feature selection scheme that incorporates dependence between states and actions and assigns priorities to various state-action combinations in the features.

In a function approximation setting, corresponding to every feasible state-action tuple, a feature vector has to be specified. The inputs for the feature selection scheme would be the broad estimates of congestion levels and elapsed times on the lanes of the road network. In Prashanth and Bhatnagar [2011a], the feature vector contained a bit each for the congestion estimate, elapsed time estimate and the sign configuration portion, respectively, for each lane of the road network. While each of these attributes are important, the approximation architecture used in Prashanth and Bhatnagar [2011a] did not take into account the dependence between features. In our novel feature selection scheme that we propose below, we incorporate dependence between the state and the action features while using graded thresholds. Another advantage of our methodology is that the dimension of the feature vector is now reduced by more than half as compared to the scheme in Prashanth and Bhatnagar [2011a]. Numerical experiments show that this new choice of features significantly outperforms the one used in Prashanth and Bhatnagar [2011a]. The various aspects of the Q-learning TLC algorithm that incorporates function approximation with the new feature selection scheme, are now described.

In the algorithm below, a state-action feature denoted by $\sigma_{i,a}$ is associated with each feasible state-action tuple (i, a). The Q-function is then approximated as

$$Q(i,a) \approx \omega^T \sigma_{i,a}. \tag{7.9}$$

Let s_n and s_{n+1} denote the states at instants n and $n+1$, respectively, and ω_n be the estimate of the parameter ω at instant n. The Q-learning algorithm with function approximation has the following update rule:

$$\boxed{\omega_{n+1} = \omega_n + b(n)\sigma_{s_n,a_n}\left(k(s_n,a_n) + \gamma \min_{v \in A(s_{n+1})} \omega_n^T \sigma_{s_{n+1},v} - \omega_n^T \sigma_{s_n,a_n}\right), \quad (7.10)}$$

where ω_0 is set arbitrarily. In (7.10), the action a_n is chosen in state s_n according to $a_n = arg\min_{v \in A(s_n)} \omega_n^T \sigma_{s_n,v}$.

The features are chosen as follows: $\sigma_{s_n,a_n} = (\sigma_1(n), \ldots, \sigma_K(n))^T$, where the scheme for selection of the feature value $\sigma_i(n)$ corresponding to lane i when action a is chosen, is explained in Table. 7.1.

The feature selection scheme is graded and assigns a value for each lane based on whether the queue length on the lane is below L_1, is between L_1 and L_2, or is above L_2, on whether the elapsed time is below T_1 or above it and also on whether the sign configuration indicates a RED or a GREEN light for the lane. For instance, if both queue length and

Table 7.1: Feature selection ($\sigma_i(n)$) scheme for lane i

State	Action	Feature
$q_i(n) < L_1$ and $t_i(n) < T_1$	RED	0
	GREEN	1
$q_i(n) < L_1$ and $t_i(n) \geq T_1$	RED	0.2
	GREEN	0.8
$L_1 \leq q_i(n) < L_2$ and $t_i(n) < T_1$	RED	0.4
	GREEN	0.6
$L_1 \leq q_i(n) < L_2$ and $t_i(n) \geq T_1$	RED	0.6
	GREEN	0.4
$q_i(n) \geq L_2$ and $t_i(n) < T_1$	RED	0.8
	GREEN	0.2
$q_i(n) \geq L_2$ and $t_i(n) \geq T_1$	RED	1
	GREEN	0

elapsed time are above the "highest" threshold level for the lane, then an action of GREEN would result in a feature value of 0 and an action of RED would result in the value 1. In essence, this choice indicates that the TLC algorithm should attempt to switch this lane to green because regardless of the value of the weight vector, the Q-value in such a case would be 0. By a similar argument, if both the queue length and the elapsed time are below the "lowest" threshold level for the lane, then the feature value chosen is just the opposite, i.e., 0 for RED and 1 for GREEN, implying that it is better to keep this lane red. The feature values corresponding to other decision choices are again suitably graded. It is clear that the assignment of feature values here takes into account the dependence between states and actions, which is unlike Prashanth and Bhatnagar [2011a] where such dependence was not incorporated.

While the cardinality of the state-action space may be high so that storing/updating Q-values for each (s, a)-tuple may be impossible, the above feature-based algorithm estimates Q-values using the parameterization (7.9), making its implementation feasible even on large road networks as the parameter ω_n has the same dimension as that of σ_{s_n,a_n}. The Q-learning with function approximation algorithm with the update rule (7.9) and our novel feature selection scheme described above, will be referred to as the QTLC-FA-NFS algorithm.

QTLC-FA-NFS with threshold tuning (QTLC-FA-NFS-TT)

As with QTLC-SA-TT, the combination of QTLC-FA-NFS with the threshold tuning algorithm (7.5) gives the QTLC-FA-NFS-TT algorithm. The recursions on the faster timescale in the case of QTLC-FA-NFS-TT algorithm are as follows: Let $\{\tilde{s}_n, n \geq 0\}$ denote a state-valued process that depends on both the tunable policy as well as the tunable parameter $\tilde{\theta}_l, l \geq 0$, where $\tilde{\theta}_l = \theta_n + \delta\triangle(n)$ for $n = \left[\frac{l}{L}\right]$, and updates of $\theta_n \equiv (L_1(n), L_2(n), T_1(n))^T$ are governed according to (7.5). For $m = nL, \ldots, (n+1)L - 1$,

$$\tilde{Z}(m+1) = \tilde{Z}(m) + b(n)\left(k(\tilde{s}_m, \hat{a}_m) - \tilde{Z}(m)\right), \quad (7.11a)$$

$$\omega_{m+1} = \omega_m + b(n)\sigma_{\tilde{s}_m,\hat{a}_m}\left(k(\tilde{s}_m, \hat{a}_m) + \gamma \min_{v \in \mathcal{A}(\tilde{s}_{m+1})} \omega_m^T \sigma_{\tilde{s}_{m+1},v} - \omega_m^T \sigma_{\tilde{s}_m,\hat{a}_m}\right). \quad (7.11b)$$

The action $\hat{a}_m$ in (7.11a)-(7.11b) is chosen to be the one that minimizes $\omega_m^T \sigma_{\tilde{s}_m,v}$ over all $v \in \mathcal{A}(\tilde{s}_m)$.

We make the following assumption here:

Assumption (A4)

(A4a) For any given $\theta \in C$, the basic underlying process $\{s_n, n \geq 1\}$ is an MDP.

(A4b) For any given policy and parameter $\theta \in C$, the process $\{s_n, n \geq 1\}$ is ergodic Markov.

7.3.3 Priority based TLC (PTLC)

The sign configuration policy here is a graded threshold-based policy that assigns different priorities to different policy levels. The thresholds here are on the queue lengths (L_1 and L_2) and elapsed times since the last switch over of lights to red (T_1) on individual lanes. The cost assigned to each lane is decided based on whether the queue length on that lane is below L_1, is between L_1 and L_2, or is above L_2 at any instant and also on whether the elapsed time is below T_1 or above it. For instance, if both queue length and elapsed time are above the "highest" threshold levels (L_2 and T_1 respectively) on a given lane, then the policy assigns the highest priority value (of 6) to that lane. The priority assignment for any lane i of the road network based on the queue length q_i and elapsed time t_i is shown in Table. 7.2. The policy then selects the sign configuration with the maximum (over all feasible sign configurations) sum of lane priority values. In essence, the TLC algorithm flushes the traffic on lanes with long waiting queues, while also giving higher priority

Table 7.2: Priority assignment for each lane in the TLC policy

Condition	**Priority value**
$q_i < L_1$ and $t_i < T_1$	1
$q_i < L_1$ and $t_i \geq T_1$	2
$L_1 \leq q_i(n) < L_2$ and $t_i(n) < T_1$	3
$L_1 \leq q_i(n) < L_2$ and $t_i \geq T_1$	4
$q_i \geq L_2$ and $t_i < T_1$	5
$q_i \geq L_2$ and $t_i \geq T_1$	6

to lanes that have been waiting on a red signal for a long time. This helps to combine efficiency with fairness.

PTLC with threshold tuning (PTLC-TT)

As with the previous TLC algorithms, we combine the threshold tuning algorithm (7.5) with PTLC to obtain the PTLC-TT algorithm. The state-valued process $\{\hat{s}_n, n \geq 0\}$ in this case under the priority based policy described above depends on the tunable parameter sequence $\hat{\theta}_l = \theta_n + \delta\triangle(n), n \geq 0$, where $\theta_n \equiv (L_1(n), L_2(n), T_1(n))^T, n \geq 0$ are updated according to (7.5). The faster timescale recursions here are given as follows: For $m = nL, \ldots, (n+1)L-1$,

$$\tilde{Z}(m+1) = \tilde{Z}(m) + b(n)(k(\hat{s}_m, \hat{a}_m) - \tilde{Z}(m)). \tag{7.12}$$

The action $\hat{a}_m$ above is selected in state $\hat{s}_m$ based on the priority assignment policy (described above), i.e., select the sign configuration that has the maximum sum of priority values (where the maximum is over all feasible sign configurations) and switch the lanes in the chosen sign configuration to green. We now make the following assumption:

Assumption (A5)

(A5a) The basic underlying process $\{\hat{s}_n, n \geq 0\}$ is a parameterized MDP.

(A5b) For any given $\theta \in C$ and the specified priority-based policy, $\{\hat{s}_n, n \geq 0\}$ is ergodic Markov.

7.4 Convergence Analysis

We show here the convergence of the threshold tuning algorithm, viz., recursions (7.5), when the updates of $\tilde{Z}$ are governed according to (7.12) in the PTLC-TT scheme. A similar analysis works in the case when these are governed as per the other schemes – (7.8) and (7.11), respectively. In particular, the actions in the single-stage cost expressions (7.8a) and (7.11a) are obtained using the policies suggested by the corresponding schemes in (7.8b) and (7.11b), respectively. The analysis that follows can be used in the same way to give optimal L_1 and L_2 values for the corresponding class of policies for each algorithm. The analysis of the Q-learning algorithm with state aggregation as per (7.7) can be conducted in the same manner as regular Q-learning with full state representations (Watkins and Dayan [1992a], Tsitsiklis [1994a], Borkar and Meyn [2000]). On the other hand, the analysis of the Q-learning update with function approximation (7.11b) cannot be shown because of the off-policy problem associated with Q-learning with function approximation. Nevertheless, QTLC-FA-NFS-TT is seen to empirically show the best results and suggests that the tuning update algorithm that is used to have the L_1 and L_2 thresholds improves the performance of Q-learning with function approximation as well. Note that because the action selection in the case of the PTLC algorithm is via the policy as prescribed in Table 7.2, one can write $\hat{a}_n = f(\hat{s}_n)$, $n \geq 0$, where the function f is a deterministic function that specifies the aforementioned policy. For simplicity, we shall consider here the case of $L = 1$, i.e., of no additional averaging on top of the two-timescale averaging. The case of general (finite) L can also be handled, see for instance, Bhatnagar et al. [2003].

Let $\bar{\pi}(x) = (\bar{\pi}_1(x_1), \bar{\pi}_1(x_2), \bar{\pi}_2(x_3))^T$ for any $x = (x_1, x_2, x_3)^T \in \mathbb{R}^3$. Define now the operators $\bar{\pi}_1$ and $\bar{\pi}_2$ as follows:

$$\bar{\pi}_1(v(x)) = \lim_{\eta \to 0} \left(\frac{\pi_1(x + \eta v(x)) - x}{\eta} \right), \bar{\pi}_2(w(x)) = \lim_{\eta \to 0} \left(\frac{\pi_2(x + \eta w(x)) - x}{\eta} \right), \tag{7.13}$$

for any continuous functions $v : [L_{\min}, L_{\max}] \to \mathcal{R}$ and $w : [T_{\min}, T_{\max}] \to \mathcal{R}$, respectively. The limits in (7.13) exist and are unique since $[L_{\min}, L_{\max}]$ and $[T_{\min}, T_{\max}]$ are convex sets. From definition, $\bar{\pi}_1(v(x)) = v(x)$ (resp. $\bar{\pi}_2(w(y)) = w(y)$) if $x \in (L_{\min}, L_{\max})$ (resp. $y \in (T_{\min}, T_{\max})$.

Consider the following ODE:

$$\dot{\theta}(t) = \bar{\pi}(-\nabla J(\theta(t))). \tag{7.14}$$

Let $R \subset I = \{\theta \in C \mid \nabla J(\theta) = 0\}$ denote the set of stable fixed points of (7.14). Note that I denotes the set of all fixed points of (7.14) that would include not just stable equilibria but also unstable ones such as local maxima, saddle points etc. The set of stable equilibria (i.e., the local minima) is in general a subset of I. For $\eta > 0$, let $R^\eta \triangleq \{\theta \in C| \parallel \theta - \theta_0 \parallel <$

η for some $\theta_0 \in R\}$, denote the set of points that are within an η-neighborhood of R. Our main results if the following and its proof will be shown later.

Theorem 7.1 *Given $\eta > 0$, there exists $\delta_0 > 0$ such that for all $\delta \in (0, \delta_0)$, the recursions $\theta(n)$ converge almost surely to R^η.*

We will assume for simplicity that the elapsed times on any lane remain bounded even though the bound can be large. Moreover, the queue lengths on any lane are bounded as well because each lane (between two junctions) can at most accommodate a bounded number of vehicles. The single-stage cost function can be seen to be a linear function of the state and remains bounded in this case. If the elapsed times are allowed to become unbounded, then one will require additional assumptions such as the existence of a stochastic Lyapunov function in order to ensure that the second moment of the system state remains uniformly bounded almost surely.

The PTLC-TT algorithm (in the case of $L = 1$) can be rewritten as follows:

$$L_i(n+1) = \pi_1\left(L_i(n) - a(n)\left(\frac{\tilde{Z}(n)}{\delta\triangle_i(n)}\right)\right), \quad i = 1, 2 \tag{7.15}$$

$$T_1(n+1) = \pi_3\left(T_1(n) - a(n)\left(\frac{\tilde{Z}(n)}{\delta\triangle_3(n)}\right)\right), \tag{7.16}$$

$$\tilde{Z}(n+1) = \tilde{Z}(n) + b(n)(k(\hat{s}_n, f(\hat{s}_n)) - \tilde{Z}(n)). \tag{7.17}$$

Lemma 7.2 *Each of the recursions (7.15)-(7.17) is uniformly bounded with probability one.*

Proof: Recursions (7.15)-(7.16) stay uniformly bounded as a consequence of the projection operators π_1 and π_2 respectively. As described previously, the single-stage cost function $k(\cdot, \cdot)$ remains uniformly bounded almost surely. Now note that since $b(n) \to 0$ as $n \to \infty$, there exists an integer $N_0 > 0$ such that for all $n \geq N_0$, $b(n) < 1$. Thus, for $n \geq N_0$, $\tilde{Z}(n+1)$ is a convex combination of $\tilde{Z}(n)$ and $k(\hat{s}_n, f(\hat{s}_n))$ (a uniformly bounded quantity). Suppose $\sup_n |k(\hat{s}_n, f(\hat{s}_n))| \leq \hat{K}$, almost surely for some $\hat{K} > 0$. Thus,

$$\sup_n |\tilde{Z}(n)| < \infty,$$

almost surely. The claim follows. ■

We analyze first the convergence of the faster timescale recursion (7.17). Define two sequences of time points $\{s(n)\}$ and $\{t(n)\}$ according to $s(0) = t(0) = 0$ and for $n \geq 1$, $s(n) = \sum_{m=0}^{n} a(m)$, $t(n) = \sum_{m=0}^{n} b(m)$. Let $\Delta(t), t \geq 0$ be defined by $\Delta(t) = \Delta(n), t \in [s(n), s(n+1)], n \geq 0$.

Let $\mathcal{F}_n = \sigma(\hat{s}_m, \theta(m), m \leq n), n \geq 0$ be a sequence of associated sigma fields. Consider the sequence $N_n, n \geq 0$, defined according to $N_0 = 0$ and for $n \geq 1$,

$$N_n = \sum_{m=0}^{n-1} b(m)\left(k(\hat{s}_m, f(\hat{s}_m)) - E\left[k(\hat{s}_m, f(\hat{s}_m)) \mid \mathcal{F}_{m-1}\right]\right) \triangleq \sum_{m=0}^{n-1} b(m) M_m.$$

Lemma 7.3 $(N_n, \mathcal{F}_n), n \geq 0$ *forms an almost surely convergent martingale sequence.*

Proof: It is easy to see that $(N_n, \mathcal{F}_n), n \geq 0$ forms a martingale sequence. Now consider the quadratic variation process associated with this martingale. Note that

$$\sum_{n=0}^{\infty} E[(N_{n+1} - N_n)^2 \mid \mathcal{F}_n] = \sum_{n=0}^{\infty} b(n)^2 M_m^2 < \infty,$$

almost surely, since M_m^2 remains uniformly bounded because the single-stage cost function $k(\cdot,\cdot)$ is uniformly bounded and moreover, $\sum_n b(n)^2 < \infty$. The claim follows from the martingale convergence theorem (cf. Theorem 3.3.4 of Borkar [1995]). ■

Consider now the following set of ODEs associated with (7.15)-(7.17).

$$\dot{L}_1(t) = 0, \quad \dot{L}_2(t) = 0, \quad \dot{T}_1(t) = 0, \tag{7.18}$$

$$\dot{\tilde{Z}}(t) = J(\theta(t) + \delta\Delta(t)) - \tilde{Z}(t), \tag{7.19}$$

respectively, where $\theta(t), t \geq 0$ is defined according to $\theta(t(n)) = \theta(n), n \geq 0$ with continuous linear interpolation on $[t(n), t(n+1)]$. Also, $\Delta(t) = \Delta(n)$ for $t \in [t(n), t(n+1)], n \geq 0$.

Before we proceed further, we recall a key result from Hirsch [1989]. Consider an ODE

$$\dot{x}(t) = f(x(t)), \tag{7.20}$$

where $f : \mathcal{R}^N \to \mathcal{R}^N$ (for some $N \geq 1$) is a Lipschitz continuous function. Given T, $\gamma > 0$, we call a bounded, measurable $y(\cdot) : \mathbb{R}^+ \cup \{0\} \to \mathbb{R}^N$, a (T, γ)-perturbation of

(7.20) if there exist $0 = T_0 < T_1 < T_2 < \cdots < T_r \uparrow \infty$ with $T_{r+1} - T_r \geq T \; \forall r$ and solutions $\theta^r(t), t \in [T_r, T_{r+1}]$ of (7.20) for $r \geq 0$, such that

$$\sup_{t \in [T_r, T_{r+1}]} \parallel \theta^r(t) - y(t) \parallel < \Delta.$$

We recall the following result given as Theorem 1, pp.339 of Hirsch [1989].

Let H denote the set of globally asymptotically stable attractors of the ODE (7.20) and given $\epsilon > 0$, let H^ϵ denote the set of points that are within an ϵ-neighborhood from the set H. In particular, $H \subset H^\epsilon$.

Lemma 7.4 (Hirsch Lemma) *Given ϵ, $T > 0$, $\exists \bar{\gamma} > 0$ such that for all $\gamma \in (0, \bar{\gamma})$, every (T, γ)-perturbation of (7.20) converges to H^ϵ.*

We now return to the analysis of the faster timescale recursion. Given $\hat{T} > 0$, let $\hat{T}_n, n \geq 0$ be defined according to $\hat{T}_0 = 0$ and $\hat{T}_n = \min\{t(n) \mid t(n) \geq \hat{T}_{n-1} + \hat{T}\},\; n \geq 1$. Then $\hat{T}_{n+1} - \hat{T}_n \approx \hat{T}, \forall n \geq 0$ and there exists a subsequence $\{l(n)\}$ of $\{n\}$ such that $\hat{T}_n = t(l(n)) \forall n$. Define $\bar{L}_1(t), \bar{L}_2(t), \bar{T}_1(t), \bar{Z}(t), t \in [t(n), t(n+1)], n \geq 0$ in the following manner: $\bar{L}_1(t(n)) = L_1(n)$, $\bar{L}_2(t(n)) = L_2(n)$, $\bar{T}_1(t(n)) = T_1(n)$, $\bar{Z}(t(n)) = \tilde{Z}(n)$, respectively, with suitable continuous linear interpolations in between intervals $[t(n), t(n+1)]$. Given $T, \epsilon > 0$, we say that $\bar{x}(\cdot)$ is a (T, ϵ)-perturbation of the ODE $\dot{x}(t) = f(x(t))$, if there exist $T_n, n \geq 0$ such that $T_{n+1} - T_n \geq T \; \forall n$ and $\sup_{t \in [T_n, T_{n+1}]} \parallel \bar{x}(t) - x(t) \parallel < \epsilon, \forall n$.

Proposition 7.5 *The functions $\bar{L}_1(t), \bar{L}_2(t), \bar{T}_1(t), \bar{Z}(t), t \geq 0$ form $(\hat{T}, \gamma)$-perturbations of the ODEs (7.18)-(7.19).*

Proof: Note that along the timescale $t(n), n \geq 0$, i.e., using the sequence of step sizes $b(n), n \geq 0$, one can rewrite (7.15)-(7.16) as follows:

$$L_i(n+1) = \pi_1 \left(L_i(n) - b(n)\chi_i(n) \right), \quad i = 1, 2, \tag{7.21}$$

$$T_1(n+1) = \pi_3 \left(T_1(n) - b(n)\chi_3(n) \right), \tag{7.22}$$

where $\chi_i(n) = \dfrac{a(n)}{b(n)} \left(\dfrac{\tilde{Z}(n)}{\delta \triangle_i(n)} \right), i = 1, 2, 3$, respectively. Now, from Assumption (A2), we have that $\dfrac{a(n)}{b(n)} \to 0$ as $n \to \infty$. Hence, $\chi_j(n) = o(1), j = 1, 2, 3$. Finally, consider the recursion (7.17). Note that by Lemma 7.3, we have that $\displaystyle\sum_{j=l(n)}^{l(n+1)-1} b(j) M_j \to 0$ as $n \to \infty$.

One can now rewrite (7.17) as

$$\tilde{Z}(n+1) = \tilde{Z}(n) + b(n)(J(\theta(n) + \delta\Delta(n)) + \chi_4(n)) + (N_{n+1} - N_n),$$

where $\chi_4(n) = E\left[k(\hat{s}_m, f(\hat{s}_m)) \mid \mathcal{F}_{m-1}\right] - J(\theta(n) + \delta\Delta(n))$. Note that $\chi_4(n), n \geq 0$ constitutes the 'Markov' noise in the update. From Assumption (A5), $\chi_4(n) \to 0$ on the 'natural timescale' that is clearly faster than the timescale of the algorithm, see Chapter 6.2 of Borkar [2008] for a detailed treatment of natural timescale recursions involving Markov noise. The claim follows. ■

Lemma 7.6 *With probability one,* $|\tilde{Z}(n) - J(\theta(n) + \delta\Delta(n)| \to 0$ *as* $n \to \infty$.

Proof: Follows by applying Lemma 7.4 (Hirsch [1989]) for every $\epsilon > 0$. ■

We now consider the slower timescale recursions (7.15)-(7.16). Recall that the parameter dimension in our case is $N = 3$. Thus, $P = 2^{\lceil log_2(N+1) \rceil} = 4$. Let $\Delta(n) = (\Delta_1(n), \Delta_2(n), \Delta_3(n))^T$, $n \geq 0$ be the perturbations obtained using the Hadamard matrix based procedure.

Lemma 7.7 *The vectors* $\Delta(n)$, $n \geq 0$ *satisfy the following properties:*

1. *For any* $s \geq 0$ *and all* $k \in \{1, \ldots, 3\}$, $\displaystyle\sum_{n=s+1}^{s+P} \frac{1}{\Delta_k(n)} = 0.$

2. *For any* $s \geq 0$ *and all* $i, j \in \{1, \ldots, 3\}$, $i \neq j$, $\displaystyle\sum_{n=s+1}^{s+P} \frac{\Delta_i(n)}{\Delta_j(n)} = 0.$

Proof: The claim is obvious from the construction. ■

Let $\theta(n) = (L_1(n), L_2(n), T_1(n))^T$. For any $x = (x_1, x_2, x_3)^T \in \mathcal{R}^3$, let $\pi(x) \triangleq (\pi_1(x_1), \pi_1(x_2), \pi_2(x_3))^T$. In view of Lemma 7.6, one can consider the following in place of (7.15)-(7.16):

$$\theta(n+1) = \pi\left(\theta(n) - a(n)J(\theta(n) + \delta\Delta(n))(\Delta(n))^{-1}\right). \tag{7.23}$$

Lemma 7.8 *Given any fixed integer* $K > 0$, *for all* $r \in \{1, \ldots, K\}$

(i) $\lim_{m\to\infty} \parallel \theta(m+r) - \theta(m) \parallel = 0$, *w.p.* 1, *and*

(ii) $\lim_{m\to\infty} \parallel \nabla J(\theta(m+r)) - \nabla J(\theta(m)) \parallel = 0$, *w.p.* 1.

Proof: (i) The proof follows in a similar manner as Lemma 2.2 of Bhatnagar et al. [2003].

(ii) Follows from (i) and Assumption (A1). ■

The proof of the following result has not been shown in Bhatnagar et al. [2003] for Hadamard matrix perturbations. Hence, we provide it below.

Lemma 7.9 *The following hold for any $k, l \in \{1, \dots, 3\}$, $k \neq l$:*

$$\| \sum_{n=m}^{m+P-1} \frac{a(n)}{a(m)} \frac{\triangle_k(n)}{\triangle_l(n)} \nabla_k J(\theta(n)) \|, \| \sum_{n=m}^{m+P-1} \frac{a(n)}{a(m)} \frac{1}{\triangle_l(n)} J(\theta(n) + \delta\Delta(n)) \| \to 0 \ \ as \ \ m \to \infty. \tag{7.24}$$

Proof: We first show that (7.24) holds. By letting $K = P$ in Lemma 7.8, it follows that $a(j)/a(m) \to 1$ as $m \to \infty$ for any $j \in \{m, \dots, m+P-1\}$. Note also that P is an even integer. As a consequence of Lemma 7.7, one can split any set $A_m \triangleq \{m, m+1, \dots, m+P-1\}$ into two disjoint subsets $A^+_{m,k,l}$ and $A^-_{m,k,l}$ each having the same number of elements, with $A^+_{m,k,l} \cup A^-_{m,k,l} = A_m$ and such that $\dfrac{\triangle_k(n)}{\triangle_l(n)}$ takes value $+1$ $\forall n \in A^+_{m,k,l}$ and -1 $\forall n \in A^-_{m,k,l}$, respectively. Thus,

$$\| \sum_{n=m}^{m+P-1} \frac{a(n)}{a(m)} \frac{\triangle_k(n)}{\triangle_l(n)} \nabla_k J(\theta(n)) \| = \| \sum_{n \in A^+_{m,k,l}} \frac{a(n)}{a(m)} \nabla_k J(\theta(n)) - \sum_{n \in A^-_{m,k,l}} \frac{a(n)}{a(m)} \nabla_k J(\theta(n)) \| .$$

The first claim in (7.24) now easily follows and the second claim follows from Lemma 7.8 and (A1). ■

Proof of Theorem 7.1. The recursion (7.23) can be rewritten as follows: For $i = 1, 2, 3$,

$$\theta_i(n+1) = \gamma_i \left(\theta_i(n) - a(n) \frac{J(\theta(n) + \delta\Delta(n))}{\Delta_i(n)} \right), \tag{7.25}$$

where $\gamma_1 = \gamma_2 = \pi_1$ and $\gamma_3 = \pi_2$, respectively. The recursion (7.25) can be rewritten as follows:

$$\theta_i(n+1) = \theta_i(n) - a(n) \frac{J(\theta(n) + \delta\Delta(n))}{\Delta_i(n)} - a(n) Z_i(n), \tag{7.26}$$

where $Z_i(n)$ is an error term that results from the projection. By a Taylor series expansion of $J(\theta(m) + \delta\triangle(m))$ around the point $\theta(m)$, one obtains: For $i = 1, 2, 3$,

$$\theta_i(m+2^P) = \theta_i(m) - \sum_{l=m}^{m+2^P-1} a(l) \nabla_i J(\theta(l)) - a(m) \sum_{l=m}^{m+2^P-1} \sum_{j=1, j\neq i}^{3} \frac{a(l)}{a(m)} \frac{\triangle_j(l)}{\triangle_i(l)} \nabla_j J(\theta(l))$$

$$- a(m) \sum_{l=m}^{m+2^P-1} \frac{a(l)}{a(m)} \frac{1}{\triangle_i(l)} J(\theta(l)) + a(m)O(\delta) - \sum_{j=m}^{m+2^P-1} a(j) Z_i(j). \tag{7.27}$$

The third and the fourth terms on the RHS of (7.27) vanish asymptotically as a consequence of Lemma 7.9. The rest now follows from a result on the convergence of projected stochastic approximation, see Theorem 5.3.1 on pp.191-196 of Kushner and Clark [1978b]. ■

7.5 Simulation Experiments

We now provide numerical results to illustrate the performance of the various threshold tuning TLC algorithms. For the purpose of traffic simulation and performance comparison of the various graded threshold based TLC algorithms, we used the Green Light District (GLD) traffic simulation software Wiering et al. [2004]. The TLC algorithms were compared using the performance metrics of average junction waiting time and the total number of road users who reached their destination, as functions of the number of cycles. We also implemented the QTLC-FA algorithm proposed in Prashanth and Bhatnagar [2011a] (see Chapter 5) and its tuning variant QTLC-FA-TT for the sake of comparisons. The QTLC-FA-TT algorithm combines the QTLC-FA algorithm with threshold tuning in a similar manner as the QTLC-FA-NFS-TT algorithm. The update rule (7.5) is again followed to tune the thresholds. However, the sign configuration policy used in this algorithm is from the QTLC-FA algorithm described in Section 5.2.

7.5.1 Implementation

We implemented the threshold tuning algorithm for all four algorithms that incorporate graded thresholds - PTLC, QTLC-SA, QTLC-FA and QTLC-FA-NFS, respectively. We compared the performance of the tuned variants of the TLC algorithms against their counterparts that involved fixed thresholds (no tuning). The algorithms studied are outlined below:

- **PTLC**: This is a graded threshold based TLC algorithm described in Section 7.3.3 and uses a priority assignment scheme based on waiting queue lengths and elapsed times in order to arrive at a sign configuration. Note that this algorithm uses fixed thresholds, and the performance of this algorithm is studied against its tuning variant PTLC-TT (below).

- **PTLC-TT**: This algorithm combines threshold tuning with the priority based TLC mentioned above. In other words, the sign configuration policy here is the same as PTLC, except that the thresholds used ($\theta = (L_1, L_2, T_1)^T$ are tuned using the online incremental algorithm described in Section 7.3.3.

- **QTLC-SA**: This is the Q-learning based TLC algorithm that incorporates state aggregation and is described in Section 7.3.1. Note that this algorithm uses fixed thresholds and the performance of this algorithm is compared against its tuning variant (QTLC-SA-TT) below.

- **QTLC-SA-TT**: This algorithm combines threshold tuning with the QTLC-SA TLC mentioned above. As in PTLC-TT, the parameter of thresholds $\theta = (L_1, L_2, T_1)^T$ used in the QTLC-SA TLC algorithm are tuned using the online incremental algorithm described in Section 7.3.1.

- **QTLC-FA**: This is the TLC algorithm presented in Prashanth and Bhatnagar [2011a] (see Chapter 5) and uses Q-learning with function approximation.

- **QTLC-FA-TT**: This is the tuning variant of the QTLC-FA algorithm mentioned above and incorporates the update (7.5) to tune the threshold parameter θ.

- **QTLC-FA-NFS**: This is the Q-learning based TLC algorithm with function approximation that incorporates the novel feature selection strategy described in Table 7.1. This algorithm is described in detail in Section 7.3.2. Note that this algorithm uses fixed thresholds and the performance of this algorithm is compared against its tuning variant (QTLC-FA-NFS-TT) below.

- **QTLC-FA-NFS-TT**: This algorithm is the tuning counterpart of QTLC-FA-NFS and tunes the threshold parameter θ using the recursions (7.5).

We performed experiments with the aforementioned TLC algorithms on the road networks shown in Fig. 7.2. In essence, we consider four different settings - a single junction road network, a four-junction corridor, a 4x4-grid network as well as a network of 9 signalized-junctions with 24 roads around the Indian Institute of Science campus in Bangalore (hereafter referred to as the IISc network), respectively for testing the TLC algorithms. Thus, we test the performance of our algorithms on both small road networks (such as the single-junction network and the four-junction corridor) as well as large size road network (such as the 4x4-grid and the IISc network). The 4x4-grid network consists of 16 edge nodes (where traffic is generated), 16 junctions with traffic lights, 40 roads, with each being 4 lanes wide and when full can house upto 1500 vehicles. Further, the cardinality of the state-action space is of the order of 10^{130} for the 4x4-grid network. We tested the performance of all the proposed TLC algorithms - PTLC, QTLC-SA and QTLC-FA-NFS (and their tuning counterparts) as well as the QTLC-FA algorithm from Prashanth and Bhatnagar [2011a] on a single junction and four-junction corridor. However, on a 4x4-grid network as well as the IISc network, we could not implement the QTLC-SA algorithm. As explained before, QTLC-SA uses full state representations and hence is not implementable on high-dimensional state-action spaces (as with the 4x4-grid network and the IISc network). However, QTLC-SA is an enhancement to the Q-learning based TLC algorithm proposed for instance in Abdulhai et al. [2003]. In fact, the Q-learning based TLC with full state representations in Abdulhai et al. [2003] was proposed only for a single-junction road network. Use of state aggregation allowed the resulting QTLC-SA algorithm to scale up to a four-junction corridor. The results in this scenario are presented here.

The simulations are conducted for 25000 cycles for all algorithms and in each simulation, the destination of the road user is fixed randomly (using a discrete uniform distribution). At each time-step, vehicles are inserted into the road network as per the spawn frequencies of the edge nodes, where spawn frequency specifies the rate at which traffic is generated randomly in GLD. The spawn frequencies of the edge nodes are set in a way that ensures that the proportion of cars flowing on the main road to those on the side roads is in the ratio 100 : 5. This ratio is close to what is practically observed and has also been used for instance in Cools et al. [2008]. The results presented below are the averages over ten independent simulations with different initial seeds. For all the algorithms, the weights in the single-stage cost function (7.2) are set as $\alpha_1 = \beta_1 = 0.5$, $\alpha_2 = 0.6$ and $\beta_2 = 0.4$, respectively. This assignment essentially gives equal weightage to queue length and elapsed time components of (7.2), while according a higher weightage to main road traffic over side road traffic. The threshold parameter θ was set to $(6, 14, 90)^T$ for all the TLC algorithms. The performance of the TLC algorithms with the above tuning parameter was then compared with the counterparts of these algorithms that incorporate tuning. This value of θ has been used in the results reported in Prashanth and Bhatnagar [2011a]. The parameters δ and L used in the threshold tuning algorithm (7.5) were set to 0.5 and 10 respectively. The discount factor γ used in (7.6) and (7.9) was set to 0.9.

Table 7.3: Total Arrived Road Users (TAR) – results from two experiments

(a) TAR for a range of weights α_1 on Single-Junction

α_1	PTLC	QTLC-FA-NFS
0.2	16514.38±674.09	26675.27±660.33
0.3	17563.46±683.51	25632.40±669.58
0.4	17585.98±663.99	27627.77±669.28
0.5	**18365.80±662.31**	**27779.77±672.60**
0.6	17486.90±669.88	27571.61±670.46
0.7	17683.87±659.67	26454.72±669.99
0.8	14722.40±646.74	25385.50±653.01

(b) TAR for various TLC algorithms on two road networks

	4x4-Grid Network	IISc Network
QTLC-FA-NFS	17408.56 ± 84.20	21751.17 ± 525.41
QTLC-FA-NFS-TT	**22917.40 ± 80.83**	**24773.38 ± 626.48**
PTLC	3807.14 ± 70.26	16235.74 ± 472.31
PTLC-TT	7662.64 ± 82.38	18878.87 ± 461.66
QTLC-FA	14418.95±80.15	18699.59 ± 415.35
QTLC-FA-TT	19369.61 ± 77.96	20531.24 ± 520.26

7.5.2 Results

Fig. 7.3 shows the average trip waiting time (ATWT) plots on a single-junction and a four-junction corridor, respectively and compares the four TLC algorithms PTLC, QTLC-SA, QTLC-FA and QTLC-FA-NFS when combined with the threshold tuning algorithm (7.5). Figs. 7.4(a) – 7.5(b) show ATWT plots comparing PTLC, QTLC-FA and QTLC-FA-NFS algorithms with their tuning counterparts on a 4x4-grid network and the IISc network respectively. Table. 7.3(b) presents the total arrived road users (TAR) for the various TLC algorithms on a 4x4-grid network and the IISc network respectively. As mentioned before, since QTLC-SA is not implementable on high-dimensional state spaces (and hence, larger networks), performance comparisons of this algorithm are presented only on a single junction and a four-junction corridor. The choice of weights $\alpha_1 = \beta_1 = 0.5$ is justified by the results obtained for different choices of α_1 in Table. 7.3(a).

While the default setting in GLD for the traffic arrival pattern is uniform, we also conducted an experiment where the traffic arrived according to a Poisson process. The results from this experiment on two road networks for the PTLC and QTLC-FA-NFS algorithms are presented in Fig. 7.6. We observe that, even in this case, our tuning scheme results in a performance improvement for both the algorithms. This is significant because our tuning scheme is seen to perform well even for cases where the arrival distribution is not uniform. Note that in the basic problem formulation, we did not make any assumption about the nature of the arrival distribution, except assuming a Markovian evolution for the system.

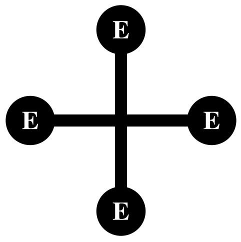
We also observe that the parameter θ converges to the optimal threshold value for each TLC algorithm - PTLC, QTLC-SA and QTLC-FA - considered in each of the road networks (See Fig. 7.2). This is illustrated by the convergence plots in Figures. 7.7–7.8. These plots as well as the ATWT plots also show that the transient phase of the threshold tuning algorithm (7.5), i.e., the initial period when the threshold parameter θ has not converged, is short.

We observe that incorporating our threshold tuning algorithm results in performance enhancements for all the TLC algorithms. Further, among the TLC algorithms studied, it is evident that QTLC-FA-NFS algorithm shows the best overall performance on all network settings considered. In particular, it outperformed the algorithm from Prashanth and Bhatnagar [2011a] (Chapter 5), which establishes the superiority of the feature selection procedure of QTLC-FA-NFS over QTLC-FA. While PTLC-TT algorithm worked second-best to QTLC-FA-NFS-TT on the single-junction road network, on larger road networks it is found to perform poorly in comparison to the QTLC-FA-NFS algorithm and its tuning variant.

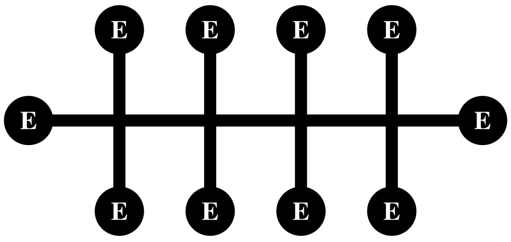
7.6 Summary

Our goal in this chapter was to design an algorithm for tuning the thresholds to their optimal values in any graded threshold based TLC algorithm. Towards this end, we introduced and analyzed three new traffic light control algorithms with threshold tuning. We developed and applied, for the first time, an SPSA-based threshold tuning algorithm that incorporates Hadamard matrix based deterministic perturbation sequences and with proven convergence to the optimal thresholds for the various algorithms. We developed three new TLC algorithms - a priority-based TLC algorithm, an enhanced version of the Q-learning with full-state representation TLC algorithm that incorporates state aggregation and a Q-learning algorithm with function approximation that incorporates a novel feature selection scheme. We combined our threshold tuning algorithm with the above mentioned TLC algorithms. In the case of the Q-learning with full state representations, our threshold tuning algorithm finds an "optimal" way of clustering states based on the aforementioned thresholds and in the case of the Q-learning with function approximation based TLC, our algorithm amounts to feature adaptation for an RL algorithm using function approximation (a topic that is of independent research interest in the RL community). Empirical observations indicate a significant gain in performance when our threshold tuning algorithm is used in conjunction with each of the TLC algorithms considered. The tuning variants of the TLC algorithms (above) clearly outperform their counterparts that use (arbitrarily set) fixed thresholds.

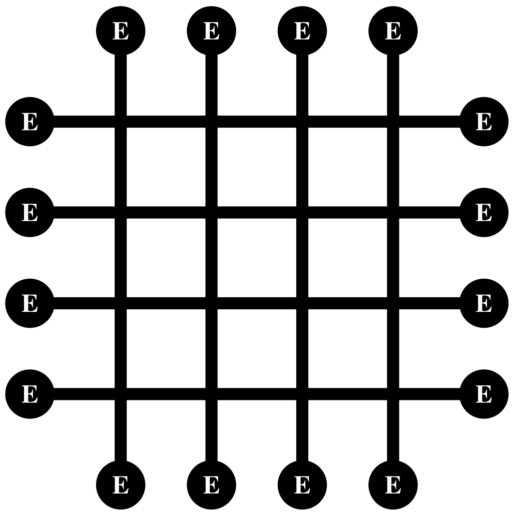
It may be noted that SPSA is a local search based optimization technique. While one could possibly consider methods such as simulated annealing to find the global minimum, the flip side of such approaches is the computational complexity requirement that is usually very high unlike SPSA. Further, in this particular application scenario, we have observed that the deterministic perturbation SPSA based threshold tuning algorithm works well and this is demonstrated by the results of the simulation experiments.



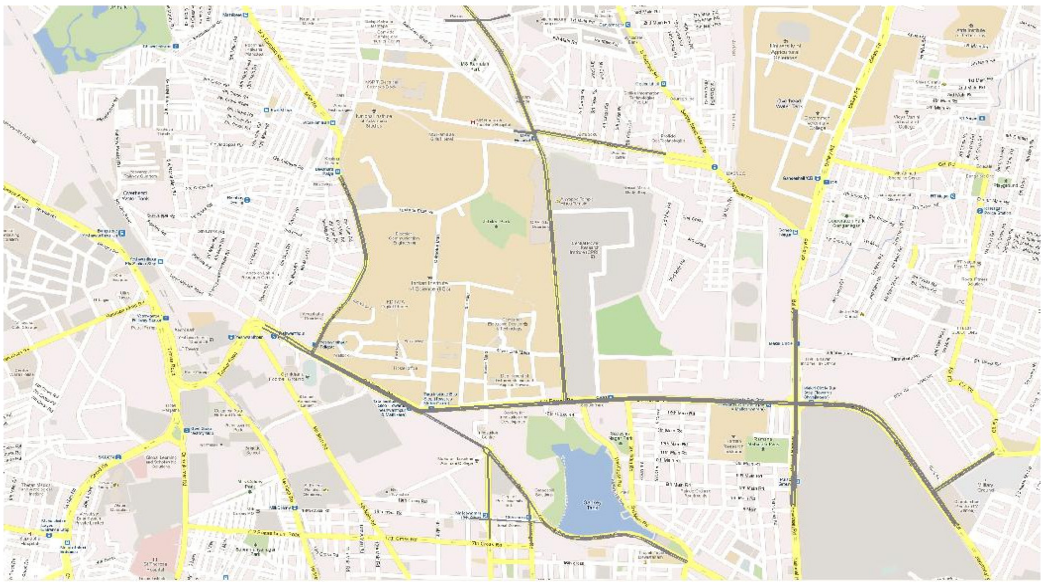
(a) Single Junction



(b) Four-Junction Corridor

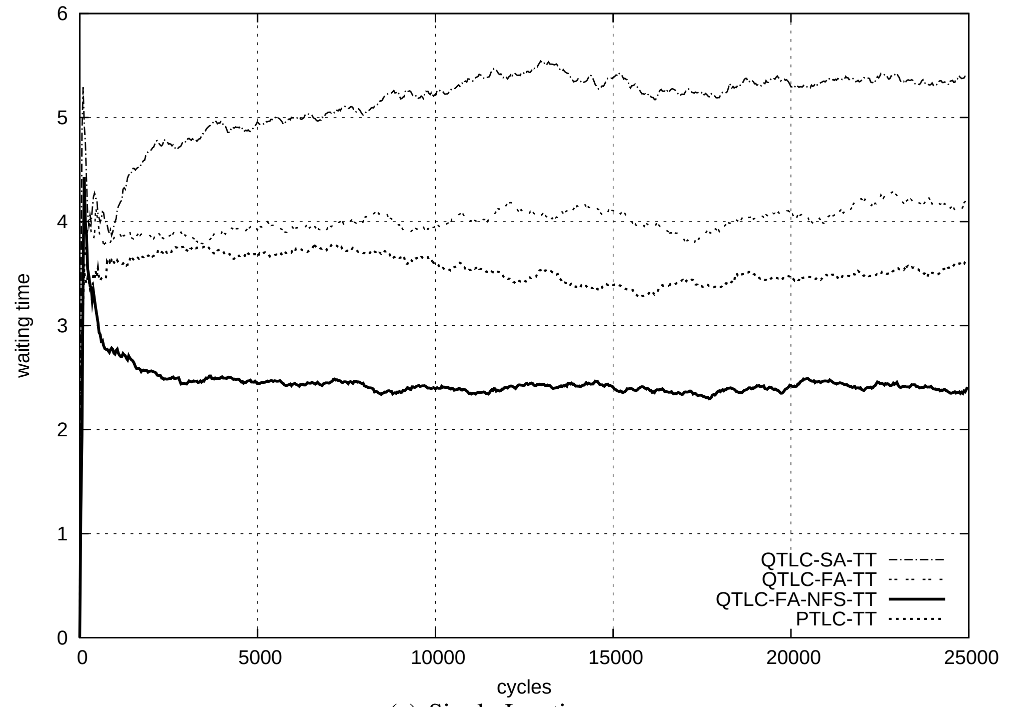


(c) 4x4-Grid Network

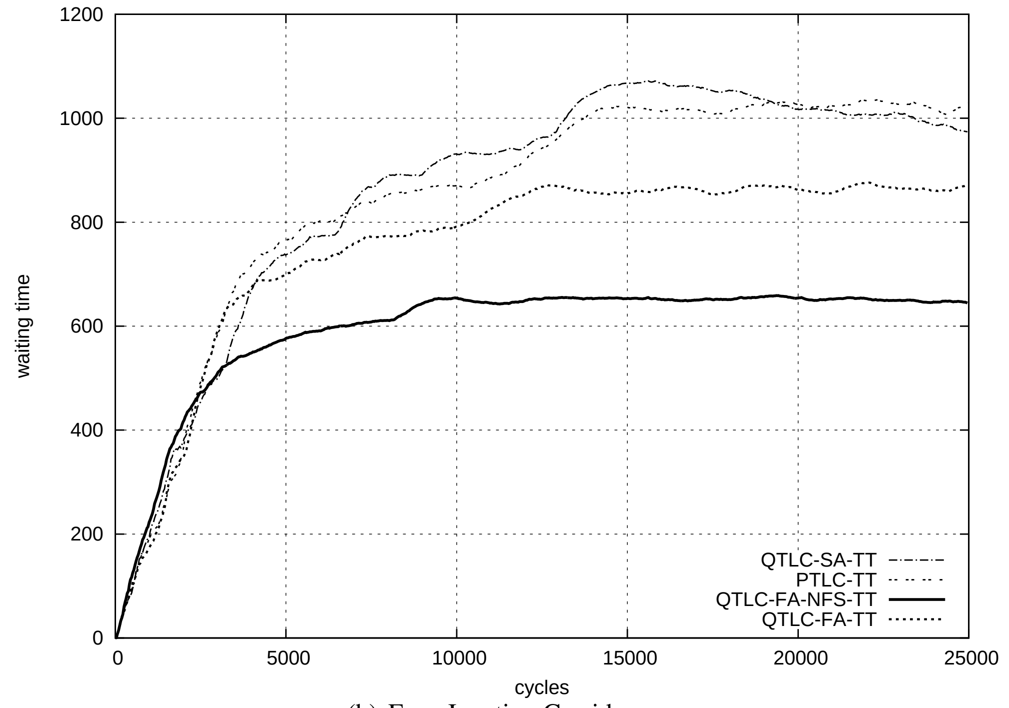


(d) The IISc Network

Figure 7.2: Road Networks used for our Experiments

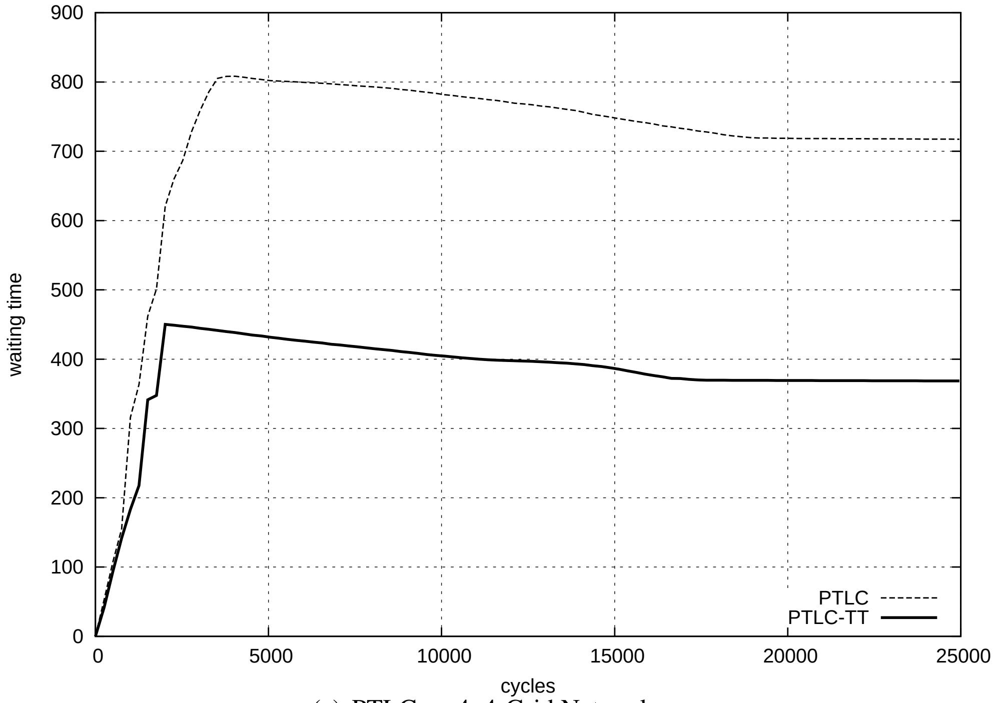


(a) Single Junction

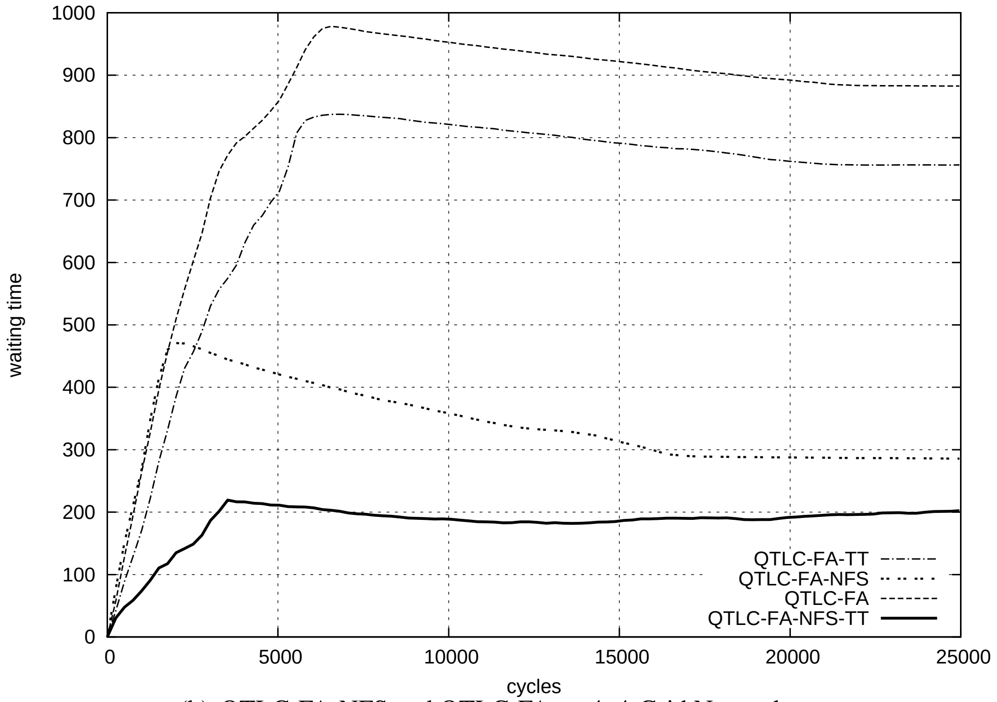


(b) Four-Junction Corridor

Figure 7.3: Performance Comparison of TLC Algorithms using Average Trip Waiting Time as the metric on two small road networks

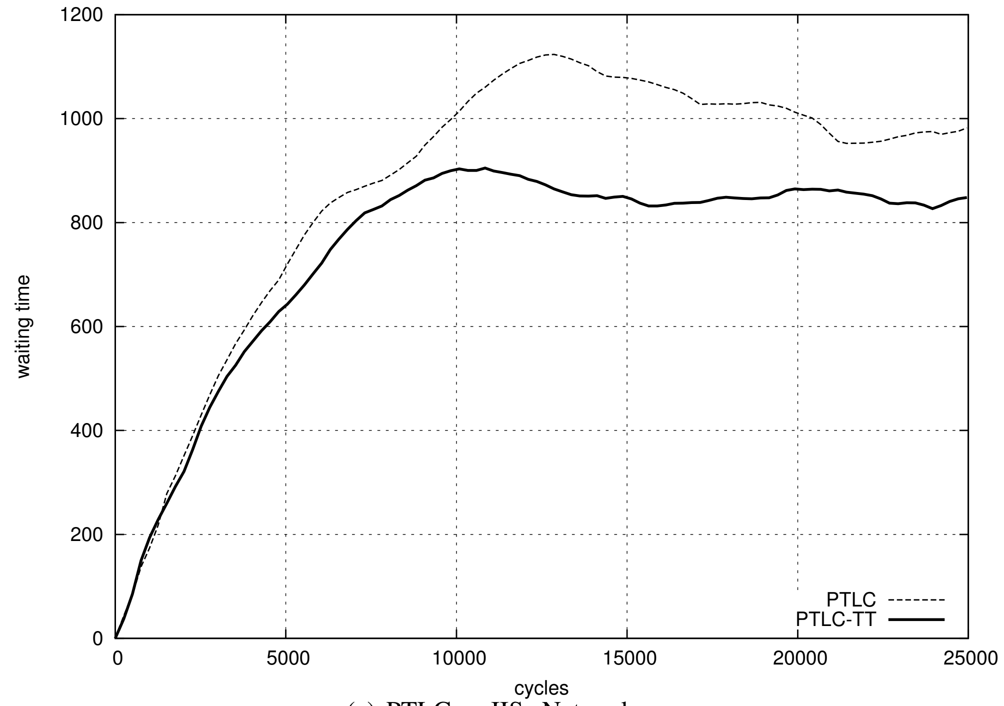


(a) PTLC on 4x4-Grid Network

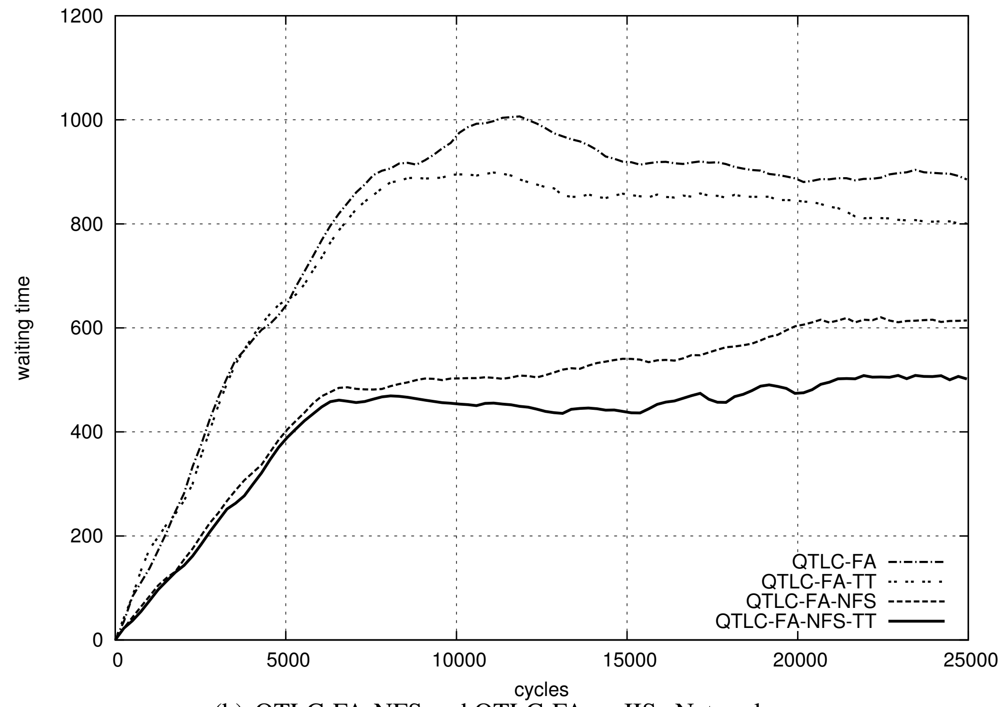


(b) QTLC-FA-NFS and QTLC-FA on 4x4-Grid Network

Figure 7.4: Performance Comparison of TLC Algorithms with their tuning counterparts on the 4x4-Grid Network

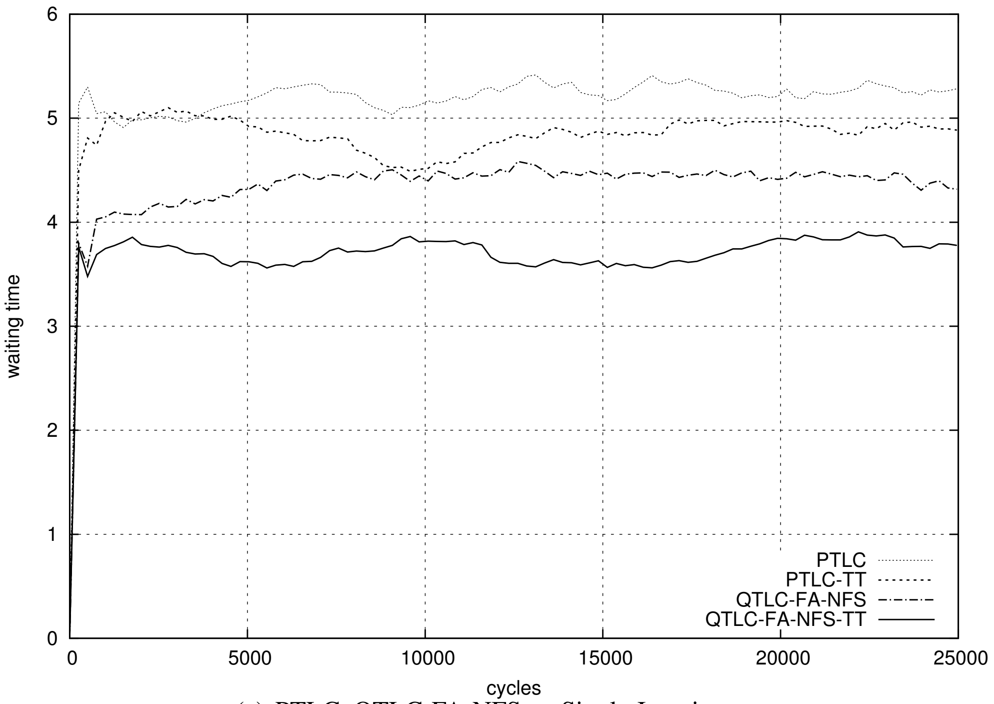


(a) PTLC on IISc Network

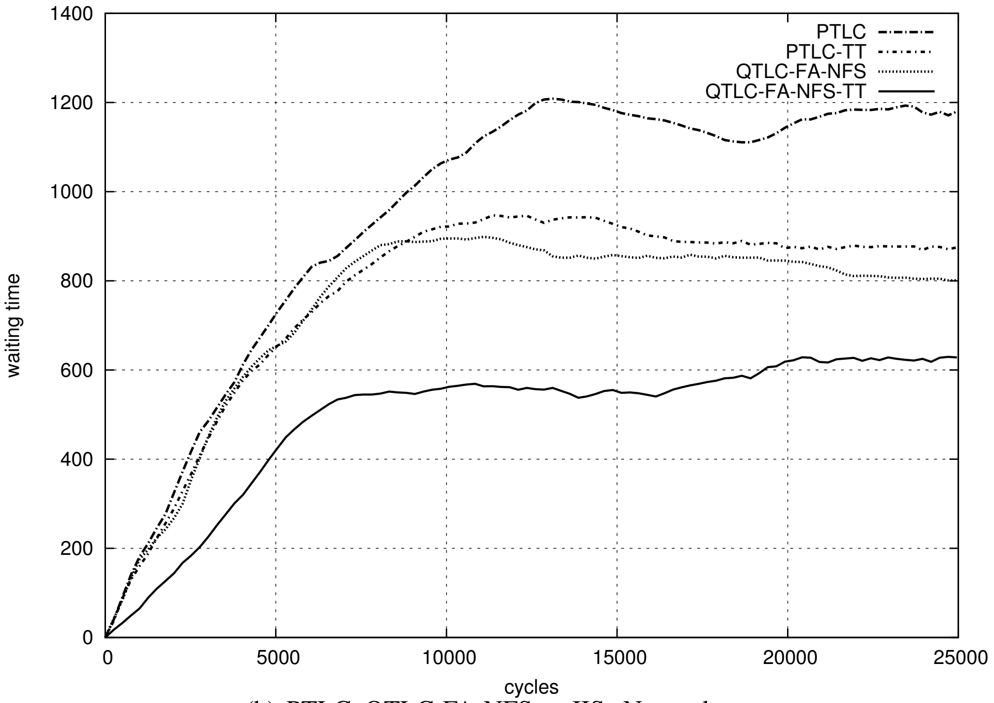


(b) QTLC-FA-NFS and QTLC-FA on IISc Network

Figure 7.5: Performance Comparison of TLC Algorithms with their tuning counterparts on the IISc Network

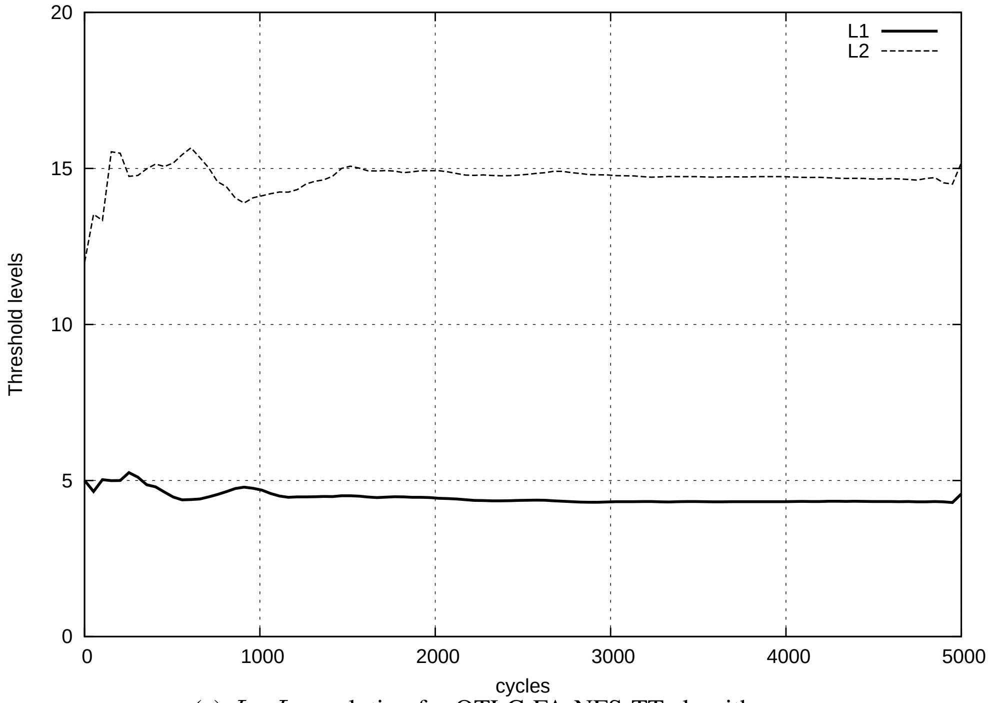


(a) PTLC, QTLC-FA-NFS on Single Junction

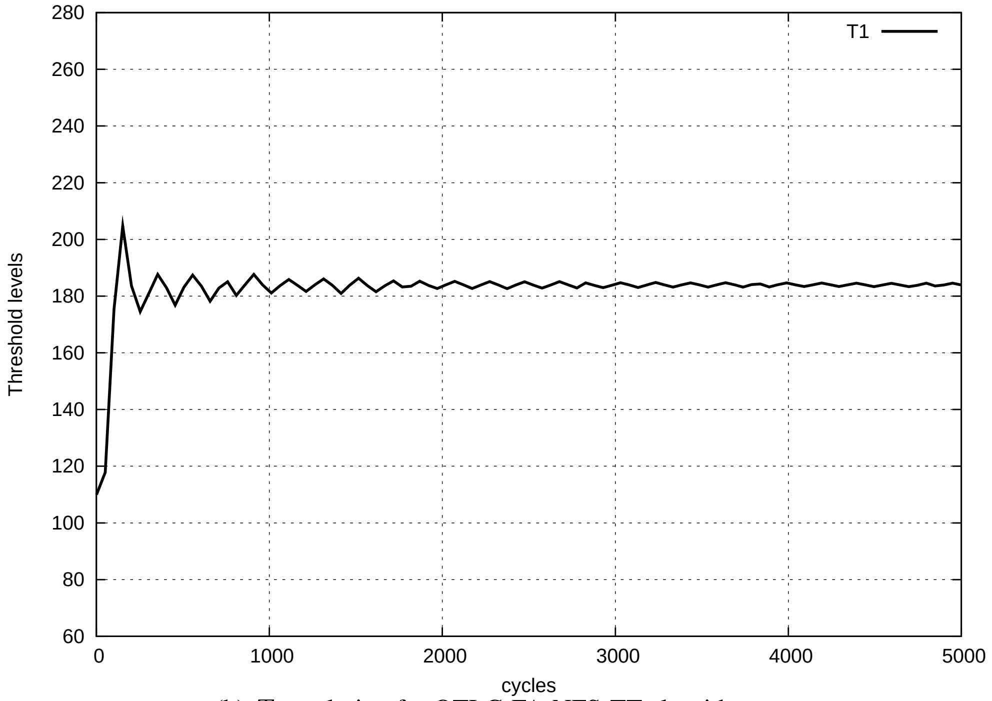


(b) PTLC, QTLC-FA-NFS on IISc Network

Figure 7.6: Performance Comparison of TLC Algorithms with Poisson vehicle arrival pattern on two road networks

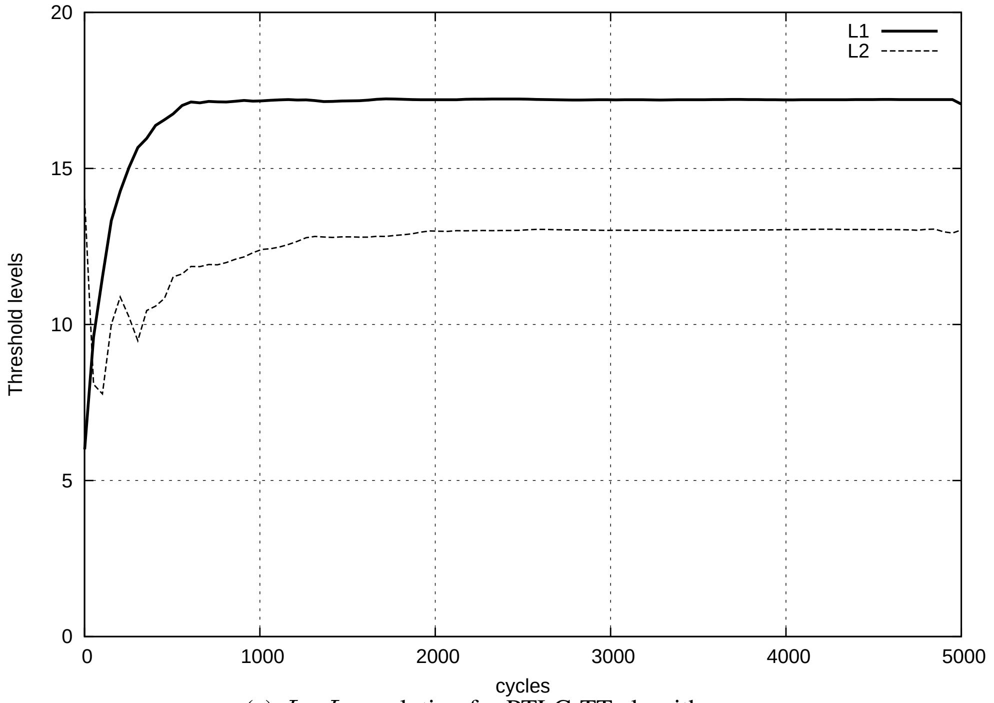


(a) L_1, L_2 evolution for QTLC-FA-NFS-TT algorithm

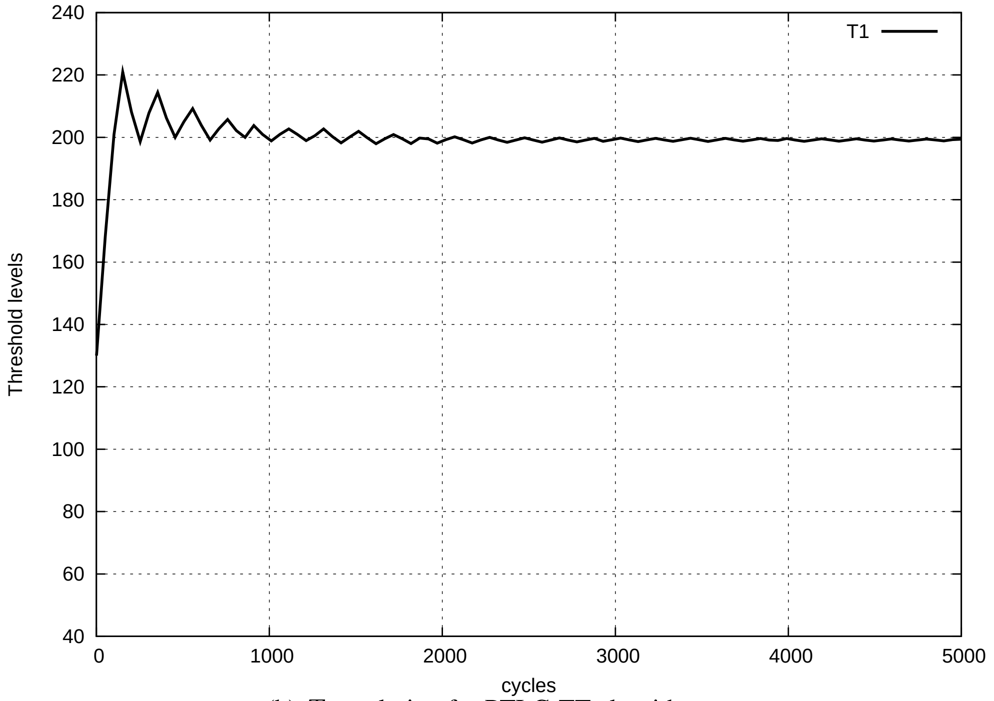


(b) T_1 evolution for QTLC-FA-NFS-TT algorithm

Figure 7.7: Convergence of θ parameter - illustration for QTLC-FA-NFS-TT algorithm in the case of a Four-Junction Corridor



(a) L_1, L_2 evolution for PTLC-TT algorithm



(b) T_1 evolution for PTLC-TT algorithm

Figure 7.8: Convergence of θ parameter - illustration for PTLC-TT algorithm in the case of a Four-Junction Corridor

Chapter 8

Online Adaptation of Features in Reinforcement Learning

In this chapter, we propose a new algorithm for feature adaptation and study its application to the problem of traffic signal control. We consider the problem of prediction (i.e., estimating the value function corresponding to a given policy) under the infinite horizon discounted cost criterion and employ the TD learning scheme with linear function approximation for this purpose. We let all feature components take values only between 0 and 1. Upon convergence of the TD scheme for a fixed set of features, we consider the converged weights in the weight vector (i.e., the parameter whose scalar product with the state-feature approximates the value of that state). We then replace two of the columns of the feature matrix that correspond to the two smallest components of the weight vector as follows. The column of the feature matrix corresponding to the smallest component of the weight vector is replaced by the most recent estimate of the value function (suitably normalized so that all its entries are between 0 and 1) obtained after running the TD scheme for a given large number of iterations using the feature matrix of the previous step, while the column corresponding to the second smallest component of the weight vector is replaced by independent and uniformly generated random numbers between 0 and 1. Thus while the worst performer (amongst the columns of the feature matrix) is replaced by the normalized value function estimate, the second-worst performer is replaced with a random search direction to aid in exploration of better features and thereby a better subspace than the previous. The remaining columns of the feature matrix are left unchanged. The TD algorithm is then run again with the new feature matrix and the process repeated.

As an application setting, we consider the problem of estimating the value of a policy in a problem of traffic signal control (Prashanth and Bhatnagar [2011a]). The general control problem in this domain is to obtain a policy for signal switching across various sign configurations so that traffic flow is maximized and levels of congestion and hence delays at the traffic junctions are minimized. In Prashanth and Bhatnagar [2011a], Q-learning with function approximation for a given feature matrix is applied for this problem. By keeping the optimal policy (obtained from the Q-learning algorithm in Prashanth and Bhatnagar [2011a]) corresponding to the original feature matrix fixed, we apply our algorithm for feature adaptation and observe consistent improvement in performance over "episodes" during each of which the feature matrix is held fixed and TD is applied. We study the performance of our scheme on two different network settings and observe performance improvements in both.

The rest of the chapter is organized as follows: In Section 8.1, we give the framework. The feature adaptation scheme is presented in Section 8.2. In Section 8.3, we present a partial analysis of the convergence behaviour of our scheme. In Section 8.4, the results of numerical experiments in the traffic signal control setting are shown. Finally, Section 8.5 presents the concluding remarks.

8.1 The Framework

Our basic setting is an MDP $\{X_n\}$ taking values in a set S (called the state space) that is governed by a control sequence $\{Z_n\}$. Let $A(i)$ be the set of feasible controls or actions in state i and $A \stackrel{\triangle}{=} \cup_{i \in S} A(i)$ be the set of all actions or the action space. We assume that both S and A are finite sets.

We consider here that the MDP $\{X_n\}$ is governed by a given (fixed) SDP μ. In such a case $\{X_n\}$ is in fact a Markov chain taking values in S. Let $c(i,a)$ denote the single-stage cost when the Markov chain is in state i and action a is picked. Given $\gamma \in (0,1)$, let

$$V^{\mu}(i) = E\left[\sum_{m=0}^{\infty} \gamma^m c(X_m, \mu(X_m)) \mid X_0 = i\right] \tag{8.1}$$

denote the value function under SDP μ when the initial state of $\{X_n\}$ is i. The value function can be obtained by solving the system of equations

$$V^{\mu}(i) = c(i, \mu(i)) + \gamma \sum_{j \in S} p(i, j, \mu(i)) V^{\mu}(j), \ i \in S, \tag{8.2}$$

or in vector notation,

$$V^\mu = c^\mu + \gamma P^\mu V^\mu, \tag{8.3}$$

where $P^\mu = [[p(i,j,\mu(i))]]_{i,j\in S}$ is the transition probability matrix of $\{X_n\}$, $c^\mu = (c(i,\mu(i)), i \in S)^T$ is the vector of single-stage costs, and $V^\mu = (V^\mu(i), i \in S)^T$ is the vector of 'values' over individual states respectively, under policy μ. The system of equations (8.3) yield the solution

$$V^\mu = (I - \gamma P^\mu)^{-1} c^\mu, \tag{8.4}$$

where I denotes the $(|S| \times |S|)$–identity matrix. When the size of the state space ($|S|$) is large, obtaining the inverse of the matrix $(I - \gamma P^\mu)$ is computationally hard. An alternative is to solve (8.3) using a value iteration procedure. However, since (8.3) corresponds to a linear system of $|S|$ equations, one expects such a procedure to be slow as well for large $|S|$.

Another problem in using a purely numerical approach is that in most real life situations, the transition probabilities $p(i,j,\mu(i))$, $i,j \in S$ are in fact not known. A way out is to use a combination of value function approximation (via a so-called approximation architecture) and stochastic approximation. The former helps to make the computational complexity manageable while the latter helps to learn the (approximated) value function with only real or simulated measurements and without any knowledge of the transition probabilities.

We approximate $V^\mu(j) \approx \phi(j)^T\theta$, where $\phi(j) = (\phi_1(j), \ldots, \phi_d(j))^T$ is a d-dimensional feature associated with state j. Also, $\theta = (\theta(1), \ldots, \theta(d))^T$ is an associated parameter. Let Φ denote the $|S| \times d$ feature matrix with $\phi(j)^T$, $j \in S$, as its rows. Thus $\Phi = [[\phi_k(s)]]_{k=1,\ldots,d, s\in S}$. Let $\phi_k \triangleq (\phi_k(s), s \in S)^T$ denote the kth column of Φ, $k \in \{1, \ldots, d\}$.

We make the following assumptions.

Assumption 8.1 *The Markov chain $\{X_n\}$ under SDP μ is irreducible.*

Assumption 8.2 *The d columns of the matrix Φ, i.e., $\phi_1, \ldots, \phi_d$ are linearly independent. Further, $d \leq |S|$.*

From Assumption 8.1, $\{X_n\}$ is also positive recurrent (since $|S| < \infty$). Let $d^\mu(i)$,

$i \in S$ be the stationary distribution of $\{X_n\}$ under policy μ. Further, let D^μ be a diagonal matrix with entries $d^\mu(i)$, $i \in S$ along the diagonal. The regular TD(0) algorithm is as follows:

The TD(0) Learning Algorithm

$$\theta_{n+1} = \theta_n + a(n)\delta_n\phi(X_n), \tag{8.5}$$

where δ_n is the *temporal difference term* that is defined according to

$$\delta_n = (c(X_n, \mu(X_n)) + \gamma\phi(X_{n+1})^T\theta_n - \phi(X_n)^T\theta_n), \; n \geq 0.$$

Further, $a(n), n \geq 0$ is a sequence of step-sizes that satisfy

$$\sum_n a(n) = \infty, \; \sum_n a^2(n) < \infty. \tag{8.6}$$

The parameter θ_n is d-dimensional and has components (say) $\theta_n(1), \ldots, \theta_n(d)$. Thus, $\theta_n = (\theta_n(1), \ldots, \theta_n(d))^T$. The algorithm (8.5) is called the temporal difference learning (TD(0)) algorithm for the discounted cost case and has been analyzed for its convergence in a more general setting (TD(λ) with $\lambda \in [0, 1]$) in Tsitsiklis and Van Roy [1997]. The average cost version of this algorithm has also been analyzed in Tsitsikis and Van Roy [1999] as well as Bhatnagar et al. [2009a].

It can be shown (see Tsitsiklis and Van Roy [1997]) that the TD(0) algorithm (8.5) converges according to $\theta_n \to \theta^*$ with probability one, as $n \to \infty$, where

$$\theta^* \triangleq (\theta_1^*, \ldots, \theta_d^*)^T = (\Phi^\top D^\mu(\gamma P^\mu - I)\Phi)^{-1}\Phi^\top D^\mu c^\mu.$$

In the next section, we describe our feature adaptation scheme.

8.2 The Feature Adaptation Scheme

We let all entries of the feature matrix Φ to take values between 0 and 1. Recall that we approximate $V^\mu(i) \approx \phi(i)^T\theta$. Thus, $V^\mu = (V^\mu(1), \ldots, V^\mu(|S|))^T$ can be approximated as

$$V^\mu \approx \Phi\theta = \begin{bmatrix} \sum_{j=1}^{d} \phi_j(1)\theta_j \\ \sum_{j=1}^{d} \phi_j(2)\theta_j \\ \cdot \\ \cdot \\ \cdot \\ \sum_{j=1}^{d} \phi_j(|S|)\theta_j \end{bmatrix}.$$

Alternatively, note that

$$V^\mu \approx \sum_{j=1}^{d} \phi_j\theta_j.$$

We run the algorithm using a nested loop sequence with the outer loop run for a total of R iterations. The feature matrix is updated in the outer loop procedure. In between two successive updates of the outer loop, i.e., for a fixed feature matrix update, the inner loop comprising of TD recursions is run for a given (large) number M of iterates in order to obtain a final parameter corresponding to the aforementioned feature matrix (update). We call each such M-iterate run of TD for a given feature matrix an episode. Thus our scheme is run for a total of R episodes. The feature adaptation scheme is described in detail below:

The Feature Adaptation Scheme:

- *Step 0 (Initialize): Select a $|S| \times d$ feature matrix Φ^0 with columns $\phi_j^0 = (\phi_j^0(s), s \in S)^T$, $j = 1, \ldots, d$. Let Φ^0 have all its entries between 0 and 1 and let it satisfy Assumption 8.2. Set $r := 0$ and $n := 0$, respectively. Set $V_0^{\mu,0} = 0$ to be the initial estimate of the value function. Let $\bar{V}_0^{\mu,0} = 0$ be the initial estimate of the normalized value function. Also set M and R to be given large integers. Set θ_0^0 as the initial parameter value.*

- *Step 1: Let Φ^r denote the feature matrix during the rth step of the algorithm. The $(n+1)$st update of TD in the rth step of the algorithm is given as follows:*

$$\theta_{n+1}^r = \theta_n^r + a(n)\delta_n^r\phi^r(X_n), \tag{8.7}$$

where $a(n)$, $n \geq 0$ satisfy (8.6) and δ_n^r, $n \geq 0$ are defined according to

$$\delta_n^r = (c(X_n, \mu(X_n)) + \gamma \phi^r(X_{n+1})^T \theta_n^r - \phi^r(X_n)^T \theta_n^r).$$

Set $n := n+1$. If $n = M-1$, set $V_M^{\mu,r} = \Phi^r \theta_M^r$, and go to Step 2; else repeat Step 1.

- *Step 2: Let $\theta_M^r \overset{\triangle}{=} (\theta_{M,1}^r, \ldots, \theta_{M,d}^r)^T$. Find $\theta_{M,k}^r$, $\theta_{M,l}^r$, $k, l \in \{1, \ldots, d\}$, $k \neq l$ such that*

$$\theta_{M,k}^r \leq \theta_{M,l}^r \leq \theta_{M,j}^r \; \forall j \in \{1, \ldots, d, j \neq k, j \neq l\}.$$

(Any tie is resolved according to some prescribed rule). Obtain a new feature matrix Φ^{r+1} as follows: First normalize $V_M^{\mu,r}$ (by first letting any negative component to be zero and then dividing every component by one with the largest value) to obtain $\bar{V}_M^{\mu,r}$. Thus, all entries of $\bar{V}_M^{\mu,r}$ lie between 0 and 1. Set $\phi_k^{r+1} := \bar{V}_M^{\mu,r}$. Next obtain ϕ_l^{r+1} by picking all its entries independently according to the uniform distribution on $[0,1]$. Retain the remaining columns in the Φ^{r+1} matrix from the matrix Φ^r itself, i.e., set $\phi_i^{r+1} := \phi_i^r$ for all $i \in \{1, \ldots, d\}$ with $i \neq k, l$. Set Φ^{r+1} to be the matrix with columns ϕ_j^{r+1}, $j = 1, \ldots, d$. Set $r := r+1$. If $r < R$, go to Step 1; else go to Step 3.

- *Step 3 (Termination): Output θ_M^R as the final parameter value and $V_M^{\mu,R} = \Phi^R \theta_M^R$ as the corresponding value function estimate.*

We summarize the notation used with regards to the value function below (as this will also be used in the convergence analysis):

- V^μ: exact value function under policy μ,
- $V_n^{\mu,r} = \Phi^r \theta_n^r$: estimate of the value function at the rth iterate of algorithm after the nth run of Step 1 ($M \geq n \geq 0$),
- $V_M^{\mu,r} = \Phi^r \theta_M^r$: estimate of the value function after completion of Step 1 during the rth iterate of algorithm (i.e., with $n = M$),
- $V_*^{\mu,r} = \Phi^r \theta_*^r$: estimate of the value function from Step 1 of algorithm at the rth iterate if Step 1 was run until convergence of TD, i.e., where $\theta_*^r = \lim\limits_{M \to \infty} \theta_M^r$ with probability one,

- $\bar{V}_M^{\mu,r}$: normalized estimate of value function after completion of Step 1 during the rth iterate of algorithm,
- $\check{V}_r^{\mu}$: projection of V^μ to S_r.

Note above that we obtain the normalized value function $\bar{V}_M^{\mu,r}$ at each iterate r of the algorithm only after termination of Step 1 on that iterate. Convergence of the TD scheme for a given feature matrix Φ^r, as $M \to \infty$, has been shown in the literature (Tsitsiklis and Van Roy [1997], Bertsekas and Tsitsiklis [1996a]) under the weighted Euclidean norm $\|\cdot\|_{D^\mu}$ defined according to

$$\| x \|_{D^\mu} = \sqrt{x^\top D^\mu x},$$

for any $x \in \mathcal{R}^{|S|}$. Let S_r, $r = 1, 2, \ldots$ denote the subspace

$$S_r = \{\Phi^r \theta \mid \theta \in \mathcal{R}^d\}.$$

Let $\check{V}_r^{\mu}$ represent the best approximation to the value function V^μ within the subspace S_r. This is obtained by simply projecting V^μ to S_r. Note that TD (or for that matter any other algorithm that operates only within the subspace S_r) does not in general converge to $\check{V}_r^{\mu}$. In fact the parameters θ_n^r, $n \geq 0$ obtained from TD, see Step 1 of the feature adaptation scheme (above), converge to θ_*^r almost surely where

$$\theta_*^r = (\Phi^{r\top} D^\mu (\gamma P^\mu - I)\Phi^r)^{-1} \Phi^{r\top} D^\mu c^\mu.$$

This would correspond to a value function estimate of $V_*^{\mu,r} = \Phi^r \theta_*^r$ which is not the same as $\check{V}_r^{\mu}$ even though both $V_*^{\mu,r}$ and $\check{V}_r^{\mu}$ are vectors in the subspace S_r. Since we run TD for a fixed number M of instants during each visit of the algorithm to Step 1, the estimate of the value function as given by TD after the rth cycle (of M iterates) is $V_M^{\mu,r} = \Phi^r \theta_M^r$. By choosing M sufficiently large, one can bring down the difference between $V_M^{\mu,r}$ and $V_*^{\mu,r}$.

The key idea behind our adaptation scheme is the following: Since one replaces the ‘worst basis vector’ by the current best estimate of the value function and the ‘next-to-worst basis vector’ by uniformly generated random numbers, one ensures that (a) the best linear combination of the basis functions in the current subspace is retained while (b) in addition, the scheme does a random exploration in order to find a potentially better subspace than the current. As our experiments demonstrate, it is indeed seen to be the case that our adaptive scheme results in significant performance improvement.

Remark 4 *One possible variation is to pick new basis vectors from a prespecified over-complete basis. These vectors could be sparsely represented. Nevertheless, the running estimate of the value function need not have a sparse representation. Thus the algorithm stores all but one sparse vectors, thus saving on storage.*

Remark 5 *Another possible tweak is to perform a Gram-Schmidt step on each new independent basis vector picked relative to the rest and replace it by the unit normal vector thus obtained. While this is aesthetically more pleasing and will avoid certain 'low probability' pathologies, it has a computational overhead. We did not use this tweak in our experiments as we obtained good results regardless.*

Remark 6 *A good error estimate for fixed point of a projected linear contraction relative to fixed point of the contraction itself, that holds with 'high probability' under certain dimensionality requirements on the range of the projection, is given in Barman [2008].*

In the next section, we give a partial analysis of the convergence behaviour of our feature adaptation scheme.

8.3 Convergence Analysis

We first introduce some notation. Let U denote the unit sphere in $\mathcal{R}^{|S|}$. Since we normalize the basis vectors, they can be identified with points in U. Let $V^{\perp}$ denote the subspace of $\mathcal{R}^{|S|}$ orthogonal to V^{μ} (recall that V^{μ} is the true value function). Let $B \subset U$ denote the ϵ-neighborhood of $V^{\perp} \cap U$ in U for a small $\epsilon > 0$. Let D denote the ϵ-neighborhood of V^{μ}. Let Π_r denote the projection to $Range(\Phi^r)$. Then $\check{V}^{\mu}_r = \Pi_r(V^{\mu})$. Let $\Psi : \mathcal{R}^{|S|} \times \mathcal{R}^{|S|} \to \mathcal{R}^{|S|}$ denote the map

$$\Psi(x,y) := \frac{x - \langle x, y/||y||\rangle y/||y||}{||x - \langle x, y/||y||\rangle y/||y|| ||},$$

i.e., the component of x orthogonal to y, normalized to have unit norm. Let

$$\beta_r = \frac{||\ V^{\mu} - V^{\mu,r}_{*}\ ||}{||\ V^{\mu} - \check{V}^{\mu}_r\ ||}.$$

We have the following lemma.

Lemma 8.1 *(a)* $\beta_r \in \left[1, \dfrac{1}{1-\gamma}\right] \; \forall r \geq 0.$

(b) For β_r close to one (for any $r \geq 0$), $V^{\mu,r}_{*} \approx \check{V}^{\mu}_r$.

Proof: (a) It is easy to see that the vectors $V^\mu - V_*^{\mu,r}$, $V^\mu - \check{V}_r^\mu$ and $\check{V}_r^\mu - V_*^{\mu,r}$ form a right angle triangle with $V^\mu - V_*^{\mu,r}$ as the hypotenuse. Further, $V^\mu - \check{V}_r^\mu$ and $\check{V}_r^\mu - V_*^{\mu,r}$ form a right angle with $\check{V}_r^\mu - V_*^{\mu,r}$ being a vector within the subspace S_r and $V^\mu - \check{V}_r^\mu$ being orthogonal to S_r. Thus,

$$\| V^\mu - V_*^{\mu,r} \| \geq \| V^\mu - \check{V}_r^\mu \|,$$

implying that $\beta_r \geq 1 \; \forall r \geq 0$. Further, from Lemma 6 of Tsitsiklis and Van Roy [1997], we also obtain that

$$\| V^\mu - V_*^{\mu,r} \| \leq \frac{1}{(1-\gamma)} \| V^\mu - \check{V}_r^\mu \| .$$

Thus we also have $\beta_r \leq \dfrac{1}{1-\gamma} \; \forall r \geq 0$.
(b) By the Pythagorean theorem, we have

$$\| V^\mu - V_*^{\mu,r} \|^2 = \| V^\mu - \check{V}_r^\mu \|^2 + \| \check{V}_r^\mu - V_*^{\mu,r} \|^2 . \tag{8.8}$$

From (8.8), we obtain

$$\beta_r^2 = 1 + \frac{\| \check{V}_r^\mu - V_*^{\mu,r} \|^2}{\| V^\mu - \check{V}_r^\mu \|^2},$$

or that

$$\begin{aligned} \| \check{V}_r^\mu - V_*^{\mu,r} \| &= \sqrt{\beta_r^2 - 1} \, \| V^\mu - \check{V}_r^\mu \| \\ &\leq 2K \sqrt{\beta_r^2 - 1}, \end{aligned}$$

where K is an upper bound on $\| V^\mu \|$. Thus if β_r is close to one, we have that $V_*^{\mu,r} \approx \check{V}_r^\mu$.
$\square$

Now let $\nu > 0$ be such that $\| \check{V}_r^\mu - V_*^{\mu,r} \| < \nu$. Let $\alpha > 0$ be such that if $V \notin D$ and $z \notin B$ is a unit vector, then for $\Pi' :=$ projection onto $Range(\Phi^r)$ and $\Pi'' :=$ projection onto span$(Range(\Phi^r) \cup \{z\})$, we have

$$\|V - \Pi''(V)\| \leq \|V - \Pi'(V)\| - \alpha.$$

Next, we will consider the actual error $\| V_M^{\mu,r} - V^\mu \|$ and show that it decreases on the average as r is increased when M is large enough. We have the following result.

Theorem 8.2 *$\exists \alpha_0 > 0$ and $M_0 > 0$ such that for $\alpha > \alpha_0$, and $M \geq M_0$,*

$$E[\| V_M^{\mu,r+1} - V^\mu \|] < E[\| V_M^{\mu,r} - V^\mu \|].$$

Proof: Note that because of the finite (M) step termination of the TD update in between two successive updates of the feature matrix, the quantity $V_M^{\mu,r}$ is random for any given $r \geq 0$. The dynamics of $\{V_M^{\mu,r}, r \geq 0\}$ can be described according to

$$V_M^{\mu,r+1} = V_*^{\mu,r+1} + \eta((V_M^{\mu,r}, \xi_{r+1}), \zeta_{r+1}),$$

where:

1. the i.i.d. variables $\{\xi_r\}$ denote the new basis vectors being inducted,
2. ζ_{r+1} captures the random inputs of the algorithm with the fixed basis, between the rth and $(r+1)$th basis change, and,
3. $\eta_{r+1} := \eta((V_M^{\mu,r}, \xi_{r+1}), \zeta_{r+1}) = V_M^{\mu,r+1} - V_*^{\mu,r+1}$ is the error due to inexact convergence (because of the finite run of TD).

Let $\chi_r := V_*^{\mu,r} - \check{V}_r^\mu$. Further, as before, let K be an upper bound on $\| V^\mu \|$. Note that by Corollary 14, Chapter 4 of Borkar [2008], $\exists M_0$ such that for $M \geq M_0$, η_{r+1} can be bounded by a given $\kappa > 0$ with probability $1 - \delta$, $\delta > 0$ prescribed. Let

$$\begin{aligned} A_r &= \{\Psi(\xi_{r+1}, V_M^{\mu,r}) \in B\}, \\ C_r &= \{\|\eta_{r+1}\| \geq \kappa\}. \end{aligned}$$

Then for $\epsilon_1 > P(A_r)$, we have

$$\begin{aligned}
E[\|V_M^{\mu,r+1} - V^\mu\|] &\leq E[\|V_*^{\mu,r+1} - V^\mu\|] + E[\|\eta_{r+1}\|] \\
&\leq E[\|\check{V}_{r+1}^\mu - V^\mu\|] + E[\|\chi_{r+1}\|] + E[\|\eta_{r+1}\|] \\
&\leq E[\|\check{V}_{r+1}^\mu - V^\mu\| I_{A_r}] + E[\|\check{V}_{r+1}^\mu - V^\mu\| I_{A_r^c}] \\
&\quad + E[\|\chi_{r+1}\|] + E[\|\eta_{r+1}\|] \\
&\leq E[\|\check{V}_r^\mu - V^\mu\|] - \alpha + 2\epsilon_1 K + E[\|\chi_{r+1}\|] + E[\|\eta_{r+1}\|] \\
&\leq E[\|V_M^{\mu,r} - V^\mu\|] - \alpha + 2\epsilon_1 K + E[\|\chi_r\|] + E[\|\eta_r\|] \\
&\qquad + E[\|\chi_{r+1}\|] + E[\|\eta_{r+1}\|] \\
&\leq E[\|V_M^{\mu,r} - V^\mu\|] - \alpha + 2((\epsilon_1 + \delta)K + (1-\delta)\kappa + \nu).
\end{aligned}$$

Let $\alpha_0 := 2((\epsilon_1 + \delta)K + (1-\delta)\kappa + \nu)$. Then for $\alpha > \alpha_0$,

$$E[\|V_M^{\mu,r+1} - V^\mu\|] < E[\|V_M^{\mu,r} - V^\mu\|].$$

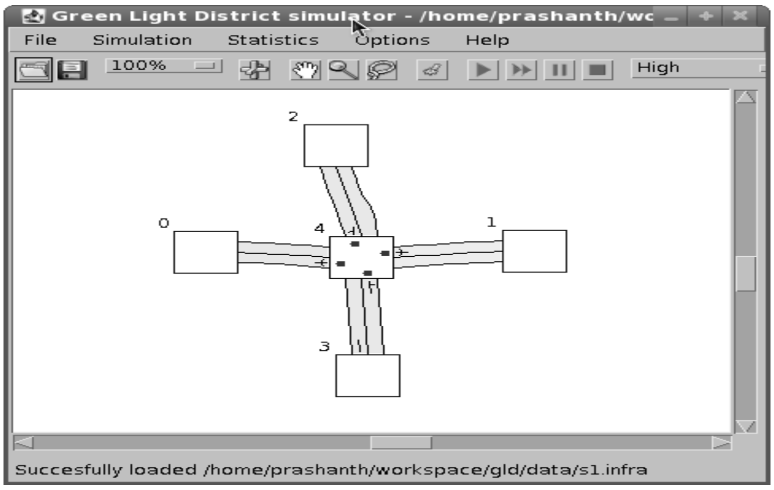
Further, $\alpha > \alpha_0$ will hold if $\delta, \nu, \kappa, \epsilon_1$ are sufficiently small and $V_M^{\mu,r} \notin D$. $\square$

From Theorem 8.2, the average norm difference between the true value function and the approximate value function to which TD(0) converges for a given set of bases decreases as the set of bases is updated after each update of the basis adaptation scheme.

8.4 Application to Traffic Signal Control

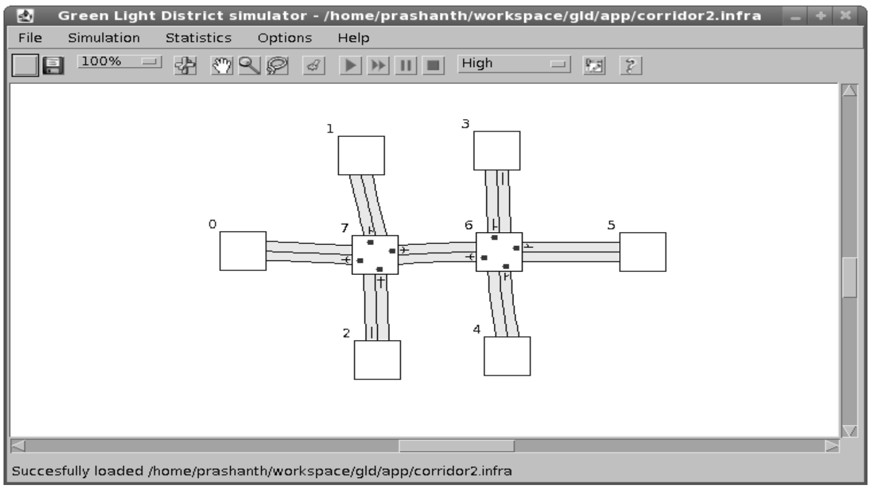
We consider the problem of traffic signal control as an application setting. The aim in general for this problem is to find an optimal policy for switching signals at traffic junctions in a road network so as to maximize flows and minimize delays. The signals that can be switched to green simultaneously form a sign configuration. Q-learning with linear function approximation has been applied for this control problem in Prashanth and Bhatnagar [2011a]. We, however, consider the problem of adaptively tuning the feature matrix when actions are chosen according to a given policy. The policy that we use is obtained using the Q-learning algorithm from Prashanth and Bhatnagar [2011a], i.e., after convergence of the Q-learning algorithm. We apply the feature adaptation scheme described in Section 8.2.

The MDP setting of states, actions and costs is exactly same as in previous chapters.We consider two different road traffic networks: (a) a network with a single traffic signal junction and (b) a corridor with two traffic signal junctions. The two networks are shown in



Succesfully loaded /home/prashanth/workspace/gld/data/s1.infra

Figure 8.1: A Single-Junction Road Network


Green Light District simulator - /home/prashanth/workspace/gld/app/corridor2.infra
File
Simulation
Statistics
Options
Help
100%
High
0
1
2
3
4
5
6
7

Succesfully loaded /home/prashanth/workspace/gld/app/corridor2.infra

Figure 8.2: A Corridor Network with Two Junctions

Figures 8.1 and 8.2, respectively. We implemented these network settings and our algorithm on the green light district (GLD) open source software for road traffic simulations Wiering et al. [2004].

We study the performance of our feature adaptation scheme using estimates of $E[\| V_M^{\mu,r} \|]$. Recall that from Theorem 8.2, the estimates $E[\| V_M^{\mu,r} - V^{\mu} \|]$ diminish with r. The value function V^{μ}, however, is not available, so we use the fact that by the foregoing, $E[\| V_M^{\mu,r} \|]$ will tend to increase. For estimating $E[\| V_M^{\mu,r} \|]$, we use the sample averages of the estimates of $\| V_M^{\mu,r} \|$. As mentioned previously, we let $V_n^r = \Phi^r \theta_n^r$ denote the nth estimate of the value function when the feature matrix is Φ^r. We obtain the aforementioned

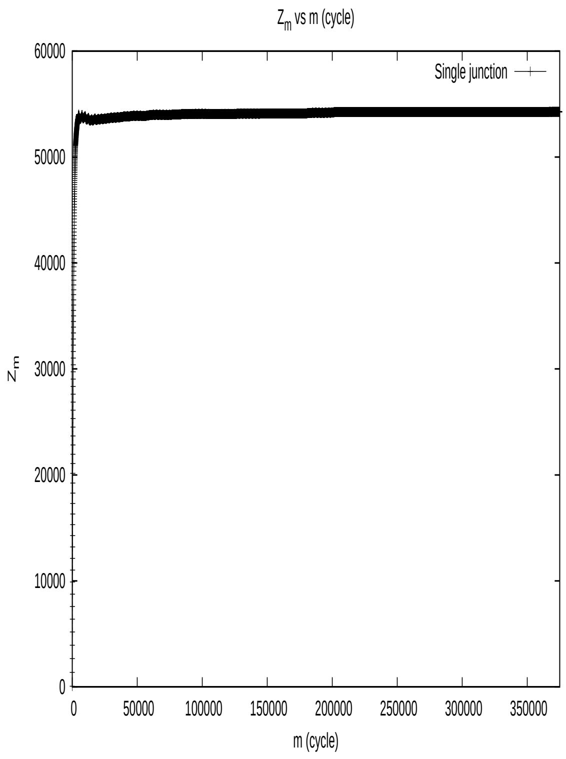


Figure 8.3: Plot of Z_m vs. m in the Case of Single-Junction Road Network

sample averages by running the recursion

$$Z^r_{n+1} = (1-a)Z^r_n + a \parallel V^r_n \parallel,$$

where for any given $r \in \{0, 1, \ldots, R-1\}$, the above recursion is run for M iterations i.e., with $n \in \{0, 1, \ldots, M-1\}$. Next, the value of r is updated and the above procedure repeated. Here a is a small step-size that we select to be 0.001. By abuse of notation, we denote Z^r_n as Z_m in these figures where m denotes the number of cycles or time instants (in absolute terms) i.e., $m \in \{0, 1, \ldots, RM-1\}$, when Z_m is updated.

We call each group of M cycles when the feature matrix Φ^r is held fixed for some r, an episode. We conducted our experiments for the cases of single-junction and two-junction-corridor networks for a total of 150 episodes in each where each episode comprised of 2,500 cycles. Thus $M = 2500$ and $R = 150$ in our experiments.

For the single-junction case, we show in Figure 8.3 the plot of Z_m as a function of m (the number of cycles). We observe that there is a significant improvement after the first episode which results in a steep jump in the Z_m value. In subsequent (r) iterations when the Φ^r matrix is updated, the performance improvement continues, though in smaller

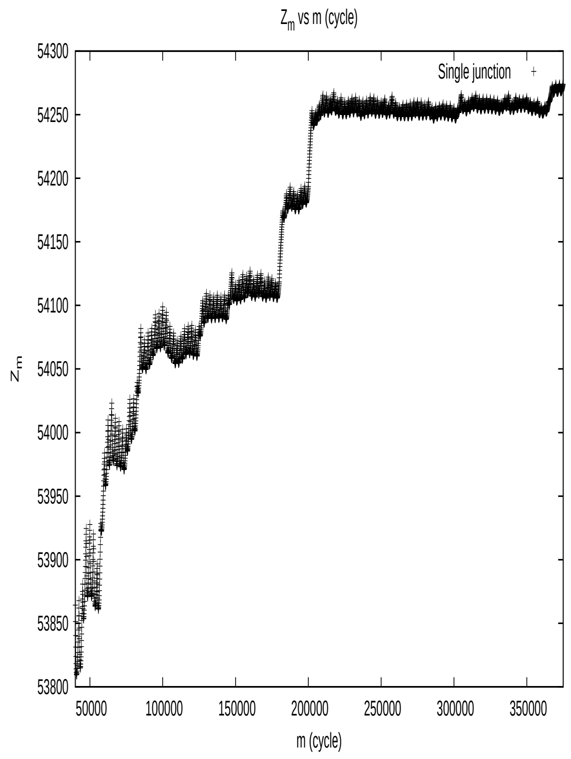


Figure 8.4: Plot of Z_m vs. m after 40,000 Cycles in the Case of Single-Junction Road Network

steps. The improvement in performance from feature adaptation can be seen more clearly in Figure 8.4, when the values of Z_m are plotted for $m \geq 40,000$. The value of Z_m as well as the difference $Z_m - Z_{M-1}$ i.e., improvement in 'Z_m performance' in relation to its value obtained after the completion of the first episode (i.e., with the originally selected feature matrix) is shown at the end of the 30th, 60th, 90th, 120th and 150th episodes respectively, in Table 8.1. As expected, the values of Z_m are seen to consistently increase here.

Next, for the case of the two-junction corridor road network, we show a similar plot of Z_m as a function of m in Figure 8.5. Further, in Figure 8.6, we show the same plot for cycles 40,000 onwards to show performance improvement resulting from feature adaptation. Similar observations as for the single-junction case hold in the case of the two-junction corridor as well.

Finally, as before, we show in Table 8.2, the values of Z_m as well as of the difference $Z_m - Z_{M-1}$ at the end of the 30th, 60th, 90th, 120th and 150th episodes respectively. The values of Z_m are again seen to consistently increase here as well.

Table 8.1: Performance Improvement with Feature Adaptation for the Single-Junction Road Network

# Cycle (m)	Z_m	$Z_m - Z_{M-1}$, ($m \geq M-1$)
2499	51042.23	
74999	54003.00	2960.76
149999	54116.59	3074.36
224999	54260.28	3218.05
299999	54255.38	3213.15
374999	54274.72	3232.49

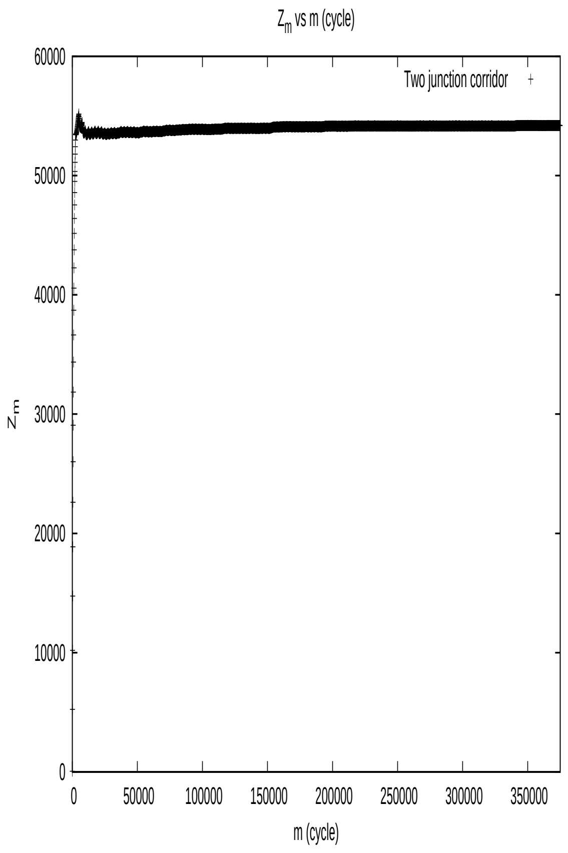


Figure 8.5: Plot of Z_m vs. m in the Case of Two-Junction Corridor Network

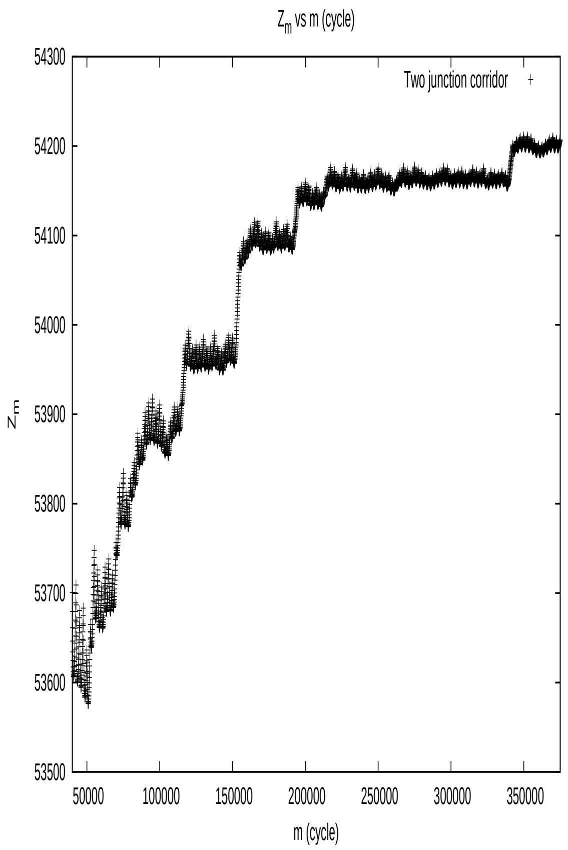


Figure 8.6: Plot of Z_m vs. m after 40,000 Cycles in the Case of Two-Junction Corridor Network

Table 8.2: Performance Improvement with Feature Adaptation for the Two-Junction Corridor Road Network

# Cycle (m)	Z_m	$Z_m - Z_{M-1}$ ($m \geq M-1$)
2499	53480.65	
74999	53834.04	353.39
149999	53985.54	504.89
224999	54166.69	686.04
299999	54167.99	687.34
374999	54207.78	727.13

8.5 Summary

We presented in this chapter a novel online feature adaptation algorithm. We observed significant performance improvements on two different settings for a problem of traffic signal control. We considered the problem of prediction here and applied our feature adaptation scheme in conjunction with the temporal difference learning algorithm. As future work, one may consider the application of our algorithm together with other schemes such as least squares temporal difference (LSTD) learning (Bradtke and Barto [1996], Boyan [1999]) and least squares policy evaluation (LSPE) (Nedic and Bertsekas [2003], Bertsekas and Nedic [2004]). Moreover, one may apply a similar scheme for the problem of control, for instance, in conjunction with the actor-critic algorithms in Konda and Tsitsiklis [2003] and Bhatnagar et al. [2009a].

Part II

Service Systems

This part discusses stochastic optimization algorithms for solving the problem of optimizing labor cost by adapting staffing levels in a service system. The core of each of the algorithms is a multi-timescale stochastic approximation scheme that incorporates a random perturbation based algorithm for 'primal descent' and couples it with a 'dual ascent' scheme for the Lagrange multipliers. The first order algorithms (Prashanth et al. [2011b], Prasad et al. [2012]) proposed use simultaneous perturbation stochastic approximation (SPSA) based techniques for gradient estimation in the primal. In particular, we developed algorithms that use SPSA with deterministic perturbations based on certain Hadamard matrices. We also developed two second order (Newton) methods (Prashanth et al. [2012b]). The first algorithm estimates the Hessian of the objective function using SPSA and involves an explicit inverse, while the second one is a novel algorithm that leverages Woodbury's identity to directly estimate the inverse of the Hessian and hence achieve compututational efficiency. These algorithms were found to perform significantly better than the state-of-the-art OptQuest optimization toolkit, which is being used in IBMs pool simulation project.

Chapter 9

Adaptive labor staffing

A *Service System (SS)* is an organization composed of (i) the resources that support, and (ii) the processes that drive service interactions so that the outcomes meet customer expectations (Alter [2008], Spohrer et al. [2007], Ramaswamy and Banavar [2008]). This chapter focuses on the SS in the data-center management domain, where customers own data centers and other IT infrastructures supporting their businesses. Owing to size, complexity, and uniqueness of these technology installations, the management responsibilities of the same are outsourced to specialized service providers. A *delivery center* is a remotely-located workplace from where the service providers manage the data-centers. Each *service request (SR)* that arrives at a delivery center requires a specific skill and is supported by a *service worker (SW)* with the corresponding skill set. The SWs work in shifts which are typically aligned to the business hours of the supported customers. Hence, a group of customers supported by a group of SWs, along with the operational model of how SRs are routed constitutes an SS in this chapter. A delivery center may consist of many SS.

In this chapter, we consider the problem of adaptive labor staffing in the context of service systems. The objective is to find the optimal staffing levels in a SS for a given dispatching policy (i.e., a map from service requests to service workers) while maintaining system steady-state and compliance to aggregate service level agreement (SLA) constraints. The staffing levels constitute the worker parameter that we optimize and specify the number of workers in each shift and of each skill level. The SLA constraints specify the target resolution time and the aggregate percentage for an SR originating from a particular customer and with a specified priority level. For instance, a sample SLA constraint could specify that 95% of all SRs from customer 1 with 'urgent' priority must be resolved within 4 hours. While the need for SLA constraints to be met is obvious, the requirement for having queues holding unresolved SRs bounded is also necessary because SLA attainments are calculated only for the work completed. The problem is challenging because analytical modeling of SS operations is difficult due to aggregate SLA constraints and also because the SS characteristics such as work patterns, technologies, and customers supported change frequently. An important aspect to consider in the design of the adaptive labor staffing algorithm is its computational efficiency. A truly adaptive labor staffing algorithm is required to have a low computational time requirement, which in turn helps in making staffing changes on a shorter timescale, for instance, every week.

We formulate this problem as a constrained hidden Markov cost process (Marbach and Tsitsiklis [2001] define an analogous Markov reward framework, i.e., in the unconstrained setting for a continuous parameter) that depends on the worker parameter. To have a sense of the search space size, an SS consisting of 30 SWs who work in 6 shifts and 3 distinct skill levels corresponds to more than 2 trillion configurations. Apart from the high cardinality of the discrete parameter set, the constrained Markov cost process involves a hidden or un-observed state component. We design a novel single stage cost function for the constrained Markov cost process in a way that balances the conflicting objectives of worker under-utilization and SLA under/over-achievement. The performance objective is a long-run average of this single stage cost function and the goal is to find the optimum steady state worker parameter (i.e., the one that minimizes this objective) from a discrete high-dimensional parameter set. However, our problem setting also involves constraints relating to queue stability and SLA compliance. Thus, the optimum worker parameter is in fact a constrained minimum. Another difficulty in finding the optimum (constrained) worker parameter is that the single-stage cost function can be estimated only via simulation. Hence, the need is for a search algorithm that incrementally updates the worker parameter along a descent direction, while adhering to a set of queue stability and SLA constraints.

The rest of this chapter is organized as follows: In Section 9.1, we survey relevant literature in service systems as well as stochastic optimization. In Section 9.2, we introduce the framework of service systems. In Section 9.3, we present the detailed problem formulation. In the following chapters of this thesis, we develop novel discrete-optimization algorithms based on SPSA to solve the above problem and then, empirically establish their usefulness by comparing against a state-of-the-art optimization toolkit OptQuest on five real-life SS.

9.1 Literature Survey

We now review literature in two different areas of related work: (1) techniques pertaining to service systems analysis and (2) developments in stochastic optimization approaches.

9.1.1 Service Systems

In Verma et al. [2011], a two step mixed-integer program is formulated for the problem of dispatching SRs within service systems. While their goal is similar to ours, their formulation does not model the stochastic variations in arrivals and service times. Further, unlike our framework, the SLAs in their formulation are not aggregated over a month long period. In Wasserkrug et al. [2008], the authors propose a scheme for shift-scheduling in the context of third-level IT support systems. Unlike this chapter, they do not validate their method against data from real-life third-level IT support. In Cezik and L'Ecuyer [2008] as well as Bhulai et al. [2008], simulation-optimization methods are proposed for finding the optimal staffing in a multiskill call center. While Cezik and L'Ecuyer [2008] propose a cutting plane algorithm for solving an integer program, Bhulai et al. [2008] rely on obtaining a linear programming solution. However, unlike SASOC algorithms, steady-state system analysis is not performed there. Instead, they consider only a single iteration of the system. Further, the SLA constraints considered there are not aggregate in nature. Also, the algorithms proposed in Cezik and L'Ecuyer [2008], Bhulai et al. [2008] are heuristic in nature, whereas the SASOC algorithms are provably convergent. In Chan [2008], the emergent behavior of a service system consisting of a large number of cells is studied by applying an agent based simulation method. Each cell contains an analytical M/M/1 queue model. The simulation helps observe how cells die and neighborhood patterns emerge among cells. While Chan [2008] exemplifies the human aspects of service systems which would be an important future direction for our work, it does not aim to propose a labor-optimization technique. In Robbins and Harrison [2008], usage of a simulation based search method is proposed for finding the optimal staffing levels in the context of a call-center domain. They evaluate the system given a staffing level with an analytical model, which is possible in their simplified domain but would not be feasible for service systems due to aggregate SLA constraints and dynamic queues. They apply simulation to search for the optimal staffing level based on a heuristic. An analysis of service systems using the ARENA simulation tool is presented in Brickner et al. [2010]. Unlike our model, the system there is not subjected to aggregate SLA constraints. Also, in their work, they do not consider preemption of low priority SRs by higher priority SRs and assignment of higher skilled SWs to growing queues of SRs requiring lower skill levels. Their model assumes a pool of dedicated SWs for each customer as well as a separate pool of shared SWs. In Banerjee et al. [2011], a simulation framework for evaluating dispatching policies is proposed. A scatter search technique is used to search over the space of SS configurations and optimize the staff allocation. While we share their simulation model, the goal in this chapter is to propose a stochastic iterative scheme in the framework of constrained hidden Markov cost processes and prove their convergence to the optimal staffing levels in a SS. In general, none of the above papers propose an optimization algorithm that is geared for SS and that leverages simulation to adapt optimization search parameters, when both the objective and the constrained functions are suitable long-run averages.

In Prashanth et al. [2011a], an algorithm based on SPSA was proposed for the problem of staffing optimization in service systems. While our SASOC-G algorithm is similar to the one proposed in Prashanth et al. [2011a], a notable difference is that SASOC-G algorithm incorporates a novel generalized projection scheme unlike Prashanth et al. [2011a]. Further, we rigorously specify the constrained hidden Markov cost process framework along with a detailed proof of convergence of all SASOC algorithms (including SASOC-G) and also provide detailed experimental results for two different dispatching policies under both flat as well as bursty SR arrival processes.

9.1.2 Stochastic Optimization

A popular and highly efficient simulation based local optimization scheme for gradient estimation is Simultaneous Perturbation Stochastic Approximation (SPSA) proposed by Spall [1992]. SPSA is based on the idea of randomly perturbing the parameter vector using i.i.d., symmetric, zero-mean random variables. This scheme has the critical advantage that it needs only two samples of the objective function to estimate the gradient for any N-dimensional parameter. In Spall [1997], a one-simulation variant of SPSA was proposed. However, the algorithm in Spall [1997] was not found to work as well in practice as its two simulation counterpart. Usage of deterministic perturbations instead of randomized ones was proposed in Bhatnagar et al. [2003]. The deterministic perturbations there were based either on lexicographic or Hadamard matrix based sequences and were found to perform better than their randomized perturbation counterparts. In particular, the one-simulation variant of the SPSA algorithm that is based on Hadamard matrices was found to significantly perform better than the one-simulation random-perturbation algorithm presented in

Spall [1997]. A Newton based SPSA algorithm that needs four system simulations with Bernoulli random perturbations was proposed in Spall [2000]. In Bhatnagar [2005], three new SPSA based estimates of the Hessian that require three, two and one system simulations, respectively, were proposed. In Bhatnagar [2007], certain smoothed functional (SF) Newton algorithms that incorporate Gaussian-based perturbations were proposed. The algorithms of Spall [2000], Bhatnagar [2005] and Bhatnagar [2007] were for unconstrained optimization. In Bhatnagar et al. [2011b], continuous optimization techniques such as SPSA and SF, have been adapted to a setting of discrete parameter optimization. Two simulation based optimization algorithms that involve randomized projections have been proposed there for an unconstrained setting. In Bhatnagar et al. [2011a], several simulation based algorithms for constrained optimization have been proposed. Two of the algorithms proposed there use SPSA for estimating the gradient, after applying the Lagrange relaxation procedure to the constrained optimization problem, while the other two incorporate smoothed functional approximation. Constrained optimization in the context of Markov decision processes has been considered, for instance, in Borkar [2005] and Bhatnagar [2010]. The algorithm proposed in Borkar [2005] is a three time-scale stochastic approximation scheme that incorporates an actor-critic algorithm for primal descent and performs dual ascent on Lagrange multipliers. However, it assumes full state representation for the underlying MDP. The algorithm proposed in Bhatnagar [2010] combines the ideas of multi-timescale stochastic approximation and reinforcement learning with function approximation to develop a simulation based online algorithm for a constrained control problem, involving function approximation. For a survey of gradient estimation techniques in the context of simulation optimization, the reader is referred to Henderson and Nelson [2006].

Our SASOC algorithms differ from the stochastic optimization approaches outlined above in various ways. Many algorithms, for instance those proposed in Spall [2000] and Bhatnagar [2005, 2007], are for unconstrained optimization and in a continuous optimization setting. However, our staff optimization problem is for a discrete worker parameter and requires SLA and queue stability constraints to be satisfied. While the algorithms of Bhatnagar et al. [2011b] have been proposed for a discrete optimization setting, our SASOC algorithms differ in the projection technique used and more importantly solve an optimization problem under functional (inequality) constraints. While the algorithms of Bhatnagar et al. [2011a] have been developed for continuous constrained optimization, our SASOC algorithms optimize a discrete parameter and also differ in the dual ascent procedure, as we do not accumulate the long-run averages of the constraint functions. Further, unlike the Newton-based methods of Bhatnagar et al. [2011a] which estimate the Hessian and perform an explicit matrix inverse, our SASOC-W algorithm directly tunes the inverse of the Hessian matrix by employing a procedure based on the Woodbury's identity. Thus, the computational complexity of SASOC-W is significantly lower than its Hessian based counterparts in Bhatnagar et al. [2011a]. To the best of our knowledge, we are the first to present adaptations of Newton-based search approaches for constrained discrete optimization. Simulation-based optimization methods in a Markov reward process framework have been proposed in Marbach and Tsitsiklis [2001]. However, these have been proposed for a continuous-valued parameter in a completely observed state setting and further, do not involve constraints. Our problem setting, on the other hand, involves un-observed or hidden state components and falls within the purview of constrained hidden Markov models.

We now briefly summarize our contributions in the context of adaptive labor staffing, which forms the content of this as well as Chapters 10 and 11.

Our contributions

1. We present three novel discrete parameter simulation optimization methods based on simultaneous perturbation stochastic approximation techniques for adaptive labor staffing in service systems. The first algorithm, referred to as SASOC-G, is a first order method that uses SPSA based gradient estimate in the primal. The second and third algorithms, referred to as SASOC-H and SASOC-W, respectively, are second-order Newton methods. To the best of our knowledge, this is the first work that develops Newton-based algorithms for constrained discrete parameter optimization.

2. All the three SASOC algorithms that we propose are online, incremental and easy to implement. Further, they possess the necessary convergence properties.

   (a) We combine the approaches discussed in Bhatnagar et al. [2011a] for continuous parameter constrained optimization as well as in Bhatnagar et al. [2011b] for discrete parameter unconstrained optimization to obtain the aforementioned three new algorithms for constrained discrete optimization. Unlike Bhatnagar et al. [2011a] where an explicit inversion of the Hessian at each update step was advocated, we incorporate the Woodbury's identity to obtain a novel update step for the inverse of the Hessian in our algorithm SASOC-W.

(b) Further, unlike Bhatnagar et al. [2011b] where fully randomized projections were used, we incorporate a generalized projection operator that is continuously differentiable in the parameter and works as a deterministic operator over a large portion of the search space and incorporates randomization over a small portion. This helps in bringing down the computational requirement as a deterministic projection scheme requires less computation than a fully randomized one.

3. We evaluate our algorithms on five real-life SS in the data-center management domain. For each of the SS, we collect operational data on work arrival patterns, service times, and contractual SLAs and feed this data into the simulation model of Banerjee et al. [2011].

4. From the simulation experiments, we observe that our algorithms show overall better performance in comparison with the state-of-the-art OptQuest optimization toolkit (Laguna [1998]). Further, our algorithms are 25 times faster than OptQuest and have a significantly lower execution runtime.

9.2 Service System Framework

Formally, a service system is characterized by the following entities.

A set of customers, denoted by $\mathcal{C}$, supported by the service system.

A set of shifts, denoted by $\mathcal{A}$, across which the service workers are distributed.

A set of skill or complexity levels, denoted by $\mathcal{B}$.

A set of priority levels, denoted by P.

A set of time intervals, denoted by $\mathcal{I}$, where during each interval the arrivals stay stationary, with the number of arrivals following a Poisson distribution whose rate parameter is given by the function α described next.

Arrival rates specified by the mapping $\alpha : \mathcal{C} \times \mathcal{I} \to \mathbb{R}$. We assume that each of the SR arrival processes from the various customers C_i are independent and Poisson distributed with $\alpha(C_i, I_j)$ specifying the rate parameter.

Service time distributions characterized by the mapping $\tau : P \times \mathcal{B} \to (r_1, r_2), r_i \in \mathbb{R}, i = 1, 2$. Here r_1 represents the mean and r_2 the standard deviation of a truncated log-normal distributed random variable corresponding to a particular priority-complexity pair. In other words, if M is a random variable following a normal distribution with mean r_1 and standard deviation r_2, then the truncated lognormal random variable is $e^M \wedge \top$, where $\top$ is a truncation constant that is chosen to be large in practice.

SLA constraints, given by the mapping $\gamma : \mathcal{C} \times P \to (r_1, r_2), r_i \in \mathbb{R}, i = 1, 2$. Here $\gamma(C_i, P_j) = (r_1, r_2)$ means that the SLA target for SRs from customer C_i and with priority P_j is (r_1, r_2), with r_1 specifying the SLA percentage target and r_2 the resolution time target (in hours). For instance, $\gamma(C_1, P_1) = (95, 4)$ translates to the requirement that at least 95% of the SRs from customer C_1 with priority level P_1 should be closed within 4 hours. Note that the SLAs are computed at the end of each month and hence the aggregate SLA targets are applicable to all SRs that are closed within the month under consideration.

Note that each arriving SR has a customer identifier ($\in \mathcal{C}$) and a priority identifier ($\in P$) and any SW works in a particular shift ($\in \mathcal{A}$) and possesses a skill level ($\in \mathcal{B}$). In other words, each customer can issue multiple SRs with their respective SLA targets and the SWs with the right skill level and relevant shift have to pull these SRs from the complexity queues and close them within the deadline specified by the SLA. The set $\mathcal{I}$ and the mapping α allow us to model the variations in arrival rates better than in a setting where the arrivals are assumed to be Poisson-distributed for the entire period. We assume that the number of arrivals during any interval ($\in \mathcal{I}$) is upper-bounded by a sufficiently large constant. Further, the time taken by a SW to complete an SR is stochastic and follows a lognormal distribution, where the parameters of the distribution are learned by conducting time and motion exercises described in Banerjee et al. [2011].

In the following we provide the specifics of the simulation framework.

- $\mathcal{I}$ contains one element for each hour of the week. Hence, $|\mathcal{I}| = 168$. Each time interval is one hour long.
- P=$\{P_1, P_2, P_3, P_4\}$, where, $P_1 > P_2 > P_3 > P_4$.
- $\mathcal{B}$=$\{$High, Medium, Low$\}$, where, High $>$ Medium $>$ Low.
- **Swing**: Swing is invoked when Low queue length> 10. When this happens, one SW is

randomly chosen with skill level Medium and the SW is assigned SRs from Low queue until Low queue length< 10.

- **Preemption**: Preemption relation $\Rightarrow$ is the transitive closure of the tuples $P_1 \Rightarrow P_2$, $P_2 \Rightarrow P_3$, and $P_2 \Rightarrow P_4$.
- **Breaks**: Each SW takes 3 breaks in a shift independently of other SWs with a total duration of 75 minutes. The maximum priority SRs preemptible by breaks is P_3. In other words, breaks are modeled as internal SRs with priority P_2.
- **On-call**: Minimum priority above which SRs arriving out of shift hours are routed to on-call SW is P_2. Hence, SRs with priorities P_3 or P_4 would wait till the next shift if they arrive in non-shift hours.
- **Infrastructure down**: The mean-time-between-failures of work infrastructure is 178 hours and the mean down time is 104 minutes (based on historical data). Further, the time between failures and down-times is distributed exponentially.

Table 9.1(a) illustrates a simple SS configuration, specifying the staffing levels across shifts and skill levels. This essentially constitutes the worker parameter that we optimize. In this example, $\mathcal{A} = \{\text{S1, S2, S3}\}$ and $\mathcal{B} = \{\text{high, medium, low}\}$. Tables 9.1(b) and 9.2(b) provide sample utilizations and SLA attainments on a SS with three shifts, two customers and four priority levels. Table 9.2(a) illustrates the target SLA requirements for the same SS.

Table 9.1: Sample workers and utilizations

(a) Workers θ_i

	Skill levels		
Shift	High	Med	Low
S1	1	3	7
S2	0	5	2
S3	3	1	2

(b) Utilizations $u_{i,j}$

	Skill levels		
Shift	High	Med	Low
S1	67%	34%	26%
S2	45%	55%	39%
S3	23%	77%	62%

Figure 9.1 shows the main components of the SS. The SRs arrive from multiple customers and the arrival rate is specific to the hour of week, i.e., within each hour of week, and for each customer-priority pair, the arrivals follow a Poisson distribution. The parameters of this distribution are learned from historical data over a period of at least 6 months. Once the SR arrives, it is queued up in a matching complexity queue by the queue manager and

Table 9.2: Sample SLA constraints

(a) SLA targets $\gamma_{i,j}$

	Customers	
Priority	Bossy Corp	Cool Inc
P_1	95%4h	89%5h
P_2	95%8h	98%12h
P_3	100%24h	95%48h
P_4	100%18h	95%144h

(b) SLA attainments $\gamma'_{i,j}$

	Customers	
Priority	Bossy Corp	Cool Inc
P_1	98%4h	95%5h
P_2	98%8h	99%12h
P_3	89%24h	90%48h
P_4	92%18h	95%144h

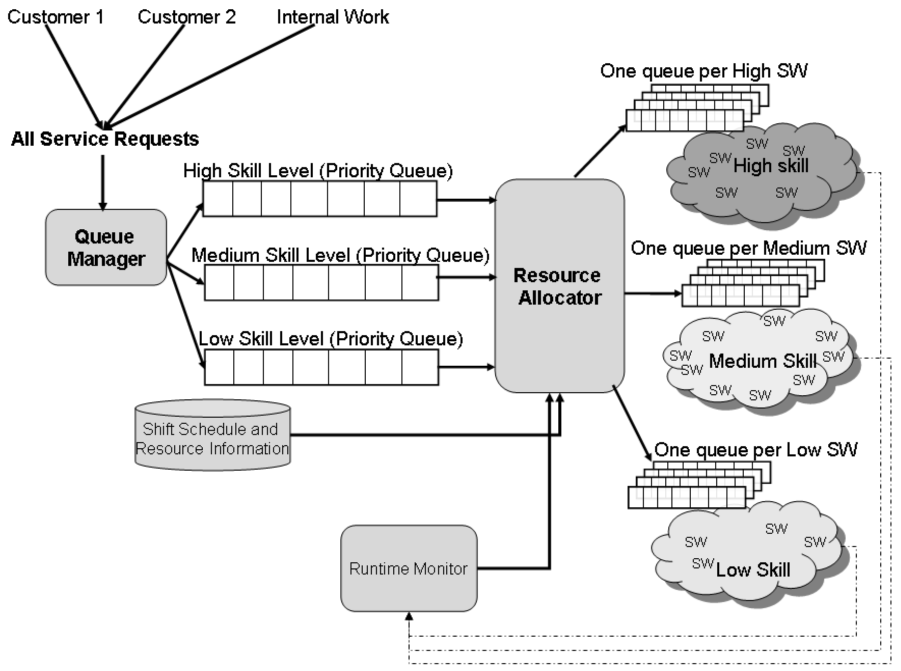


Figure 9.1: Operational model of an SS

the dispatcher would then assign it to a SW based on the dispatching policy. For instance, in the PRIO-PULL policy, SRs are queued in the complexity queues based directly on the priority assigned to them by the customers. On the other hand, in the EDF policy, the time left to SLA target deadline is used to assign the SRs to the SWs i.e., the SW works on the SR that has the earliest deadline. Note that we have a finite buffer system, i.e., the number of SRs in each of the complexity queues is upper-bounded by a sufficiently large constant. Any arriving SR that finds the corresponding complexity queue full will depart the system.

A SW works in exactly one shift (working days and times) and a SS may operate in multiple shifts. We say that a particular configuration of workers across shifts and skill levels is feasible if (a) the SLA constraints are met and (b) the complexity queues do not become unbounded when using this configuration. While the need for (a) is obvious, the requirement for having bounded complexity queues is also necessary. This is because SLA attainments are calculated only for work completed and not for that waiting for completion in the complexity queues. For instance, say in a given month, 100 SRs arrive at various times from a customer to a SS and only 50 of them are completed within the target completion time stipulated by the SLA constraints. The remaining 50 SRs are still in progress without a known completion time and hence do not have an impact on the SLA attainment measures. Hence, a healthy SLA attainment alone is insufficient and the bound on the growth of complexity queues fills the gap.

9.3 Constrained parameterized hidden Markov Cost Process

In this section, we describe our problem framework. A similar framework is considered for instance in Marbach and Tsitsiklis [2001], except that the setting considered there is unconstrained and the parameter is continuous-valued. Moreover, in Marbach and Tsitsiklis [2001], the state is fully observable and rewards (and not costs) are considered. Our setting, however, involves a discrete-time, continuous-space hidden Markov process represented by $\{(X_n, Y_n), n \geq 0\}$. The idea here is that whereas $\{(X_n, Y_n)$ is Markov for a given parameter θ that we describe below, the portion X_n of this process is observed while Y_n is hidden. We describe X_n and Y_n more clearly in Section 9.3.2. The transition probabilities of this process depend on the worker parameter $\theta = (\theta_1, \ldots, \theta_N)^T \in \mathcal{D}$, where $N = |A| \times |B|$. In the above, θ_i indicates the number of service workers whose skill level is $(i-1)\%|B|$ and whose shift index is $(i-1)/|B|$, with the indices into the set B starting from 0, i.e., $i \in \{0, 1, \ldots, |B|-1\}$. As an example, the worker parameter for the setting in Table 9.1(a) is $\theta = (\theta_1, \ldots, \theta_9)^T = (1, 3, 7, 0, 5, 2, 3, 1, 2)^T$. The parameter vector θ takes values in the set $\mathcal{D}$, where $\mathcal{D} \stackrel{\triangle}{=} \{0, 1, \ldots, W_{\max}\}^N$. Here $W_{\max}$ is an upper bound for the number of workers in any shift and of any skill level. Note that one can enumerate all the points in $\mathcal{D}$ as $\mathcal{D} = \{D^1, D^2, \ldots, D^p\}$ for some $p > 1$.

As illustrated in Fig 9.2, the system stochastically transitions from one state to another, while incurring a state-dependent cost. In addition, there are state-dependent single-stage (constraint) functions described via $g_{i,j}(X_n), h(X_n), i = 1, \ldots, |C|, j = 1, \ldots, |P|$. These shall correspond to the SLA and queue stability constraints. Note that these functions depend explicitly on only the observed part X_n of the state process $(X_n, Y_n), n \geq 0$. The state

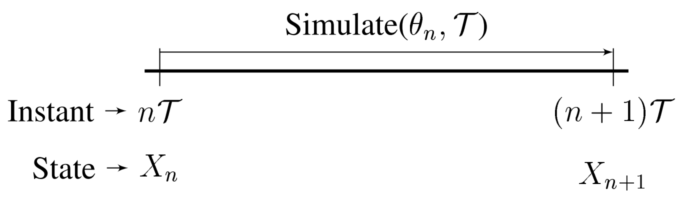


Figure 9.2: A portion of the time-line illustrating the evolution of the process

together with the cost and constraint functions constitutes the constrained hidden Markov cost process. The nth system transition of this underlying process involves a simulation of the service system for a fixed period $\mathcal{T}$ with the current worker parameter θ_n. For instance, in our experiments, $\mathcal{T} = 10$, i.e., we simulate the service system for a period of ten months with the staffing levels specified by θ_n. Also, note that this is a continuously running simulation, where at discrete time instants $n\mathcal{T}$ we update the worker parameter θ_n and the simulation output causes a probabilistic transition from the current state (X_n, Y_n) to the next state (X_{n+1}, Y_{n+1}), while incurring a single stage cost $c(X_n)$. The precise definitions of the state, the cost and the constraint functions are given in Section 9.3.2. By an abuse of notation, we refer to the state at instant $n\mathcal{T}$ as (X_n, Y_n).

9.3.1 The Objective

We use the long-run average cost as the performance objective in our setting. Thus, we are interested in optimizing the steady-state system performance. The optimization problem is the following:

$$\boxed{\begin{aligned}
&\text{Find } \min_{\theta} J(\theta) \triangleq \lim_{n\to\infty} \tfrac{1}{n} \sum_{m=0}^{n-1} c(X_m)\\
&\text{subject to}\\
&G_{i,j}(\theta) \triangleq \lim_{n\to\infty} \tfrac{1}{n} \sum_{m=0}^{n-1} g_{i,j}(X_m) \leq 0,\\
&\qquad\qquad\qquad \forall i = 1,\ldots,|C|, j = 1,\ldots,|P|,\\
&H(\theta) \triangleq \lim_{n\to\infty} \tfrac{1}{n} \sum_{m=0}^{n-1} h(X_m) \leq 0.
\end{aligned} \qquad (9.1)}$$

We assume below that the Markov process $\{(X_n, Y_n)\}$ under any parameter θ is ergodic. In such a case, the limits in (9.1) are well-defined. If this is not the case, one may replace the "lim" with "limsup" in the definitions of $J(\theta), G_{i,j}(\theta)$ and $H(\theta)$ in (9.1). Given the above constrained Markov cost process formulation, the optimization problem (9.1) essentially stipulates that the optimal worker parameter θ^* should minimize the long-run average cost objective $J(\cdot)$ while maintaining queue stability in steady-state (i.e., the long-run average of $h(X_n)$ should not be above zero) and adhering to contractual SLAs, i.e., that the long-run average of $g_{i,j}(X_n)$ should not be above zero as well, for any feasible (i,j)-tuple.

The SASOC algorithms that we design subsequently (see Section 10.2) use the cost $c(X_n)$ and constraint functions $g_{i,j}(X_n)$ and $h(X_n)$ to tune the worker parameter θ_n at instant $n\mathcal{T}$, and the system simulation would then continue with the updated worker parameter. While it is desirable to find the optimum $\theta^* \in S$, i.e.,

$$\boxed{\theta^* = \text{argmin}\left\{J(\theta) \text{ s.t. } \theta \in \mathcal{D}, G_{i,j}(\theta) \leq 0, i = 1,\ldots,|C|, j = 1,\ldots,|P|, H(\theta) \leq 0\right\},}$$

it is in general very difficult to achieve a global minimum. We apply the Lagrange relaxation procedure to the above problem and then provide SPSA based algorithms - both first as well as second order, for finding a locally optimum parameter θ^*. We now describe in detail the state, single stage cost and constraint functions that we adopt for the constrained hidden Markov cost process formulated for optimizing the staffing in the context of service systems.

9.3.2 State, Cost and Constraints

The observed part X_n of the state at instant n is the vector of the lengths of waiting SR queues corresponding to each skill level, the current utilization of workers for each shift and skill level, and the current SLA attainments for each customer and SR priority. The un-observed or hidden part Y_n is the vector of residual service times, i.e., the pending time to service completion of SRs which are being worked upon by individual SWs. Thus,

$$X_n = (\mathcal{N}(n), u(n), \gamma'(n), q(n)), \tag{9.2}$$
$$Y_n = (\mathcal{R}(n)), \tag{9.3}$$

where,

- $\mathcal{N}(n)$ is a vector of complexity queue lengths and is defined as $\mathcal{N}(n) = (\mathcal{N}_1(n),\ldots,\mathcal{N}_{|B|}(n))$, with $\mathcal{N}_i(n)$ denoting the complexity queue length corresponding to the skill level i, where $i = 1, \ldots, |B|$. For instance, $\mathcal{N}_0(n)$ would give the length of the complexity queue housing 'low' complexity SRs and $\mathcal{N}_1(n), \mathcal{N}_2(n)$ would give the corresponding numbers for 'medium' and 'high' complexity SRs. Note that since all the complexity queues are of finite size, we have $\mathcal{N}_i(n) \leq \varsigma, i = 1, \ldots, |B|$, where $\varsigma > 0$ is a sufficiently large constant.

- $\mathcal{R}(n)$ is a vector of residual service times and is defined as $\mathcal{R}(n) = (\mathcal{R}_{1,1,1}(n), \ldots, \mathcal{R}_{1,1,W_{\max}}(n), \ldots, \mathcal{R}_{|A|,|B|,W_{\max}}(n))$, where $\mathcal{R}_{i,j,k}(n)$ denotes the residual service time of the SR currently being processed at instant $n\mathcal{T}$ by the kth worker in shift i and of skill level j. By convention, we set $\mathcal{R}_{i,j,k}(n) = \kappa$ if there is no worker corresponding to the shift i, skill level j and index k with this shift-skill combination. Here κ is a special value used to signify the absence of a worker in a particular shift-skill level combination.

- The utilization vector $u(n) = (u_{1,1}(n), \ldots, u_{|A|,|B|}(n))$, where each $u_{i,j}(n) \in [0, 1]$ is the average utilization of the workers in shift i and skill level j, at instant n.

- The SLA attainment vector $\gamma'(n) = (\gamma'_{1,1}(n), \ldots, \gamma'_{|C|,|P|}(n))$, where $\gamma'_{i,j}(n) \in [0, 1]$ denotes the SLA attainment for customer i and priority j, at instant n.

- $q(n)$ is an indicator variable that denotes the queue feasibility status of the system at instant n. In other words, $q(n)$ is 0 if the growth rate of the SR queues (for each complexity) is beyond a threshold and is 1 otherwise. We need $q(n)$ to ensure system steady-state which is independent of SLA attainments because the latter are computed only on the SRs that were completed and not on those queued up in the system.

Remark 7 *Let S denote the state space. We observe that S is a compact set. This is because each of the state components in X_n and Y_n are closed and bounded. In particular, each element of $u(n)$, $\gamma'(n)$ takes values in $[0, 1]$ and $0 \leq q(n) \leq 1$. The system SR queues $\mathcal{N}$ are also of finite length and hence, X_n is bounded. The residual time vector in Y_n also takes values in a compact set in lieu of the fact that each element of $\mathcal{R}$ is upper bounded by the total service time at the SR queues and that in turn takes values in $[0, \top]$.*

Remark 8 *In our setting, the service times follow a truncated log-normal distribution. The residual service time at any point cannot be precisely estimated in a real system. Hence, $\mathcal{R}(n)$ is the hidden state component for the Markov process. Strictly speaking, the setting falls into the realm of hidden Markov process. However, the cost function $c(\cdot)$ and other* *constraint functions of X_n formulated later in this section do not depend on the value of $\mathcal{R}(n)$ and instead work with the observable components of the state. Further, the algorithms we propose aim to minimize a long average of the cost function, while satisfying the constraints (also long-run averages) and hence, do not require $\mathbb{R}(n)$ in their update rules.*

Considering that the queue lengths, utilizations and SLA attainments at instant $n+1$ depend only on the state at instant n, we observe that $\{(X_n(\theta), Y_n(\theta)), n \geq 0\}$ is a constrained hidden Markov cost process for any given (fixed) parameter θ. We now describe the single stage cost function, whose long-run average sum we try to optimize in (9.1). We let the cost function $c(X_n)$ have the form:

$$\boxed{c(X_n) = r \times \left(1 - \sum_{i=1}^{|A|} \sum_{j=1}^{|B|} \alpha_{i,j} \times u_{i,j}(n)\right) + s \times \left(\frac{\sum_{i=1}^{|C|} \sum_{j=1}^{|P|} |\gamma'_{i,j}(n) - \gamma_{i,j}|}{|C| \times |P|}\right), \tag{9.4}}$$

where $r, s \geq 0$ and $r + s = 1$. Further, $0 \leq \gamma_{i,j} \leq 1$ denotes the contractual SLA for customer i and priority j. The single stage cost function here can be seen to be bounded. In fact, from (9.4), we observe that $0 \leq c(X_n) \leq 1$. This is because $u_{i,j}(n), \gamma_{i,j}, \gamma'_{i,j}(n) \in [0,1]$ and each component in (9.4) is upper-bounded by 1.

The cost function is designed to balance between the conflicting objectives of maximizing the utilization of workers and meeting the SLA requirements simultaneously. By the first component in (9.4), we seek to minimize the under-utilization of workers as it is more fine-grained and hence, allows tighter minimization in comparison to minimizing just the sum of workers across shifts and skill levels. The second component in (9.4) represents the over/under-achievement of SLAs, which is the distance between attained and the contractual SLAs. While the need for meeting the target SLAs motivates the under-achievement part in the second component, it is also necessary to minimize over-achievement of SLAs. This is because an over-achieved SLA, for instance meeting 100% instead of the target of 95% for a particular customer, requires more time and effort than necessary from some workers and no additional reward is obtained in this case.

Note that the first term in (9.4) uses a weighted sum of utilizations over workers from each shift and across each skill level. Further, the weights $\alpha_{i,j}$ are fixed and not time-varying. Using historical data on SR arrivals, the percentage of workload arriving in each shift and for each skill level is obtained. These percentages decide the weights $\alpha_{i,j}$ used in

(9.4), that in turn satisfy

$$0 \leq \alpha_{i,j} \leq 1, \text{ and } \sum_{i=1}^{|A|}\sum_{j=1}^{|B|} \alpha_{i,j} = 1,$$

for $i = 1, 2, \ldots, |A|$, and $j = 1, 2, \ldots, |B|$. This prioritization of workers helps in optimizing the worker set based on the given workload. For instance, if 70% of the SRs requiring low skill worker attention arrive in shift 1, then one may set $\alpha_{1,0} = 0.7$, in the cost function (9.4), where 0 denotes the low skill level index.

The single-stage constraint functions $g_{i,j}(\cdot), h(\cdot), i = 1, \ldots, |C|, j = 1, \ldots, |P|$, are given by:

$$g_{i,j}(X_n) = \gamma_{i,j} - \gamma'_{i,j}(n), \forall i = 1, \ldots, |C|, j = 1, \ldots, |P|, \tag{9.5}$$

$$h(X_n) = 1 - q(n). \tag{9.6}$$

Here (9.5) specifies that the attained SLA levels should be equal to or above the contractual SLA levels for each customer-priority tuple. Further, (9.6) ensures that the SR queues for each complexity in the system stay bounded. In the constrained optimization problem formulated below, we attempt to satisfy these constraints in the long-run average sense (see (9.1)).

The SASOC algorithms treat the parameter as continuous-valued and tune it accordingly. Let us denote this continuous version of the worker parameter by $\bar{\theta} = (\bar{\theta}_1, \ldots, \bar{\theta}_N)$. Note that $\bar{\theta}_i \in [0, W_{\max}], i = 1, 2, \ldots, N$. We now design a smooth projection operator Γ that projects $\bar{\theta}$ on to the discrete space $\mathcal{D}$ so that the same can be used for performing the simulation of the service system. We call the Γ-operator as a generalized projection scheme as it lies in between a fully deterministic projection scheme based on mere rounding off and a completely randomized scheme, whereby depending on the value of $\bar{\theta}_j$ (for any $j = 1, \ldots, N$) one can find points $\mathcal{D}^k$ and $\mathcal{D}^{k+1}$ with $\mathcal{D}^k < \mathcal{D}^{k+1}$, $\mathcal{D}^k, \mathcal{D}^{k+1} \in \mathcal{D}$ such that $\mathcal{D}^k$ and $\mathcal{D}^{k+1}$ are the immediate neighbours of $\bar{\theta}_j$ in the set $\mathcal{D}$. Then, one sets the corresponding discrete parameter as

$$\theta_j = \begin{cases} \mathcal{D}^{k+1} & \text{w.p. } \dfrac{\bar{\theta}_j - \mathcal{D}^k}{\mathcal{D}^{k+1} - \mathcal{D}^k}, \\ \mathcal{D}^k & \text{w.p. } \dfrac{\mathcal{D}^{k+1} - \bar{\theta}_j}{\mathcal{D}^{k+1} - \mathcal{D}^k}. \end{cases} \tag{9.7}$$

9.3.3 A Generalized Projection Operator

For any $\bar{\theta} = (\bar{\theta}_1, \ldots, \bar{\theta}_N)$ with $\bar{\theta}_j \in [0, W_{\max}], j = 1, 2, \ldots, N$, we define a projection operator $\Gamma(\bar{\theta}) = (\Gamma_1(\bar{\theta}_1), \ldots, \Gamma_N(\bar{\theta}_N)) \in \mathcal{D}$ which projects any $\bar{\theta}$ onto the discrete set $\mathcal{D}$ as follows:

Let $\zeta > 0$ be a fixed real number and $\bar{\theta}_i$ be such that $\mathcal{D}^j \leq \bar{\theta}_i \leq \mathcal{D}^{j+1}, \mathcal{D}^j < \mathcal{D}^{j+1}$ for some $\mathcal{D}^j, \mathcal{D}^{j+1} \in \mathcal{D}$. Let us consider an interval of length 2ζ around the midpoint of $[\mathcal{D}^j, \mathcal{D}^{j+1}]$ and denote it as $[\tilde{\mathcal{D}}_1, \tilde{\mathcal{D}}_2]$, where $\tilde{\mathcal{D}}_1 = \frac{\mathcal{D}^j + \mathcal{D}^{j+1}}{2} - \zeta$ and $\tilde{\mathcal{D}}_2 = \frac{\mathcal{D}^j + \mathcal{D}^{j+1}}{2} + \zeta$. Then, $\Gamma_i(\bar{\theta}_i)$ for $\theta_i \in [\mathcal{D}^j, \tilde{\mathcal{D}}_1] \cup [\tilde{\mathcal{D}}_2, \mathcal{D}^{j+1}]$ is defined by

$$\Gamma_i(\bar{\theta}_i) = \begin{cases} 0 & \text{if } \bar{\theta}_i < 0 \\ \mathcal{D}^j & \text{if } \bar{\theta}_i \leq \frac{\mathcal{D}^j + \mathcal{D}^{j+1}}{2} - \zeta \\ \mathcal{D}^{j+1} & \text{if } \bar{\theta}_i \geq \frac{\mathcal{D}^j + \mathcal{D}^{j+1}}{2} + \zeta \\ W_{\max} & \text{if } \bar{\theta} \geq W_{\max}. \end{cases} \tag{9.8}$$

Further, $\Gamma_i(\bar{\theta}_i)$ for $\bar{\theta}_i \in [\tilde{\mathcal{D}}_1, \tilde{\mathcal{D}}_2]$ is given by

$$\Gamma_i(\bar{\theta}_i) = \begin{cases} \mathcal{D}^j & \text{w.p. } f(\frac{\tilde{\mathcal{D}}_2 - \bar{\theta}_i}{2\zeta}) \\ \mathcal{D}^{j+1} & \text{w.p. } 1 - f(\frac{\tilde{\mathcal{D}}_2 - \bar{\theta}_i}{2\zeta}) \end{cases} \tag{9.9}$$

In the above, w.p. stands for with probability and f is any continuously differentiable function defined on $[0, 1]$ such that $f(0) = 0$ and $f(1) = 1$. Note that we deterministically project onto either $\mathcal{D}^j$ or $\mathcal{D}^{j+1}$ if $\bar{\theta}_i$ is outside of the interval $[\tilde{\mathcal{D}}_1, \tilde{\mathcal{D}}_2]$. Further, for $\bar{\theta}_i \in [\tilde{\mathcal{D}}_1, \tilde{\mathcal{D}}_2]$, we project randomly using a smooth function f. It is necessary to have a smooth projection operator to ensure convergence of our SASOC algorithms as opposed to a deterministic projection operator that would project $\bar{\theta}_i \in [\mathcal{D}^j, \frac{\mathcal{D}^j + \mathcal{D}^{j+1}}{2})$ to $\mathcal{D}^j$ and $\bar{\theta}_i \in [\frac{\mathcal{D}^j + \mathcal{D}^{j+1}}{2}, \mathcal{D}^{j+1}]$ to $\mathcal{D}^{j+1}$. The problem with such a deterministic operator is that there is a jump at the midpoint and hence, when extended for any θ in the convex hull $\bar{\mathcal{D}}$, the transition dynamics of the process $\{(X_n, Y_n), n \geq 0\}$ is not necessarily continuously differentiable in the parameter. In other words, a non-smooth projection operator does not allow us to mimic a continuous parameter system.

The SASOC algorithms that we present subsequently tune the worker parameter in the convex hull of $\mathcal{D}$, denoted by $\bar{\mathcal{D}}$, a set that can be defined as $\bar{\mathcal{D}} = [0, W_{\max}]^N$. This idea has been used in Bhatnagar et al. [2011b] for an unconstrained discrete optimization problem.

However, the projection operator used there was a fully randomized operator. The generalized projection scheme that we incorporate has the advantage that while it ensures that the transition dynamics of the parameter extended Markov process is smooth (as desired), it requires a lower computational effort because in a large portion of the parameter space (assuming ζ is small), the projection operator is essentially deterministic.

We also require another projection operator $\bar{\Gamma}$ that projects any $\theta \in \mathbb{R}^N$ onto the set $\bar{\mathcal{D}}$ and is defined as $\bar{\Gamma}(\theta) = (\bar{\Gamma}_1(\theta_1), \ldots, \bar{\Gamma}_N(\theta_N))$, where $\bar{\Gamma}_i(\theta_i) = \min(0, \max(\theta_i, W_{\max}))$, $i = 1, \ldots, N$. Thus, $\bar{\Gamma}(\cdot)$ keeps the parameter updates within the set $\bar{\mathcal{D}}$ and $\Gamma(\cdot)$ projects them to the discrete set $\mathcal{D}$. The projected updates are then used as the parameter values for conducting the simulation of the service system.

9.3.4 Assumptions

We now make the following assumptions on the process, cost and step-sizes. For our algorithms that we present in the next chapter, one of (A2) or (A2') (below) will be assumed depending on whether the algorithm incorporates a Newton-based search technique or a gradient-based procedure.

(A1) The Markov process $\{(X_n(\theta), Y_n(\theta)), n \geq 0\}$ under a given dispatching policy and parameter θ is ergodic.

(A2) The single-stage cost functions $c(\cdot)$, $g_{i,j}(\cdot)$ and $h(\cdot)$ are all continuous. The long-run average cost $J(\cdot)$ and constraint functions $G_{i,j}(\cdot), H(\cdot)$ are twice continuously differentiable with bounded third derivative.

(A2') The single-stage cost functions $c(\cdot)$, $g_{i,j}(\cdot)$ and $h(\cdot)$ are all continuous. The long-run average cost $J(\cdot)$ and constraint functions $G_{i,j}(\cdot), H(\cdot)$ are continuously differentiable with bounded second derivative .

(A3) The step-sizes $\{a(n)\}$, $\{b(n)\}$ and $\{d(n)\}$ satisfy

$$\sum_n a(n) = \sum_n b(n) = \sum_n d(n) = \infty; \sum_n (a^2(n) + b^2(n) + d^2(n)) < \infty,$$

$$\frac{b(n)}{d(n)}, \frac{a(n)}{b(n)} \to 0 \text{ as } n \to \infty.$$

Assumption (A1) ensures that the process $\{(X_n, Y_n)\}$ is stable for any given θ and ensures that the long-run averages of the single stage cost and constraint functions in (9.1) are well-defined. (A2)/(A2') is a technical requirement needed to push through suitable Taylor's arguments in order to prove the convergence of the corresponding algorithm. The first two conditions in (A3) are standard requirements for step-size sequences and the last part ensures the separation of time scales between the different recursions in SASOC algorithms discussed in detail in Section 10.1.

Remark 9 *As seen before, the state space S is compact and ergodicity of the underlying Markov process $\{(X_n, Y_n)\}$ will follow if one ensures that there is at least one worker available for each complexity class.*

9.4 Summary

In this chapter, we motivated the discrete optimization problem of adaptively determining optimal staffing levels in SS. We formulated the problem as a constrained hidden Markov cost process, with the aim of finding an optimum worker parameter that minimizes a certain long-run cost objective, while adhering to a set of constraint functions, which are also long run averages. All SASOC algorithms that we describe in the next chapter are simulation based optimization methods as the single-stage cost and constraint functions are observable only via simulation and no closed form expressions are available. The single-stage cost that we designed balanced the conflicting objectives of maximizing worker utilizations and minimizing the over-achievement of SLAs. We also described a smooth (generalized) projection operator that projects a continuous-valued worker parameter onto a discrete set. As we will see in the next chapter, this helps in imitating a continuous parameter system with suitably defined transition dynamics and hence facilitates the usage of simultaneous perturbation technique.

In the next chapter, we present three novel SASOC algorithms for optimizing staff allocation in the context of SS. These algorithms solve the constrained optimization problem by applying the Lagrange relaxation procedure and incorporate an SPSA based scheme for performing gradient descent in the primal and at the same time, ascent in the dual, for the Lagrange multipliers.

Chapter 10

Simultaneous Perturbation Methods for Adaptive Labor Staffing

In this chapter, we develop three novel discrete parameter simulation-based optimization algorithms for solving the problem considered in Chapter 9, that of adaptive labor staffing. Henceforth, we shall refer to these algorithms as *SASOC (Staff Allocation using Stochastic Optimization with Constraints)* algorithms. The overall optimization scheme in all algorithms has two main stages: (a) evaluation of a candidate solution (worker parameter) to the long-run constrained optimization problem, and (b) updation of the worker parameter in the negative descent direction of the long-run performance objective. For the evaluation step (a) above, we leverage the simulation-based operational model developed in Banerjee et al. [2011]. All the SASOC algorithms that we propose fall into the category of simulation-based optimization methods. These methods do not require a priori knowledge of the service system dynamics in general and the single stage cost function in particular. Instead, they use the output of the service system simulation to find the optimum of a certain long term performance objective. Further, all SASOC algorithms that we propose are based on the crucial idea of perturbing the parameter vector using independent, identically distributed symmetric Bernoulli random variables. This idea constitutes the simultaneous perturbation stochastic approximation (SPSA) scheme that is an integral part of all the SASOC algorithms proposed here. More general perturbation random variables may however be used, see for instance, Spall [1992, 2000]. All SASOC algorithms involve a certain generalized smooth projection operator, which is essential to project the continuous-valued worker parameter tuned by the SASOC algorithms onto the discrete set. The smoothness is necessary to ensure that the underlying transition dynamics of the constrained Markov cost process is itself smooth (as a function of the continuous-valued parameter) - a critical requirement to prove the convergence of all SASOC algorithms.

The rest of this chapter is organized as follows: In Section 10.1, we introduce our solution methodology and in Section 10.2, we present our stochastic optimization techniques, specifically the first order method SASOC-G as well as two second order methods - SASOC-H and SASOC-W, respectively. In Section 10.3, we provide the proofs of convergence of all the SASOC algorithms. In the next chapter, we discuss the implementation of our algorithms as well as OptQuest, and present the performance simulation results.

10.1 Solution Methodology

The constrained long-run average cost optimization problem (9.1) can be expressed using the standard Lagrange multiplier theory as an unconstrained optimization problem given below.

$$\max_{\lambda} \min_{\theta} L(\theta, \lambda) \triangleq \lim_{n \to \infty} \frac{1}{n} \sum_{m=0}^{n-1} E \left\{ c(X_m) + \sum_{i=1}^{|C|} \sum_{j=1}^{|P|} \lambda_{i,j} g_{i,j}(X_m) + \lambda_f h(X_m) \right\} \tag{10.1}$$

where $\lambda_{i,j} \geq 0, \quad \forall i = 1, \ldots, |C|, j = 1, \ldots, |P|$ represent the Lagrange multipliers corresponding to constraints $g_{i,j}(\cdot)$ and λ_f represents the Lagrange multiplier for the constraint $h(\cdot)$, in the optimization problem (9.1). Also, $\lambda = (\lambda_{i,j}, \lambda_f, i = 1, \ldots, |C|, j = 1, \ldots, |P|)^T$. The function $L(\theta, \lambda)$ is commonly referred to as the Lagrangian. An optimal (θ^*, λ^*) is a saddle point for the Lagrangian, i.e., $L(\theta, \lambda^*) \geq L(\theta^*, \lambda^*) \geq L(\theta^*, \lambda)$. Thus, it is necessary to design an algorithm which descends in θ and ascends in λ in order to find the optimum point. The simplest iterative procedure for this purpose would use the gradients of the Lagrangian with respect to θ and λ to descend and ascend respectively. However, for the given system, the computation of gradient with respect to θ would be intractable due to lack of a closed form expression of the Lagrangian. Thus, a simulation based algorithm is required. The above explanation suggests that an algorithm for computing an optimal (θ^*, λ^*) would need three stages in each of its iterations.

1. The inner-most stage which performs one or more simulations over several time steps and aggregates data, i.e., does the averaging of the single-stage cost and constraint functions $c(\cdot)$, $g_{i,j}(\cdot)$ and $h(\cdot)$ for any given θ and λ updates.

2. The next outer stage which estimates the gradient of the Lagrangian along θ and updates θ along a descent direction. This stage would perform several iterations for a given λ and find a good estimate of θ; and

3. The outer-most stage which updates the Lagrange multipliers λ along an ascent direction, using the converged values of the inner two loops.

The above three steps will have to be performed iteratively till the solution converges to a saddle point described previously. Note that the loops are nested in the sense that the loop in iteration (1) would be a sub-loop for iteration (2). Likewise, iteration (2) would be a sub-loop for iteration (3). Thus, in between two successive updates of an outer loop (iterations (2) or (3)), one would potentially have to wait for a long time for convergence of the inner loop procedure (iteration (1) or iterations (1) and (2), respectively). This problem gets addressed by using simultaneous updates to all three stages in a stochastic recursive scheme but with different step-size schedules, the outer-most having the smallest while the inner-most having the largest of step-sizes. The resulting scheme is a multiple time-scale stochastic approximation algorithm [Borkar, 2008, Chapter 6].

10.2 Our Algorithms

In a deterministic optimization setting (i.e., without stochastic noise), any negative descent algorithm for obtaining the minimum (in θ) of the Lagrangian (10.1) (for a given λ) would have the form

$$\boxed{\theta(n+1) = \bar{\Gamma}(\theta(n) - \gamma_k \mathcal{H}(\theta(n))^{-1} \nabla_\theta L(\theta, \lambda)), \qquad (10.2)}$$

where $\mathcal{H}(\theta(n))$ is a positive definite and symmetric matrix and $\gamma_k > 0$ is a given step-size parameter. It is easy to see that $-\mathcal{H}(\theta(n))^{-1}\nabla_\theta L(\theta, \lambda)$ is a descent direction, since $\mathcal{H}(\theta(n))^{-1}$ is a positive definite matrix. Using the fact that $L(\theta, \lambda)$ has bounded second derivatives (see (A2)) and with suitable assumptions on the step-sizes γ_k, it would follow that (10.2) converges.

However, if the single stage cost function $c(\cdot)$ and the constraint functions $g_{i,j}(\cdot)$ and $h(\cdot)$ are observable only via simulation or in other words, that $\nabla_\theta L(\theta, \lambda)$ is not computable, then a stochastic approximation based algorithm for obtaining a saddle point of the Lagrangian (10.1) would update the worker parameter along a descent direction as follows:

$$\theta(n+1) = \bar{\Gamma}(\theta(n) - \gamma_k \mathcal{H}_n^{-1} h_n). \tag{10.3}$$

In the above, h_n represents the estimate of the gradient and $\mathcal{H}_n$, the positive definite and symmetric matrix used at update instant n. The convergence of our algorithms, which have the form (10.3) is established by using a diminishing stepsize sequence (see (A3) below) and employing a biased estimate using SPSA of the gradient, with the bias vanishing asymptotically.

All the SASOC algorithms that we propose are noisy variants of (10.3) and use SPSA techniques to estimate the gradient of the Lagrangian w.r.t. θ. In addition, two of our algorithms, SASOC-H and SASOC-W, use SPSA techniques to generate a sequence of positive definite and symmetric matrices. The three algorithms mainly differ in the choice of $\mathcal{H}_n$ and hence also the descent direction:

1. *SASOC-G*: Here $\mathcal{H}_n = I$ (identity matrix) and hence, this algorithm tunes the worker parameter in the direction of the negative gradient. The gradient is estimated using a two-measurement version of SPSA.

2. *SASOC-H*: Here $\mathcal{H}_n = \nabla^2 L(\theta, \lambda)(n)$, the Hessian of L w.r.t. $\theta(n)$ and hence, this uses a Newton update for optimizing the worker parameter. We use a two perturbation sequence version of SPSA to simultaneously estimate the gradient and the Hessian. For simplicity in the numerical experiments, as with Bhatnagar [2005], we implement the Jacobi variant of SASOC-H wherein $\mathcal{H}_n$ is a diagonal matrix with diagonal entries corresponding to the second-order partial derivatives (same as corresponding entries in the Hessian) but with all other elements set to zero.

3. *SASOC-W*: Here, as with SASOC-H, $\mathcal{H}_n$ is the Hessian of L. However, in this algorithm, the inverse of the Hessian matrix is tuned directly by using a procedure based on the Woodbury's identity. In contrast, SASOC-H estimates the Hessian and performs a matrix inverse. We use the full Hessian in our experiments because of ease of computation here over the case of direct Hessian inversion.

Thus, SASOC-G is a first-order method performing only gradient estimation while the other two algorithms SASOC-H and SASOC-W, are second-order methods that estimate both the gradient and the Hessian.

The overall flow of all SASOC algorithms can be diagrammatically represented as in Fig. 10.1. Each iteration of the algorithm involves two simulations (each for a period $\mathcal{T}$) -

one with $\Gamma(\theta(n))$, i.e., the current estimate of the parameter projected using the generalized projection operator so that it takes values in the discrete set $\mathcal{D}$ and the other with the (projected) perturbed parameter, $\Gamma(\theta(n) + p(n))$. Note that the perturbation $p(n)$ is algorithm-specific. For SASOC-G, $p(n) = \delta\Delta(n)$ and for SASOC-H/W, $p(n) = \delta_1\Delta(n) + \delta_2\hat{\Delta}(n)$. The rationale behind the choice of $p(n)$ will be subsequently clarified when the individual SASOC algorithms are presented. In every stage of the SASOC algorithms, the two simulations are carried out as shown in Fig. 10.1. Using the state values of the two simulations, $X(n)$ and $\hat{X}(n)$, the worker parameter θ is updated in an algorithm-specific manner. Algorithm 2 gives the structure of all three of our incremental update SASOC algorithms.

10.2.1 Algorithm structure

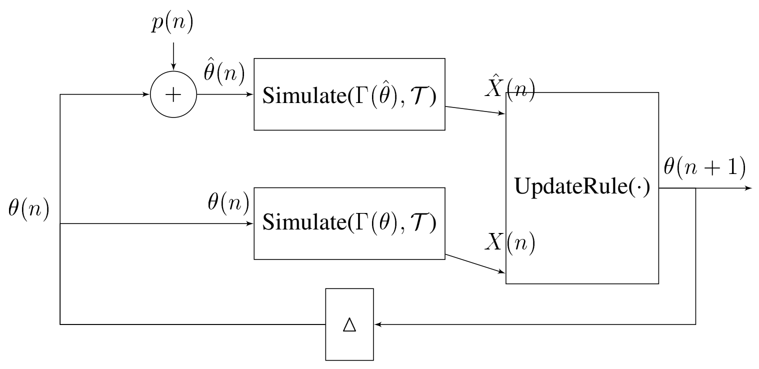


Figure 10.1: Overall flow of the algorithm 2.

Algorithm 2 Skeleton of SASOC algorithms

Input:

- R, a large positive integer;
- θ_0, initial parameter vector; $p(\cdot)$; Δ; $K \geq 1$
- UpdateRule(), the algorithm-specific update rule for the worker parameter θ and Lagrange multiplier λ.
- Simulate($\theta, \mathcal{T}$) $\rightarrow X$, the simulator of the SS

Output: $\theta^* \overset{\triangle}{=} \Gamma(\theta(R))$.

```
$\theta \leftarrow \theta_0$, $n \leftarrow 1$
```

```
begin loop
    $X \leftarrow$ Simulate($\Gamma(\theta(n)), \mathcal{T}$).
    $\hat{X} \leftarrow$ Simulate($\Gamma(\theta(n) + p(n)), \mathcal{T}$).
    UpdateRule().
    $n \leftarrow n + 1$
    if $n = R$ then
        Terminate and output $\Gamma(\theta(R))$.
    end if
end loop
```

10.2.2 SASOC-G Algorithm

SASOC-G is a three time-scale stochastic approximation algorithm that does primal descent using a two-measurement SPSA while performing dual ascent on the Lagrange multipliers.

SPSA based gradient estimate

Here, the gradient of the Lagrangian w.r.t. θ is obtained according to

$$\nabla_\theta L(\theta, \lambda) = \lim_{\delta \downarrow 0} E\left[\left(\frac{L(\theta + \delta\Delta, \lambda) - L(\theta, \lambda)}{\delta}\right)\Delta^{-1}\middle|\theta, \lambda\right], \tag{10.4}$$

where Δ is a vector of perturbation random variables that are independent, zero-mean, ± 1-valued and have the symmetric Bernoulli distribution. More general distributions on these random variables can be chosen as described in Spall [1992, 2000]. In (10.4), Δ^{-1} represents a vector of element-wise inverses of the Δ vector. Note that (10.4) is a one-sided estimate whose convergence is shown in [Chen et al., 1999, Lemma 1]. For a sufficiently large $K > 1$ and small $\delta > 0$, the estimate of the gradient can be approximated as follows:

$$\nabla_\theta L(\theta, \lambda) \approx \frac{1}{K}\sum_{i=1}^{K}\left[\left(\frac{L(\theta + \delta\Delta(i), \lambda) - L(\theta, \lambda)}{\delta}\right)\Delta(i)^{-1}\right].$$

Update rule of SASOC-G

From the form of the gradient estimator, it is clear that the Lagrangian function would be needed to compute the gradient estimate. However, for our problem, obtaining a closed-form expression for the Lagrangian itself is an intractable task. We overcome this by running two simulations with parameters $\Gamma(\theta(n))$ and $\Gamma(\theta(n)+p(n))$. Here, $p(n)=\delta\Delta(n)$, a choice motivated by the form of the gradient estimate in (10.4). Using the output of the two simulations, we estimate the quantities $L(\theta+\delta\Delta,\lambda)$ and $L(\theta,\lambda)$, respectively on the faster timescale. These estimates are in turn used to tune the worker parameter θ in the negative gradient descent direction. For $\lambda_{i,j}$ and λ_f, values of $g_{i,j}(\cdot)$ and $h(\cdot)$ respectively provide a stochastic ascent direction, proof of which will be given later in Theorem 10.6. Since maximization of the Lagrangian w.r.t. $\lambda_{i,j}$ and λ_f represents the outer most step, these parameters are updated on the slowest time-scale. The overall update rule for this scheme, SASOC-G, is as follows: For all $n\geq 0$,

$$\left.\begin{aligned}
&\theta_i(n+1)=\bar{\Gamma}_i\left(\theta_i(n)+b(n)\left(\tfrac{\bar{L}(nK)-\bar{L}'(nK)}{\delta\Delta_i(n)}\right)\right),\forall i=1,2,\ldots,N,\\
&\text{where for } m=0,1,\ldots,K-1,\\
&\bar{L}(nK+m+1)=\bar{L}(nK+m)+\\
&d(n)(c(X_{nK+m})+\sum_{i=1}^{|C|}\sum_{j=1}^{|P|}\lambda_{i,j}(nK)g_{i,j}(X_{nK+m})+\lambda_f h(X_{nK+m})-\bar{L}(nK+m)),\\
&\bar{L}'(nK+m+1)=\bar{L}'(nK+m)+\\
&d(n)(c(\hat{X}_{nK+m})+\sum_{i=1}^{|C|}\sum_{j=1}^{|P|}\lambda_{i,j}(nK)g_{i,j}(\hat{X}_{nK+m})+\lambda_f h(\hat{X}_{nK+m})-\bar{L}'(nK+m)),\\
&\lambda_{i,j}(n+1)=\left(\lambda_{i,j}(n)+a(n)g_{i,j}(X_n)\right)^+,\forall i=1,2,\ldots,|C|,j=1,2,\ldots,|P|,\\
&\lambda_f(n+1)=\left(\lambda_f(n)+a(n)h(X_n)\right)^+.
\end{aligned}\right\}\tag{10.5}$$

We now explain the various quantities in (10.5).

- $K\geq 1$ is a fixed parameter which controls the rate of update of θ in relation to that of $\bar{L}$ and $\bar{L}'$. This parameter allows for accumulation of updates to $\bar{L}$ and $\bar{L}'$ for K iterations in between two successive θ updates;
- X_m represents the state at iteration m from the simulation run with nominal parameter $\theta_{[\frac{n}{K}]}$ while $\hat{X}_m$ represents the state at iteration m from the simulation run with perturbed

parameter $\theta_{[\frac{n}{K}]} + \delta\Delta_{[\frac{n}{K}]}$. Here $[\frac{n}{K}]$ denotes the integer portion of $\frac{n}{K}$. For simplicity, hereafter we use θ to denote $\theta_{[\frac{n}{K}]}$ and $\theta + \delta\Delta$ to denote $\theta_{[\frac{n}{K}]} + \delta\Delta_{[\frac{n}{K}]}$;

- $\delta > 0$ is a fixed perturbation control parameter while Δ is a vector of perturbation random variables that are independent, zero-mean and have the symmetric Bernoulli distribution;
- The operator $\bar{\Gamma}(\cdot)$ ensures that the updated value for θ stays within the convex hull $\bar{\mathcal{D}}$ and is defined in Section 9.3.3; and
- $\bar{L}$ and $\bar{L}'$ represent Lagrange estimates corresponding to θ and $\theta+\delta\Delta$ respectively. Thus, for each iteration, two simulations are carried out, one with the nominal parameter θ and the other with the perturbed parameter $\theta + \delta\Delta$, the results of which are used to update $\bar{L}$ and $\bar{L}'$.

We achieve separation of time-scales between the recursions of $\theta_i, \bar{L}, \bar{L}'$ and λ via the difference in the step-sizes $a(n), b(n)$ and $d(n)$ (see *(A3)*). The chosen step-sizes ensure that the recursions of Lagrange multipliers $\lambda_{i,j}$ proceed 'slower' in comparison to those of the worker parameter θ, while the updates of the average cost - $\bar{L}$ and $\bar{L}'$ proceed the fastest.

10.2.3 SASOC-H Algorithm

This is a second-order algorithm for adaptive labour staffing which uses SPSA based techniques to estimate both the gradient and the Hessian. As discussed before, the overall algorithm structure is represented by Fig. 10.1 with $p(n) = \theta(n) + \delta_1\Delta(n) + \delta_2\hat{\Delta}(n)$ in this case. Thus, each iteration of the algorithm involves two simulations (each for a period $\mathcal{T}$) - one with $\Gamma(\theta(n))$ and the other with $\Gamma(\theta(n) + \delta_1\Delta(n) + \delta_2\hat{\Delta}(n))$. As explained in the next section, the two perturbation sequences Δ and $\hat{\Delta}$ are used to simultaneously estimate the gradient and the Hessian of the Lagrangian w.r.t. θ.

SPSA based simultaneous estimates for the gradient and the Hessian

Suppose the Lagrangian in (10.1) is twice differentiable w.r.t. θ, then we can look at possible second order schemes for computing updates to θ. If the Lagrangian (10.1) were a second-degree equation, then the exact solution for the θ update to reach the minimum point would have been $-[\nabla^2_\theta L(\theta_0)]^{-1}\nabla_\theta L(\theta_0)$ with θ_0 as the starting point, i.e.,

$$\theta^* = \theta_0 - [\nabla^2_\theta L(\theta_0)]^{-1}\nabla_\theta L(\theta_0),$$

would be the optimal parameter. For a higher-degree Lagrangian, the above solution can be used with a step-size parameter iteratively till convergence to an optimal θ^*. Let Δ and $\widehat{\Delta}$ be two independent vectors of perturbation random variables that are independent, zero-mean, ± 1-valued and have the symmetric Bernoulli distribution. More general distributions for Δ and $\widehat{\Delta}$ may however be used, see Spall [1992, 2000]. We use the following estimates for the gradient and Hessian, respectively, as described in [Bhatnagar et al., 2011a, Section 3.2.1]:

$$\nabla_\theta L(\theta,\lambda) = \lim_{\delta_1,\delta_2 \downarrow 0} E\left[\left(\frac{L(\theta+\delta_1\Delta+\delta_2\widehat{\Delta},\lambda) - L(\theta,\lambda)}{\delta_2} \right) \widehat{\Delta}^{-1} \middle| \theta,\lambda \right],$$

$$\nabla_\theta^2 L(\theta,\lambda) = \lim_{\delta_1,\delta_2 \downarrow 0} E\left[\Delta^{-1} \left(\frac{L(\theta+\delta_1\Delta+\delta_2\widehat{\Delta},\lambda) - L(\theta,\lambda)}{\delta_1\delta_2} \right) \left(\widehat{\Delta}^{-1}\right)^T \middle| \theta,\lambda \right],$$

where Δ^{-1} and $\widehat{\Delta}^{-1}$ represent vectors of element-wise inverses of the Δ and $\widehat{\Delta}$ vectors, respectively. Thus, the inner terms of the above two expectations can be used for estimating the Hessian and also updating θ.

Update rule of SASOC-H

We have the following update rule of SASOC-H using two perturbation sequences with the rest of the recursions the same as with the previous algorithm. For $n \geq 0$, we have

$$\theta_i(n+1) = \bar{\Gamma}_i \left(\theta_i(n) + b(n) \sum_{j=1}^{N} M_{i,j}(n) \left(\frac{\bar{L}(nK) - \bar{L}'(nK)}{\delta_2 \widehat{\Delta}_j(n)} \right) \right), \qquad (10.6)$$

$$H_{i,j}(n+1) = H_{i,j}(n) + b(n) \left(\frac{\bar{L}'(nK) - \bar{L}(nK)}{\delta_1 \Delta_j(n) \delta_2 \widehat{\Delta}_i(n)} - H_{i,j}(n) \right), \qquad (10.7)$$

for $i, j = 1, \ldots, N$. Note that

- The update equations corresponding to $\bar{L}, \bar{L}'$, $\lambda_{i,j}, i = 1, \ldots, |C|, j = 1, \ldots, |P|$ and λ_f are the same as in SASOC-G (10.5). However, note that the perturbed parameter in this case is $(\theta(n) + \delta_1\Delta(n) + \delta_2\widehat{\Delta}(n))$. Thus, unlike SASOC-G, $\hat{X}_m$ represents the state at iteration m from the simulation run with perturbed parameter $\Gamma(\theta(n) + \delta_1\Delta(n) + \delta_2\widehat{\Delta}(n))$, while X_m continues to have the same interpretation as in the case of SASOC-G.

- $\delta_1, \delta_2 > 0$ are fixed perturbation control parameters while $\Delta(n)$ and $\widehat{\Delta}(n), n \geq 0$ are two independent sequences of vectors of perturbation random variables that are independent, zero-mean, ± 1-valued, and have the symmetric Bernoulli distribution;
- $H(n) = [H_{i,j}(n)]_{i=1,j=1}^{|A|\times|B|,|A|\times|B|}$ represents the nth Hessian estimate of the Lagrangian. $H(0)$ is a positive definite and symmetric matrix. We let $H(0) = cI$, with $c > 0$ and I being the identity matrix; and
- $M(n) = \Upsilon(H(n))^{-1} = [M(n)_{i,j}]_{i=1,j=1}^{|A|\times|B|,|A|\times|B|}$ represents the inverse of the Hessian estimate $H(n)$ of the Lagrangian, where $\Upsilon(\cdot)$ is a projection operator ensuring that the Hessian estimates remain symmetric and positive definite. The Υ operation is assumed to satisfy assumption *(A4)*.

Assumption (A4)

The projection operator $\Upsilon(\cdot)$ projects a square matrix to a symmetric positive definite matrix. If $\{A_n\}$ and $\{B_n\}$ are sequences of matrices in $\mathcal{R}^{N\times N}$ such that $\lim\limits_{n\to\infty} \parallel A_n - B_n \parallel = 0$, then $\lim\limits_{n\to\infty} \parallel \Upsilon(A_n) - \Upsilon(B_n) \parallel = 0$ as well. Further, for any sequence $\{C_n\}$ of matrices in $\mathcal{R}^{N\times N}$, if $\sup\limits_n \parallel C_n \parallel < \infty$, then $\sup_n \parallel \Upsilon(C_n) \parallel < \infty$ and $\sup_n \parallel \{\Upsilon(C_n)\}^{-1} \parallel < \infty$, as well.

We show in a later section with simulation results that second-order approaches exhibit better convergence behaviour.

10.2.4 SASOC-W Algorithm

The SASOC-H algorithm is more robust than SASOC-G. However, it requires computation of the inverse of the Hessian H at each stage which is a computationally intensive operation. We propose an enhancement using Woodbury's identity to the previous algorithm that results in significant computational gains. In particular, an application of Woodbury's identity brings down the computational complexity from $O(n^3)$[1] to $O(n^2)$, where $n = |A| \times |B|$.

Woodbury's Identity

Woodbury's identity states that

$$(A + BCD)^{-1} = A^{-1} - A^{-1}B\left(C^{-1} + DA^{-1}B\right)^{-1} DA^{-1}$$

[1]The popular Gauss-Jordan procedure for matrix inverse of a matrix has computational complexity $O(n^3)$.

where A and C are invertible square matrices and B and D are rectangular matrices of appropriate sizes. The Hessian update in (10.7) without projection can be rewritten as

$$H(n+1) = (1-b(n))H(n) + P(n)Z(nK)Q(n)$$

where
$P(n) = \frac{1}{\delta_1}\left[\frac{1}{\Delta_1(n)}, \frac{1}{\Delta_2(n)}, \ldots, \frac{1}{\Delta_{|A|\times|B|}(n)}\right]^T, Q(n) = \frac{1}{\delta_2}\left[\frac{1}{\widehat{\Delta}_1(n)}, \frac{1}{\widehat{\Delta}_2(n)}, \ldots, \frac{1}{\widehat{\Delta}_{|A|\times|B|}(n)}\right],$
and $Z(nK) = b(n)\left(\bar{L}'(nK) - \bar{L}(nK)\right)$.

Now, applying Woodbury's identity to $H(n+1)^{-1} = M(n+1)$, gives us

$$M(n+1) = \left(\frac{M(n)}{1-b(n)}\left[I - \frac{b(n)\left(\bar{L}'(nK) - \bar{L}(nK)\right)P(n)Q(n)M(n)}{1-b(n)+b(n)\left(\bar{L}'(nK) - \bar{L}(nK)\right)Q(n)M(n)P(n)}\right]\right),$$

which is a recursive update rule for directly updating the matrix $M(n)$, which is the inverse of $H(n), n \geq 0$.

Update rule of SASOC-W

The modified update scheme, named SASOC-W, is given below. For $n \geq 0$,$i = 1, \ldots, N$,

$$\begin{aligned}\theta_i(n+1) =& \bar{\Gamma}_i\left(\theta_i(n) + b(n)\sum_{j=1}^{N} M_{i,j}(n)\left(\frac{\bar{L}(nK) - \bar{L}'(nK)}{\delta_1\widehat{\Delta}_j(n)}\right)\right), \qquad (10.8)\\ M(n+1) =& \Upsilon\left(\frac{M(n)}{1-b(n)}\left[I - \frac{b(n)\left(\bar{L}'(nK) - \bar{L}(nK)\right)P(n)Q(n)M(n)}{1-b(n)+b(n)\left(\bar{L}'(nK) - \bar{L}(nK)\right)Q(n)M(n)P(n)}\right]\right).\end{aligned}$$

In the above, $M(0)$ is initialized to cI, I being the identity matrix and $c > 0$. The rest of the update rule corresponding to $\bar{L}, \bar{L}'$, $\lambda_{i,j}, i = 1, \ldots, |C|, j = 1, \ldots, |P|$ and λ_f stays the same as with SASOC-G/H.

Our SASOC algorithms differ from the algorithms of Bhatnagar et al. [2011a] in the following ways: (i) Unlike the algorithms of Bhatnagar et al. [2011a] which are for a continuous-valued parameter, our SASOC algorithms are for a constrained discrete optimization setting and involve a generalized projection operator that renders the transition probabilities of the extended Markov process for any $\theta \in \bar{\mathcal{D}}$ smooth. (ii) We allow the average cost estimates $\bar{L}$ and $\bar{L}'$ to accumulate and update the worker parameters W_i and

Lagrange multipliers $\lambda_{i,j}$ every K instants. (iii) Unlike Bhatnagar et al. [2011a], we do not need to estimate the average cost for the constraint functions $G_{i,j}(\theta)$ and $H(\theta)$. Instead, we directly use the sample $g_{i,j}(\cdot)$ for performing gradient ascent in Lagrange multipliers. (iv) Since the SASOC-W algorithm does not involve explicit computation of the Hessian inverse, it is computationally more efficient than the second-order algorithms of Bhatnagar et al. [2011a].

10.3 Convergence analysis

We provide a sketch of the convergence of the SASOC-G, SASOC-H and SASOC-W algorithms. The first step in the convergence analysis is common to all the SASOC algorithms and involves the extension of the dynamics of the underlying Markov process from a discrete-valued parameter dependent process with the parameter taking values in the set $\mathcal{D}$ to a continuous-valued parameter dependent process with the parameter taking values in $\bar{\mathcal{D}}$. The following subsections then establish the convergence of the three SASOC algorithms in the continuous parameter.

10.3.1 Extension of the transition dynamics to the continuous parameter

Recall that the discrete parameter θ of the Markov process $\{(X_n(\theta), Y_n(\theta))\}$ takes values in the set $\mathcal{D}$ defined earlier. Using the members of $\mathcal{D}$, one can extend the transition dynamics $p_\theta(i,j)$ of the underlying Markov process to any θ in the convex hull $\bar{\mathcal{D}}$ as follows:

$$p_\theta(i,j) = \sum_{k=1}^{p} \beta_k(\theta) p_{D^k}(i,j), \quad \forall \theta \in \bar{\mathcal{D}}, i,j \in S, \tag{10.9}$$

where the weights $\beta_k(\theta)$ satisfy $0 \leq \beta_k(\theta) \leq 1, k = 1,\ldots,p$ and $\sum_{k=1}^{p} \beta_k(\theta) = 1$. For this choice of $\beta_k(\theta)$, $p_\theta(i,j), i,j \in S, \theta \in \bar{\mathcal{D}}$ can be seen to satisfy the properties of transition probabilities. We now explain the manner in which these weights are obtained. It is worth noting here that the weights $\beta_k(\theta)$ must be continuously differentiable in order to ensure that the extended transition probabilities are continuously differentiable as well and our SASOC algorithms converge. Moreover, in the SASOC algorithms, we do not require an explicit computation of these weights while trying to solve the constrained optimization problem (9.1). Consider the case when $\theta = \theta_1$ and suppose θ_1 lies between D^j and D^{j+1} (both members of $\mathcal{D}$). By construction, $\beta_k(\theta)$ will correspond to the probability with which projection is done on $[D^j, D^{j+1}]$ and is obtained using the Γ-projection operator as follows: Let us consider an interval of length 2ζ around the midpoint of $[D^j, D^{j+1}]$ and denote it as $[\tilde{D}_1, \tilde{D}_2]$, where $\tilde{D}_1 = \frac{\mathcal{D}^j + \mathcal{D}^{j+1}}{2} - \zeta$ and $\tilde{D}_2 = \frac{\mathcal{D}^j + \mathcal{D}^{j+1}}{2} + \zeta$. Then, the weights $\beta_k(\theta)$ are set in the following manner: $\beta_k(\theta) = 0, \forall k \notin \{j, j+1\}$ and $\beta_j(\theta), \beta_{j+1}(\theta)$ is given by:

$$(\beta_j(\theta_1), \beta_{j+1}(\theta_1)) = \begin{cases} (1,0) & \text{if } \theta_1 \in \left[D^j, \tilde{D}_1\right] \\ \left(f(\frac{\tilde{D}_2 - \theta_1}{2\zeta}), 1 - f(\frac{\tilde{D}_2 - \theta_1}{2\zeta})\right) & \text{if } \theta_1 \in \left[\tilde{D}_1, \tilde{D}_2\right] \\ (0,1) & \text{if } \theta_1 \in \left[\tilde{D}_2, D^{j+1}\right] \end{cases} \tag{10.10}$$

In the above, f to be a continuously differentiable function defined on $[0,1]$ such that $f(0) = 0$ and $f(1) = 1$ and the Γ-projection is derived from such an f. The above can be similarly extended when the parameter θ has N components. It can thus be seen that $\beta_k(\theta), k = 1, \ldots, p$ are continuously differentiable functions of θ. Thus, from (10.9) and the fact that $\beta_k(\theta)$ are continuously differentiable, it can be seen that the extended transition dynamics $p_\theta(i,j), \forall \theta \in \bar{\mathcal{D}}, i, j \in S$ are continuously differentiable.

We now claim the following:

Lemma 10.1 *For any $\theta \in \bar{\mathcal{D}}$, $\{(X_n(\theta), Y_n(\theta)), n \geq 0\}$ is ergodic Markov.*

Proof: Follows in a similar manner as Lemma 2 of Bhatnagar et al. [2011b]. ■

Now, define analogues of the long-run average cost and constraint functions for any $\theta \in \bar{\mathcal{D}}$ as follows:

$$\begin{aligned} \bar{J}(\theta) &\triangleq \lim_{n\to\infty} \frac{1}{n} \sum_{m=0}^{n-1} c(X_m(\theta)), \\ \bar{G}_{i,j}(\theta) &\triangleq \lim_{n\to\infty} \frac{1}{n} \sum_{m=0}^{n-1} g_{i,j}(X_m(\theta)) \leq 0, \\ &\qquad \forall i = 1, \ldots, |C|, j = 1, \ldots, |P|, \\ \bar{H}(\theta) &\triangleq \lim_{n\to\infty} \frac{1}{n} \sum_{m=0}^{n-1} h(X_m(\theta)) \leq 0. \end{aligned} \tag{10.11}$$

The difference between the above and the corresponding entitites defined in (9.1) is that θ can now take values in $\bar{\mathcal{D}}$.

Lemma 10.2 *$\bar{J}(\theta), \bar{G}_{i,j}(\theta), i = 1, \ldots, |C|, j = 1, \ldots, |P|$, and $\bar{H}(\theta)$ are continuously differentiable in $\theta \in \bar{\mathcal{D}}$.*

Proof: Follows in a similar manner as Lemma 3 of Bhatnagar et al. [2011b]. ■

We now prove the SASOC algorithms described previously are equivalent to their analogous continuous parameter ($\bar{\theta}$) counterparts under the extended Markov process dynamics.

Lemma 10.3 *Under the extended dynamics $p_\theta(i,j), i,j \in S$ of the Markov process $\{(X_n(\theta), Y_n(\theta))\}$ defined over all $\theta \in \bar{\mathcal{D}}$, we have*

(i) SASOC-G algorithm is analogous to its continuous counterpart where $\Gamma(\theta)$ and $\Gamma(\theta + \delta\triangle)$ are replaced by $\bar{\Gamma}(\theta)$ and $\bar{\Gamma}(\theta + \delta\triangle)$ respectively.

(ii) SASOC-H and SASOC-W algorithms are analogous to their continuous counterparts where $\Gamma(\theta)$ and $\Gamma(\theta + \delta_1\triangle + \delta_2\hat{\triangle})$ are replaced by $\bar{\Gamma}(\theta)$ and $\bar{\Gamma}(\theta + \delta_1\triangle + \delta_2\hat{\triangle})$ respectively.

Proof: (i): Consider the SASOC-G algorithm (10.5). Let $\theta(m)$ be a given parameter update that lies in $\bar{D}^o$ (where $\bar{D}^o$ denotes the interior of the set $\bar{D}$). Let $\delta > 0$ be sufficiently small so that $\bar{\theta}^1(m) = (\bar{\Gamma}_j(\theta_j(m) + \delta\Delta_j(m)), j = 1,\ldots,N)^T = (\theta_j(m) + \delta\Delta_j(m)), j = 1,\ldots,N)^T$.

Consider now the Γ-projected parameters $\theta^1(m) = (\Gamma_j(\theta_j(m)+\delta\Delta_j(m)), j = 1,\ldots,N)^T$ and $\theta^2(m) = (\Gamma_j(\theta_j(m)), j = 1,\ldots,N)^T$, respectively. By the construction of the generalized projection operator, these parameters are equal to $\theta^k \in C$ with probabilities $\beta_k((\theta_j(m) + \delta\Delta_j(m), j = 1,\ldots,N)^T)$ and $\beta_k(\theta_j(m), j = 1,\ldots,N)^T)$, respectively. When the operative parameter is θ^k, the transition probabilities are $p_{\theta^k}(i,l), i,l \in S$. Thus with probabilities $\beta_k((\theta_j(m)+\delta\Delta_j(m), j = 1,\ldots,N)^T)$ and $\beta_k((\theta_j(m), j = 1,\ldots,N)^T)$, respectively, the transition probabilities in the two simulations equal $p_{\theta^k}(i,l)$, $i,l \in S$.

Next, consider the alternative (extended) system with parameters $\bar{\theta}^1(m) = (\bar{\Gamma}_j(\theta_j(m)+\delta\Delta_j(m))$ and $\bar{\theta}^2(m) = \bar{\Gamma}_j(\theta_j(m))$, respectively. The transition probabilities are now given by

$$p_{\bar{\theta}^i(m)}(j,l) = \sum_{k=1}^{p} \beta_k(\bar{\theta}^i(m)) p_{\theta^k}(j,l),$$

$i = 1,2,\ j,l \in S$. Thus with probability $\beta_k(\bar{\theta}^i(m))$, a transition probability of $p_{\theta^k}(j,l)$ is obtained in the ith system. Thus the two systems (original and the one with extended dynamics) are analogous.

Now consider the case when $\theta(m) \in \partial\bar{\mathcal{D}}$, i.e., is a point on the boundary of $\bar{\mathcal{D}}$. Then, one or more components of $\theta(m)$ are extreme points. For simplicity, assume that only one component (say the ith component) is an extreme point as the same argument carries over if there are more parameter components that are extreme points. By the ith component of $\theta(m)$ being an extreme point, we mean that $\theta_i(m)$ is either 0 or $W_{\max}$. By assumption, the other components $j = 1, \ldots, N, j \neq i$ are not extreme. Thus, $\theta_i(m) + \delta\Delta_i(m)$ can lie outside of the interval $[0, W_{\max}]$. For instance, suppose that $\theta_i(m) = W_{\max}$ and that $\theta_i(m) + \delta\Delta_i(m) > W_{\max}$ (which will happen if $\Delta_i(m) = +1$). In such a case, $\theta_i^1(m) = \Gamma_i(\theta_i(m) + \delta\Delta_i(m)) = W_{\max}$ with probability one. Then, as before, $\theta^1(m)$ can be written as the convex combination $\theta^1(m) = \sum_{k=1}^{p} \beta_k(\theta^1(m))\theta^k$ and the rest follows as before. The same argument carries over when more than one parameter component is an extreme point.

(ii): Follows in a similar manner as part (i) above. ■

As a consequence of Lemma 10.3, we can analyze the SASOC algorithms with the continuous parameter $\bar{\theta}$ used in place of θ and under the extended transition dynamics (10.9). By an abuse of notation, we shall henceforth use θ to refer to the latter.

10.3.2 SASOC-G

The convergence analysis of SASOC-G can be split into four stages:

(I) The fastest time-scale in SASOC-G corresponds to the step-size sequence $\{d(n)\}$ that is used to update the Lagrangian estimates $\bar{L}$ and $\bar{L}'$ corresponding to simulations with θ and $\theta + \delta\Delta$ respectively. First, we show that these estimates indeed converge to the Lagrangian values $L(\theta, \lambda)$ and $L(\theta + \delta\Delta, \lambda)$ defined in equation (10.1). Note that the θ and λ which are updated on slower time-scales, can be assumed to be time invariant quantities for the purpose of analysis of these Lagrangian estimates.

(II) Next, we show that the parameter updates $\theta(n)$ using SASOC-G converge to a limit point of the ODE

$$\dot{\theta}(t) = \check{\Gamma}\left(-\nabla_\theta L(\theta(t), \lambda)\right), \tag{10.12}$$

where $\check{\Gamma}$ is defined as follows: For any bounded continuous function $\epsilon(\cdot)$,

$$\check{\Gamma}(\epsilon(\theta(t))) = \lim_{\eta \downarrow 0} \frac{\bar{\Gamma}(\theta(t) + \eta\epsilon(\theta(t))) - \theta(t)}{\eta}. \tag{10.13}$$

The projection operator $\check{\Gamma}(\cdot)$ ensures that the evolution of θ stays within the bounded set $\mathcal{D}$. Again for the analysis of the θ-update, the value of λ which is updated on the slowest time-scale is assumed constant.

(III) We then show that $\lambda_{i,j}$s and λ_f converge respectively to the limit points of the ODEs

$$\begin{aligned}
&\dot{\lambda_{i,j}}(t) = \Pi\left(G_{i,j}(\theta^*)\right), \forall i = 1, 2, \ldots, |C|, j = 1, 2, \ldots, |P|,\\
&\dot{\lambda_f}(t) = \Pi\left(H(\theta^*)\right),
\end{aligned}$$

where θ^* is the converged parameter value of SASOC-G corresponding to the Lagrange parameter $\lambda(t) \overset{\triangle}{=} (\lambda_{i,j}(t), \lambda_f(t), i = 1, \ldots, |C|, j = 1, \ldots, |P|)^T$. For any bounded and continuous function $\bar{\epsilon}(\cdot)$,

$$\Pi(\bar{\epsilon}(\lambda(t))) = \lim_{\eta \downarrow 0} \frac{(\lambda(t) + \eta\bar{\epsilon}(\lambda(t)))^+ - \lambda(t)}{\eta}.$$

Here again, the projection operator Π ensures that the evolution of λ stays non-negative. From the definition of the Lagrangian given in equation (10.1), the gradient of the Lagrangian w.r.t. $\lambda_{i,j}$ can be seen to be $G_{i,j}(\theta^*)$ and that w.r.t. λ_f to be $H(\theta^*)$. Thus, the above ODEs suggest that in SASOC-G, $\lambda_{i,j}$s and λ_f are ascending in the Lagrangian value and converge to a local maximum point.

(IV) Finally, we show that the algorithm indeed converges to a (local) saddle point of the Lagrangian with local maximum in $\lambda_{i,j}$s and λ_f, and local minimum in θ.

Lemma 10.4 $||\bar{L}(n) - L(\theta(n), \lambda(n))|| \to 0$ *w.p. 1, as* $n \to \infty$.

Proof: The θ and λ values are updated on slower time-scales, thus assumed to be constant in this proof. Let

$$l(X_{nK+m}) \overset{\triangle}{=} c(X_{nK+m}) + \sum_{i=1}^{|C|}\sum_{j=1}^{|P|} \lambda_{i,j}(nK) g_{i,j}(X_{nK+m}) + \lambda_f h(X_{nK+m}).$$

The $\bar{L}$ update can be re-written as

$$\bar{L}(m+1) = \bar{L}(m) + d(m)\left(L(\theta(m), \lambda(m)) + \xi_1(m) - \bar{L}(m) + M_{m+1}\right),$$

where $\xi_1(m) = (E[l(X_m)|\mathcal{F}_{m-1}] - L(\theta(m), \lambda(m)), m \geq 0$ and $\mathcal{F}_m = \sigma(X_n, Y_n, \lambda(n), \theta(n), n \leq m), m \geq 0$ are the associated σ-fields. Also, $M_{m+1} = l(X_m) - E[l(X_m)|\mathcal{F}_{m-1}], m \geq 0$ is a martingale difference sequence. Let $N_m = \sum_{n=0}^{m} d(n)M_{n+1}$. It can be easily verified that $(N_m, \mathcal{F}_m), m \geq 0$ is a square-integrable martingale obtained from the corresponding martingale difference $\{M_m\}$. Further, from the square summability of $d(n), n \geq 0$, and the facts that S is compact and l is Lipschitz continuous, it can be verified from the martingale convergence theorem that $\{N_m, m \geq 0\}$ converges almost surely.

Now from Lemma 10.1 $\{(X_m, Y_m)\}$ is ergodic Markov for any given $\theta(m) \in \bar{\mathcal{D}}$. Hence, $|\xi_1(m)| \to 0$ almost surely on the 'natural timescale', as $m \to \infty$. The 'natural timescale' is clearly faster than the algorithm's timescale and hence $\xi_1(m)$ can be ignored in the analysis of $\bar{L}$-recursion, see [Borkar, 2008, Chapter 6.2] for detailed treatment of natural timescale algorithms. The rest of the proof follows by applying Lemma 7.4 (Hirsch [1989]). ■

On similar lines, $||\bar{L}'(n) - L(\theta(n) + \delta\Delta(n), \lambda(n))|| \to 0$ w.p. 1, as $n \to \infty$. Thus, θ updates which are on the slower time scale $\{b(n)\}$, can be re-written as

$$W_i(n+1) = W_i(n) - b(n)\left(\frac{L(\theta(n)+\delta\Delta(n),\lambda) - L(\theta(n),\lambda)}{\delta\Delta_i(n)}\right) + b(n)\chi_{n+1}, \quad (10.14)$$

$\forall i = 1, 2, \ldots, |A| \times |B|$, where $\chi_n = o(1)$ in view of Lemma 10.4. Note here that $\lambda(n) \equiv \lambda, \quad \forall n$. Now for the ODE (10.12), $V^{\lambda}(\cdot) = L(\cdot, \lambda)$ serves as an associated Lyapunov function and the stable fixed points of this ODE lie within the set $K^{\lambda} = \{\theta \in S : \check{\Gamma}\left(-\nabla L(\theta(t), \lambda)\right) = 0\}$.

Theorem 10.5 *Under (A1),(A2'),(A3), in the limit as $\delta \to 0$, $\theta(R) \to \theta^* \in K^{\lambda}$ almost surely as $R \to \infty$.*

Proof: From assumption (A2), $L(\theta, \lambda)$ is continuous in θ, for any given λ. Hence over the compact set $\bar{\mathcal{D}}$, $L(\theta, \lambda)$ is uniformly bounded. Thus, from Lasalle's invariance theorem (Lasalle and Le fschetz [1961], see also [Kushner and Yin, 1997, Theorem 2.3, pp. 76]), $\theta(R) \to \theta^* \in K^{\lambda}$ a.s. as $R \to \infty$. ■

Thus (10.14) can be seen to be an Euler discretization of (10.12) and converges a.s. to K^{λ} in the limit as $\delta \to 0$.

For $\{\lambda(n)\}$ updates on the slowest time-scale $\{a(n)\}$, one can replace $\theta(n)$ by $\theta^*(n) \in K^{\lambda(n)}$. Let

$$F^{\theta^*} = \{\lambda \geq 0 : \Pi\left(G_{i,j}(\theta^*)\right) = 0, \forall i = 1, 2, \ldots, |C|, j = 1, 2, \ldots, |P|; \Pi\left(H(\theta^*)\right) = 0\}.$$

Theorem 10.6 *$\lambda(R) \rightarrow \lambda^* \in F^{\theta^*}$ w.p. 1 as $R \rightarrow \infty$.*

Proof: The λ update in (10.5) can be re-written as

$$\lambda_{i,j}(n+1) = \lambda_{i,j}(n) + a(n)\left[G_{i,j}(\theta^*(n)) + N_{n+1} + M_{n+1}\right],$$

where $N_{n+1} = E[g_{i,j}(X_n)|\mathcal{F}_{n-1}] - G_{i,j}(\theta^*(n))$, $M_{n+1} = g_{i,j}(X_n) - E[g_{i,j}(X_n)|\mathcal{F}_{n-1}]$. It is easy to see from Lemma 10.1 that $N_n \rightarrow 0$ as $n \rightarrow \infty$ along the natural timescale (see Lemma 10.4). Further, $\{M_n\}$ is a martingale difference sequence with $\sum_{i=0}^{n} a(i)M_{i+1}, n \geq 0$, being the associated martingale that can be seen to be a.s. convergent (See Prop. 4.4 of Bhatnagar et al. [2011a]). Thus from [Borkar, 2008, Extension 3 of Section 2.2], the result follows for $\lambda_{i,j}$s. A similar argument holds for λ_f. ■

Now, we need to show that the convergence of the algorithm is indeed to a saddle point, i.e., $\theta^* \in K^{\lambda^*}$ and $\lambda^* \in F^{\theta^*}$. This can be shown by first observing that the ODE for the λ-update corresponds to $\dot{\lambda}_{i,j}(t) = \Pi(\nabla_{\lambda_{i,j}}(L(\theta^{\lambda}, \lambda_{i,j}(t)))$, which corresponds to an ascent in the space of Lagrange multipliers and then invoking the envelope theorem of mathematical economics [Mas-Colell et al., 1995, pp. 964-966]; see remark (2) in [Bhatnagar et al., 2011a, pp 15].

10.3.3 SASOC-H

Convergence analysis of SASOC-H follows along similar lines as that of the SASOC-G algorithm as given below.

1. As in Lemma 10.4, one can see that $\bar{L}$ and $\bar{L}'$ iterations converge almost surely as follows:

$$\|\bar{L}(n) - L(\theta(n), \lambda(n))\|, \|\bar{L}'(n) - L(\theta(n) + \delta_1\Delta(n) + \delta_2\widehat{\Delta}(n), \lambda(n))\| \rightarrow 0 \text{ as } n \rightarrow \infty.$$

2. Next, we show that the parameter updates $\theta(n)$ of SASOC-H converge to a limit point of the ODE

$$\dot{\theta}(t) = \check{\Gamma}\left(-\Upsilon(\nabla^2_{\theta}L(\theta(t), \lambda))^{-1}\nabla_{\theta}L(\theta(t), \lambda)\right), \tag{10.15}$$

   where $\check{\Gamma}$ is as defined in equation (10.13).

3. The rest of the analysis of slower time-scale updates of $\lambda_{i,j}$s and λ_f, and saddle point behaviour follows from that of SASOC-G.

Lemma 10.7

$$\left\|\frac{L(\theta(n)+\delta_1\Delta(n)+\delta_2\widehat{\Delta}(n),\lambda(n))-L(\theta(n),\lambda(n))}{\delta_2\widehat{\Delta}_i(n)}-\nabla_{\theta_i}L(\theta(n),\lambda(n))\right\|\to 0 \text{ w.p. 1},$$

with $\delta_1,\delta_2\to 0$ *as* $n\to\infty\quad\forall i\in\{1,2,\ldots,|A|\times|B|\}$.

Proof: Follows from [Bhatnagar et al., 2011a, Proposition 4.10]. ■

Lemma 10.8

$$\left\|\frac{L(\theta(n)+\delta_1\Delta(n)+\delta_2\widehat{\Delta}(n),\lambda(n))-L(\theta(n),\lambda(n))}{\delta_1\Delta_i(n)\delta_2\widehat{\Delta}_j(n)}-\nabla^2_{\theta_{i,j}}L(\theta(n),\lambda(n))\right\|\to 0 \text{ w.p. 1},$$

with $\delta_1,\delta_2\to 0$ *as* $n\to\infty,\quad\forall i,j\in\{1,2,\ldots,|A|\times|B|\}$.

Proof: Follows from [Bhatnagar et al., 2011a, Proposition 4.9]. ■

Lemma 10.9

$$\left\|H_{i,j}(n)-\nabla^2_{\theta_{i,j}}L(\theta(n),\lambda(n))\right\|\to 0 \text{ w.p. 1},$$

with $\delta_1,\delta_2\to 0$ *as* $n\to\infty,\quad\forall i,j\in\{1,2,\ldots,|A|\times|B|\}$.

Proof: Follows from Lemma 10.8 applied to the Hessian update of SASOC-H. ■

Lemma 10.10

$$\left\|M(n)-\Upsilon(\nabla^2_\theta L(\theta(n),\lambda(n)))^{-1}\right\|\to 0 \text{ w.p. 1},$$

with $\delta_1,\delta_2\to 0$ *as* $n\to\infty,\quad\forall i,j\in\{1,2,\ldots,|A|\times|B|\}$.

Proof: Follows from Lemma 10.9 and [Bhatnagar, 2007, Lemma A.9]. ■

Let

$$\bar{K}^\lambda=\left\{\theta\in S:\frac{dL(\theta(t),\lambda)}{dt}=-\nabla_\theta L(\theta(t),\lambda)^T\Upsilon(\nabla^2_\theta L(\theta(t),\lambda))^{-1}\nabla_\theta L(\theta(t),\lambda)=0\right\}.$$

Theorem 10.11 *Under assumptions (A1),(A2),(A3) and (A4), in the limit as* $\delta_1,\delta_2\to$

0, $\theta(R) \to \theta^* \in \bar{K}^\lambda$ *almost surely as* $R \to \infty$.

Proof: Following Lemmas 10.4, 10.7 and 10.10, with $\delta_1, \delta_2 \to 0$, the update of the parameter θ can we re-written in vector form as

$$\theta_{n+1} = \bar{\Gamma}\left(\theta_n - b(n)\Upsilon(\nabla^2_\theta L(\theta(n), \lambda))^{-1}\nabla_\theta L(\theta(n), \lambda) + b(n)\chi_n\right)$$

with $\chi_n = o(1)$. Thus, the update of parameter θ can be viewed as a noisy Euler discretization of the ODE (10.15) using a standard approximation argument as in [Kushner and Clark, 1978a, pp. 191-196]. Note that $V^\lambda(\cdot) = L(\cdot, \lambda)$ itself serves as the associated Lyapunov function [Kushner and Yin, 1997, pp. 75] for the ODE (10.15) with stable limit points of the ODE lying within the set $\bar{K}^\lambda$. From assumption (A2), $L(\theta, \lambda)$ is assumed to be continuous. Hence over the compact set $\bar{\mathcal{D}}$, $L(\theta, \lambda)$ is uniformly bounded. Thus, from Lasalle's invariance theorem (Lasalle and Le fschetz [1961]), $\theta(n) \to \theta^* \in K^\lambda$ a.s. as $n \to \infty$. ■

10.3.4 SASOC-W

Convergence analysis of SASOC-W follows from that of SASOC-H with the following lemma in place of Lemma 10.10.

Lemma 10.12

$$\left\|M(n) - \Upsilon(\nabla^2_\theta L(\theta(n), \lambda(n)))^{-1}\right\| \to 0 \text{ w.p. 1},$$

with $\delta_1, \delta_2 \to 0$ *as* $n \to \infty$, $\quad \forall i, j \in \{1, 2, \ldots, |A| \times |B|\}$.

Proof: From Woodbury's identity, since $M(n), n \geq 1$ sequence of SASOC-W is identical to the $\Upsilon(H(n))^{-1}, n \geq 1$ sequence of SASOC-H, the result follows from Lemma 10.10. ■

10.4 Summary

In this chapter, we presented three novel discrete parameter simulation-optimization algorithms for optimizing staff allocation in the context of SS. The problem was formulated as a constrained hidden Markov cost process in Chapter 9. For solving the constrained optimization problem, we applied the Lagrange relaxation procedure and used an SPSA based scheme for performing gradient descent in the primal and at the same time, ascent in the dual, for the Lagrange multipliers. All SASOC algorithms also incorporated a smooth (generalized) projection operator that helped imitate a continuous parameter system with suitably defined transition dynamics. Using the theory of multi-timescale stochastic approximation, we presented the convergence proof of our algorithms.

The next chapter presents the results from the numerical experiments carried out to evaluate each of the algorithms. These experiments are based on real-life SS data and compare SASOC algorithms against the state-of-the-art simulation optimization toolkit OptQuest. As we will see in the next chapter, the empirical results demonstrate that the SASOC algorithms in general show overall superior performance compared to OptQuest, while also being computationally efficient.

Chapter 11

Performance Analysis of SASOC Algorithms

In this chapter, we present the simulation framework and the results from the empirical evaluation of the SASOC algorithms described in Chapter 10. The simulation experiments are performed on five real-life SS in the data-center management domain. For each of the SS, we collect operational data on work arrival patterns, service times, and contractual SLAs and feed this data into the simulation model of Banerjee et al. [2011]. As will be seen later from the simulation experiments, we observe that our algorithms show overall better performance in comparison with the state-of-the-art OptQuest optimization toolkit (Laguna [1998]). Further, our algorithms are about 25 times faster than OptQuest and have a significantly lower execution runtime - a significant advantage aiding the adoption of our SASOC algorithms for adaptive labor staffing for even short-term staffing changes.

The rest of this chapter is organized as follows: In Section 11.1, we discuss the implementation of our algorithms as well as the OptQuest algorithm. In Section 11.2 we present the performance simulation results on both flat-arrival as well as bursty-arrival SS pools. Finally, in Section 11.3 we provide the concluding remarks.

11.1 Implementation

We use the simulation framework developed in Banerjee et al. [2011] for implementing all our algorithms. A number of dispatching policies have been developed in Banerjee et al. [2011]. In particular, we study the PRIO-PULL and EDF policies for performance comparisons of the various algorithms. We implemented the following labor staffing algorithms:

- **SASOC-G**: This is a first order method that estimates $\nabla L(\theta, \lambda)$ using SPSA and is described in Section 10.2.2.
- **SASOC-H**: This is a second order Newton method that involves an explicit inverse of the Hessian matrix and is described in Section 10.2.3.
- **SASOC-W**: This is a second order Newton method, which unlike SASOC-H does away with the inversion of the Hessian matrix and leverages the Woodbury identity in order to estimate the inverse of the Hessian directly. This algorithm is described in Section 10.2.4.
- **OptQuest**: This is an algorithm that uses the state-of-the-art optimization tool-kit OptQuest. In particular, we used the scatter search based variant of OptQuest for our experiments.

OptQuest employs an array of techniques including scatter and tabu search, genetic algorithms, and other meta-heuristics for the purpose of optimization and is quite well-known as a tool for solving simulation optimization problems (Laguna [1998]). OptQuest along with several other engines from Frontline Systems won the INFORMS impact award[1] in the year 2010.

We choose five real-life SS from two different countries providing server support to IBM's customers. The five SS cover a variety of characteristics such as high vs. low workload, small vs. large number of customers to be supported, small vs. big staffing levels, and stringent vs. lenient SLA constraints. Collectively, these five SS staff more than 200 SWs with 40%, 30%, and 30% of them having low, medium, and high skill level, respectively. Also, these SS support more than 30 customers each, who make more than 6500 SRs every week with each customer having a distinct pattern of arrival depending on its business hours and seasonality of business domain. Fig 11.1(a) shows the total work hours per SW per day for each of the SS. The bottom part of the bars denotes customer SR work, i.e., the SRs raised by the customers whereas the top part of the bars denotes internal SR work, i.e., the SRs raised internally for overhead work such as meetings, report generation, and HR activities. This seggregation is important because the SLAs apply only to customer SRs. Internal SRs do not have deadlines but they may contribute to queue growth. Note that while average work volumes are significant, they may not directly correlate to SLA attainment. Fig 11.1(b) shows the effort data, i.e., the mean time taken to

[1]http://www.solver.com/press201008.htm

resolve an SR (a lognormal distributed random variable in our setting) across priority and complexity classes. As shown in Figs. 11.2(a)–11.2(b), the arrival rates for SS4 and SS5 show much higher peaks than SS1, SS2, and SS3, although their average work volumes are comparable. The variations are significant because during the peak periods, many SRs may miss their SLA deadlines and influence the optimal staffing result.

We implemented our SASOC algorithms on the simulation framework from Banerjee et al. [2011] for both the perturbed and the unperturbed simulations (See X and $\hat{X}$ computations in Algorithm 2). For our SASOC algorithms, the simulations were conducted for 1000 iterations, with each iteration having 20 simulation replications - ten each with unperturbed parameter θ and perturbed parameter $\hat{\theta}$, respectively. Each replication simulated the operations of the respective SS for a 30 day period. Thus, we set $R = 1000$ and $K = 10$ for SASOC algorithms. On the other hand, for the OptQuest algorithm, simulations were conducted for 5000 iterations, with each iteration of 100 replications of the SS.

For all the SASOC algorithms, we set the weights in the single-stage cost function $c(X_m)$, see (9.4), as $r = s = 0.5$. We thus give equal weightage to both the worker utilization and the SLA over-achievement components. The indicator variable q used in the constraint (9.6) was set to 0 (i.e., infeasible) if the queues were found to grow by 1000% over a two-week period during simulation. Each of the experiments are run on a machine with dual core Intel 2.1 GHz processor and 3 GB RAM.

The Υ operator implemented for SASOC-W can be described as follows. Let $\hat{H}$ be the Hessian update which needs to be projected. The following sequence of operations represent this projection. (i) $\hat{H} \leftarrow \frac{(\hat{H}+\hat{H}^T)}{2}$; (ii) Perform eigen decomposition on $\hat{H}$ to get all eigen-values and corresponding eigen-vectors; (iii) Project each eigen-value to $\left[\epsilon, \frac{1}{\epsilon}\right]$ where $1 > \epsilon > 0$. ϵ is chosen to be a small number so as to allow for larger range of values, but not too small to avoid singularity. The upper limit in the projection range is to avoid singularity of the inverse of the Hessian estimate; and (iv) Reconstruct $\hat{H}$ using the projected eigen-values but with same eigen-vectors. The Υ operator in the case of SASOC-H with diagonal Hessian is one that simply projects each diagonal entry to $\left[\epsilon, \frac{1}{\epsilon}\right]$. It is easy to see that the Υ operator satisfies assumption *(A4)*.

On each SS, we compare our SASOC algorithms with the OptQuest algorithm using W_{sum} and mean utilization as the performance metrics. Here $W_{sum} \triangleq \sum_{i=1}^{|A|} \sum_{j=1}^{|B|} \theta_{i,j}$ is the sum of workers across shifts and skill levels. The mean utilization here refers to a weighted average of the utilization percentage achieved for each skill level, with the weights being the fraction of the workload corresponding to each skill level.

As evident in Figs. 11.2(a) and 11.2(b), the SS pools SS1, SS2 and SS3 are characterized by a flat SR arrival pattern, whereas SS4 and SS5 are characterized by a bursty such pattern. We present and analyze the results on these pools separately, starting with the flat arrival pools in the next section.

11.2 Results

11.2.1 Flat-Arrival SS pools

Figs. 11.3(a) and 11.4(a) compare the W^*_{sum} achieved for OptQuest and SASOC algorithms using PRIO-PULL and EDF on three real life SS with a flat SR arrival pattern (see Fig. 11.2(a)). Here W^*_{sum} denotes the value obtained upon convergence of W_{sum}. On these SS pools, namely SS1, SS2 and SS3, respectively, we observe that our SASOC algorithms find a better value of W^*_{sum} as compared to OptQuest. Note in particular that on SS1, SASOC algorithms perform significantly better than OptQuest with an improvement of nearly 100%. Further, on SS2, OptQuest is seen to be infeasible whereas all the SASOC algorithms obtain a feasible and good allocation.

It is evident that SASOC algorithms consistently outperform the OptQuest algorithm on these SS pools. Further, among the SASOC algorithms, we observe that SASOC-W finds better solutions in general as compared to the other two SASOC algorithms. Further, we observe that in all our experiments that include both flat as well as bursty arrival pools, the optimal worker parameter obtained by all our SASOC algorithms is feasible, i.e., satisfies both the SLA as well as queue stability constraints.

Fig. 11.4(a) presents similar results for the case of the EDF dispatching policy. The behavior of OptQuest and SASOC algorithms here was found to be similar to that when PRIO-PULL is used with SASOC algorithms showing performance improvements over OptQuest here as well. We present the utilization percentages across different skill levels (low, medium and high) in Figs. 11.3(b) and 11.4(b). We observe mean utilization of workers is a crucial factor for a labor staffing algorithm and it is evident from Figs. 11.3(b) and 11.4(b) that SASOC algorithms exhibit a higher mean utilization of workers and hence, better overall performance in comparison to the OptQuest algorithm.

11.2.2 Bursty-Arrival SS pools

Figs. 11.5(a) and 11.6(a) compare the W^*_{sum} achieved for OptQuest and SASOC algorithms using PRIO-PULL and EDF on two real life SS with a bursty SR arrival pattern (see Fig. 11.2(b)). The utilization percentages for these pools are presented in Figs. 11.5(b) and 11.6(b). From these performance plots, we observe that OptQuest is seen to be slightly better than SASOC-G and SASOC-W when the underlying dispatching policy is PRIO-PULL, whereas in the case of EDF dispatching policy, the SASOC algorithms clearly outperform OptQuest. The execution time advantage of SASOC algorithms over OptQuest hold in the case of these pools as well.

Observation 1 SASOC algorithms are computationally efficient and hence well-suited for adaptive labor staffing as compared to OptQuest.

Computational efficiency is a significant factor for any adaptive labor staffing algorithm. For instance, if a candidate labor staffing algorithm takes too long to find the optimal staffing levels, it is not amenable for making staffing changes in a real SS. Both from the number of simulations required as well as the wall clock run time standpoints, SASOC algorithms are better than OptQuest. This is because OptQuest requires 5000 iterations with each iteration of 100 replications, whereas the SASOC algorithms require 1000 iterations of 20 replications each in order to find W^*_{sum}. This results in a $25X$ speedup for SASOC algorithms and also manifests in the wall clock runtimes of SASOC algorithms because simulation run-times are proportional to the number of SS simulations. We observe that the SASOC algorithms result in at least 10 to 15 times improvement as compared to OptQuest from the wall clock runtimes perspective. For instance, on SS1 the typical run-time of OptQuest was found to be 24 hours, whereas SASOC algorithms took less than 2.5 hours each to converge. As mentioned before, these runtimes were measured on a machine with dual core Intel 2.1 GHz processor and 3 GB RAM.

In fact, we observed in the case of SS2 that OptQuest does not find a feasible solution even after repeated runs for 5000 search iterations. Also, because OptQuest depends heavily on SLA attainments and respective confidence intervals of previous iterations, it requires higher number of replications than SASOC. Further, we observed that SASOC algorithms converge within 500 iterations in all our experiments. Thus, SASOC algorithms require 50 times less number of simulations as compared to OptQuest, while searching for the optimal SS configuration. This runtime advantage ensures that an SS manager can make staffing changes even at the granularity of every week by making use of SASOC algorithms and the same may not be possible with OptQuest due to its longer runtimes.

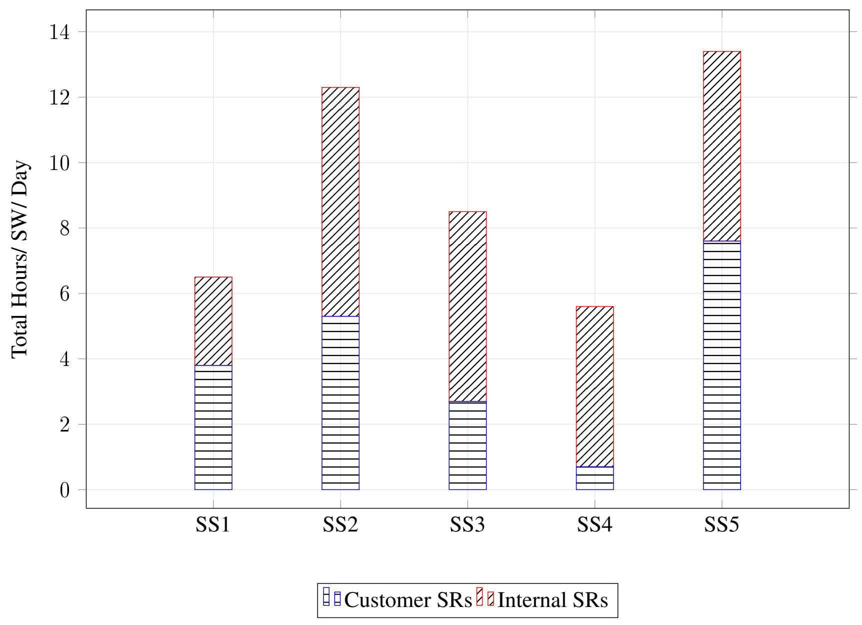
Observation 2 The parameter vector θ converges to the optimum value in all SASOC algorithms.

We observe that the parameter θ (and hence W_{sum}) converges to the optimum value for each of the SS pools considered. This is illustrated by the convergence plots of 'W_{sum} vs number of cycles' in Figs. 11.7(a) and 11.7(b). This is an important feature of SASOC as we established convergence of our algorithm in section 10.3 and the plots confirm the same. In contrast, the OptQuest algorithm is not proven to converge to the optimum even after repeated runs, as illustrated in the case of SS2 in Fig 11.3(a).

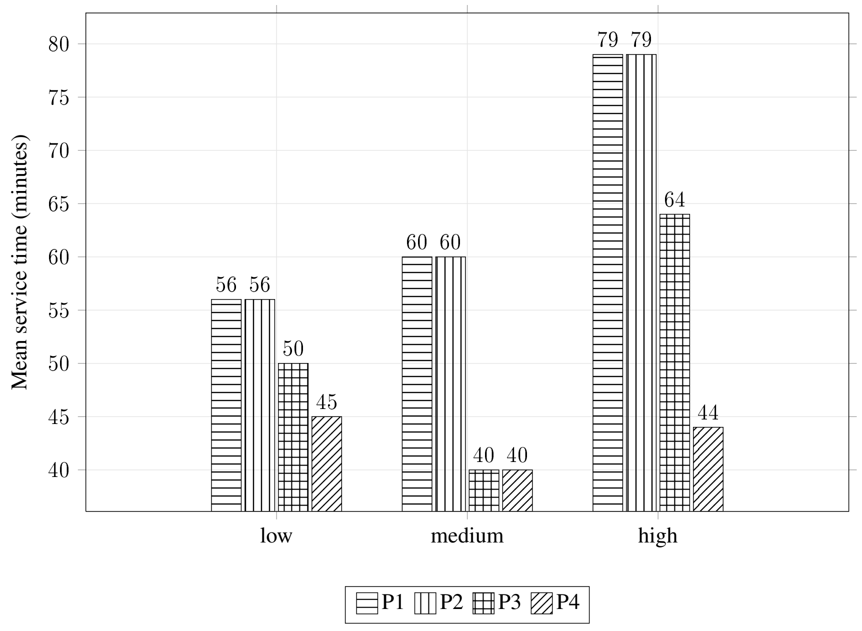
From the above performance comparisons over SS pools with flat as well as bursty SR arrival patterns, it is evident that our SASOC algorithms show overall better performance in comparison with OptQuest. Among the SASOC algorithms, we observe that the second order algorithms (SASOC-H and SASOC-W) perform better than the first order algorithm (SASOC-G) in many cases, with SASOC-W being marginally better than SASOC-H.

11.3 Summary

In this study of staff allocation algorithms in the context of service systems, presented in Chapters 9, 10 and 11, we motivated the discrete optimization problem of adaptively determining optimal staffing levels in SS. We presented three novel SASOC algorithms for optimizing staff allocation in the context of SS. We formulated the problem as a constrained hidden Markov cost process, with the aim of finding an optimum worker parameter that minimizes a certain long run cost objective, while adhering to a set of constraint functions, which are also long run averages. All SASOC algorithms are simulation based optimization methods as the single-stage cost and constraint functions are observable only via simulation and no closed form expressions are available. The single stage cost that we designed balanced the conflicting objectives of maximizing worker utilizations and minimizing the over-achievement of SLAs. For solving the constrained optimization problem, we applied the Lagrange relaxation procedure and used an SPSA based scheme for performing gradient descent in the primal and at the same time, ascent in the dual, for the Lagrange multipliers. All SASOC algorithms also incorporated a smooth (generalized) projection operator that helped imitate a continuous parameter system with suitably extended transition dynamics. Using the theory of multi-timescale stochastic approximation, we presented the convergence proof of our algorithms. Numerical experiments were performed to evaluate each of the algorithms based on real-life SS data against the state-of-the-art simulation optimization toolkit OptQuest. SASOC algorithms in general showed overall superior performance compared to OptQuest, as they (a) exhibited more than an order of magnitude faster convergence than OptQuest, (b) consistently found solutions of good quality and in most cases better than those found by OptQuest, and (c) showed guaranteed convergence even in scenarios where OptQuest did not find feasibility after repeated runs for 5000 iterations. Given the quick convergence of SASOC algorithms (in minutes), they are particularly suitable for adaptive labor staffing where a few days of optimization run like in OptQuest would fail to keep up with the changes. By comparing the results of the SASOC algorithms on two independent dispatching policies, we showed that SASOC's performance is independent of the operational model of SS. As future work, one may consider single-stage cost function enhancements that include worker salaries as well as other relevant monetary costs.

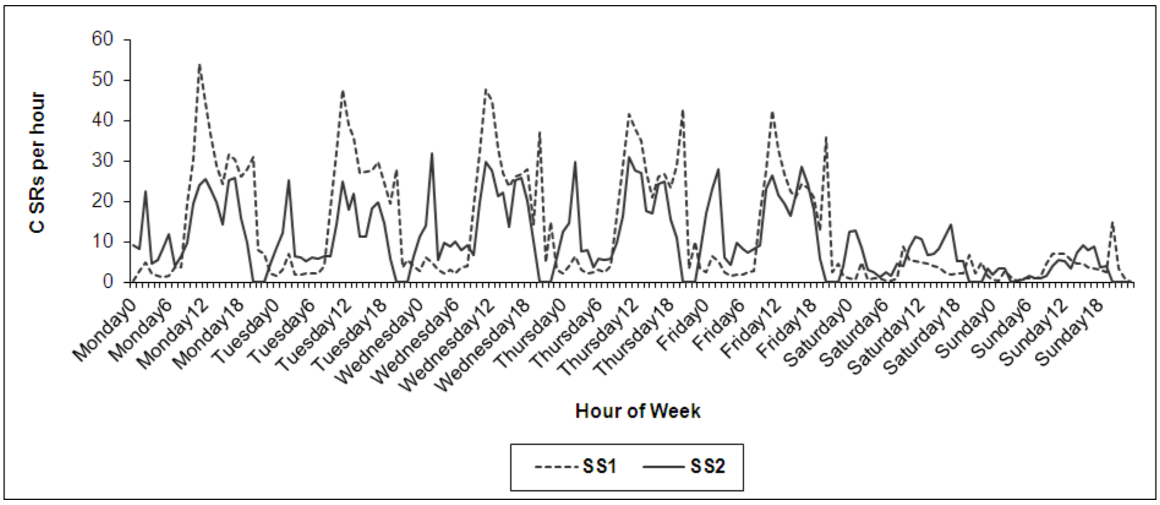


(a) Total work volume statistics for each SS

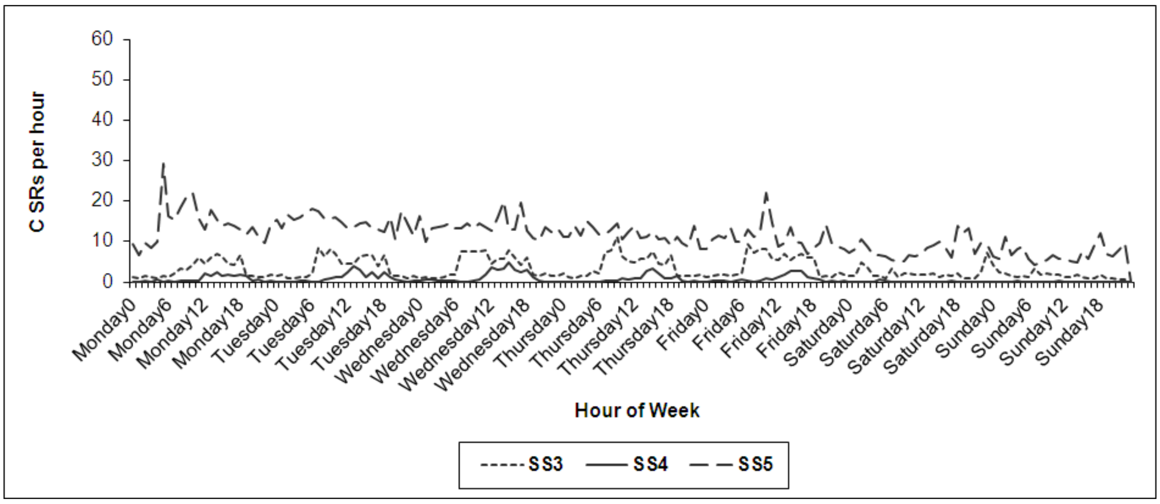


(b) Estimated mean service times for a SS

Figure 11.1: Characteristics of the service systems used for simulation

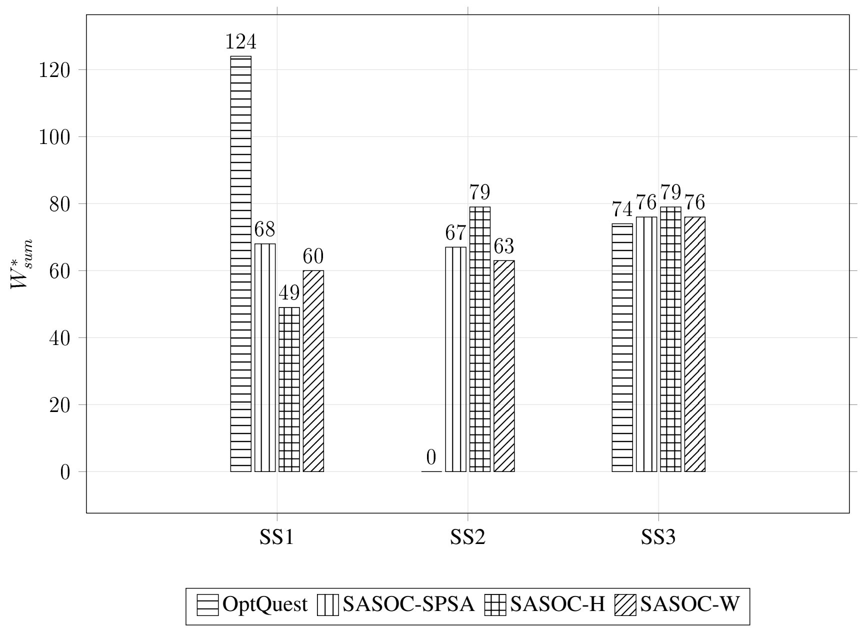


(a) SS1 and SS2 work arrival pattern

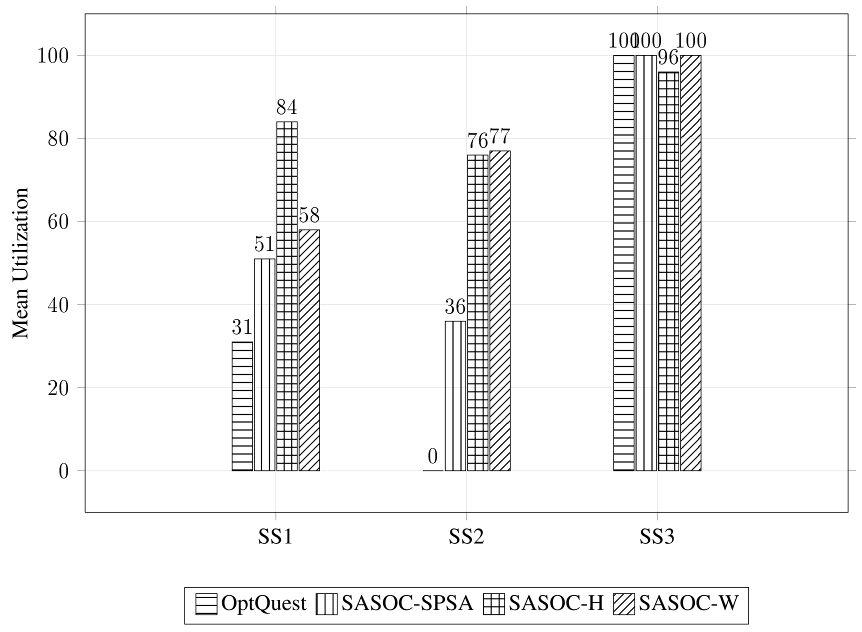


(b) SS3, SS4 and SS5 work arrival pattern

Figure 11.2: Work arrival patterns over a week for each SS

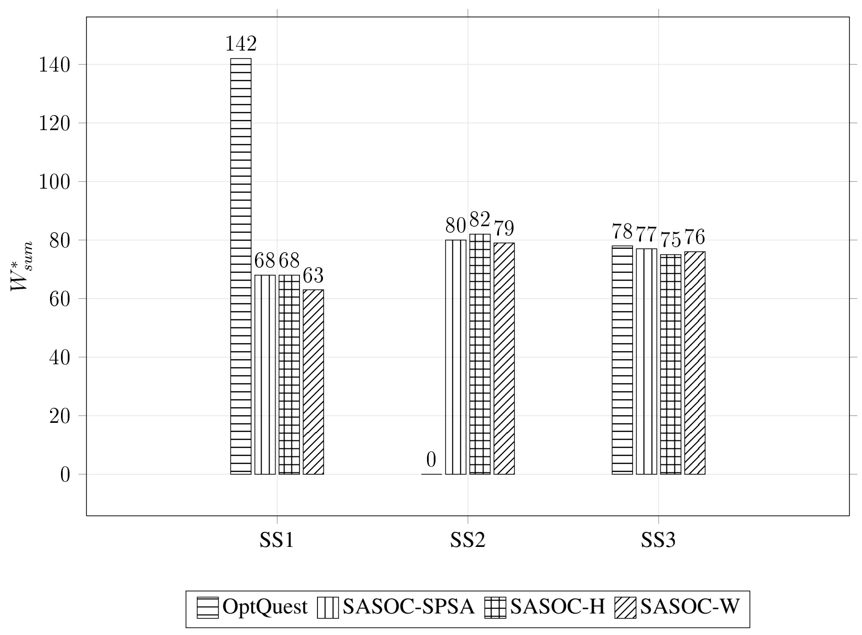


(a) W^*_{sum} achieved

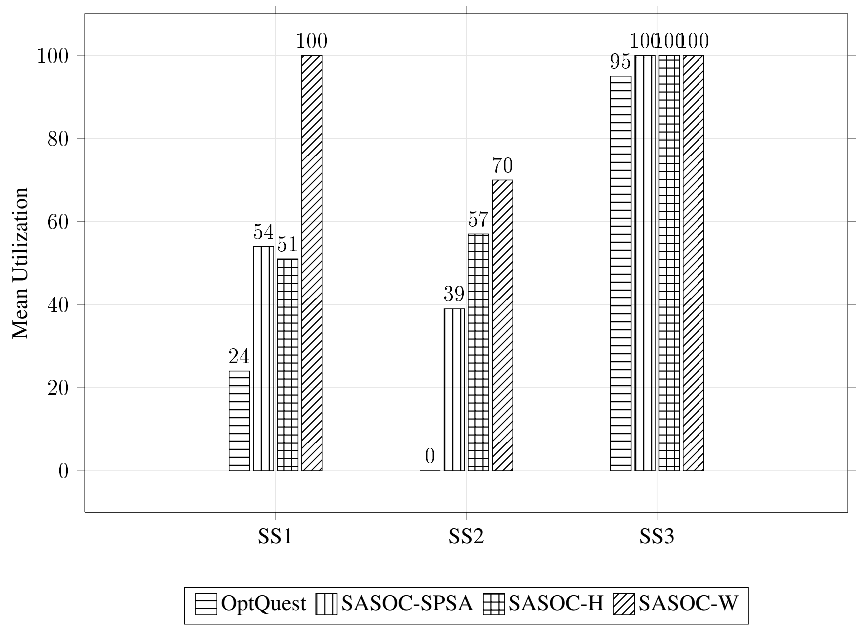


(b) Utilization of the workers across skill levels

Figure 11.3: Performance of OptQuest and SASOC for PRIO-PULL dispatching policy on SS1, SS2 and SS3 (*Note:* OptQuest is infeasible over SS2)

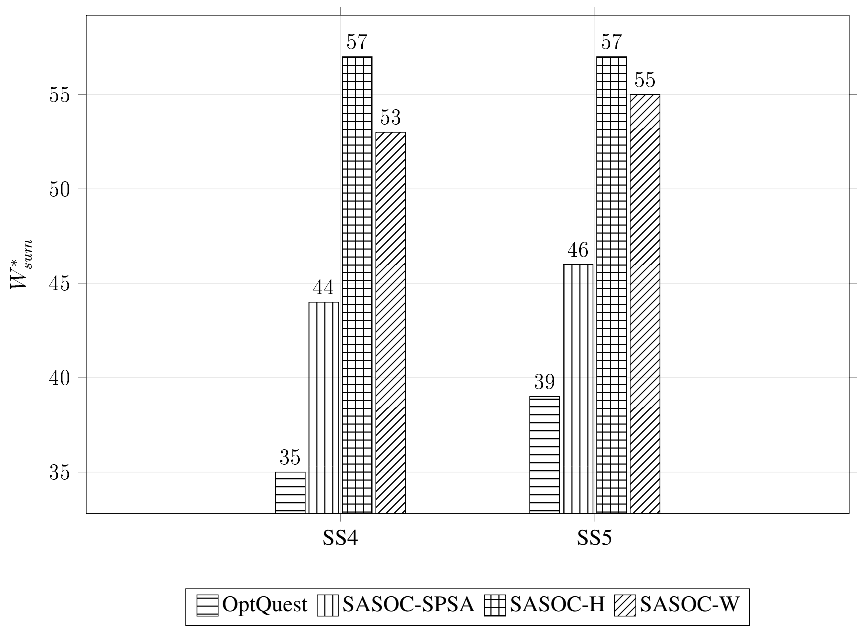


(a) W^*_{sum} achieved

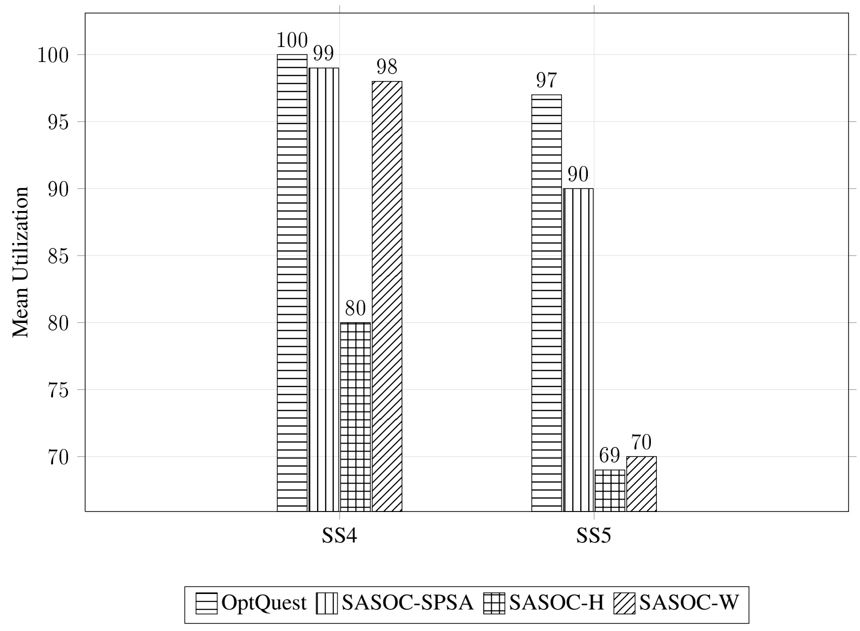


(b) Utilization of the workers across skill levels

Figure 11.4: Performance of OptQuest and SASOC for EDF dispatching policy on SS1, SS2 and SS3 (*Note:* OptQuest is infeasible over SS2). The mean utilization values have been rounded to nearest integer.

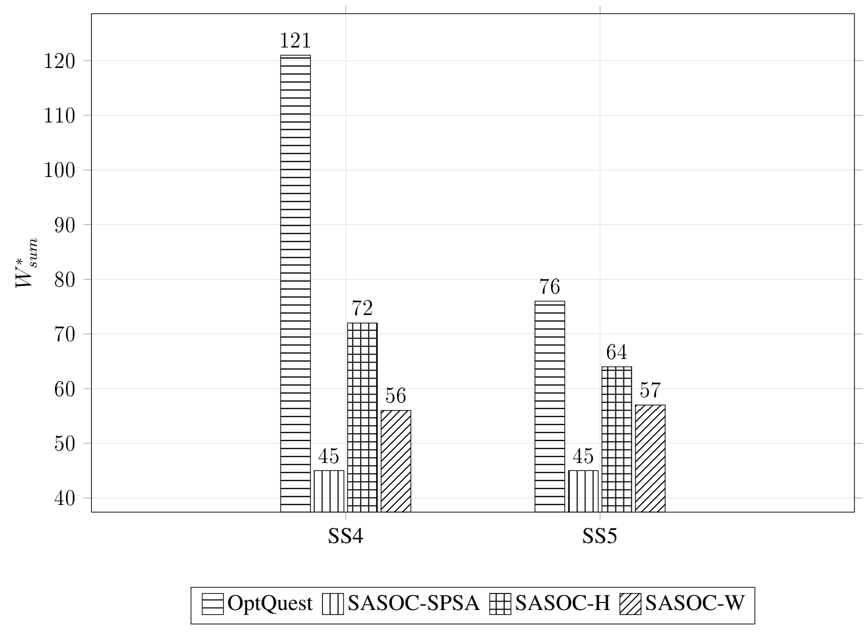


(a) W^*_{sum} achieved

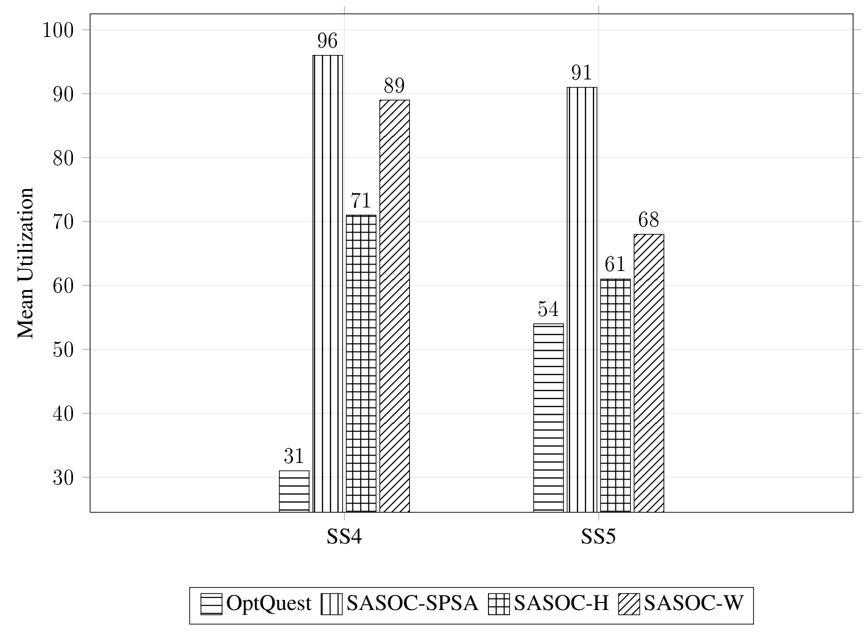


(b) Utilization of the workers across skill levels

Figure 11.5: Performance of OptQuest and SASOC for PRIO-PULL dispatching policy on SS4 and SS5

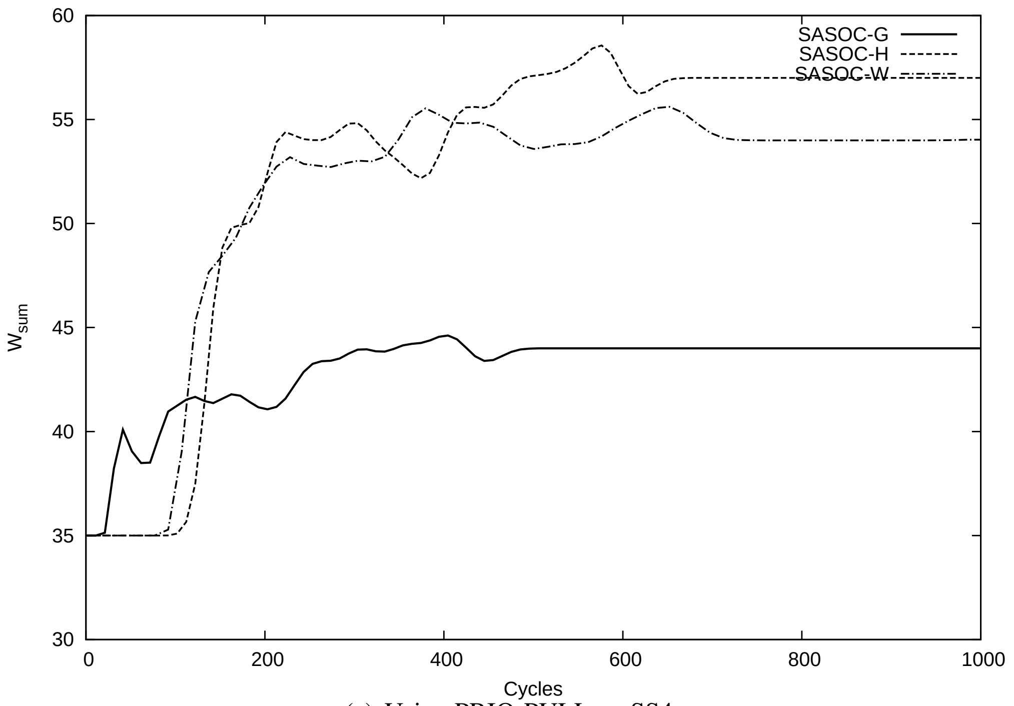


(a) W^*_{sum} achieved

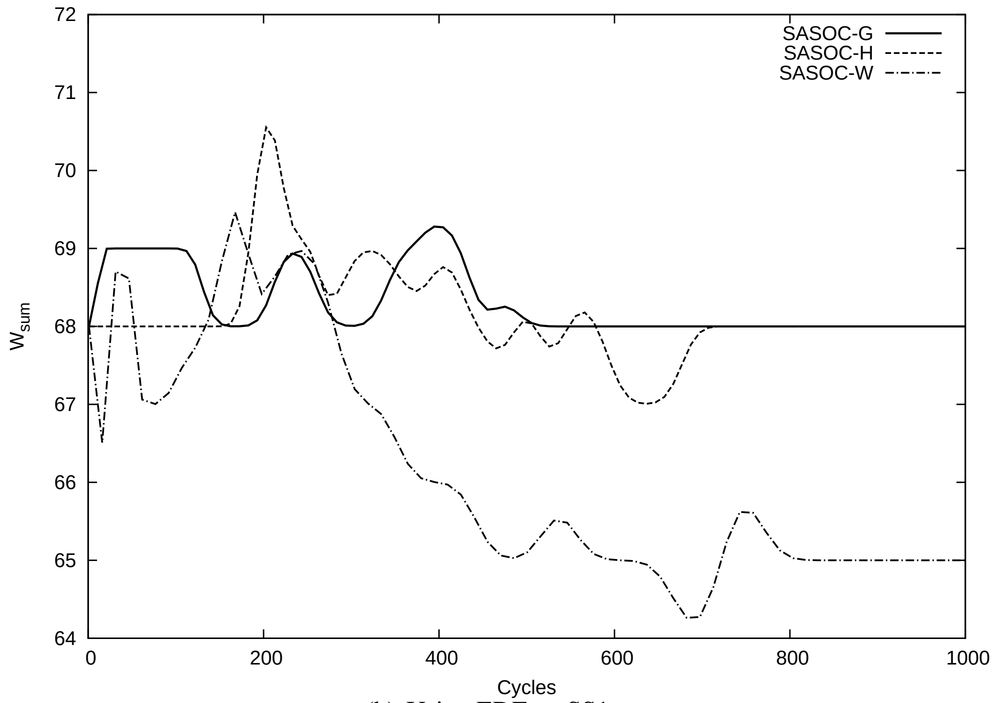


(b) Utilization of the workers across skill levels

Figure 11.6: Performance of OptQuest and SASOC for EDF dispatching policy on SS4 and SS5. The mean utilization values have been rounded to nearest integer.



(a) Using PRIO-PULL on SS4



(b) Using EDF on SS1

Figure 11.7: Convergence of W_{sum} as a function of number of cycles for different SASOC algorithms - Illustration on SS1 and SS4 for two dispatching policies

Part III

Sensor Networks

We study the problem of tracking a single object moving randomly through a dense network of wireless sensors such that the entire sensing region has no discontinuities. However, we wish to conserve energy by optimizing the sleep times of the sensors in a way that does not compromise on the tracking quality. We remove the waking channel assumption, i.e., a sleeping sensor can be communicated with and model the sleep–wake scheduling problem as a partially-observable Markov decision process (POMDP). We propose novel RL-based algorithms - with both long-run discounted and average cost objectives - for solving this problem. All our algorithms incorporate function approximation and feature-based representations to handle the curse of dimensionality. Further, the feature selection scheme used in each of the proposed algorithms intelligently manages the energy cost and tracking cost factors, which in turn, assists the search for the optimal sleeping policy. The results from the simulation experiments suggest that our proposed algorithms perform better than two recently proposed algorithms from Fuemmeler and Veeravalli [2008], Fuemmeler et al. [2011].

Chapter 12

Sleep–Wake Scheduling for Object Tracking

Considering the potential range of applications and the low deployment and maintenance costs, a lot of research attention has gone into the design of Wireless Sensor Networks or WSNs, specifically on the power management or energy efficiency aspect of it. A typical WSN consists of tiny compact sensors that are energy-limited, low-powered inexpensive devices. These sensors, when deployed in a real environment perform network discovery and establish routing paths to form a connected network, with little or no external intervention. They sense a physical phenomena such as temperature, pressure, etc., collate the sensing data, and send it to a central node or fusion center.

We consider a WSN application scenario where the objective is to track an object moving through a given area covered by sensors. This is illustrated in the one-dimensional and two-dimensional networks described in Figures 12.1 and 12.2 respectively. The movement of the object is assumed to be random with a given distribution (an assumption that we relax later). In this setting, it is necessary to optimize the energy usage of each sensor through sleep-wake cycling and hence increase the lifetime of the network. The sleep–wake scheduling algorithm has to balance two conflicting objectives - minimize the energy usage of the individual sensors and track the object to a reasonable accuracy.

The setting we consider is similar to Fuemmeler and Veeravalli [2008], Atia et al. [2010], as we also remove the waking channel assumption and assume a sleep sensor cannot be communicated with. However, while Fuemmeler and Veeravalli [2008], Atia et al. [2010] proposed sub-optimal algorithms, we adopt the reinforcement learning (RL) approach to develop sleep–wake scheduling algorithms that find the ‘optimal’ sleeping policy. RL has the advantages of being model-free (no model of the system is assumed) and RL algorithms are in general easy to implement, online and have proven convergence guarantees.

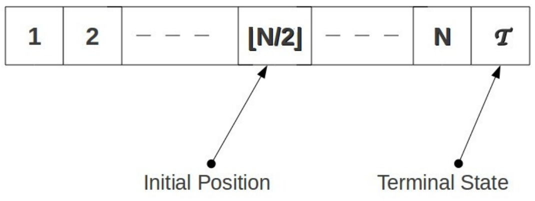


Figure 12.1: Field of sensors and the movement of object in a one-dimensional network

Our contributions

- We consider the problem of sleep-wake scheduling for object tracking using a wireless sensor network and formulate this as a partially-observed Markov decision process (POMDP), similar to Fuemmeler and Veeravalli [2008], Fuemmeler et al. [2011]. However, unlike the total cost objective considered in Fuemmeler and Veeravalli [2008], we consider both discounted as well as average cost objectives in the infinite horizon.
- In the discounted cost POMDP setting, we propose two RL-based algorithms with function approximation for solving this problem.
  - The first sleep–wake scheduling algorithm proposed is based on Q-learning with linear function approximation, and

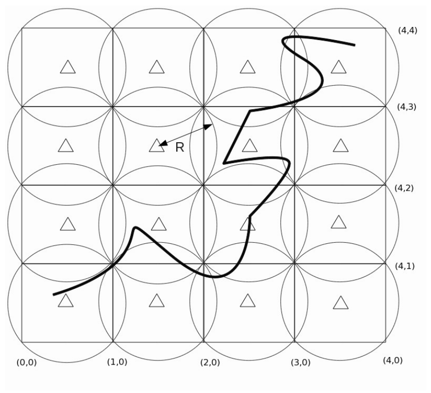


Figure 12.2: Field of sensors and the movement of object in a two-dimensional network

  - the second sleep–wake scheduling algorithm proposed is a two timescale convergent Q-learning algorithm, which also uses linear function approximation. This algorithm is adapted from Bhatnagar and Lakshmanan [2012] and unlike the above algorithm, has proven convergence to the optimal policy.

  To the best of our knowledge, we are the first to propose reinforcement learning with function approximation for sleep–wake control.

- In the average cost POMDP setting, we propose two RL-based algorithms with function approximation.

  - The first algorithm uses the Q-learning analogue for the average cost setting with function approximation.
  - The second algorithm proposed is a novel two timescale algorithm that performs on-policy Q-learning, while employing function approximation. This algorithm uses policy gradient using a one-simulation SPSA estimate on the faster timescale, while the Q-value parameter (arising out of linear function approximation architecture for the Q-values) is updated in an on-policy TD-like fashion on the slower timescale. Further, on the slower timescale, the average cost is also estimated and used for updating the Q-value parameter. This algorithm has been presented (in any setting) for the first time and so we present a rigorous proof of convergence of this algorithm.

- Function approximation and feature-based representations are incorporated in each of our algorithms to handle the curse of dimensionality associated with high-dimensional state spaces (as in the case of the sleep–wake control problem considered in this paper). Further, these techniques also considerably simplify the implementation of our algorithms, owing to the computational (in terms of both space and time) advantages of function approximation. The feature selection scheme used in each of the proposed algorithms intelligently manages the energy cost and tracking cost factors, which in turn, assists the search for the optimal sleeping policy.

- Using a bottom-up approach, we study our algorithms first on a one-dimensional (1d) setting with no overlaps to establish the usefulness of the RL-based sleep–wake scheduling algorithms. Next, we consider a two-dimensional (2d) setting with overlaps for the performance comparisons. We also implement two recently proposed algorithms – FCR and Q_{MDP} proposed in Fuemmeler and Veeravalli [2008], Fuemmeler et al. [2011] – for the sake of comparison. Empirically, our algorithms are seen to be easily implementable over all simulation settings and more importantly, they converge rapidly with a short initial transient phase. Further, the proposed algorithms consistently outperform the FCR and Q_{MDP} algorithms from the perspective of both the number of sensors as well as the tracking error metrics.

- We also incorporate an online estimation procedure for the transition probability matrix that captures the movement of the object. This is motivated by the fact that the distribution of the object movement is often unknown in practice and the sleep–wake scheduling algorithms proposed above as well as those in the literature (cf. Fuemmeler and Veeravalli [2008], Fuemmeler et al. [2011]) base their decisions on the probability vector that specifies the object's location. We combine the estimation procedure for the distribution of the object's movement with the algorithms outlined above using multi-timescale stochastic approximation. From the numerical experiments we observe that the procedure for estimating the transition probability matrix, converges to the true transition probability matrix in all the network settings considered.

The rest of this chapter is organized as follows: In Section 12.1, we review relevant literature concerned with recent developments in the area of sleep–wake scheduling using WSNs. In Section 12.2, we formulate the sleep-wake scheduling problem as a POMDP in the discounted cost setting. In the next chapter, we present novel RL-based sleep-wake scheduling algorithms for the infinite horizon discounted as well as average cost objectives and also present the estimation procedure for the transition probability matrix that describes the object's movement. Later, in Chapter 14, we present the results from numerical experiments that include cases when the distribution of object movement is known as well as when the same is unknown.

12.1 Literature Review

We now review some of the sleep-wake scheduling algorithms previously proposed in the literature. In Premkumar and Kumar [2008], the authors develop a Markov decision process (MDP) model for intrusion detection in a WSN. They present three sleep–wake scheduling algorithms to control the number of sensors in the wake state. In Liu and Elhanany [2006], the authors propose a MAC protocol for WSNs that is based on reinforcement learning (RL). The scheduling of the sleep and active periods of sensor nodes is done following a

Q-learning algorithm that attempts to maximize the throughput while being energy efficient. In Fuemmeler and Veeravalli [2008], the sleep-wake scheduling problem is formulated as a partially observable Markov decision process (POMDP) and several sub-optimal algorithms are proposed for its solution. Sub-optimal algorithms instead of dynamic programming techniques are used in order to tackle the curse of dimensionality. In Atia et al. [2010], an overlapping sensing range has been considered while following a similar approach as in Fuemmeler and Veeravalli [2008] while continuous Gaussian observations have been incorporated in Fuemmeler et al. [2011]. In Gui and Mohapatra [2004], the authors have proposed two sleep-wake scheduling algorithms for single object tracking with waking channel assumption. A dynamic clustering mechanism has been used for balancing energy cost and tracking error in Jin et al. [2006]. In Jiang et al. [2008], a sleep scheduling algorithm based on the target's moving direction has been proposed and a particle filter based distributed target tracking scheme has been designed in Jiang and Ravindran [2011] with low communication costs. In Dousse et al. [2006], the authors study the time delay for a mobile intruder to be detected by the sensor with a connected path to the sink and contrast it with the detection time by a sensor network with arbitrary connectivity. Pal et al. [2009] provide a cooperative node-to-node activation scheme for sleep-wake scheduling for object tracking with waking channel assumption. Lai and Paschalidis [2008] study sleep-wake scheduling of nodes to preserve energy while ensuring that network latency is as low as possible. In Li et al. [2011], object tracking using partial information broadcasting scheme among sensor nodes for decentralized tracking has been studied. Hong et al. [2009] proposed a prediction based target tracking scheme in which it is not necessary for the sensors to be aware of their own location. An actor-critic based RL approach has been used for energy conservation in embedded sensor network by Sridhar et al. [2007]. A Q- learning based approach has been used for a different problem setting for sleep scheduling in WSNs in Niu [2010] and for routing problem in Ouferhat and Mellouk [2007]. Finding an efficient sleep-wake policy for the sensors while maintaining good tracking accuracy by solving an MDP has been studued in Beccuti et al. [2009]. Naderan et al. [2009] present a survey of the state-of-the art techniques used in mobile target tracking applications while Anastasi et al. [2009] survey energy conservation techniques in WSNs. Di and Joo [2007] provide a survey of machine learning techniques used in WSNs. RL based techniques have also been used in problems with decentralized settings, see Shah and Kumar [2007], Mihaylov et al. [2011].

12.2 The Setting

We study the problem of tracking a single object moving through a dense network of sensors with the sensors totally covering the sensing region. We start off with a simple one-dimensional network as shown in Figure 1. The sensing range of the sensors is limited and non-overlapping with that of other sensors, and completely covers the entire region of the network. We then also consider a two-dimensional grid of sensors with overlapping sensing regions as shown in Figure 2. The object follows a random movement pattern which can be assumed to be either known *a priori* or estimated on-line. Clearly, having sleeping sensors in the network that cannot be woken up can result in tracking errors. Thus, we have to design sleeping policies for the sensors to optimize the trade off between energy savings and tracking errors. We assume that there is a central controller which does this optimization and assigns sleep times to sensors based on the observations sent by those sensors that are awake under the constraint that the sensors completely cover the sensing region.

We formulate this problem in the setting of partially observable MDPs and develop two RL algorithms for solving the same. We now present the adaptation of the sleep–wake scheduling problem to the MDP framework.

12.2.1 Sleep–wake control POMDP

1-d model (without overlap)

First we consider a network with N sensors placed along a line, as shown in Fig. 12.1. We assume for simplicity that the sensors cover the area of interest without overlaps[1]. Each sensor can be in either of the two states: awake or sleep. A sensor in the sleep state consumes negligible energy and object sensing can be performed only in the awake state. The control center collects sensing information at discrete time instants (denoted $n, n+1, \ldots$) and decides on the sleeping policy for the sensors that are awake. When a sleeping sensor wakes up, it remains awake for at least one time unit. We assume that the sleep time received by a sensor is upper-bounded by Λ.

The object movement is described by a Markov chain whose state is the current location of the object, to within the accuracy of a sensing region. There is one additional state in this Markov chain, namely $\mathcal{T}$ that represents the object's departure from the network and is an absorbing state. The transition probability matrix of this Markov chain is given by an

[1]This is a feasible assumption in one-dimensional networks because if the maximum sensing radius of each sensor is R, then we can place two sensors $2R$ distance apart in order to have a consistent model.

$(N+1) \times (N+1)$ matrix $\mathbf{P}$ such that P_{ij} is the probability of the object being in state j in the next time step given that it is currently in state i. We assume that there is a positive probability that an object moves from any region to the terminal state. Now, we formulate the problem as a POMDP, where the states, actions and costs are given as follows:

State, Action and Observation

We define the state as

$$\mathbf{s}_k = (l_k, \mathbf{r}_k), \text{ where} \tag{12.1}$$

- l_k refers to the location of the object at instant k and can take values $1, \ldots, n, T$.
- $\mathbf{r}_k = (r_k(1), ..., r_k(N))$ where $r_k(i)$ denotes the remaining sleep time of the i^{th} sensor.

While the object transitions from one location to another as specified by the transition probability matrix P, the remaining sleep time vector $\mathbf{r}_k$ evolves as follows

$$r_{k+1}(i) = (r_k(i) - 1)\mathcal{I}_{\{r_k(i)>0\}} + a_k(i)\mathcal{I}_{\{r_k(i)=0\}}, \tag{12.2}$$

In (12.2), the remaining sleep time of sensor i (if it is in sleep state at time k) is its current sleep time decremented by one. Otherwise, the remaining sleep time is the action chosen $(a_k(i))$ for sensor i at the current step (k), see below for definition of action $a_k(i)$. Note that $\mathcal{I}$ denotes the indicator function in (12.2).

However, it is not always possible to track the object at each time instant as the sensors at the object's location at a given time instant may be in the sleep state. Hence, we formulate this problem as a POMDP with imperfect state information. Following the notation from Fuemmeler and Veeravalli [2008], the observation z_k available to the control center is given by

$$z_k = \begin{cases} l_k & \text{if } l_k \neq \mathcal{T} \text{ and } r_k(l_k) = 0, \\ \kappa & \text{if } l_k \neq \mathcal{T} \text{ and } r_k(l_k) > 0, \\ \mathcal{T} & \text{if } l_k = \mathcal{T}. \end{cases} \tag{12.3}$$

In the above, κ denotes a special value signifying that the object is not detected.

The action $\mathbf{a}_k$ at instant k is the vector of chosen sleep times of the sensors. However if the i^{th} sensor is asleep, we make sure that $a_k(i) = r_k(i)$. Each $a_k(i)$ can take values between 0 and Λ. Thus, the total information available to the control center at instant k is given by

$$I_k = (z_0, \ldots, z_k, a_0, \ldots, a_{k-1}),$$

where $I_0 = z_0$ denotes the initial state of the system. The action a_k specifies the chosen sleep configuration of the n sensors and is a function of I_k. As pointed out in Fuemmeler et al. [2011], in the above POMDP setting, a sufficient statistic is $\mathbf{p}_k = P(l_k | I_k)$. In other words, $\mathbf{p}_k = (p_k(1), ..., p_k(N), p_k(\mathcal{T}))$ is the distribution at time step k of the object being in one of the locations $1, 2, ..., N, \mathcal{T}$ and evolves according to

$$\boxed{\mathbf{p}_{k+1} = \mathbf{p}_k \mathbf{P} \mathcal{I}_{\{r_{k+1}(l_{k+1})>0\}} + \mathbf{e}_{l_{k+1}} \mathcal{I}_{\{r_{k+1}(l_{k+1})=0\}} + \mathbf{e}_{\mathcal{T}} \mathcal{I}_{\{l_{k+1}=\mathcal{T}\}}. \qquad (12.4)}$$

In the above, $\mathbf{e}_b$ denotes the degenerate probability distribution with 1 at the b^{th} position and 0 elsewhere and l_{k+1} denotes the location of the object at time $k+1$.

Single-stage cost

The single stage cost function is defined such that if the object is still within the network at time k, then the cost incurred is the energy consumed by the sensors that are awake (i.e., with remaining sleep time of 0) and a constant times an additional cost if the object is currently at a location where the sensor is asleep (i.e., has a remaining sleep time > 0). The single stage cost is given by

$$\boxed{g(\mathbf{s}_k, \mathbf{a}_k) = \mathcal{I}_{\{l_k \neq \mathcal{T}\}} \left(\sum_{\{i: r_k(i)=0\}} c + \mathcal{I}_{\{r_k(l_k)>0\}} \mathcal{K} \right), \qquad (12.5)}$$

where $c > 0$ is the cost per each sensor in the wake state, l_k is the location of the object at time step k and $\mathcal{K}$ is a constant. The constant $\mathcal{K}$ is chosen such that a tracking error results in a significantly higher cost (contributed by the second term in brackets above), in comparison to the energy savings achieved by putting sensors to sleep (contributed by the first term in brackets above). Note that our cost formulation differs from Fuemmeler and Veeravalli [2008], Fuemmeler et al. [2011], since a tracking error resulted in a cost of 1 there.

Remark 10 *The single stage reward is the negative of the single stage cost, that is,* $r(\mathbf{s}_k, \mathbf{a}_k) = -g(\mathbf{s}_k, \mathbf{a}_k)$*. All the algorithms that we present next attempt to find the optimal sleeping policy that maximizes the long-run discounted sum of rewards* $r(\cdot, \cdot)$*, which is equivalent to minimizing the long-run discounted sum of costs* $g(\cdot, \cdot)$ *given by* (12.5)*.*

Remark 11 *As stated before, we consider the infinite horizon discounted cost framework, i.e., we attempt to minimize the long-run discounted sum of the single-stage costs* $g(\cdot,\cdot)$. *In this setting, the discount factor* γ *plays a crucial role. A lower* γ *serves to discount the future costs more, whereas a higher value of* γ *puts more emphasis on the future costs. We set* $\gamma = 0.9$ *in all our experiments.*

2-d model (with overlap)

In this case, due to overlapping sensing regions, the object can be observed by multiple sensors if they are in the wake state. The observation z_k available to the central controller in this case evolves according to

$$z_k = \begin{cases} l_k & \text{if } l_k \neq \mathcal{T} \text{ and } \exists j \in \mathcal{C}(l_k) \text{ such that } r_j(l_k) = 0, \\ \kappa & \text{if } l_k \neq \mathcal{T} \text{ and } r_j(l_k) > 0, \forall j \in \mathcal{C}(l_k) \\ \mathcal{T} & \text{if } l_k = \mathcal{T}. \end{cases} \tag{12.6}$$

In the above, $\mathcal{C}(k)$ denotes the set of sensors that observe the location k and κ, as before, denotes a special value to indicate that the object is not detected. The single-stage cost $g(\cdot)$ is redefined as follows:

$$\boxed{g(\mathbf{s}_k, \mathbf{a}_k) = \mathcal{I}_{\{l_k \neq \mathcal{T}\}} \left(\sum_{\{i : r_k(i) = 0\}} c + \mathcal{I}_{\{r_j(l_k) > 0, \forall j \in \mathcal{C}(l_k)\}} \mathcal{K} \right), \qquad (12.7)}$$

The states, actions and the single-stage cost function for both the settings discussed above constitute a POMDP. While the formulation here is similar to that of Fuemmeler et al. [2011], the objective in our case is to minimize the long-run discounted sum of the single-stage costs (12.5)–(12.7), unlike the total cost setting of Fuemmeler et al. [2011] (that is equivalent to the stochastic shortest path problem). Further, using the sufficient statistic $\mathbf{p}_k$, we develop two novel RL-based sleep–wake scheduling algorithms – $\mathcal{QSA}$ and $\mathcal{TQSA}$ – to find the optimal sleeping policy. As we demonstrate later in our experiments, our algorithms outperform the FCR and Q_{MDP} algorithms proposed in Fuemmeler and Veeravalli [2008] as well as Fuemmeler et al. [2011]. Further, $\mathcal{TQSA}$ possesses convergence guarantees to the optimal sleeping policy, while not making assumptions on the state evolution (12.4).

Chapter 13

Sleep–Wake Scheduling Algorithms

While solutions to MDP can be obtained using well-known DP methods such as value iteration and policy iteration (see Bertsekas [1995]), these require the specification of transition dynamics which is often unavailable. Reinforcement learning (RL) algorithms are simulation based methods that find the optimal policy, even in the absence of precise knowledge of the transition dynamics (see Kaelbling et al. [1996], Sathiya Keerthi and Ravindran [1994], Sutton and Barto [1998c]). We first develop a sleep–wake scheduling algorithm that is based on the well-known Q-learning algorithm, see Watkins and Dayan [1992b] and Tsitsiklis [1994b]. While Q-learning finds the optimal policy that maximizes a long run discounted reward (2.10) even in the absence of a system model, it requires full state representations and hence is useful only on manageable-sized state spaces. To alleviate this problem on high-dimensional state and action spaces, we incorporate feature-based function approximation methods. The resulting algorithm is henceforth referred to as $\mathcal{QSA}$ and is described in Section 13.1.1. Next, we develop a sleep–wake scheduling algorithm that employs two-timescale stochastic approximation and performs on-policy updates in a Q-learning-like fashion. This algorithm has been adapted from Bhatnagar and Lakshmanan [2012] and solves the off-policy problem that arises when the original Q-learning algorithm is combined with function approximation. The resulting algorithm in our setting is henceforth referred to as $\mathcal{TQSA}$.

Background on the Q-learning algorithm and its function approximation variant were presented in Section 2.2. In the next section, we present the algorithms for discounted cost objective and later in Section 13.2 we address the average cost setting.

13.1 Sleep–wake scheduling for discounted reward objective

13.1.1 Q-learning based Sleep–wake Scheduling Algorithm ($\mathcal{QSA}$)

While Q-learning does not require knowledge of the system model, it does suffer from the computational problems associated with huge state and action spaces as it stores the $Q(\mathbf{s}, \mathbf{a})$ values in a look-up table and requires updates of all $Q(\mathbf{s}, \mathbf{a})$ values at each step for convergence. In our setting, this algorithm becomes intractable as the state-action space is huge. Even when we quantize probabilities as multiples of 0.01, and with 7 sensors, the cardinality of the state-action space $|S \times A(S)|$ is approximately $100^8 \times 4^7 \times 4^7$ if we use κ of 3. So, it is not practically feasible to update and store Q-values for such large numbers of state-action tuples while running the Q-learning algorithm from (2.13). The situation gets aggravated when we consider larger sensing regions (and hence higher number of sensors). To deal with this problem of the curse of dimensionality, we develop a feature based Q-learning algorithm as in Chapter 5.

We now describe our Q-learning based algorithm with linear function approximation for finding the optimal sleeping policy. While the full state Q-learning algorithm in (2.13) cannot be used on even moderately sized sensing regions, its function approximation based variant uses feature based representations and is computationally efficient both in terms of space and time.

Update rule of $\mathcal{QSA}$

Let $\mathbf{s}_k$ and $\mathbf{s}_{k+1}$ denote the state at instants k and $k+1$, respectively. Let θ_k be the k^{th} update of the parameter. Our algorithm works with a single online simulation trajectory of states and actions, and updates θ according to

$$\boxed{\theta_{k+1} = \theta_k + \alpha(k)\sigma_{\mathbf{s}_k,\mathbf{a}_k}\left(r(\mathbf{s}_k, \mathbf{a}_k) + \gamma \max_{\mathbf{b}\in A(\mathbf{s}_{k+1})} \theta_k^T \sigma_{\mathbf{s}_{k+1},\mathbf{b}} - \theta_k^T \sigma_{\mathbf{s}_k,\mathbf{a}_k}\right), \qquad (13.1)}$$

where θ_0 is set arbitrarily. In (13.1), the action $\mathbf{a}_k$ is chosen in state $\mathbf{s}_k$ according to an $\epsilon-$greedy policy, i.e., with a probability of $(1-\epsilon)$, a greedy action given by $arg\max_{b\in A(\mathbf{s}_k)}(\theta_k^T \sigma_{\mathbf{s}_k,\mathbf{b}})$ is chosen and with probability ϵ, another action is randomly chosen.

Feature selection

The features are chosen as follows:

$$\sigma_{s_k,a_k} = (\sigma_{s_k,a_k}(1), ..., \sigma_{s_k,a_k}(N))^T, \tag{13.2}$$

where $\sigma_i(k), i \leq N$ is the feature value corresponding to sensor i. These are defined as follows:

$$\text{Let } \rho_k = c(\Lambda - a_k(i)) - \sum_{j=1}^{a_k(i)} [\mathbf{pP}^j]_i. \tag{13.3}$$

Then,

$$\sigma_{\mathbf{s_k},\mathbf{a_k}}(i) = \begin{cases} V \times sgn(\theta_k(i)) & \text{if } 0 \leq |\rho_k| \leq \epsilon, \\ -V \times sgn(\theta_k(i)) & \text{otherwise .} \end{cases} \tag{13.4}$$

The first term in (13.3) represents the energy cost incurred when sensor i is assigned a sleep time of $a(i)$, while the second component there represents the tracking error incurred. In particular, the second component sums the probability of the object being in location i at each of the future time instants until $a(i)$ time slots. The idea behind the feature selection scheme (13.4) then is to (a) first prune the actions so as to shortlist only those actions that ensure that the energy cost is ϵ-close to the tracking error and (b) select an action, among the ϵ-optimal actions that minimizes the approximate Q-value.Thus, the feature selection scheme intelligently balances the energy cost and tracking error component, resulting in the choice of an action that effectively ensures that as many sensors are kept awake while incurring minimal energy cost. Note that the choice of features is identical for both this algorithm as well as the $\mathcal{TQSA}$ algorithm described in the next section.

Algorithm 3 gives the structure of the $\mathcal{QSA}$ algorithm.

13.1.2 Two-timescale Q-learning based sleep–wake scheduling algorithm ($\mathcal{TQSA}$)

Although Q-learning with function approximation has been successful in many cases, it is theoretically difficult to prove that it converges to the optimal solution. In fact, there have been instances in which it has been shown to be unstable. A possible reason behind this problem is the *off-policy* characteristic of $\mathcal{QSA}$ accompanied by the resolution problem

Algorithm 3 Pseudo-code for $\mathcal{QSA}$

```
1:  Initialize value of $\theta$ arbitrarily.
2:  Set $M$ to be a large integer.
3:  Let $s_0 = s$ be a given initial state.
4:  Let $\theta_0$ be initial value of parameters.
5:  begin loop
6:      for each episode $k = 0, 1, \ldots, M-1$
7:      begin loop
8:          for each time step
9:          Choose action $\mathbf{a}_k$ from $A(\mathbf{s}_k)$ using an $\epsilon-$greedy policy
10:         Take action $\mathbf{a}_k$, observe the instantaneous reward $r(\mathbf{s}_k, \mathbf{a}_k)$ and the next state $\mathbf{s}_{k+1}$
11:         Compute $\theta_{k+1}$ using (13.1)
12:     end loop
13: end loop
```

introduced by the feature based Q-learning with Function Approximation. The *off-policy* problem here arises because of the presence of the "max" operation inside the Q-learning update. Note that if instead of the "max" operation, actions are selected according to a given policy, then the Q-learning update would resemble a temporal difference (TD) learning update for the joint (state-action) Markov chain. It has been shown in Tsitsiklis and Van Roy [1997] that TD with linear function approximation converges. The "max" operation in Q-learning introduces a nonlinearity in the update step that precisely results many times in non-convergence of the Q-learning update. The resolution problem arises due to the fact that the feature dimension is much less than the cardinality of the state-action space leading to inaccuracies in the estimates. A variant of Q-learning (see Bhatnagar and Lakshmanan [2012]) has been recently proposed and has been shown to be convergent. This algorithm uses two-timescale simultaneous perturbation stochastic approximation (SPSA) with deterministic perturbation sequences (Bhatnagar et al. [2003]).

The policy π is a randomized stationary policy with a Boltzmann distribution, parameterized by $\mathbf{w}$ and has the form

$$\pi_{\mathbf{w}}(\mathbf{s}, \mathbf{a}) = \frac{e^{\mathbf{w}^\top \sigma_{\mathbf{s},\mathbf{a}}}}{\sum_{\mathbf{a}' \in A(\mathbf{s})} e^{\mathbf{w}^\top \sigma_{\mathbf{s},\mathbf{a}'}}}, \quad \forall \mathbf{s} \in S, \ \forall \mathbf{a} \in A. \tag{13.5}$$

Remark 12 *Note that any policy that satisfies the following assumption can be used in place of the above π:*

(A1) *For any state–action pair (s, a), $\pi(s, a)$ is continuously differentiable in the parameter*

θ.

We also make the following assumption on the underlying Markov process $\{X_n, n \geq 0\}$:

(A2) The Markov process $\{X_n\}$ under any randomized stationary policy π is ergodic Markov.

The $\mathcal{TQSA}$ algorithm is a two-timescale stochastic approximation algorithm that employs a linear approximation architecture and parameterizes the policy. The function approximation parameter θ is tuned on the slower timescale in a TD-like fashion, while the policy parameter $\mathbf{w}$ is tuned on the faster timescale in the negative gradient descent direction using SPSA. Let $\pi'_n = \pi_{(\mathbf{w}_n+\delta\Delta_n)} \overset{\triangle}{=} (\pi_{(\mathbf{w}_n+\delta\Delta_n)}(\mathbf{i},\mathbf{a}), \mathbf{i} \in S, \mathbf{a} \in A(\mathbf{i}))^T$, where $\delta > 0$ is a given small constant, be the randomized policy parameterized by $(\mathbf{w}_n + \delta\Delta_n)$ during the nth instant. Here $\Delta_n, n \geq 0$ are perturbations obtained from certain Hadamard matrices as described below.

Update rule of $\mathcal{TQSA}$

The update rule of the $\mathcal{TQSA}$ algorithm is given by

$$\begin{aligned}\theta_{n+1} = {} & \Gamma_1\left(\theta_n + b(n)\sigma_{\mathbf{s}_n,\mathbf{a}_n}\left(r(\mathbf{s}_n,\mathbf{a}_n) + \gamma\theta_n^T\sigma_{\mathbf{s}_{n+1},\mathbf{a}_{n+1}} - \theta_n^T\sigma_{\mathbf{s}_n,\mathbf{a}_n}\right)\right),\\ \mathbf{w}_{n+1} = {} & \Gamma_2\left(\mathbf{w}_n + a(n)\frac{\theta_n^T\sigma_{\mathbf{s}_n,\mathbf{a}_n}}{\delta}\Delta_n^{-1}\right),\end{aligned} \tag{13.6}$$

In the above, the choice of features $\sigma_{\mathbf{s}_n,\mathbf{a}_n}$ is the same as in the previous algorithm. Further, $\Gamma_1 : \mathbb{R}^d \to D$, $\Gamma_2 : \mathbb{R}^N \to C$ denote two projection operators that project any $x \in \mathbb{R}^d$ and $x \in \mathbb{R}^N$ to D and C, respectively. Here C and D are compact and convex subsets of $\mathbb{R}^N$ and $\mathbb{R}^d$ respectively. For any $\mathbf{x} \in \mathbb{R}^N$, $\Gamma_2(\mathbf{x}) \in C$ is the closest point in C to $\mathbf{x}$, i.e., $\Gamma_2(x) = \arg\min_{y \in C} \parallel y - x \parallel$. Likewise, for any $\mathbf{y} \in \mathbb{R}^d$, $\Gamma_1(\mathbf{y}) \in D$ is the closest point in D to $\mathbf{y}$, i.e., $\Gamma_1(z) = \arg\min_{r \in D} \parallel r - z \parallel$. We make the following assumption on the step-size sequences $\{a(n)\}$ and $\{b(n)\}$ below:

(A3) The step-sizes $\{a(n)\}$ and $\{b(n)\}$ satisfy

$$\sum_n a(n) = \sum_n b(n) = \infty; \sum_n (a^2(n) + b^2(n)) < \infty, \text{ and } \lim_{n\to\infty} \frac{b(n)}{a(n)} = 0.$$

The first two conditions in (A3) are standard stochastic approximation requirements on the step-sizes, while the last condition above ensures that the θ-update recursion proceeds 'slower' in comparison to that of the policy parameter w.

Remark 13 *Note that the two projection operators Γ_1 and Γ_2 ensure that the iterates θ and w remain uniformly bounded. In other words, $\sup_n ||w_n||, \sup_n ||\theta_n|| < \infty$. This is a critical requirement to ensure convergence of our algorithm.*

Remark 14 *The construction of the perturbation sequence $\Delta(n)$ using Hadamard matrices is described in Section 7.2.1.*

Algorithm 4 gives the structure of the $\mathcal{TQSA}$ algorithm.

Algorithm 4 Pseudo-code for $\mathcal{TQSA}$

```
1: Initialize value of $\theta$ arbitrarily.
2: Set $M$ to be a large integer.
3: Let $\mathrm{s}_0 = \mathrm{s}$ be a given initial state.
4: Let $\theta_0, w_0$ be initial values of the parameters $\theta$ and $w$.
5: begin loop
6:     for each episode $n = 0, 1, \ldots, M-1$
7:     begin loop
8:         for each time step
9:         Choose action $\mathbf{a}_n$ from $A(\mathbf{s}_n)$ using the Boltzmann policy $\pi$ (13.5)
10:        Take action $\mathbf{a}_n$, observe the instantaneous reward $r(\mathbf{s}_n, \mathbf{a}_n)$ and the next state $\mathbf{s}_{n+1}$
11:        Compute $\theta_{n+1}, \mathbf{w}_{n+1}$ using (13.6)
12:    end loop
13: end loop
```

13.2 Sleep–wake scheduling for average reward objective

While the states, actions and single-stage cost function for the average-cost setting remain the same as those specified for the discounted-cost POMDP in Section 12.2, the primary difference is in the long-run performance objective that we seek to maximize via an optimal policy. In an infinite horizon average reward framework, the aim is to find a policy π so as to maximize the long-run average reward defined below:

$$J(\pi) = \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} r(s_n, a_n), \tag{13.7}$$

starting from any given state i (i.e., with $s_0 = i$).

Let $h(i)$ be the differential reward function corresponding to state i, under policy π. Then $h(.)$ satisfies

$$J(\pi) + h(i) = \max_a \sum_j p(i,j,a)(r(i,a) + h(j)), \forall i \in \mathcal{S}. \tag{13.8}$$

Define the Q-factors $Q(i,a), i \in \mathcal{S}, a \in \mathcal{A}(i)$ as

$$Q(i,a) = \sum_j p(i,j,a)(r(i,a) + h(j)). \tag{13.9}$$

From (13.8) and (13.9), we have

$$J(\pi) + h(i) = \max_a Q(i,a), \forall i \in \mathcal{S}. \tag{13.10}$$

Now from (13.9) and (13.10), we have

$$Q(i,a) = \sum_j p(i,j,a)(r(i,a) + \max_b Q(j,b) - J(\pi))\text{or} \tag{13.11}$$

$$J(\pi) + Q(i,a) = \sum_j p(i,j,a)(r(i,a) + \min_{b \in \mathcal{A}(j)} Q(j,b)), \tag{13.12}$$

for all $i \in \mathcal{S}, a \in \mathcal{A}(i)$.

Note that in order to solve this equation, one requires the knowledge of the transition probabilities $p(i,j,a)$ that constitute the system model. Moreover, the state and action spaces should be manageable in size. As in the case of the discounted setting, the algorithms presented subsequently for the average reward objective work with a lack of system model (like any RL algorithm) and further, effectively handle large state and action spaces by incorporating function approximation and feature based representations.

We first present a sleep–wake scheduling algorithm which is the function approximation analogue of the Q- learning with average reward algorithm proposed in Abounadi et al. [2002]. The second algorithm is a two-timescale algorithm with proven convergence to the optimal policy, unlike the Q-learning based algorithm. We shall henceforth refer to the first algorithm as $\mathcal{QSA}$-$\mathcal{A}$ and the second as $\mathcal{TQSA}$-$\mathcal{A}$.

13.2.1 Q-learning based Sleep–wake Algorithm – Average-cost variant ($\mathcal{QSA}$-$\mathcal{A}$)

The Q-learning algorithm, described previously for the discounted reward setting, is a stochastic approximation version of the value iteration procedure for MDPs. However, in an average reward setting, the extension is non-trivial and involves using the relative value iteration procedure. Let $\mathbf{s}_{n+1}$ denote the state of the system at instant $(n+1)$ when the state at instant n is $\mathbf{i}$ and action chosen is $\mathbf{a}$. Let $Q_n(\mathbf{i},\mathbf{a})$ denote the Q-value estimate at instant n associated with the tuple $(\mathbf{i},\mathbf{a})$. The relative Q-value iteration (RQVI) scheme is

$$Q_{n+1}(\mathbf{i},\mathbf{a}) = \sum_{\mathbf{j}} p(\mathbf{i},\mathbf{j},\mathbf{a})(r(\mathbf{i},\mathbf{a}) + \max_{\mathbf{b}\in A(\mathbf{j})} Q_n(\mathbf{j},\mathbf{b})) - \max_{\mathbf{r}\in A(\mathbf{s})} Q_n(\mathbf{s},\mathbf{r}), \tag{13.13}$$

where $\mathbf{s} \in S$ is an arbitrarily chosen prescribed state.

The Q-learning analogue for the average reward setting estimates the 'Q-factors' $Q(\mathbf{i},\mathbf{a})$ of all feasible state-action tuples $(\mathbf{i},\mathbf{a})$, i.e., those with $\mathbf{i} \in S$ and $\mathbf{a} \in A(\mathbf{i})$ using the relative value iteration of Q-factors (Abounadi et al. [2002]). The update rule for this algorithm is then given by

$$\boxed{Q_{n+1}(\mathbf{i},\mathbf{a}) = Q_n(\mathbf{i},\mathbf{a}) + \alpha(n)(r(\mathbf{i},\mathbf{a}) + \max_{\mathbf{b}\in A(\mathbf{j})} Q_n(\mathbf{j},\mathbf{b}) - \max_{\mathbf{r}\in A(\mathbf{s})} Q_n(\mathbf{s},\mathbf{r})), \quad (13.14)}$$

for all $\mathbf{i} \in S$ and $\mathbf{a} \in A(\mathbf{s})$. In the above, $\alpha(n), n \geq 0$ are the step-sizes that satisfy the standard stochastic approximation conditions. The last term $\max_{\mathbf{r}\in A(\mathbf{s})} Q_n(\mathbf{s},\mathbf{r})$ in (13.14) asymptotically converges to the optimal average reward per stage. Further, the iterates in (13.14) converge to the optimal Q-values $Q^*(\mathbf{i},\mathbf{a})$ that satisfy the corresponding Bellman equation (2.4) and
$\max_{\mathbf{a}\in A(\mathbf{i})} Q_n(\mathbf{i},\mathbf{a})$ then gives the optimal differential reward $h^*(\mathbf{i})$. The optimal action in state $\mathbf{i}$ corresponds to $arg \max_{\mathbf{a}\in A(\mathbf{i})} Q^*(\mathbf{i},\mathbf{a})$. The reader is referred to Abounadi et al. [2002] for a detailed convergence proof of this algorithm.

For the function approximation variant of the above algorithm, we associate with each state-action tuple $(\mathbf{i},\mathbf{a})$, a state-action feature denoted by $\sigma_{\mathbf{i},\mathbf{a}}$ as described in Section 13.1.1. The Q-function is then approximated as

$$Q(\mathbf{i},\mathbf{a}) \approx \theta^T \sigma_{\mathbf{i},\mathbf{a}}. \tag{13.15}$$

Let $\mathrm{s}_n, \mathrm{s}_{n+1}$ denote the state at instants $n, n+1$, respectively, measured online. Let θ_n be the estimate of the parameter θ at instant n. Let s be any fixed state in S.

The algorithm $\mathcal{QSA}$-$\mathcal{A}$ uses the following update rule:

$$\boxed{\theta_{n+1} = \theta_n + \alpha(n)\sigma_{\mathbf{s}_n,\mathbf{a}_n}\left(r(\mathbf{s}_n,\mathbf{a}_n) + \max_{\mathbf{v}\in A(\mathbf{s}_{n+1})} \theta_n^T \sigma_{\mathbf{s}_{n+1},\mathbf{v}} - \max_{\mathbf{r}\in A(\mathbf{s})} \theta_n^T \sigma_{\mathbf{s},\mathbf{r}}\right), \quad (13.16)}$$

where θ_0 is set arbitrarily. In (13.16), the action $\mathbf{a}_n$ is chosen in state $\mathbf{s}_n$ according to $\mathbf{a}_n = arg\max_{\mathbf{v}\in A(\mathbf{s}_n)} \theta_n^T \sigma_{\mathbf{s}_n,\mathbf{v}}$.

The overall algorithm here is similar to Algorithm 3, except that θ is updated according to (13.16) in Step 11 of the algorithm.

13.3 Two-timescale Q-learning based Sleep–wake algorithm – Average-cost variant ($\mathcal{TQSA}$-$\mathcal{A}$)

As in the case of the discounted setting, the Q-learning algorithm with function approximation is not guaranteed to converge even in average reward setting as well (cf. Prashanth and Bhatnagar [2011c]). In this section, we develop a novel convergent two-timescale Q-learning algorithm in the average cost setting. This algorithm falls under the realm of Policy-Gradient Reinforcement Learning. The idea here is to parameterize the policy and perform a stochastic gradient descent in the policy parameter. The descent here is w.r.t. the Q-value function. However, owing to the high-dimensional state space, we parameterize the Q-value function using a linear function approximation architecture, similar to (13.15). Further, we estimate the gradient of the approximate Q-value function using a one-simulation SPSA scheme, which employs certain Hadamard matrices for perturbation. We now formalize these notions below.

As in the case of the earlier three algorithms, we approximate the Q-value function as in (13.15). The policy here is setup in a manner similar to $\mathcal{TQSA}$. In other words, what we have is an SRP π_w, parameterized by w. The parameter $w = (w_1, \ldots, w_N)^T$ is assumed to take values in a compact and convex set $C \subset \mathbb{R}^N$. Further, the parameter $\theta = (\theta_1, \ldots, \theta_d)^T \in \mathbb{R}^D$ is assumed to take values in a compact and convex set $D \subset \mathbb{R}^d$.

Further, the gradient of the approximate Q-value function w.r.t. w is estimated using a one-simulation SPSA scheme with the following estimate as in Bhatnagar and Lakshmanan

[2012]: For any function f,

$$\nabla f(w_n) \approx \frac{f(w+\delta\Delta_n)}{\delta}\Delta_n^{-1}, \tag{13.17}$$

where $\delta > 0$ is a fixed small real number and $\Delta_n = (\Delta_n(1), \ldots, \Delta_n(N))^T$ is the perturbation vector constructed using Hadamard matrices, as described in Section 7.2.1.

Thus, we have a two timescale algorithm with the following characteristics:

- On the slower timescale, the Q-value parameter is updated using the regular Q-learning update for a given policy;
- On the faster timescale, the policy parameter is updated along a gradient descent direction using an SPSA-like estimate (13.17);
- Further, on the slower timescale, we also estimate the average reward (13.7) and use the estimate in the update of the Q-value parameter. Note that, unlike the discounted reward setting, the Q-values here satisfy a Bellman equation (13.12) that includes the average reward λ as well.

The update rule for $\mathcal{TQSA}$-$\mathcal{A}$ algorithm is given by

$$\theta_{n+1} = \Gamma_1\left(\theta_n + b(n)\sigma_{\mathbf{s}_n,\mathbf{a}_n}\left(r(\mathbf{s}_n,\mathbf{a}_n) - \hat{J}_{n+1} + \theta_n^T\sigma_{\mathbf{s}_{n+1},\mathbf{a}_{n+1}} - \theta_n^T\sigma_{\mathbf{s}_n,\mathbf{a}_n}\right)\right), \tag{13.18}$$

$$\hat{J}_{n+1} = \hat{J}_n + c(n)\left(r(\mathbf{s}_n,\mathbf{a}_n) - \hat{J}_n\right), \tag{13.19}$$

$$\mathbf{w}_{n+1} = \Gamma_2\left(\mathbf{w}_n + a(n)\frac{\theta_n^T\sigma_{\mathbf{s}_n,\mathbf{a}_n}}{\delta}\Delta_n^{-1}\right). \tag{13.20}$$

In the above,

- The step-sizes $a(n)$ and $b(n)$ satisfy (A3) as in the case of the $\mathcal{TQSA}$ algorithm. Further, $c(n) = kb(n)$, for a positive scalar k.
- Γ_1 and Γ_2 are projection operators that project the iterates θ_n and $\mathbf{w}_n$ to the compact sets D and C, respectively and are the same as the ones used in $\mathcal{TQSA}$ algorithm (see Section 13.1.2). The recursions (13.18) and (13.20) are stable recursions because of these projection operators.

The overall algorithm here is similar to Algorithm 4, except that θ is updated according to (13.18) in Step 11 of the algorithm.

13.3.1 Convergence analysis for $\mathcal{TQSA}$-$\mathcal{A}$

The ODE approach is adopted for analyzing the convergence of θ and w recursions (13.6). In essence, the two-timescale stochastic approximation architecture employed in the $\mathcal{TQSA}$-$\mathcal{A}$ algorithm allows (i) the faster timescale analysis of the w-recursion in (13.18) assuming that the slower θ-recursion is constant (quasi-static), and (ii) the slower timescale analysis of the θ-recursion in (13.18) assuming that the faster w-recursion has converged.

The convergence analysis comprises of the following important steps:

- Theorem 13.3, in effect, states that the w-recursion performs a gradient descent using one-simulation SPSA and converges to a set of points in the neighborhood of the local minimum of the approximate Q-value function $R(\theta, w)$ (defined below). Note that this analysis is for the w-recursion on the faster timescale, assuming the Q-value function parameter θ to be a constant.

- Analyzing θ-recursion on the slower timescale. Theorem 13.8 claims that the iterates (w, θ) asymptotically converge to (w^*, θ^*), where w^* belongs to a neighborhood set of the local minimum of $R(\theta, w)$, while θ^* solves a projected form of the Bellman equation and depends on w.

We present below the precise statements of these results. Let $\mathcal{C}(C)$ (resp. $\mathcal{C}(D)$) denote the space of all continuous functions from C (resp. D) to $\mathcal{R}^N$ (resp. $\mathcal{R}^d$). We define two operators $\hat{\Gamma}_1 : \mathcal{C}(D) \to \mathcal{C}(\mathcal{R}^d)$ and $\hat{\Gamma}_2 : \mathcal{C}(C) \to \mathcal{C}(\mathcal{R}^N)$ as follows:

$$\begin{aligned}\hat{\Gamma}_1(u(\theta)) &= \lim_{\eta \downarrow 0} \left(\frac{\Gamma_1(\theta + \eta u(\theta)) - \theta}{\eta} \right),\\ \hat{\Gamma}_2(v(w)) &= \lim_{\alpha \downarrow 0} \left(\frac{\Gamma_2(w + \alpha v(w)) - w}{\alpha} \right).\end{aligned}$$

Consider the ODE associated with the w-recursion on the faster timescale, assuming $\theta(t) \equiv \theta$ (a constant independent of t):

$$\dot{w}(t) = \hat{\Gamma}_2 \left(-\nabla_w R(\theta, w(t)) \right). \tag{13.21}$$

Theorem 13.3 establishes that the w-recursion tracks the above ODE. In the above,

$$R(\theta, w) \triangleq \sum_{i \in S, a \in A(i)} f_w(i,a) \theta^T \sigma_{i,a},$$

where $f_w(i,a)$ are the stationary probabilities $f_w(i,a) = d^{\pi_w}(i)\pi_w(i,a)$, $i \in S$, $a \in A(i)$ for the joint process $\{(X_n, Z_n)\}$, obtained from the state-action tuples at each instant. Here $d^{\pi_w}(i)$ is the stationary probability distribution for the Markov chain $\{X_n\}$ under policy π_w being in state $i \in S$. The ergodicity of the joint process $\{(X_n, Z_n)\}$ and the existence of stationary distribution $f_w(i,a)$ follows from the proposition below:

Proposition 13.1 *Under (A1) and (A2), the process $(X_n, Z_n), n \geq 0$ with Z_n, $n \geq 0$ obtained from the SRP π_w, for any given $w \in C$, is an ergodic Markov process.*

Proof: See [Bhatnagar and Lakshmanan, 2012, Proposition 1, Section 3]. ■

Lemma 13.2 *Under (A1) and (A2), the stationary probabilities $f_w(i,a)$, $i \in S$, $a \in A(i)$ are continuously differentiable in the parameter $w \in C$.*

Proof: See [Bhatnagar and Lakshmanan, 2012, Lemma 1, Section 3]. ■

Let K_θ denote the set of asymptotically stable equilibria of (13.21), i.e., the local minima of the function $R(\theta, \cdot)$ within the constraint set C. Given $\epsilon > 0$, let K_θ^ϵ denote the ϵ-neighborhood of K_θ, i.e.,

$$K_\theta^\epsilon = \{w \in C \mid \parallel w - w_0 \parallel < \epsilon, w_0 \in K_\theta\}.$$

Theorem 13.3 *Let $\theta_n \equiv \theta, \forall n$, for some $\theta \in D \subset \mathcal{R}^d$. Then, given $\epsilon > 0$, there exists $\delta_0 > 0$ such that for all $\delta \in (0, \delta_0]$, $\{w_n\}$ governed by (13.6) converges to a point $w^* \in K_\theta^\epsilon$ almost surely.*

Proof: See [Bhatnagar and Lakshmanan, 2012, Theorem 2, Section 3]. ■

We now analyze the θ-recursion, which is the slower recursion in (13.18).

Lemma 13.4 *With probability one, $|\hat{J}_n - J(\pi)| \to 0$ as $n \to \infty$, where $J(\pi)$ is the average reward under π.*

Proof: The $\hat{J}$ update can be re-written as

$$\hat{J}_{n+1} = \hat{J}_n + b(n)\left(J(\pi) + \xi_n - \hat{J}_n + M_{n+1}\right). \tag{13.22}$$

In the above,

- $\mathcal{F}_n = \sigma(\xi_m, M_m; m \leq n), n \geq 0$ is a set of σ-fields.
- $\xi_n = (E[r(s_n, a_n)|\mathcal{F}_{n-1}] - J(\pi)), n \geq 0$ and
- $M_{n+1} = r(s_n, a_n) - E[r(s_n, a_n)|\mathcal{F}_{n-1}], n \geq 0$ is a martingale difference sequence.

Let $N_m = \sum_{n=0}^{m} d(n)M_{n+1}$. Clearly, $(N_m, \mathcal{F}_m), m \geq 0$ is a square-integrable and almost surely convergent martingale. Further, from Proposition 13.1, $|\xi_n| \to 0$ almost surely on the 'natural timescale', as $n \to \infty$. The 'natural timescale' is faster than the algorithm's timescale and hence ξ_n can be ignored in the analysis of $\hat{J}$-recursion, see [Borkar, 2008, Chapter 6.2] for detailed treatment of natural timescale algorithms.

The ODE associated with (13.22) is

$$\dot{\hat{J}}(t) = J(\pi) - \hat{J}(t) \triangleq H(\hat{J}(t)). \tag{13.23}$$

Let $H_\infty(\hat{J}(t)) = \lim_{c\to\infty} \dfrac{H(c\hat{J}(t))}{c} = -\hat{J}(t)$. Note that the ODE

$$\dot{\hat{J}}(t) = -\hat{J}(t)$$

has the origin as it unique globally asymptotically stable equilibrium. Further, the ODE (13.23) has $\hat{J}^* = J(\pi)$ as it unique asymptotically stable equilibrium. The claim follows from Theorems 2.1-2.2 of Borkar and Meyn [2000]. ∎

Lemma 13.5 *Given $\epsilon > 0$, there exists a $\delta_0 > 0$ such that for all $\delta \in (0, \delta_0]$, $(w_n, \theta_n) \to \{(w, \theta) \mid w \in K_\theta^\epsilon, \theta \in D\}$ almost surely as $n \to \infty$.*

Proof: Follows in a similar manner as [Bhatnagar and Lakshmanan, 2012, Lemma 7] with $\check{X}(n) = \phi_{X_n,Z_n}(g(X_n, Z_n) - J(\pi) + \theta_n^T \phi_{X_{n+1},Z_{n+1}} - \theta_n^T \phi_{X_n,Z_n})$ used in place of $\check{X}(n)$ defined there. ∎

Lemma 13.6 *The map $w : D \to C$ is continuous.*

Proof: Follows in a similar manner as [Bhatnagar and Lakshmanan, 2012, Lemma 8]. ■

We now recall an important theorem related to convergence of general projected stochastic approximation from Bhatnagar and Lakshmanan [2012]. We use this result to prove Theorem 13.8 below. Let $\Gamma : \mathcal{R}^L \to E \subset \mathcal{R}^L$ denote a projection operator mapping any $r \in \mathcal{R}^L$ to the set E. Consider the following L-dimensional stochastic recursion

$$r_{n+1} = \Gamma\big(r_n + c(n)(h(r_n) + \xi_n + \gamma_n)\big), \tag{13.24}$$

under the assumptions (A1)–(A5) listed below. Also, consider the following ODE associated with (13.24):

$$\dot{r}(t) = \bar{\Gamma}(h(r(t))), \tag{13.25}$$

where for any continuous $y : E \to \mathcal{R}^L$,

$$\bar{\Gamma}(y(r)) = \lim_{\beta \to 0} \left(\frac{\Gamma(r + \beta y(r)) - r}{\beta} \right).$$

Let S denote the set of all asymptotically stable equilibria of the ODE (13.25). Let $t(n), n \geq 0$ be a sequence of positive real numbers defined according to $t(0) = 0$ and for $n \geq 1$, $t(n) = \sum_{j=0}^{n-1} c(j)$. Let $m(t) = \max\{n \mid t(n) \leq t\}$.

(A1) The function $h : \mathcal{R}^L \to \mathcal{R}^L$ is continuous.

(A2) The step-sizes $c(n), n \geq 0$ satisfy

$$c(n) > 0 \forall n, \ \sum_n c(n) = \infty, \ c(n) \to 0 \text{ as } n \to \infty.$$

(A3) The sequence $\gamma_n, n \geq 0$ is a bounded random sequence with $\gamma_n \to 0$ almost surely as $n \to \infty$.

(A4) There exists $T > 0$ such that $\forall \epsilon > 0$,

$$\lim_{n \to \infty} P\left(\sup_{j \geq n} \max_{t \leq T} \left| \sum_{i=m(jT)}^{m(jT+t)-1} a(i)\xi_i \right| \geq \epsilon \right) = 0.$$

(A5) The set $E \subset \mathcal{R}^L$ is compact and convex.

As a consequence of (A2), $t(n) \to \infty$ as $n \to \infty$. Hence, $m(t) \to \infty$ as $t \to \infty$.

Theorem 13.7 *[Kushner and Clark, 1978b, Theorem 5.3.1 (pp. 191-196)] Under (A1)–(A5), the recursions $r_n, n \geq 0$ governed by (13.24) converge to a point $r^* \in S$.*

Let $T_w : \mathcal{R}^{|S\times A(S)|} \rightarrow \mathcal{R}^{|S\times A(S)|}$ be the operator given by

$$T_w(J)(i,a) = g(i,a) - J(\pi)e + \sum_{j\in S, b\in A(j)} p_w(i,a;j,b)J(j,b), \tag{13.26}$$

or in more compact notation

$$T_w(J) = G - J(\pi)e + \gamma P_w J,$$

where G is the column vector with components $g(i,a), i \in S, a \in A(i)$ and P_w is the transition probability matrix with components $p_w(i,a;j,b)$. Here $p_w(i,a;j,b)$ denotes the transition probabilities of the joint process $\{(X_n, Z_n)\}$.

In compact notation, the ODE associated with the θ-recursion of (13.6) corresponds to

$$\dot{\theta}(t) = \hat{\Gamma}_1\left(\Phi^T \mathbb{F}_{w(\theta(t))}(T_{w(\theta(t))}(\Phi\theta(t)) - \Phi\theta(t))\right). \tag{13.27}$$

In the above, $\mathbb{F}_w$ denotes the diagonal matrix with elements along the diagonal being $f_w(i,a)$, $i \in S$, $a \in A(i)$. The notation $w \equiv w(\theta)$ refers to any point in K_θ, the set of equilibrium points of (13.21) and $w(\cdot)$ can be seen as a map from $D \subset \mathcal{R}^d$ to $C \subset \mathcal{R}^N$. Also, Φ denotes the matrix with rows $\sigma_{s,a}^T$, $s \in S, a \in A(s)$. The number of rows of this matrix is thus $|S \times A(S)|$, while the number of columns is N. Then $\Phi = (\Phi(i), i = 1, \ldots, d)$ where $\Phi(i)$ is the column vector

$$\Phi(i) = (\sigma_{s,a}(i), s \in S, a \in A(s))^T, \; i = 1, \ldots, N.$$

Consider now the set of equilibria of (13.27):

$$\hat{C} \triangleq \{\theta \in D \mid \hat{\Gamma}_1\left(\Phi^T \mathbb{F}_{w(\theta)}(T_{w(\theta)}(\Phi\theta) - \Phi\theta)\right) = 0\}.$$

The main result is then given as follows:

Theorem 13.8 *Under Assumptions (A1)-(A3), given $\epsilon > 0$, there exists $\delta_0 > 0$ such that for all $\delta \in (0, \delta_0]$, the iterates (w_n, θ_n), $n \geq 0$ governed by (13.6) satisfy*

$$(w_n, \theta_n) \rightarrow \{(w^*, \theta^*) \mid w^* \in K^\epsilon_{\theta^*}, \theta^* \in \hat{C}\}, \textit{ with probability one.}$$

Proof: The θ update recursion (13.18) can be re-written as follows:

$$\begin{aligned}\theta_{n+1} =&\Gamma_1\Bigg(\theta_n + b(n)\sum_{(i,a)} f_{w(\theta_n)}(i,a)\big(g(i,a) - J(\pi) + \gamma{\theta_n}^T\sum_{(j,b)} p_{w(\theta_n)}(i,a;j,b)\sigma_{j,b}\\ &- {\theta_n}^T\sigma_{i,a}\big)\sigma_{i,a} + b(n)\xi_1(n) + b(n)Y(n+1)\Bigg),\end{aligned}$$

where $\xi_1(n)$ constitute the 'Markov noise' that vanishes asymptotically along the natural timescale and is defined as follows:

$$\begin{aligned}\xi_1(n) =&E\left[\left(g(X_n,Z_n) - J(\pi) + \theta_n^T\sigma_{X_{n+1},Z_{n+1}} - \theta_n^T\sigma_{X_n,Z_n}\right)\sigma_{X_n,Z_n} \mid \mathcal{G}(n)\right]\\ &- \sum_{(i,a)} f_{w(\theta_n)}(i,a)\big(g(i,a) - J(\pi) + {\theta_n}^T\sum_{(j,b)} p_{w(\theta_n)}(i,a;j,b)\sigma_{j,b} - {\theta_n}^T\sigma_{i,a}\big).\end{aligned}$$

Further, $Y(n+1)$ is defined as follows:

$$\begin{aligned}Y(n+1) = &\left(g(X_n,Z_n) - J(\pi) + \theta_n^T\sigma_{X_{n+1},Z_{n+1}} - \theta_n^T\sigma_{X_n,Z_n}\right)\sigma_{X_n,Z_n}\\ &- E\left[\left(g(X_n,Z_n) - J(\pi) + \theta_n^T\sigma_{X_{n+1},Z_{n+1}} - \theta_n^T\sigma_{X_n,Z_n}\right)\sigma_{X_n,Z_n} \mid \mathcal{G}(n)\right],\end{aligned}$$

where $\mathcal{G}(n) = \sigma(\theta_r, X_r, Z_r, r \leq n), n \geq 0$ is a sequence of associated sigma fields. Let $N(n) = \sum_{m=0}^{n-1} b(m)Y(m+1)$. Then, $(N(n), \mathcal{G}(n)), n \geq 0$ can be seen to be a square-integrable and almost surely convergent martingale.

The main result now follows from Theorem 13.7 with the following correspondance: $r_n \equiv \theta_n$, $\Gamma(\cdot) = \Gamma_1(\cdot)$ and $c(n) = b(n), n \geq 0$. The aforementioned theorem requires assumptions (A1)-(A5) to be satisfied. We verify these assumptions here.

- (A1) requires that the map $\hat{g}(\theta)$ (defined below) be continuous in θ.

  $$\hat{g}(\theta) = \sum_{(i,a)} f_{w(\theta)}(i,a)\big(g(i,a) + \gamma\theta^T\sum_{(j,b)} p_{w(\theta)}(i,a;j,b)\phi_{j,b} - \theta^T\phi_{i,a})\phi_{i,a}\big).$$

  This follows from the facts that $w(\cdot)$ is continuous from Lemma 13.6, $f_{w(\theta)}(i,a)$ is continuous from Lemma 13.2 and $p_{w(\theta)}(i,a;j,b)$ is continuous from (A1) and Lemma 13.6.

- (A2) holds as a consequence of the assumption that we made earlier on the step-size sequences $\{a(n)\}$ and $\{b(n)\}$.

- (A3) requires that $\xi_n \to 0$ almost surely as $n \to \infty$ and $\sup_n |\xi_1(n)| < \infty$. This follows from Proposition 13.8.

- (A4) holds as a consequence of the fact that $N(n), n \geq 0$, is a convergent martingale sequence and hence, $Y(n)$ can be used in place of ξ_n in Theorem 13.7.
- Since the updates to θ take place in the set D, which is a compact and convex subset of $\mathbb{R}^N$, (A5) holds with D in place of E in Theorem 13.7.

Hence, by an application of Theorem 13.7, we see that $\theta_n \to \theta^* \in \hat{C}$. Further, as we established in Lemma 13.5, $w_n \to w^* \in K_{\theta^*}$. Hence proved. ■

13.3.2 Convergence result for $\mathcal{TQSA}$

We provide the main convergence result (cf. Theorem 13.9) for the $\mathcal{TQSA}$ algorithm. The detailed convergence proof is available in [Bhatnagar and Lakshmanan, 2012, Section 3]. Consider the mapping $T_w : \mathcal{R}^{|S\times A(S)|} \to \mathcal{R}^{|S\times A(S)|}$ given by

$$T_w(J)(i,a) = g(i,a) + \gamma \sum_{j\in S, b\in A(j)} p_w(i,a;j,b)J(j,b), \tag{13.28}$$

or in more compact notation

$$T_w(J) = G + \gamma P_w J,$$

where G and P_w are as before in Section 13.3.1.

In compact notation, the ODE associated with the θ-recursion of (13.6) corresponds to

$$\dot{\theta}(t) = \hat{\Gamma}_1 \left(\Phi^T \mathbb{F}_{w(\theta(t))}(T_{w(\theta(t))}(\Phi\theta(t)) - \Phi\theta(t))\right). \tag{13.29}$$

Note that the difference of this ODE with (13.27) because of the change in operator T_w that is now defined as in (13.28). In the above, $\mathbb{F}_w$, as before in Section 13.3.1, refers to the diagonal matrix with elements $f_w(i,a)$, $i \in S$, $a \in A(i)$. Recall that the set K_θ denotes the set of equilibrium points of (13.21) and $w(\cdot)$ can be seen as a map from $D \subset \mathcal{R}^d$ to $C \subset \mathcal{R}^N$. Consider now the set of equilibria of (13.29):

$$\hat{C} \triangleq \{\theta \in D \mid \hat{\Gamma}_1 \left(\Phi^T \mathbb{F}_{w(\theta)}(T_{w(\theta)}(\Phi\theta) - \Phi\theta)\right) = 0\}.$$

We now state the convergence result for the joint parameter vector sequence $(\theta_n, w_n), n \geq 0$.

Theorem 13.9 *Under Assumptions (A1)-(A4), given $\epsilon > 0$, there exists $\delta_0 > 0$ such that for all $\delta \in (0, \delta_0]$, the iterates (w_n, θ_n), $n \geq 0$ governed by (13.6) satisfy*

$$(w_n, \theta_n) \to \{(w^*, \theta^*) \mid w^* \in K^{\epsilon}_{\theta^*}, \theta^* \in \hat{C}\}, \textit{ with probability one.}$$

Proof: See [Bhatnagar and Lakshmanan, 2012, Theorem 3, Section 3]. ■

13.4 Extension to unknown P

The sleep–wake scheduling algorithms presented in the previous section attempt to optimize the sleep-times of the individual sensors, using the distribution of the object movement as input. However, in practice, this distribution of the object movement is unknown in advance and hence a more challenging problem is to solve the sleep–wake scheduling POMDP under various long-run performance objectives under the absence of knowledge of the transition probability matrix P, which specifies the distribution of object movement.

We develop an online scheme for estimating P in this section. This scheme can be combined with any of the four algorithms presented before, via multi-timescale stochastic approximation. The idea is to run the P estimation procedure on the faster timescale, while having any of the earlier sleep–wake scheduling algorithms run on the slower timescale. This, in effect means that the recursions of each algorithm essentially see a converged value of P from the estimation procedure.

We make an assumption that P is stationary, i.e., it does not change with time. Let P_0 be the initial estimate of the transition probability matrix P. We propose an update procedure given below.

$$P_{k+1} = \Gamma\left(P_k + \beta(k)\hat{p}_k\hat{p}_{k+1}^T\right), \tag{13.30}$$

where P_k is the estimate of P at instant k, $\hat{p}_k = [p_k(i) : i = 1, 2, \ldots, n+1]^T$ is the estimate of probability distribution at instant k (this is explained better below) and $\beta(k)$ is the step length such that $\sum_{k=1}^{\infty} \beta(k) = \infty$, $\sum_{k=1}^{\infty} \beta^2(k) < \infty$ and $\alpha(k) = o(\beta(k))$. Also, $\Gamma(\cdot)$ is a projection operator that ensures that P_{k+1} satisfies the properties of a transition probability matrix. Further $\beta(k)$, $k \geq 0$, is a faster step-size schedule than $\alpha(k)$, so corresponding to the θ_k update, P_k appears to have practically converged to P, see Chapter 6 of Borkar [2008] for a general treatment of multi-timescale stochastic approximation algorithms. In our simulation experiments we show that the above update procedure gives a "good enough" estimate of P such that the main iteration of θ provides the desired results.

Discussion Consider the case where location of the object at instants k and $k+1$, i.e., l_k and l_{k+1}, respectively, is known. Then, $\hat{p}_k$ (respectively $\hat{p}_{k+1}$) would be vectors taking the value 1 in the component l_k (respectively l_{k+1}) and 0 elsewhere, Thus, the update $\hat{p}_k\hat{p}_{k+1}^T$ would be a matrix with 1 at row l_k and column l_{k+1} and 0 in the remaining entries. The above update procedure in such cases would exhibit a sample averaging behaviour to estimate the transition probability matrix P. In the remaining cases, i.e., (i) known l_k to unknown l_{k+1}, (ii) unknown l_k to known l_{k+1}, and (iii) unknown l_k to unknown l_{k+1}, a similar behaviour can be expected.

Chapter 14

Performance Analysis of Sleep–Wake Scheduling Algorithms

14.1 Simulation Experiments - Discounted Setting

In this section, we present numerical results from the performance analysis of our discounted cost algorithms.

14.1.1 Implementation

We implemented the following sleep–wake scheduling algorithms:

- $\mathcal{QSA}$: This is the Q-learning based sleep–wake scheduling algorithm that incorporates function approximation and is described in Section 13.1.1.
- $\mathcal{TQSA}$: This is the two-timescale Q-learning based sleep–wake scheduling algorithm, adapted from Bhatnagar and Lakshmanan [2012] and is described in Section 13.1.2.
- FCR: This is an algorithm proposed in Fuemmeler and Veeravalli [2008] and is briefly described below in Section 14.1.1.
- $\mathrm{Q}_{\mathrm{MDP}}$: This is another algorithm proposed in Fuemmeler and Veeravalli [2008] based on the popular policy iteration procedure for MDPs and is briefly described below in Section 14.1.1.
- $\mathcal{QSA}$-$\mathcal{UP}$: This is the sleep–wake scheduling algorithm that results from a combination of the $\mathcal{QSA}$ algorithm with the estimation procedure (13.30).

- $\mathcal{TQSA}$-$\mathcal{UP}$:This is the sleep–wake scheduling algorithm where the estimation procedure for P (13.30) is combined with the $\mathcal{TQSA}$ algorithm.

We now briefly review the FCR and Q_{MDP} algorithms from Fuemmeler and Veeravalli [2008], Fuemmeler et al. [2011], that we implement for the sake of comparison.

FCR

This algorithm approximates the state evolution (12.4) by

$$\mathbf{p}_{t+1} = \mathbf{p}_t \mathbf{P}, \tag{14.1}$$

and then attempts to find the sleep time for each sensor by solving the following balance equation:

$$V^{(l)}(\mathbf{p}) = \min_u \left(\sum_{j=1}^{u} [\mathbf{p}\mathbf{P}^j]_l + \sum_{i=1}^{N} c[\mathbf{p}\mathbf{P}^{u+1}]_i + V^{(l)}(\mathbf{p}\mathbf{P}^{u+1}) \right). \tag{14.2}$$

Thus, the sleeping policy here is obtained locally for each sensor by solving a per-sensor Bellman equation (14.2), with a strong approximation on the state evolution. Note that our algorithms - $\mathcal{QSA}$ and $\mathcal{TQSA}$ make no such assumptions and attempt to find the optimal sleeping policy in the global sense (i.e., considering all the sensors) and not in the local sense (i.e., treating the sensors individually).

$\mathrm{Q}_{\mathrm{MDP}}$

In this approach, the decomposition into the per sensor problem is the same as in FCR. However here, the underlying assumption is that the location of the object will always be known in the future. Thus, instead of (12.4) the state evolves here according to

$$\mathbf{p}_{k+1} = \mathbf{e}_{l_{k+1}} \mathbf{P}. \tag{14.3}$$

The objective function for a sensor l, given the state component $\mathbf{p}$, is given by

$$V^{(l)}(\mathbf{p}) = \min_u \left(\sum_{j=1}^{u} [\mathbf{p}\mathbf{P}^j]_l + \sum_{i=1}^{N} c[\mathbf{p}\mathbf{P}^{u+1}]_i + \sum_{i=1}^{N} [\mathbf{p}\mathbf{P}^{u+1}]_i V^{(l)}(\mathbf{e}_i) \right). \tag{14.4}$$

It can be noted that the only difference between (14.4) and (14.2) is in the third term on the right hand side representing the future cost. In the case of Q_{MDP}, the future cost is the conditional expectation of the cost incurred from the object location after u time units given the current distribution as its location. Thus, one can solve $V^{(l)}(\mathbf{p})$ for any $\mathbf{p}$ once $V^{(l)}(\mathbf{e}_i), 1 \leq i \leq N$ are known. The Q$_{\text{MDP}}$ algorithm then attemps to find a solution to (14.4) using the well-known dynamic programming procedure - policy iteration for MDPs.

Remark 15 *An important drawback with the dynamic programming approaches is the curse of dimensionality (i.e., the computational complexity with solving the associated Markov decision process increases exponentially with the dimension and cardinality of the state and action spaces). Reinforcement learning algorithms that incorporate function approximation techniques alleviate this problem and make the computational complexity manageable, while still ensuring that the RL algorithms converge to a 'good enough' policy.*

14.1.2 Experimental Setup

We perform our experiments on two different network setups, described as follows:

1-D network Here we have a one-dimensional network of 41 sensors in a line. We assume that the overall sensing region is continuous and without overlaps. The mobility model of the object in this setting follows a uniform distribution for the probability of reaching a particular location, with the object's location at the next time step being either its current location or one of the locations on its left or right.

2-D network In this case, we have a 11×11 grid of 121 sensors. The assumption of continuous sensing range still holds but the sensor regions now overlap. We have used a simple overlap model where each sensor node's sensing region overlaps with that of 8 other nodes – the ones to the right, left, front and back and to the 4 corners: left-front,left-back,right-back and right-front, respectively. The mobility model of the object in this setting also follows a uniform distribution. However, the object's location at the next time step here is either its current location or one of the adjacent eight locations to its current location. Note the probability of the object moving to a particular location (including staying in its current location) is $\frac{1}{9}$ and if some of these locations fall outside the network then those probabilities are summed up and assigned to the probability that the object moves outside the network.

The simulations were conducted for a total of 10000 cycles for all algorithms. In all our experiments, we set $\Lambda = 5$. Recall that Λ denotes the maximum sleep time assigned to any sensor. Further, we set c and $\mathcal{K}$ for the single-stage cost function (12.5) to 1 and $N^2 \times c$ respectively. The rationale behind this assignment is that we consider encountering a tracking error to be as bad as keeping the square of the number of sensors in the network awake. Next, for both $\mathcal{QSA}$ as well as $\mathcal{TQSA}$, we set $\gamma = 0.9$ in our experiments. For $\mathcal{QSA}$, the step-sizes are chosen to be $\alpha(n) = \dfrac{1}{n}, n \geq 1$. For $\mathcal{TQSA}$, the choice of step-sizes is as follows:

$$b(n) = \frac{1}{n}, \quad a(n) = \frac{1}{n^{0.55}}, n \geq 1. \tag{14.5}$$

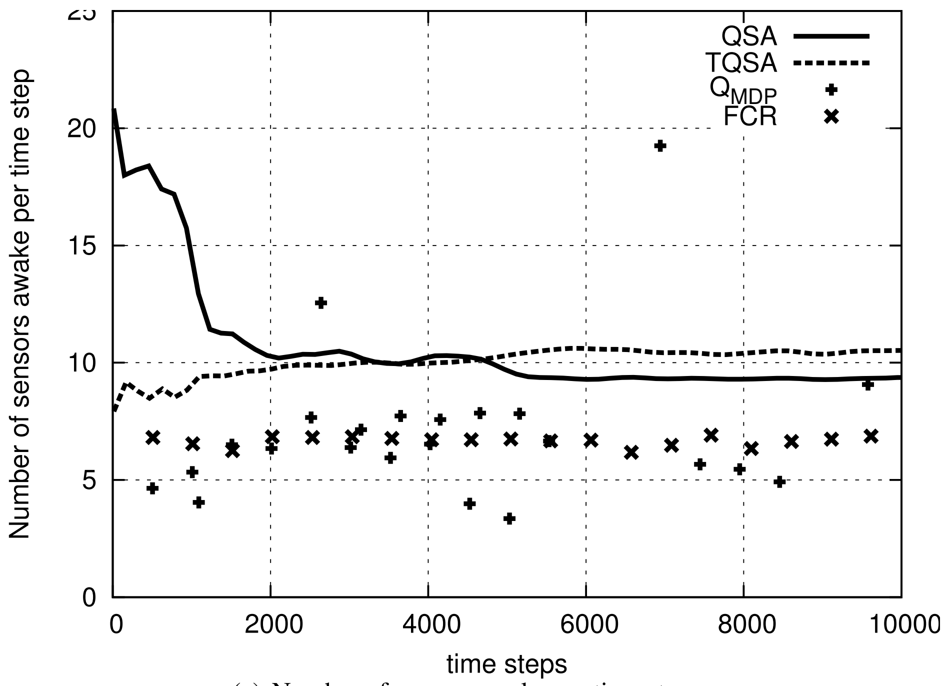
Further, for $\mathcal{TQSA}$, we set $\delta = 0.001$. This value is seen to show the best results for a range of δ-values in terms of the number of sensors awake as well as the number of detects (see Table 14.1 for a sensitivity analysis).

At each time instant, the object either moves randomly to an adjacent location or stays in its current location. However due to the large size of the action space, it is not feasible to evaluate each and every action vector. So, we deterministically prune off certain actions and present a smaller action space to the agent to choose from. For instance, if at location i, the probability of presence of the object is nearly 0 and the closest location j which has a reasonable probability of the object being there, is more than V_{max} distance away from i, it can safely be given a sleep time that is directly proportional to its distance from the current possible location of the object. We narrow down the action space in this manner and hence, we need to determine the sleep times for those sensors that are awake and within V_{max} proximity of the probable locations of the object's presence. This makes our algorithm fast irrespective of the network size. We have used the value of V_{max} as 1 in all our experiments. That is in one time step, the object can travel from its current location to any of its adjacent locations with uniform probability.

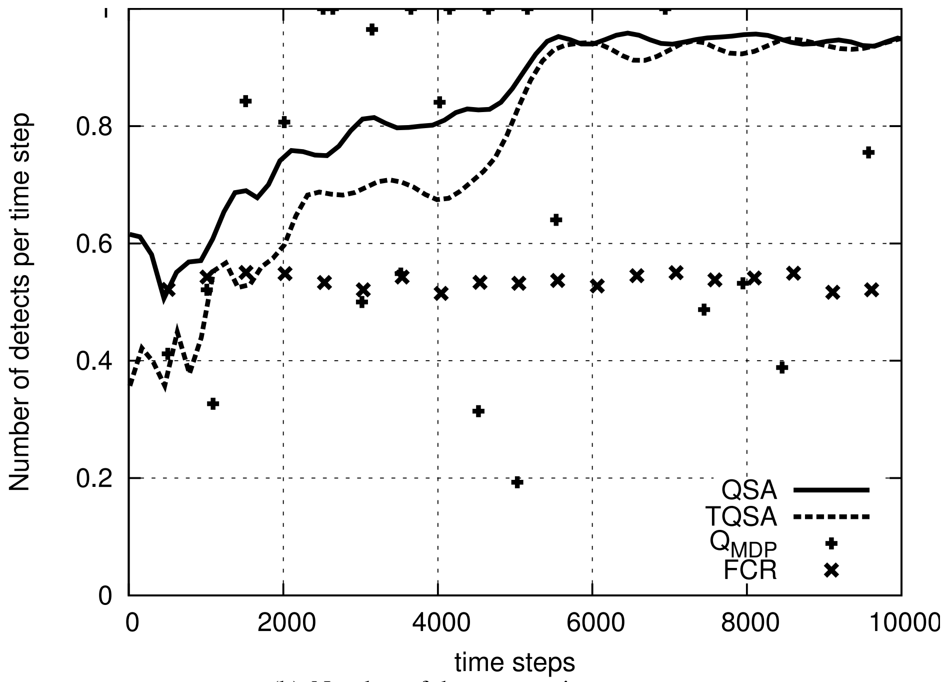
14.1.3 Results

We use the number of sensors awake per time step and the number of detects per time step as performance metrics for comparing the sleep–wake scheduling algorithms. The first metric is the ratio of the total number of sensors in the wake state to the number of time-steps, whereas the second metric is a similar ratio where the number of successful detects of the object is used instead.

Figure 14.1 presents the results of the number of sensors awake and the number of detects per time step on a 1-d network. Figure 14.2 presents similar results on a 2-d network.

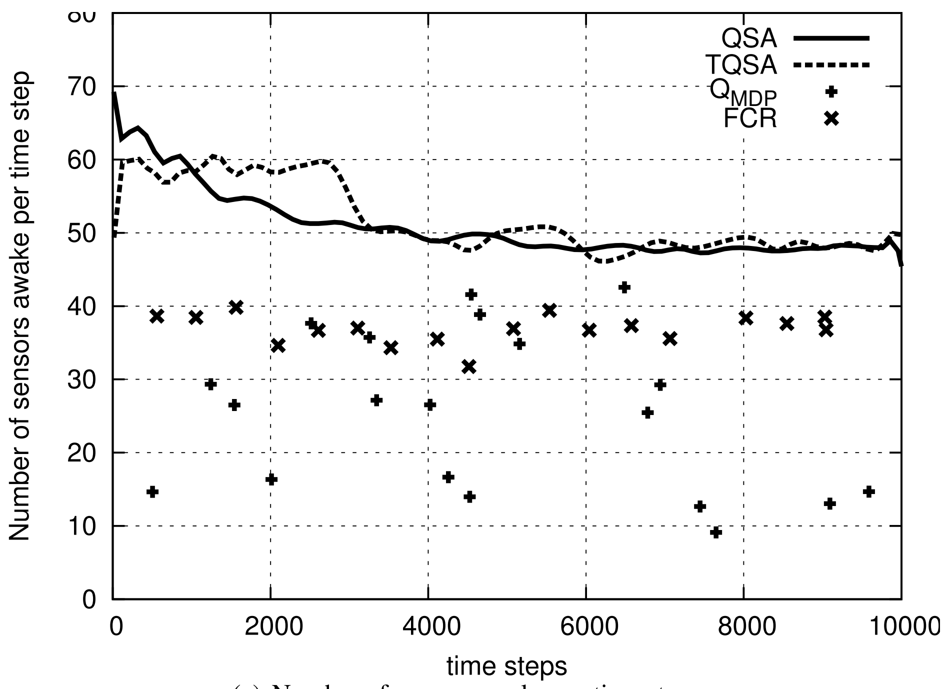


(a) Number of sensors awake per time step

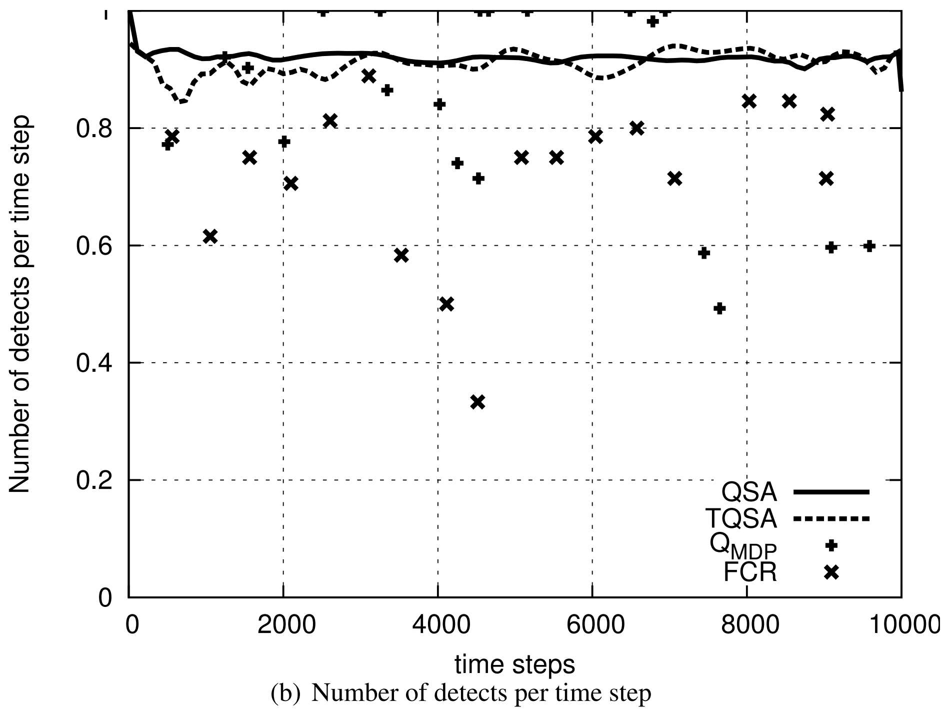


(b) Number of detects per time step

Figure 14.1: Tradeoff characteristics in 1-d network



(a) Number of sensors awake per time step



(b) Number of detects per time step

Figure 14.2: Tradeoff characteristics in 2-d network

The settings for these networks are shown in Figures 12.1 and 12.2, respectively.

Observation 1 $\mathcal{QSA}$ and $\mathcal{TQSA}$ achieve satisfactory levels of energy cost to tracking cost tradeoff.

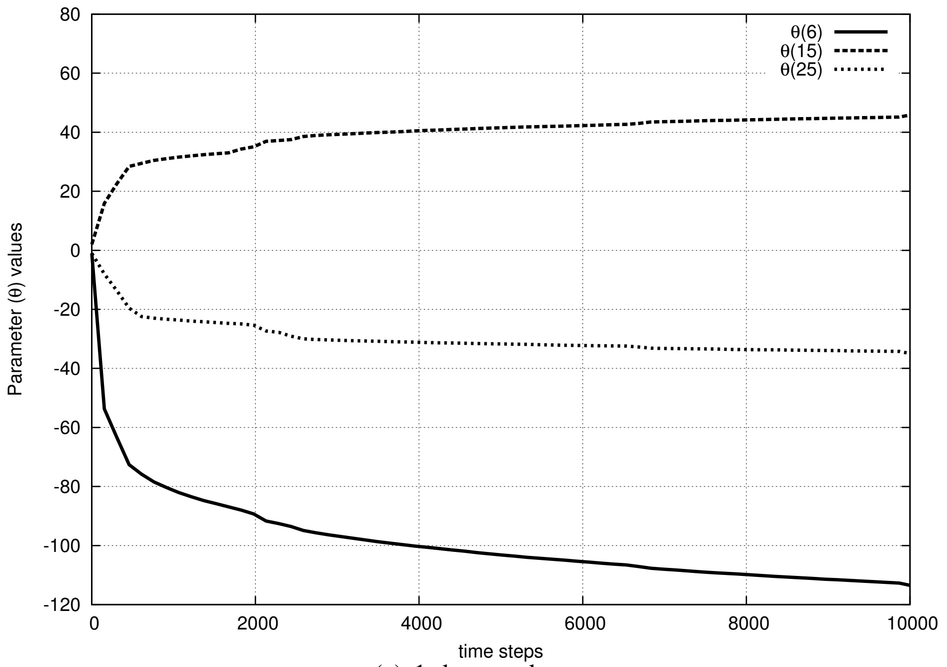
This is evident from the plots in Figures 14.1 and 14.2. While the number of sensors awake for FCR algorithm is lower than that for $\mathcal{QSA}$ and $\mathcal{TQSA}$ algorithms, the tracking accuracy is also significantly lower in comparison. This is true in results from both 1-d as well as the 2-d networks. On comparing our algorithms with $\mathrm{Q_{MDP}}$, we observe that while $\mathrm{Q_{MDP}}$ keeps a lower number of sensors awake, it also results in lower tracking accuracy. Further, note from Figure 14.1 that whenever $\mathrm{Q_{MDP}}$ has a higher tracking accuracy in comparison to our algorithms (cf. the numbers around the 2200th time instant), it pushes the number of sensors awake higher. Further, $\mathrm{Q_{MDP}}$ is not an incremental update learning algorithm and it does not stabilize the number of sensors awake and the tracking errors in the long-term. This is because, at each time instant, $\mathrm{Q_{MDP}}$ attempts to solve the balance equation (14.4) in an approximate fashion and no information about the solution thus obtained is carried forward to the future instants. This can also be observed in the variations in the number of sensors awake and the tracking accuracy of the $\mathrm{Q_{MDP}}$ algorithm. On the contrary, our algorithms learn the optimal sleeping policy for the individual sensors with contextual information being carried forward from one time step to the next. As evident in the performance plots of Figures 14.1 and 14.2, this results in a stable regime for the number of sensors awake and the tracking accuracy, unlike $\mathrm{Q_{MDP}}$.

Observation 2 The parameter vector θ converges for the $\mathcal{TQSA}$ algorithm.

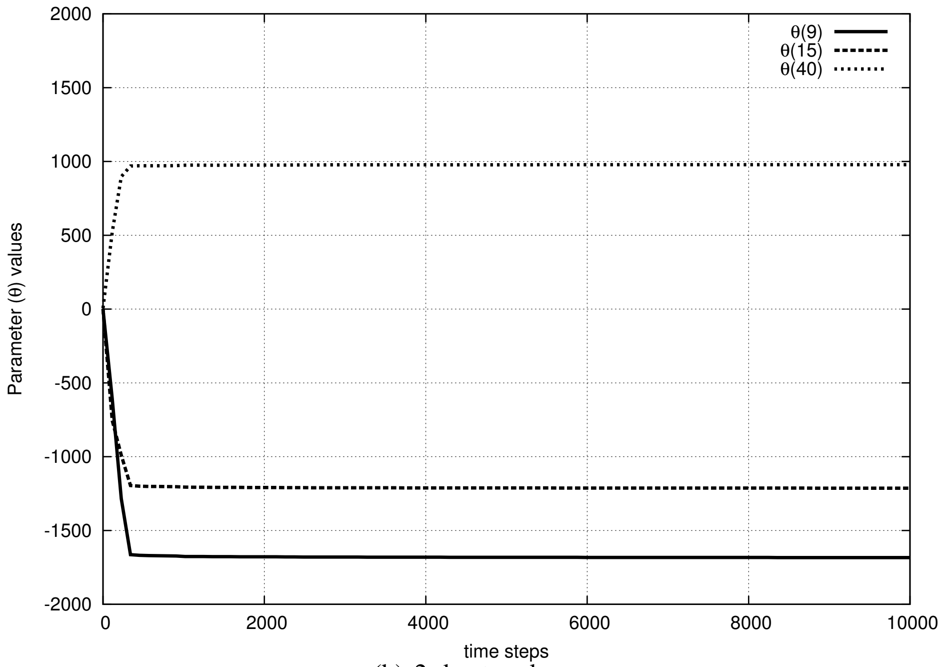
From Figures 14.3 (a) and (b), we observe that the Q-value parameter θ converges in both 1-d as well as 2-d networks. This is a significant feature of $\mathcal{TQSA}$ as it possesses theoretical convergence guarantees, unlike $\mathcal{QSA}$.

Observation 3 $\mathcal{QSA}$ and $\mathcal{TQSA}$ are empirically faster than $\mathrm{Q_{MDP}}$.

For instance, on a 1-d setting, the runtime of our algorithms $\mathcal{QSA}$ and $\mathcal{TQSA}$ was under 3 hours, whereas $\mathrm{Q_{MDP}}$ took 12 hours. The runtime advantage of our algorithms over the $\mathrm{Q_{MDP}}$ algorithm is probably due to the fact that $\mathrm{Q_{MDP}}$ employs a policy iteration procedure per sensor in each time step to determine the sleeping policy, whereas our algorithms employ function approximation techniques to achieve computational efficiency. This is also reflected in the run-times observed in a 2-d setting, where our algorithms were seen to converge in 3 hours, whereas the $\mathrm{Q_{MDP}}$ algorithm required several days.



(a) 1-d network



(b) 2-d network

Figure 14.3: Convergence trends of θ with $\mathcal{TQSA}$ - Illustration on 1-d and 2-d networks

Table 14.1: Tradeoff characteristics in 1-d network with $\mathcal{TQSA}$ for different values of δ

δ	**Number of sensors awake**	**Number of detects**
0.0001	10.17 ± 0.65	0.82 ± 0.14
0.001	$\mathbf{10.53 \pm 0.59}$	$\mathbf{0.94 \pm 0.08}$
0.01	10.54 ± 0.54	0.93 ± 0.08
0.1	10.46 ± 0.59	0.93 ± 0.08

14.2 Simulation Experiments – Average Setting

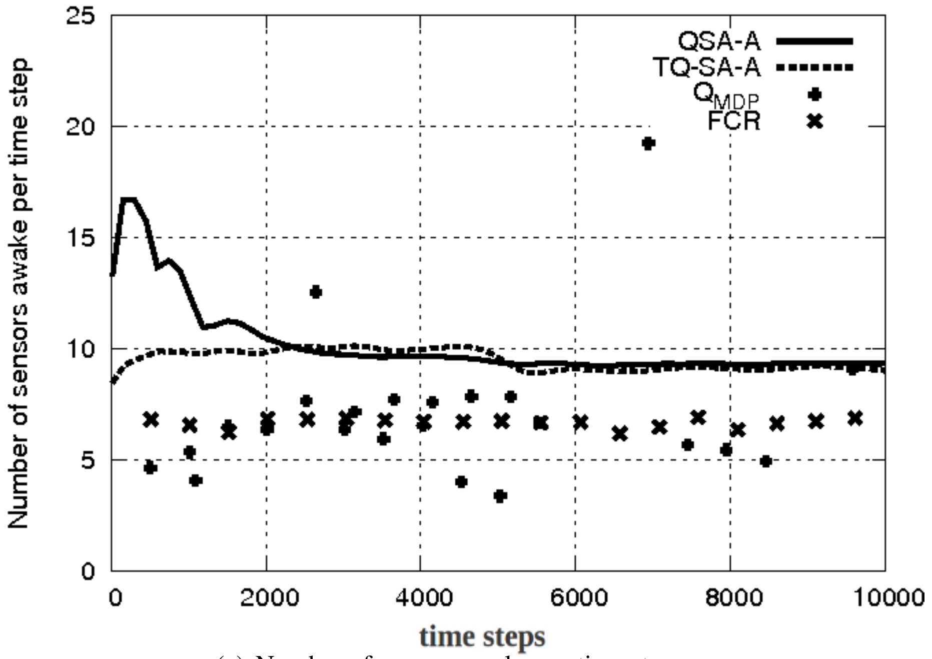
In this section, we present numerical results obtained from simulations for the average reward setting. We implemented the $\mathcal{QSA}$-$\mathcal{A}$ algorithm, which is the Q-learning with function approximation variant for the average setting as well as the $\mathcal{TQSA}$-$\mathcal{A}$ algorithm, which is the convergent two-timescale algorithm. For performance comparisons, we have once again used FCR and Q_{MDP} that were described in section 14.1. The network setups here are the same as the ones used in the discounted cost setting, i.e., a 1-d network with 41 sensors without overlap and a 2-d network with a 11×11 grid of 121 sensors with overlap. The overlap model is also the same as in section 14.1.

14.2.1 Implementation

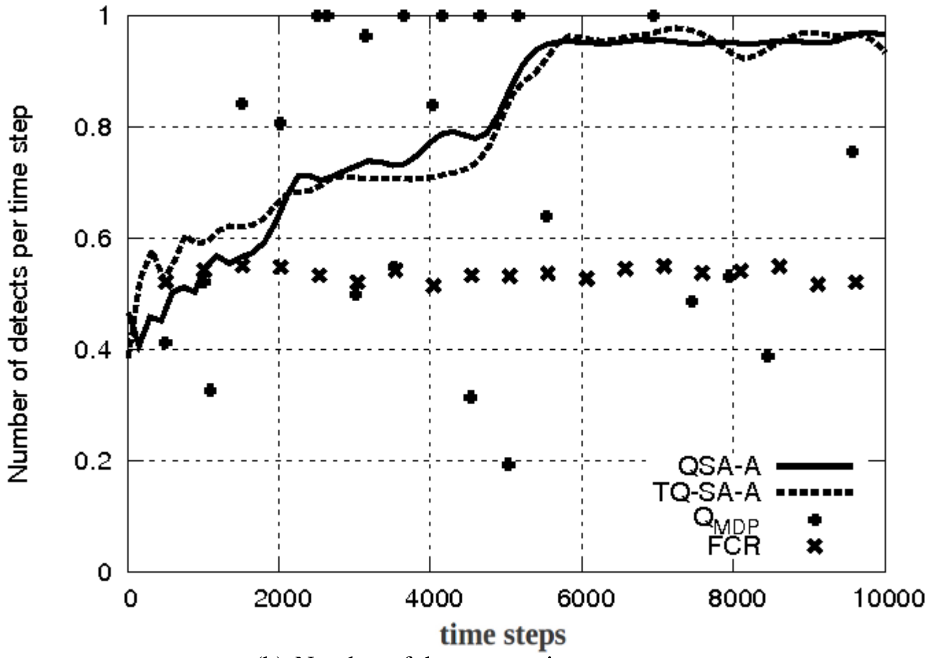
For $\mathcal{QSA}$-$\mathcal{A}$, we have taken the fixed state $\mathbf{s}''$ as $\langle \mathbf{p}_0, \mathbf{r} \rangle$ where $\mathbf{p}_0$ is the initial belief distribution and $\mathbf{r}$ is a random sleep time vector. The rationale here is that for this choice of the special state, there is a positive probability that the object will stay in its starting location after one time step. Moreover, as we explore the action space more towards the start of the experiment, it results in a positive probability of a random action being chosen at the initial time step leading to random remaining sleep times of the sensors after the first time step. Thus there is a positive probability that $\mathbf{s}''$ will be observed within the first few steps from the start of the experiment. The fact that a state satisfying such a criterion can be used as the fixed state has been established in Bertsekas [1995]. Other parameters remain the same as in section 14.1.

14.2.2 Results

Figures 14.4 and 14.5 present the number of sensors awake and the number of detects per time step, for each of the algorithms studied on two different network settings, respectively. As in the case of the discounted cost setting, we observe that both $\mathcal{QSA}$-$\mathcal{A}$ and

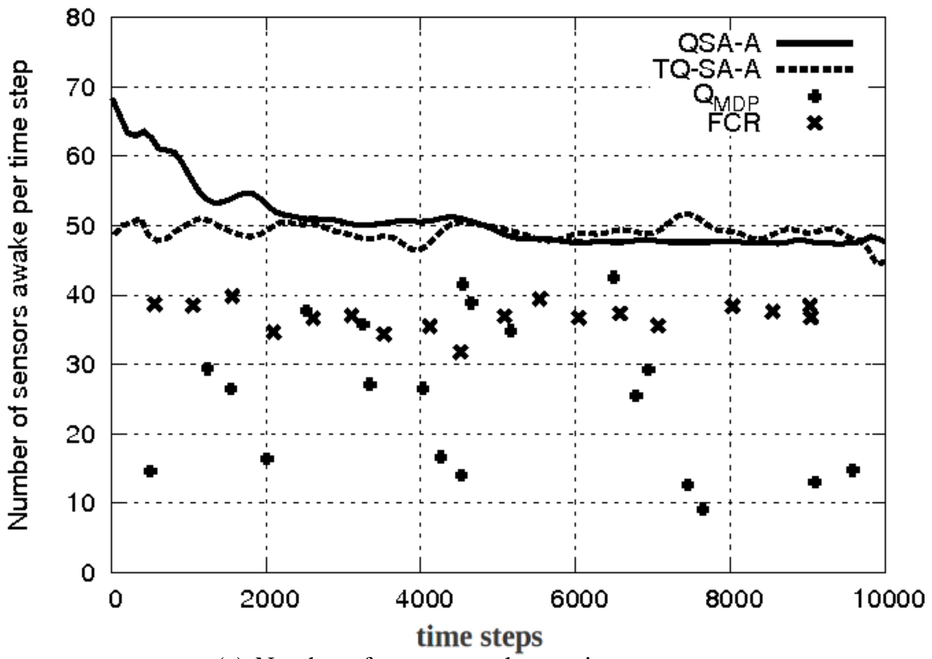


(a) Number of sensors awake per time step

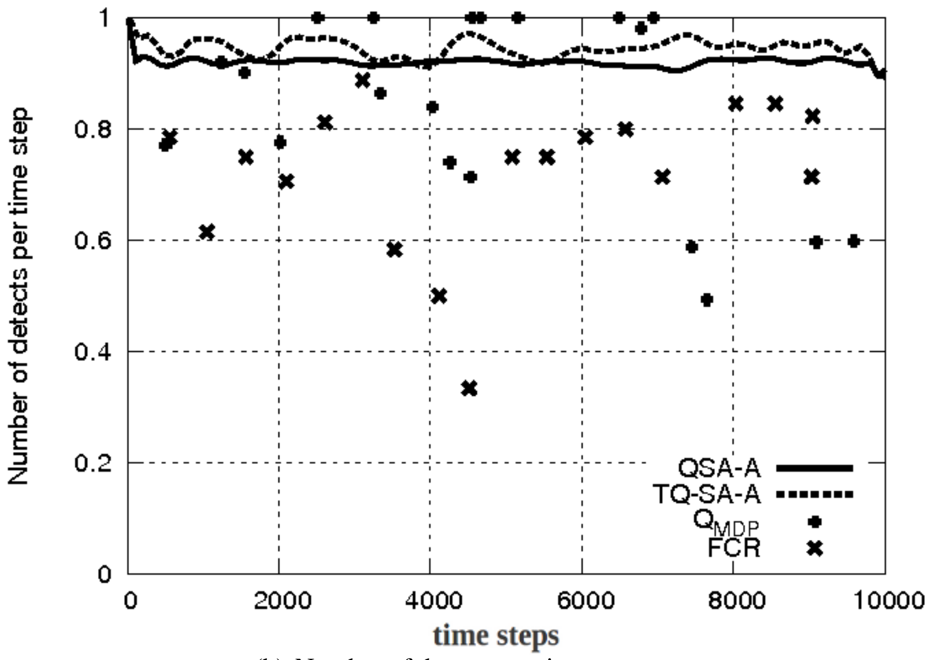


(b) Number of detects per time step

Figure 14.4: Tradeoff characteristics in 1-d network



(a) Number of sensors awake per time step



(b) Number of detects per time step

Figure 14.5: Tradeoff characteristics in 2-d network

$\mathcal{TQSA}$-$\mathcal{A}$ have been able to adaptively learn fairly good policies and consistently achieve better tradeoff in comparison to FCR and Q$_{\text{MDP}}$. The Q$_{\text{MDP}}$ exhibits fluctuating behaviour with extremely good tradeoffs for many episodes while poor tradeoffs at certain episodes. This, we suspect, is due to the underlying requirement of complete future observations in Q$_{\text{MDP}}$. This leads to assuming the propagation in the belief distribution and consequently when in the simulation there happen to be more frequent tracking errors, Q$_{\text{MDP}}$ fails to adapt to that situation and results in poor performance. Meanwhile our algorithms, $\mathcal{QSA}$ and $\mathcal{TQSA}$ adaptively learn the Q-value parameter θ which dictates the choice of actions that subsequently lead to a satisfactory tradeoff with greater consistency. The same arguments hold for the 2-d case as is evident from Figures 14.2 (a) and (b).

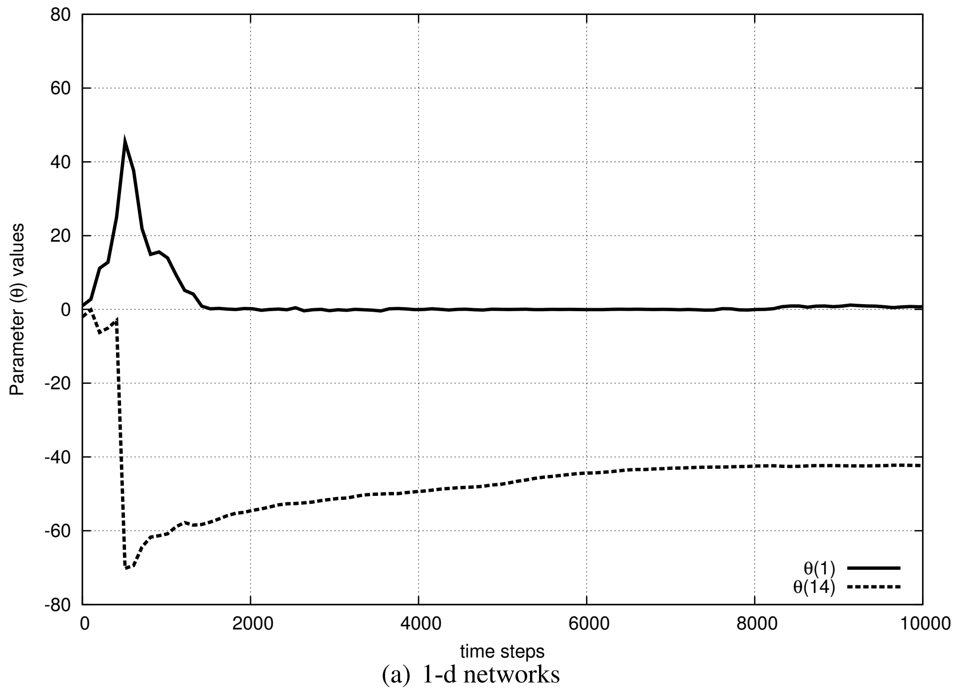
We observe from Figures 14.6 that the θ-values have converged for $\mathcal{TQSA}$-$\mathcal{A}$ in both 1-d and 2-d networks thereby indicating stability of the algorithm. This is an important advantage of $\mathcal{TQSA}$-$\mathcal{A}$ over $\mathcal{QSA}$-$\mathcal{A}$, as it possesses theoretical convergence guarantees and empirically performs on par (and sometimes better) with $\mathcal{QSA}$-$\mathcal{A}$.

14.3 Results – unknown P case

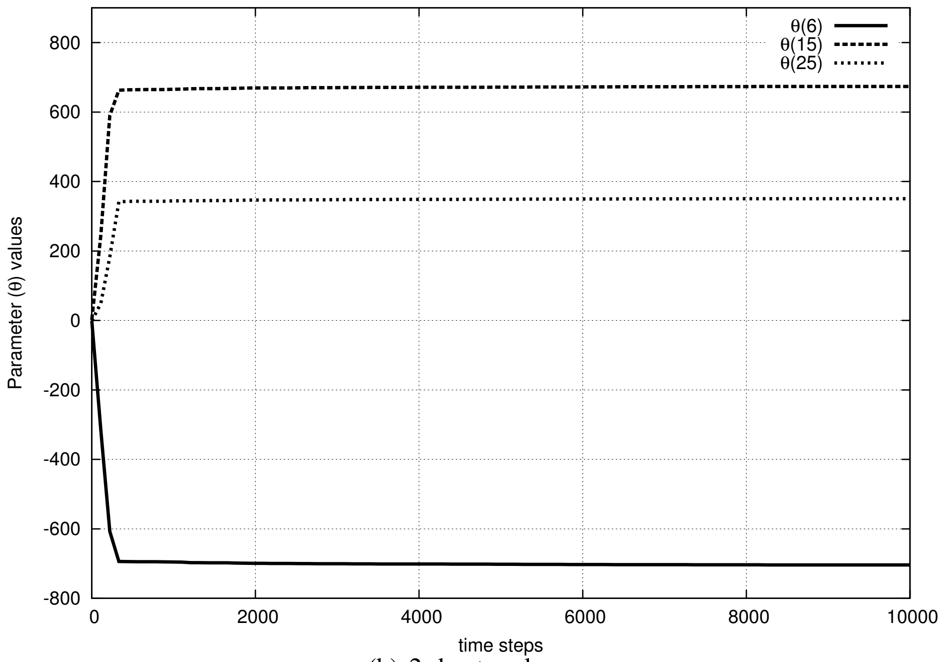
Figure 14.7 presents the performance plots (number of sensors awake and number of detects) from the experiments on a 2-d network, where $\mathcal{QSA}$ is combined with the P-estimation procedure (13.30). Figure 14.8 presents similar results for the $\mathcal{TQSA}$ algorithm. As illustrated in Figures 14.7–14.8, even with an unknown P setting, the algorithms $\mathcal{QSA}$-$\mathcal{UP}$ and $\mathcal{TQSA}$-$\mathcal{UP}$ show performance on par with their corresponding variants with known P, i.e., $\mathcal{QSA}$ and $\mathcal{TQSA}$ algorithms, respectively.

In the average setting, we studied the performance of the algorithms $\mathcal{QSA}$-$\mathcal{A}$-$\mathcal{UP}$ and $\mathcal{TQSA}$-$\mathcal{A}$-$\mathcal{UP}$. These algorithms combine $\mathcal{QSA}$-$\mathcal{A}$ and $\mathcal{TQSA}$-$\mathcal{A}$ respectively, with the P estimation procedure (13.30). We observe that, as in the discounted setting, $\mathcal{QSA}$-$\mathcal{A}$-$\mathcal{UP}$ and $\mathcal{TQSA}$-$\mathcal{A}$-$\mathcal{UP}$ exhibit performance on par with their known P variants, i.e., $\mathcal{QSA}$-$\mathcal{A}$ and $\mathcal{TQSA}$-$\mathcal{A}$ respectively. This is evident from Figure 14.10.

Further, we also observe that in the algorithms for the discounted setting ($\mathcal{QSA}$-$\mathcal{UP}$ and $\mathcal{TQSA}$-$\mathcal{UP}$) as well as the algorithms for the average setting ($\mathcal{QSA}$-$\mathcal{A}$-$\mathcal{UP}$ and $\mathcal{TQSA}$-$\mathcal{A}$-$\mathcal{UP}$), the estimates P_k of the transition probability matrix P converge to the true P. This is illustrated by the convergence plots corresponding to a 2-d network in Figure 14.9 for the discounted setting and in Figure 14.11 for the average setting. In particular, Figure 14.9 shows that for a 2-d network, the value of $P_k(i,j)$, where i corresponds to the



(a) 1-d networks



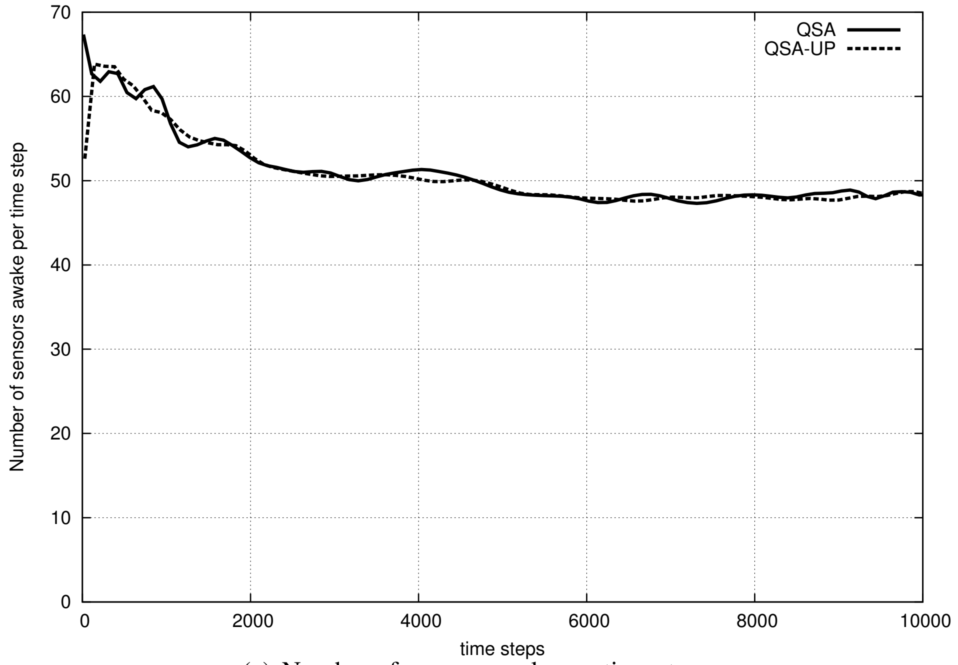
(b) 2-d networks

Figure 14.6: Convergence trends of θ with $\mathcal{TQSA}$-$\mathcal{A}$ – Illustration on 1-d and 2-d networks

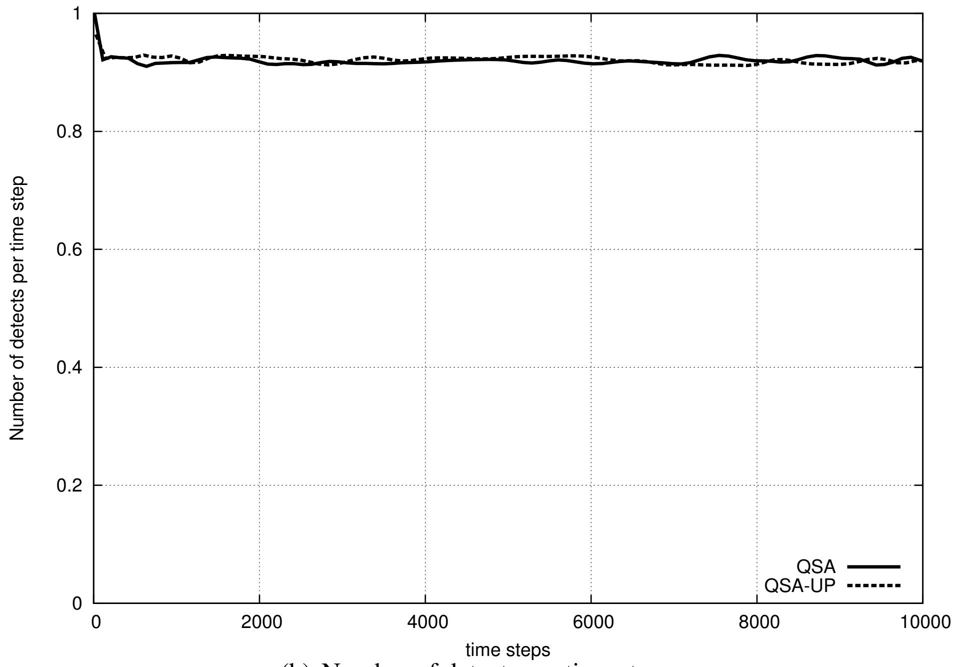
$(6,6)$th cell and j corresponds to the $(6,5)$th cell converges to the actual $P(i,j)$ value of 0.11, for both $\mathcal{QSA}$-$\mathcal{UP}$ as well as $\mathcal{TQSA}$-$\mathcal{UP}$. In contrast, the Q_{MDP} algorithm requires full knowledge of the distribution of the object movement and hence, cannot be applied in the setting of unknown P.

14.4 Conclusions and Future Work

Wireless sensor networks show a lot of promise when we consider monitoring applications, as they are cheap, easy to deploy and require little maintenance. Designing a WSN that achieves close to optimal performance from an energy efficiency perspective in an object tracking application is a challenging task. In other words, the problem we studied is to track an object moving randomly through a dense network of wireless sensors such that the entire sensing region has no discontinuities and the aim is to conserve energy by optimizing the sleep times of the sensors in a way that does not compromise on the tracking quality. Reinforcement learning presents an interesting paradigm for solving such design problems. We propose novel RL-based algorithms - with both long-run discounted and average cost objectives - for solving this problem. All our algorithms incorporate function approximation and feature-based representations to handle the curse of dimensionality. Further, the feature selection scheme used in each of the proposed algorithms intelligently manages the energy cost and tracking cost factors, which in turn, assists the search for the optimal sleeping policy. The results from the simulation experiments suggest that our proposed algorithms perform better than two recently proposed algorithms from Fuemmeler and Veeravalli [2008]. Amongst our algorithms, we observed the convergent two-timescale Q-learning algorithm showed similar performance as compared to Q-learning. However, as explained previously, $\mathcal{TQSA}$ is seen to converge theoretically while Q-learning with function approximation may not converge (Baird [1995], Tsitsiklis and Van Roy [1997]). Hence, $\mathcal{TQSA}$ is a reliable and provably convergent algorithm unlike $\mathcal{QSA}$ that may or may not work in certain settings, see Bhatnagar and Lakshmanan [2012] for applications in multi-stage queueing networks where $\mathcal{TQSA}$ is clearly better in comparison to $\mathcal{QSA}$. We also incorporated a novel estimation scheme for the transition probability matrix when the object movement distribution is not known. We combined this online estimation procedure for P with the two proposed algorithms $\mathcal{QSA}$ and $\mathcal{TQSA}$, respectively, using multi-timescale stochastic approximation. Empirically, we observed that the transition probability estimates converged to the true values in all experimental settings

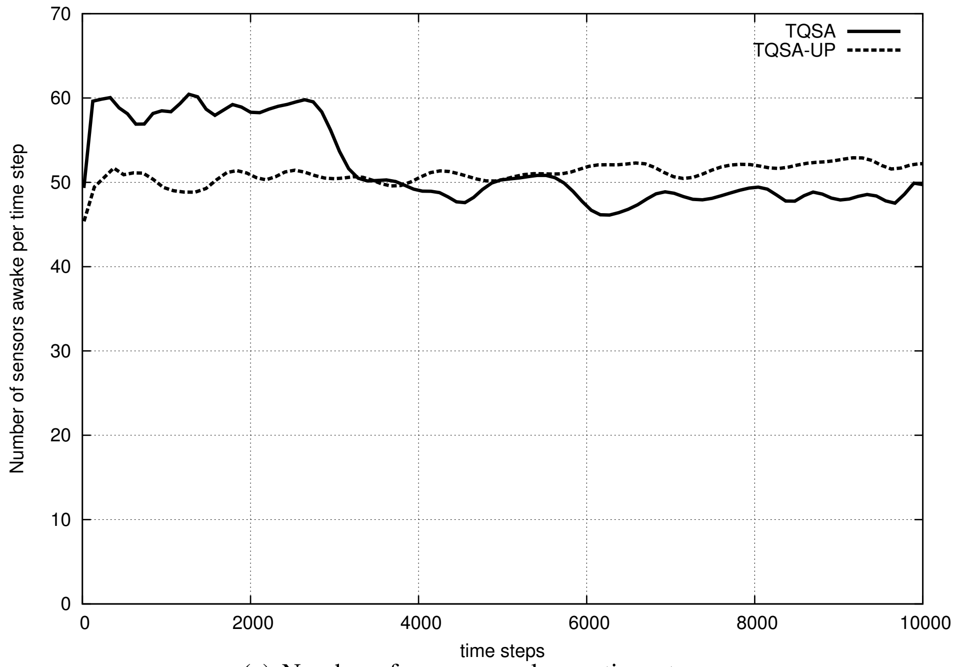


(a) Number of sensors awake per time step

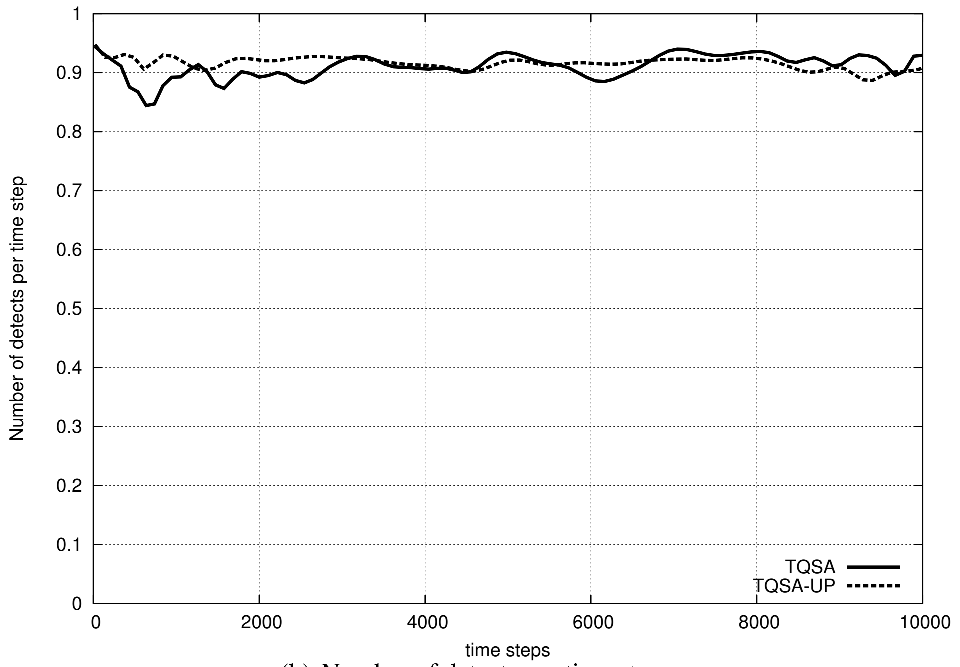


(b) Number of detects per time step

Figure 14.7: Tradeoff characteristics in 2-d network with known and unknown P for $\mathcal{QSA}$ algorithm

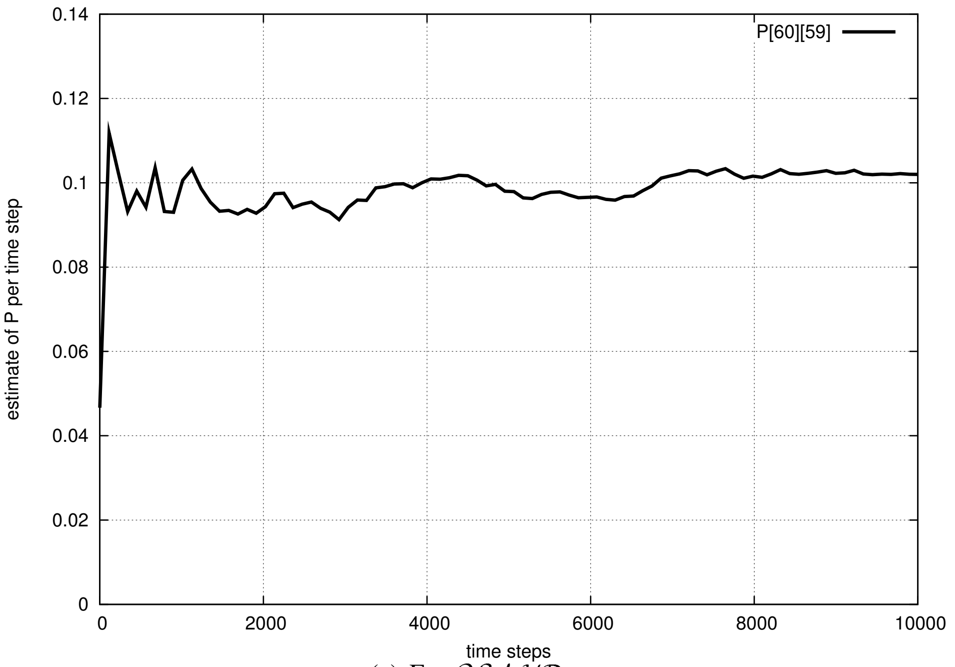


(a) Number of sensors awake per time step

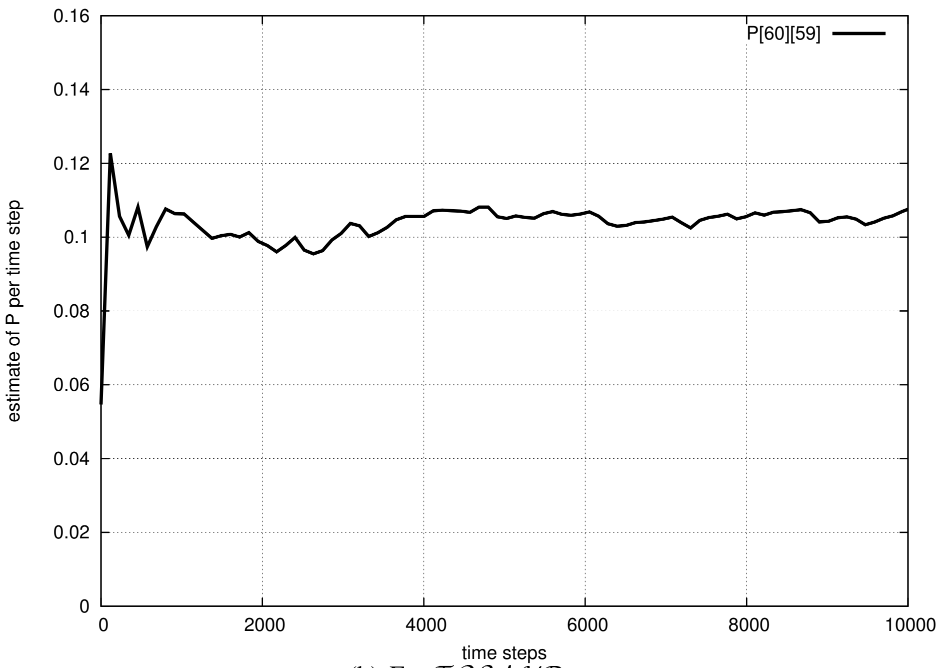


(b) Number of detects per time step

Figure 14.8: Tradeoff characteristics in 2-d network with known and unknown P for $\mathcal{TQSA}$ algorithm

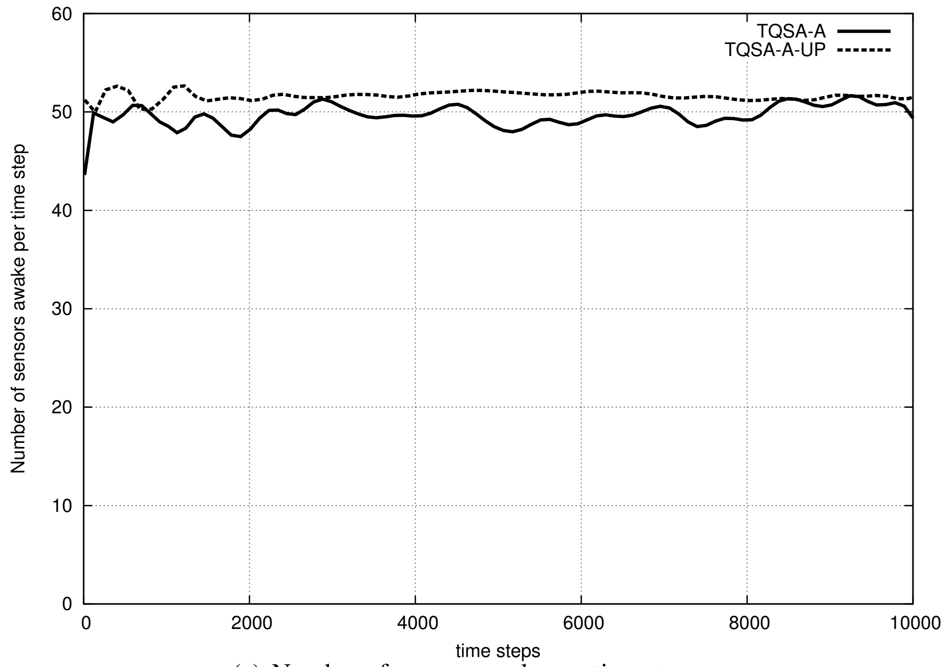


(a) For $\mathcal{QSA}$-$\mathcal{UP}$

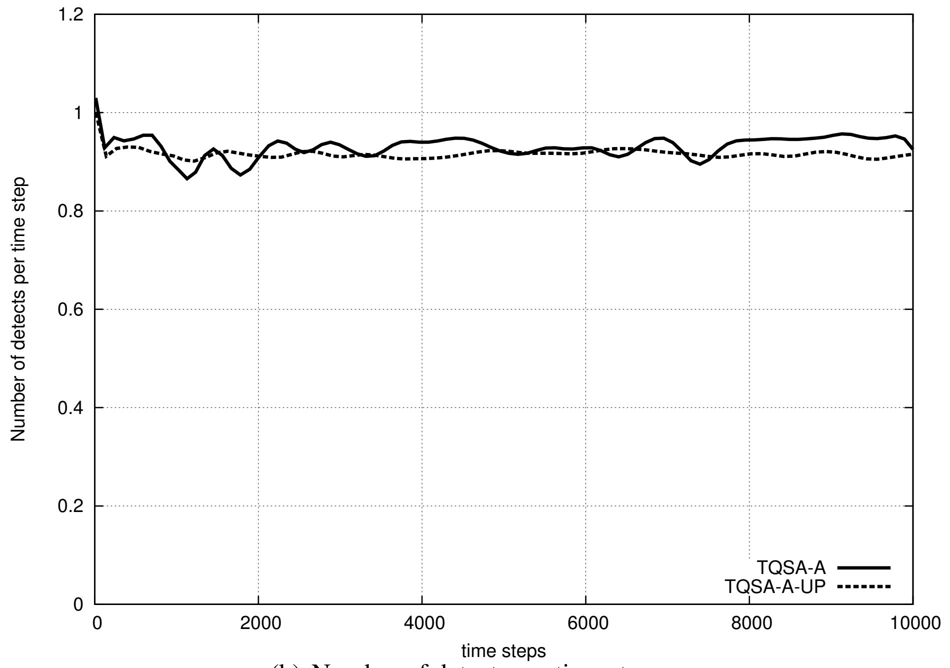


(b) For $\mathcal{TQSA}$-$\mathcal{UP}$

Figure 14.9: Convergence of P_k for both $\mathcal{QSA}$ and $\mathcal{TQSA}$ algorithms - illustration using $P_k(i, j)$ where i corresponds to the $(6, 6)$th cell and j to the $(6, 5)$th cell in the 2-d network.

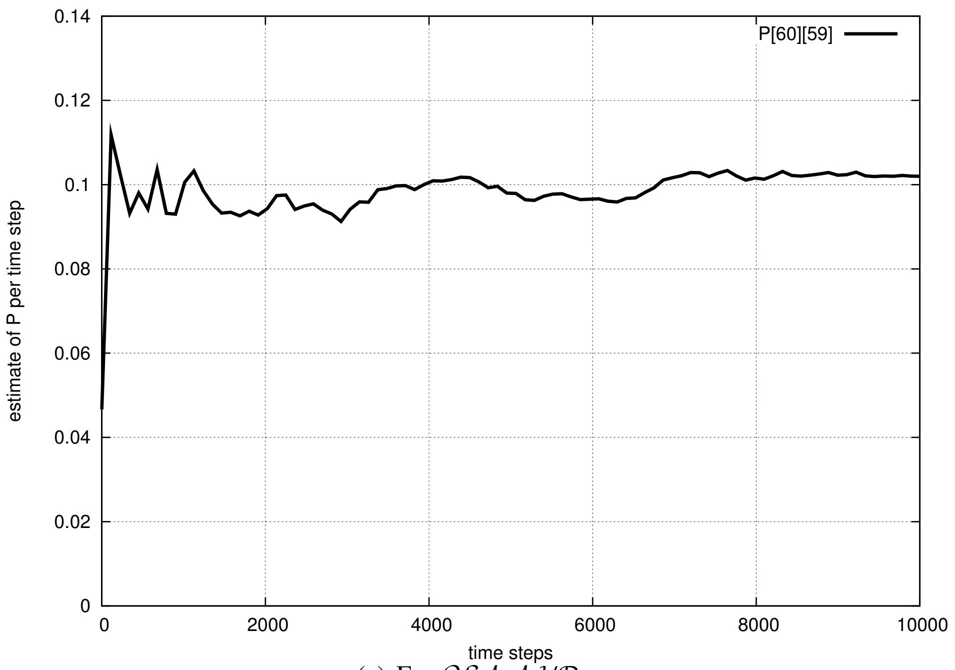


(a) Number of sensors awake per time step

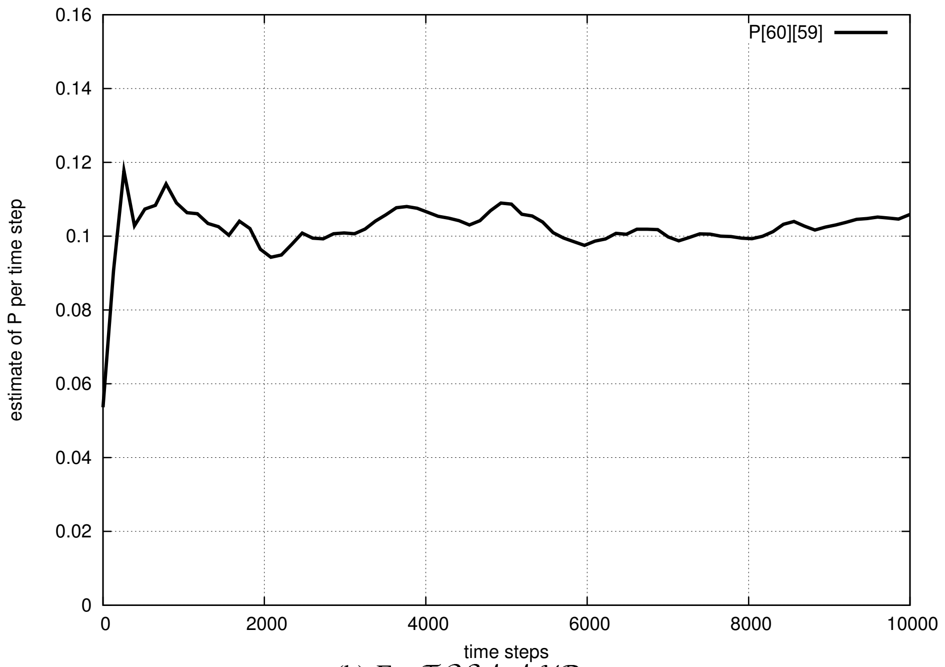


(b) Number of detects per time step

Figure 14.10: Tradeoff characteristics in 2-d network with known and unknown P for $\mathcal{TQSA}$-$\mathcal{A}$ algorithm



(a) For $\mathcal{QSA}$-$\mathcal{A}$-$\mathcal{UP}$



(b) For $\mathcal{TQSA}$-$\mathcal{A}$-$\mathcal{UP}$

Figure 14.11: Convergence of P_k for both $\mathcal{QSA}$-$\mathcal{A}$ and $\mathcal{TQSA}$-$\mathcal{A}$ algorithms - illustration using $P_k(i, j)$ where i corresponds to the $(6, 6)$th cell and j to the $(6, 5)$th cell in the 2-d network.

considered.

However, this work has several open avenues for further research. Some of these are:

- Sleep-Wake scheduling for object tracking under noisy observations by the sensors. Random interference leads to noisy observations by the sensors. Thus, the central controller needs probabilistic data association techniques to distinguish between observations that truly associate with the data and false reports.

- Sleep-Wake scheduling for multi-object tracking. In this case, the objective will be to find a tradeoff between observing more number of objects while keeping many sensors asleep.

- Decentralized object tracking in absence of central controller using cooperative decision making by the sensor nodes. In the absence of the central controller, sensor nodes would have to participate in decision making and routing. Especially in the case of overlapping sensing ranges, the nodes can compete among themselves for greater sleep times in order to save more energy. This problem can be addressed by applying both cooperative and non-cooperative game theoretic techniques and designing a mechanism for the same.

Part IV

Mechanism Design

Several key economic scenarios involve agents having limited capacities whose types change with time, e.g., service workers attending to service requests, power plants supplying to power grids, and machines connected to computing grids. Dynamic mechanisms have been proposed to address the issue of dynamic types. Also, a few mechanisms have been proposed to account for limited capacities in static settings. This part proposes two novel mechanisms – first, a static mechanism having capacity constraints and second, a dynamic mechanism in a setting where the types are dynamic and include a unit cost and capacity. We show that each of our mechanisms is ex-post incentive compatible, ex-post individually rational, and socially efficient.

Chapter 15

Static mechanism design with capacity constraints

Mechanism design has been a key technique for decision making in various economic scenarios. In many of the economic scenarios of interest, the agents' types change with time, e.g., the cost of production of a good. Also, in most real economic scenarios, agents have limited capacities, for instance, the supply capacity of a power plant on a given day. In this chapter, we consider a static setting and handle the extension to a dynamic setting in which the private types evolve over time according to a Markov process, in the next chapter. We first propose an enhanced static mechanism that provides a correction to the transfer scheme of Dash et al. [2007] to deal with over-reported capacity. The novelty here is a variable penalty scheme that penalizes an agent to the extent of damage his misreport caused in the allocation to the other agents. Next, we extend this idea for the domain that lies at the intersection of the two scenarios – agents having limited capacities and types that change with time in Chapter 16.

We consider a procurement setting in which a single buyer attempts to obtain D units of a good from a set of n sellers. Each seller i is capacity-constrained in the sense that they can produce up to c_i units at a constant unit price u_i. The capacity and unit price are private information. While a simple VCG-like transfer scheme ensures truthful unit price reports from the individual agents, the same is not true w.r.t. capacity type element (See Example 1).

Several mechanisms have been proposed to deal with the case of limited capacity of agents in the static case, but we focus on the work of Dash et al. [2007]. In their work, the limited capacity of agents is accommodated by ensuring that an agent is never allocated more than its reported capacity and by imposing a fixed penalty of a positive δ to ensure truthtelling w.r.t. the capacity type element. Here, if an agent misreports its capacity, produces less units than allocated, and is penalized as above, the agent would derive utility of $-\delta$. In a static setting, such a fixed penalty scheme is shown to be strategyproof, i.e., truthful type reports is a dominant strategy for each of the agents, see Dash et al. [2007].

In spite of the fact that a fixed δ-penalty achieves strategyproofness as shown by Dash et al. [2007], in practical competitive scenarios, agents could be willing to incur a loss of δ, if it is relatively small compared to the major losses inflicted to other agents due to the misreport. We incorporate the preference of agent to harm other agents via capacity misreport through an additive term that captures the agent's allocation under the reported capacity. Note that the reported capacity of an agent may be overstated, resulting in an increased allocation at the cost of other agents. In this modified utility setting, as we demonstrate via examples later, the mechanism of Dash et al. [2007] does not ensure truthful capacity reports. In other words, a fixed δ-penalty does not deter capacity misreports by an agent because the degree of losses inflicted on other agents via a capacity misreport is a variable, which depends on the allocation and hence the individual agents' reported types.

We present a static mechanism $\mathcal{MC}$ that enhances the mechanism from Dash et al. [2007], to incorporate a novel variable penalty scheme. The penalty imposed on agents misreporting their capacities is the same as the loss of allocation to other agents due to misreported capacities. We formally show that $\mathcal{MC}$ is strategyproof. In particular, we show that $\mathcal{MC}$ ensures truthful capacity reports from the individual agents in a setting where the agents have a preference to overstate capacity and hence, negatively impact the allocation of other agents.

Dynamic mechanism design is a key area, especially when the center makes allocations to participating agents repeatedly. For example, service workers employed by a service provider would typically attend to thousands of service requests during their employment period. Each service worker has a limited amount of time (which keeps changing every day) and her estimate of the amount of time required to resolve service requests also changes with skill gain or decay. For an efficient allocation, i.e., for minimizing the total time required to resolve a set of service requests, it is not enough for the provider to consider the current types of service workers but also the future horizon. Significant results have been already established in this area, see Bergemann and Välimäki [2010], Cavallo [2008] and Cavallo et al. [2009]. We focus on the work of Bergemann and Valimaki (B&V) (Bergemann and Välimäki [2010]), that proposes a dynamic pivot mechanism that is truthful, efficient, and individually rational. The optimal allocation is computed as the expectation on the discounted sum of valuations of each agent with his payoff computed based on his marginal contribution. However, B&V's mechanism considers a setting where a common resource is allocated to an agent at every time step and the agent's capacities are not considered. Hence the question of capacity misreports and the consequent need to penalize misreporting agents does not arise in their work.

Unfortunately, capacity constraints are nontrivial to accommodate in the design of truthful and socially efficient dynamic mechanisms. The main challenge is that in a given time step, the payoff to an agent cannot be determined until the time that the actual units produced by an agent is known. If the allocation π to agent i to produce π_i units is made at time t_1 based on the expected discounted sum of current and future valuations of all agents, and the actual number of units $\bar{c}_i$ produced by agent i is known at a later time $t_2 > t_1$, then the payoff to agent i for the allocation π cannot be made prior to t_2 because it depends on $\bar{c}_i$. However, because the valuations are discounted in the future, the marginal contribution computed at t_1 may not yield allocative efficiency at t_2. This is because, intuitively, a payment made at t_1 is worth more than the same payment at t_2. The question then is what should the payoff to agent i be at time t_2 for allocation π such that the resulting mechanism is truthful in terms of unit costs as well as capacities and is socially efficient? Note that this question does not arise for B&V's work because the payoffs can be made at t_1, i.e., the time of allocations.

To answer the above question, we develop a delayed transfer scheme that consists of a marginal contribution component and a penalty component. The marginal contribution component is similar to the transfer scheme in the dynamic pivot mechanism (Bergemann and Välimäki [2010]). However, we establish via a counterexample that providing the marginal contribution alone is not sufficient to ensure incentive compatibility of the mechanism. For this purpose, a penalty component is necessary in the transfer to the agents. With this motivation, we propose $\mathcal{DMC}$, the first dynamic mechanism to tackle capacity constraints. While $\mathcal{DMC}$ is an extension of $\mathcal{MC}$ for the dynamic case, a key novelty of $\mathcal{DMC}$ is the delayed transfer scheme where payoffs are adjusted based on the period elapsed since allocation. We establish that $\mathcal{DMC}$ is allocatively efficient, within period ex-post incentive compatible, and individually rational.

The rest of this chapter is organized as follows: In Section 15.1, we survey previously proposed literature that covers both static as well as dynamic mechanisms. In Section 15.2, we describe in detail the static mechanism $\mathcal{MC}$ with a novel variable penalty structure.

In the following chapter, we present the dynamic mechanism framework with capacity constraints and develop an extension of the static mechanism $\mathcal{MC}$ to the dynamic setting.

15.1 Literature review

For a survey of the core results to date in the area of dynamic mechanism design, the reader is referred to Cavallo [2009]. The authors in Cavallo et al. [2009] present a dynamic mechanism without capacity constraints wherein agents become unavailable for a period after the allocation is made and the payoffs can only happen when the agents eventually become available. The payoffs are adjusted upwards by compounding the discount factor for all the time points during which the agent is unavailable. In Athey and Segal [2007], the authors obtain a Bayes-Nash incentive compatible, efficient and budget-balanced mechanism for a persistent-population, dynamic-type environment with private values and independent type transitions. In Nath et al. [2011], the authors propose a two-phased dynamic mechanism that involves type-reporting and value-reporting stages, while the transfer scheme is similar to that of the dynamic pivot mechanism (Bergemann and Välimäki [2010]). In Deb [2008], a method for a dynamic setting where the buyers learn their value of an object over multiple periods is proposed. At each period, the agents update their valuations based only on their consumption experience. Such a setting applies to our work wherein the agents learn the cost of performing a unit of work and capacity over multiple periods, resulting in a higher social welfare over periods. In Kakade et al. [2011], the authors consider the problem of maximizing revenues while preserving efficiency in dynamic settings. While revenue maximization is an important property, it remains to be seen whether it can be achieved without compromising incentive compatibility and individual rationality in the dynamic settings. An indirect, efficient, and revenue maximizing mechanism for a setting where the arrivals of buyers, the demand, and supply are stochastic is Said [2010].

In Gerding et al. [2010], the authors consider a static case where the work units come with deadlines and the center wants to maximize the probability of work completion within the deadlines. They show how this can be achieved by having the provider agents report their service times. Deadlines are considered in the context of a dynamic setting in Mierendorff [2009]. An optimal mechanism is derived there for the case when incentive constraint for the deadline can be neglected as well as for a specific case of two buyers where the incentive constraint cannot be neglected.

While the works outlined above consider interesting and realistic extensions to dynamic mechanism design, they do not consider capacity constraints unlike our work.

15.2 Static Mechanism with constraints ($\mathcal{MC}$)

15.2.1 The Setting

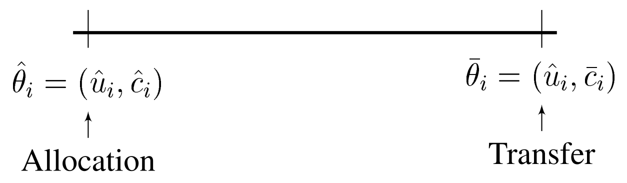
Consider a setting involving N agents (numbered $1, 2, \ldots, N$) that are capable of performing certain manufacturing tasks [1]. A central controller receives a task of manufacturing D items of a certain commodity. Each agent i is characterized by his type information - a tuple (u_i, c_i), where u_i is the unit price and c_i is the maximum capacity that agent i can provide. We assume here that the agents are aware of their capacities and that their collective capacity exceeds the demand D. We also assume that the collective capacity exceeds the demand D even in the absence of an agent i, where $i = 1, \ldots, N$. The type vector θ is given by $\theta = (\theta_1, \ldots, \theta_N)$, where $\theta_i = (u_i, c_i)$. We let $\theta_i \in \Theta_i$, where Θ_i denotes the type space of agent i. The joint type Θ is then given by $\Theta = \Theta_1 \times \ldots \times \Theta_N$. The problem at the central controller is to obtain an efficient allocation such that the demand is met and the total cost is minimized as well. We denote an allocation by $y = (y_1, \ldots, y_N)$, where y_i is the number of items that have to be produced by agent i.

The central controller thus solves the following optimization problem:

$$\begin{aligned}
&\text{Find } \pi(\theta) = \underset{y \in \mathcal{Y}}{\text{argmin}} \sum_{j=1}^{N} u_j y_j \\
&\text{s.t. } 0 \leq y_j \leq c_j, j = 1, 2, \ldots, N, \\
&\text{and } \sum_{j=1}^{N} y_j = D.
\end{aligned} \tag{15.1}$$

In the above, $\mathcal{Y}$ is the set of all possible allocations. The policy $\pi(\theta)$ obtained as a solution to (15.1) is called socially efficient and we use π_i to denote the allocation to agent i under the efficient policy π. For the purpose of defining payments to agent i in the marginal sense, we also require the solution to (15.1) with agent i removed, i.e., a solution to the following optimization problem:

[1]Note that the results of this chapter would apply to any domain where the agents are allocated some work for the external customers.


$\hat{\theta}_i = (\hat{u}_i, \hat{c}_i)$
Allocation
$\bar{\theta}_i = (\hat{u}_i, \bar{c}_i)$
Transfer

$$\begin{aligned} &\text{Find } \pi_{-i}(\theta) = \underset{y \in \mathcal{Y}}{\operatorname{argmin}} \sum_{j \neq i} u_j y_j \\ &\text{s.t. } 0 \leq y_j \leq c_j, j \neq i, \\ &\text{and } \sum_{j \neq i} y_j = D. \end{aligned} \tag{15.2}$$

The allocation to agent j under the policy obtained as a solution to (15.2) is denoted $\pi_{-i,j}$. The problem now is to define a mechanism that results in truthful type reports by the individual agents. This will happen if the agent's utility function is maximized under true reports, which constitutes *strategyproofness*. Before formalizing our mechanism, we define the utility function U_i of an agent i in our setting.

Definition 15.1 *The utility function U_i of agent i is* quasi-linear *and is given by*

$$U_i(\pi, t_i, \hat{\theta}) = t_i - u_i \bar{c}_i(\hat{\theta}) + \pi_i((u_i, \hat{c}_i), \hat{\theta}_{-i}), \tag{15.3}$$

In the above, $\bar{c}_i(\hat{\theta})$ denotes the achieved capacity of agent i, given that its allocation was $\pi_i(\hat{\theta})$. Note that if the agent has overstated its capacity, then the achieved capacity can in fact be lesser than the allocation (See Figure 15.2.1). The second term in (15.3) signifies the cost to agent i under the allocation policy π and t_i is the payment that agent i receives from the central controller. The rationale behind the final term $\pi_i(\cdot)$ above is that the utility an agent derives increases linearly with its allocation, even if it is at the cost of other agents. This is evident by noting that $\pi_i(\theta) = D - \sum_{j \neq i} \pi_j(\theta)$. Thus, the final term captures the preference of an agent to ruin others by overstating capacity. This is unlike the utility function formulation in Dash et al. [2007], where the utility function contained the first two terms of (15.3).

Remark 16 *One could also use a reward function $f(\cdot)$ that converts from allocation units to money or reward points in the utility function (15.3) as follows:*

$$U_i(\pi, t_i, \hat{\theta}) = t_i - u_i \bar{c}_i(\hat{\theta}) + f(\pi_i((u_i, \hat{c}_i), \hat{\theta}_{-i})), \tag{15.4}$$

The exact details of the reward function $f(\cdot)$ is immaterial for the discussion here, as long as it is ensured to be monotonic in its parameter. Further, we do not incorporate the reward function in the material next and the analysis can be seen to be trivially extended to include $f(\cdot)$.

Formally, a mechanism is a tuple (π, t), where π is the policy component that maps the types to an allocation vector and t denotes the transfer scheme, with $t_i : \Theta \to \mathbb{R}$ denoting the monetary payment received by agent i. The policy employed in our mechanism is socially efficient (as defined by (15.1)). Further, the transfer scheme t ensures strategyproofness and individual rationality - properties desirable of any mechanism. We later prove these properties formally.

Definition 15.2 *A mechanism is* strategyproof *if truth-telling constitutes a dominant strategy for the agents, i.e., for all agents $i = 1, 2, \dots, N$,*

$$U_i(\pi, t_i(\theta_i, \hat{\theta}_{-i}), (\theta_i, \hat{\theta}_{-i})) \geq U_i(\pi, t_i(\hat{\theta}), \hat{\theta}), \quad \forall \hat{\theta} \in \Theta$$

where $\hat{\theta}$ denotes the reported types of all agents and $\hat{\theta}_{-i}$ the reported types without agent i.

Definition 15.3 *A* strategyproof *mechanism is* individually rational *if the agents have an incentive to participate than leave it i.e., for all agents $i = 1, 2, \dots, n$, for all types $\theta \in \Theta$*

$$U_i(\pi, t_i(\theta), \theta) \geq 0.$$

We assume that the utility an agent derives by leaving the mechanism is 0.

An important aspect of the mechanism $\mathcal{MC}$ worth noting here is that on completion of a production task by agent i, the allocation is re-computed with achieved capacity of agent i being used instead of the reported capacity. The policy that results from this allocation is used to decide the transfer for agent i (see Figure 15.2.1). Table 15.1 summarizes the notation used for the policies obtained by solving (15.1) under different type vector arguments. Note that the allocation to individual agents is obtained via subscripting. For instance, $\pi_j(\bar{\theta}_i, \hat{\theta}_{-i})$ denotes the allocation to agent j when (15.1) is solved with the type argument $(\bar{\theta}_i, \hat{\theta}_{-i})$.

15.2.2 Motivation

Before formalizing the various aspects of $\mathcal{MC}$, we present a motivating example that considers a limited capacity setting and shows that the popular VCG mechanism's transfer

Table 15.1: Summary of the various allocation policies used in $\mathcal{MC}$

Notation	**Description**	**Input type**
$\pi(\hat{\theta})$	Efficient allocation with reported types	$\hat{\theta} = (\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_N)$, where $\hat{\theta}_i = (\hat{u}_i, \hat{c}_i)$
$\pi(\bar{\theta}_i, \hat{\theta}_{-i})$	Efficient allocation with achieved type of agent i and reported types of all other agents	$(\bar{\theta}_i, \hat{\theta}_{-i})$, where $\bar{\theta}_i = (\hat{u}_i, \bar{c}_i)$

scheme as well as the transfer scheme featuring a fixed δ-penalty proposed by Dash et al. [2007] fail to ensure truthful capacity reports. However, note that eliciting truthful unit price reports is not a problem and the VCG mechanism's transfer scheme ensures this part. It is the capacity constraints that make the problem non-trivial. In all the example scenarios through the paper, the demand D is assumed to be 150.

Example 1 *Consider a scenario where there are three agents with their types given by* $(u_1, c_1) = (1, 100)$, $(u_2, c_2) = (2, 50)$ *and* $(u_3, c_3) = (3, 130)$. *Suppose agent* 1 *misreports his capacity to be* 125 *while reporting his unit price truly. The other agents report their true types. Now, the efficient allocation with reported types is* $(125, 25, 0)$. *However, after the agents complete their task, the achieved capacities would be* $(100, 25, 0)$. *A VCG-like payment scheme in this setting would be according to:*

$$t_i = \sum_{j \neq i} \widehat{u}_j \pi_{-i,j}(\hat{\theta}_{-i}) - \sum_{j \neq i} \widehat{u}_j \pi_j(\hat{\theta}). \tag{15.5}$$

As per the above payment rule, agent 1*'s payoff would be* $t_1 = (2 \times 50 + 3 \times 100) - (2 \times 25) = 350$. *On the other hand, if agent* 1 *reported his capacity truthfully, he would have received as transfer* $t_1 = (2 \times 50 + 3 \times 100) - (2 \times 50) = 300$. *Thus, it is evident that agent* 1 *has an incentive to misreport his capacity and receive a higher transfer at the cost of other agents.*

To handle capacity constrants, the mechanism of Dash et al. [2007] incorporated a fixed δ*-penalty based transfer scheme, where the payment* t_i *to an agent* i *is according to:*

$$t_i = \sum_{j \neq i} \widehat{u}_j \pi_{-i,j}(\hat{\theta}_{-i}) - \sum_{j \neq i} \widehat{u}_j \pi_j(\bar{\theta}_i, \hat{\theta}_{-i}) - \delta \beta_i. \tag{15.6}$$

In the above, β_i *is a binary variable which is equal to* 1 *if* $\bar{c}_i < \pi_i(\hat{\theta})$, *i.e., when the agent has over-stated his capacity, and* 0 *otherwise. The first two terms in* (15.6) *refer to the marginal contribution of agent* i. *This is calculated using the achieved capacity of agent* i, *denoted by $\bar{c}_i$.*

Thus, as per (15.6)*, agent* 1*'s payoff would be* $t_1 = (2 \times 50 + 3 \times 100) - (2 \times 50) - \delta = 300 - \delta$*. On the other hand, if agent* 1 *reported his capacity truthfully, he would have received as transfer* $t_1 = (2 \times 50 + 3 \times 100) - (2 \times 50) = 300$*.*

While the mechanism proposed in Dash et al. [2007] achieved strategyproofness, the utility function of agent i in their setting was $U_i(\pi, t_i, \theta) = t_i - u_i \bar{c}_i(\theta)$. However, in our setting with the utility function as defined in (15.3), strategyproofness isn't necessarily ensured using the transfer of Dash et al. [2007]. This is because, an agent i has an incentive to misreport his capacity and improve his allocation at the cost of others (see the last term in (15.3)). To make this precise, the utility U_1 of agent 1 in our setting, i.e., with U_i given by (15.3), turns out to be $(300 - \delta - 1 \times 100 + 125) = 325 - \delta$, whereas with true capacity report it is $(300 - 1 \times 100 + 100) = 300$. Thus, it is clear that truthtelling w.r.t. the capacity element does not guarantee a higher utility for all values of δ.

The choice of utility function in our setting is realistic because in practical competitive scenarios, agents could choose to inflict major losses to other agents. The mechanism of Dash et al. [2007] penalized a misreporting agent by a fixed δ, which in our setting may not be enough to ensure truthful capacity reports. This is because, an agent may be willing to incur a loss of δ, if it is relatively small compared to the major losses inflicted to other agents due to the misreport. This is evident in the above example where agent 1 may choose to misreport his capacity and significantly impact the allocation of agent 2 (reduced by 25 units) while incurring a penalty of a potentially relatively small δ. On the other hand, a large value of δ also is not advisable. Consider again the above example and assume that $\delta = 1000$. It can be the case that agent 1 was probably capable of producing 125 units but fell short due to some unforeseen issues that he faced during production. A large δ may not be appropriate in this scenario. In general, precisely because the degree of losses to others that can be caused by capacity misreports in a given market are unknown, it is difficult to determine a fixed δ that effectively deters capacity misreports. Instead, it is necessary to devise a transfer scheme that penalizes the agent with the extent of damage (in the marginal sense) his misreport caused. $\mathcal{MC}$ achieves this by having a novel variable penalty scheme, that we present below.

Remark 17 *As we observed in the above example, an agent has an incentive to overstate its capacity, while it has no incentive to under-state its capacity. Suppose an agent under-reports its capacity and this inuences the optimal allocation (1). This would imply that the* *optimal allocation (1) would be performed with tighter constraints and hence the marginal contribution of the agent is reduced, implying a reduced utility to the agent. In a similar fashion, it can be argued that an agent has no incentive to over-report its unit price.*

15.2.3 The Mechanism

Given the above problem description, we now define our centralized mechanism. Our mechanism builds on the one proposed by Dash et al. [2007], while incorporating a novel variable penalty scheme. The penalty scheme proposed here enables our mechanism to ensure zero loss to other agents when one or more agents misreport their capacities. Our mechanism is direct where each agent reports his type information and the central controller then calculates an efficient allocation. The transfer, however, happens with a delay and is made only after the agent completes the allotted production task (See Figure 15.2.1).

Transfer Scheme

The transfer t_i that an agent i receives is a sum of two components and is made when agent i's production task is completed. The first component, denoted x_i, is the marginal contribution of agent i to the system, while the second component is a penalty, denoted p_i, imposed on the agents who misreport their capacities to influence the efficient allocation. The transfer components are given by:

$$\begin{aligned} t_i &= x_i + p_i, \\ \text{where} & \\ x_i &= \sum_{j \neq i} \widehat{u}_j \pi_{-i,j}(\hat{\theta}_{-i}) - \sum_{j \neq i} \widehat{u}_j \pi_j(\bar{\theta}_i, \hat{\theta}_{-i}) \\ p_i &= \sum_{j \neq i} \pi_j(\hat{\theta}) - \sum_{j \neq i} \pi_j(\bar{\theta}_i, \hat{\theta}_{-i}). \end{aligned} \tag{15.7}$$

The first component x_i of the payment t_i to the agent i is his marginal contribution towards reducing the total production cost, i.e., the difference between total costs of allocation that other agents obtain without agent i and with agent i, respectively. The penalty component p_i is also marginal in the sense that it is the difference between the total costs of allocation that other agents obtain with agent i's reported capacity and achieved capacity, respectively. Hence, the loss caused by agent i's misreport is compensated via p_i.

Remark 18 *Considering that* $\sum_{j=1}^{N} \pi_j = D$*, the penalty component can be seen as equivalent to*

$$p_i = \pi_i(\bar{\theta}_i, \hat{\theta}_{-i}) - \pi_i(\hat{\theta}).$$

Remark 19 *The reward function* $f(\cdot)$ *mentioned previously, can be incorporated into the penalty component as follows:*

$$p_i = \sum_{j \neq i} f(\pi_j(\hat{\theta})) - \sum_{j \neq i} f(\pi_j(\bar{\theta}_i, \hat{\theta}_{-i})).$$

15.2.4 Discussion

I) Now we demonstrate the usefulness of our penalty scheme using example 1, which involved a single agent misreporting his capacity. In example 1, $\pi_{-1}(\hat{\theta}_{-1}) = (50, 100)$ and $\pi(\bar{\theta}_1, \hat{\theta}_{-1}) = (100, 50, 0)$. $\mathcal{MC}$ would result in a payment of $x_1 = (2 \times 50) - (2 \times 50) = 0$ to agent 1 and the penalty component would be $p_1 = 25 - 50 = -25$. Note that while the marginal contribution component with $\mathcal{MC}$ is 0 as for Dash et al. [2007], the penalty is variable and calculated based on the damage caused by agent 1's misreport, unlike Dash et al. [2007] where it was a fixed quantity δ. As explained before, the utility U_1 derived by agent 1 under true capacity report is 300, while under capacity misreport, $U_1 = 325 - \delta$ in the case of Dash et al. [2007]. On the other hand, U_1 in the corresponding case for $\mathcal{MC}$ turns out to be $(300 - 25) - 1 \times 100 + 125 = 300$, which is equal to the utility derived under true report.

II) We now provide an example to illustrate that truth-telling in unit price is a weakly dominant strategy in $\mathcal{MC}$. Suppose there are three agents with $(u_1, c_1) = (3, 125)$, $(u_2, c_2) = (2, 50)$ and $(u_3, c_3) = (4, 100)$. Assume that the agents report their true capacities. With a unit price report of $(\widehat{u}_1, \widehat{u}_2, \widehat{u}_3) = (1, 2, 4)$, i.e., with agent 1 misreporting his unit price, the optimal allocation is $\pi(\hat{\theta}) = (125, 25, 0)$. In this case, agent 1's payment is $(2 \times 50 + 4 \times 100) - 2 \times 25 = 450$, whereas with the true type reports, it is $(2 \times 50 + 4 \times 100) - 2 \times 50 = 400$. Hence, the utility derived by agent 1 in the former case (with unit price misreport) is $450 - 3 \times 125 + 100 = 175$, whereas with true type report, agent 1's utility is $400 - 3 \times 100 + 100 = 200$. This illustrates that $\mathcal{MC}$ forces truth telling by agent 1 in the unit price component and hence is dominant.

III) We now provide an example where both unit price and capacity elements are misreported. Suppose there are three agents with $(u_1, c_1) = (3, 75)$, $(u_2, c_2) = (2, 50)$ and $(u_3, c_3) = (4, 100)$. Agent 1 misreports its type as $(1, 125)$, whereas the other agents report their true types. The optimal allocation $\pi(\hat{\theta}) = (125, 25, 0)$ and $\pi(\bar{\theta}_1, \theta_{-1}) = (75, 50, 25)$. The transfer components x_1 and p_1 to agent 1 can be calculated as follows:

$$\begin{aligned} x_1 =& (2 \times 50 + 4 \times 100) - (2 \times 50 + 4 \times 25) = 300, \\ p_1 =& 75 - 125 = -50. \end{aligned}$$

Thus, the utility of agent 1 is $(300 - 50) - 3 \times 75 + 75 = 100$. On the other hand, the optimal allocation under true type report by all agents (including agent 1) is $\pi(\theta) = (75, 50, 25)$. Since the penalty to agent 1 is 0, the transfer to agent 1 can be calculated from the marginal contribution component x_1 as $(2 \times 50 + 4 \times 100) - (2 \times 50 + 4 \times 25) = 300$. Thus, the utility derived by agent 1 is $300 - 3 \times 75 + 75 = 150$, which is strictly greater than the utility of 100 derived under the case when agent 1 misreported its type.

15.2.5 Strategyproofness of $\mathcal{MC}$

Strategyproofness of $\mathcal{MC}$ will be established by the following steps:

(i) First, in Proposition 15.5, we show that reporting the true unit price is utility maximizing for agent i, regardless of the reported capacity of agent i and of what other agents report, i.e., $U_i(\hat{\theta}) \leq U_i((u_i, \hat{c}_i), \hat{\theta}_{-i})$.

(ii) Next, in Proposition 15.6, we show that reporting the true capacity is utility maximizing for agent i, given true unit price report and regardless of what other agents report, i.e., $U_i((u_i, \hat{c}_i), \hat{\theta}_{-i}) \leq U_i(\theta_i, \hat{\theta}_{-i})$.

(iii) Theorem 15.7 then establishes that reporting true type is always a utility-maximizing strategy, regardless of what other agents do, i.e., $U_i(\hat{\theta}) \leq U_i(\theta_i, \hat{\theta}_{-i})$.

Lemma 15.4 *If $\alpha \leq \beta$, then*

(i) $\pi_i((\alpha, \hat{c}_i), \hat{\theta}_{-i}) \geq \pi_i((\beta, \hat{c}_i), \hat{\theta}_{-i})$.

(ii) $\pi_i((\hat{u}_i, \alpha), \hat{\theta}_{-i}) \leq \pi_i((\hat{u}_i, \beta), \hat{\theta}_{-i})$.

Proof: The proof is straightforward from the definition of π in (15.1). ■

Proposition 15.5 $U_i(\hat{\theta}) \leq U_i((u_i, \hat{c}_i), \hat{\theta}_{-i})$.

Proof: Using the definitions of utility function (15.3) and the transfer scheme (15.7), the LHS of the above claim is:

$$\begin{aligned} U_i(\hat{\theta}) =& \sum_{j \neq i} \widehat{u}_j \pi_{-i,j}(\hat{\theta}_{-i}) - \sum_{j \neq i} \widehat{u}_j \pi_j((\hat{u}_i, \bar{c}_i), \hat{\theta}_{-i}) \\ &+ \pi_i((\hat{u}_i, \bar{c}_i), \hat{\theta}_{-i}) - \pi_i(\hat{\theta}) - u_i \bar{c}_i(\hat{\theta}) + \pi_i((u_i, \hat{c}_i), \hat{\theta}_{-i}) \end{aligned} \tag{15.8}$$

The RHS can be simplified as follows:

$$\begin{aligned} U_i((u_i, \hat{c}_i), \hat{\theta}_{-i}) =& \sum_{j \neq i} \widehat{u}_j \pi_{-i,j}(\hat{\theta}_{-i}) - \sum_{j \neq i} \widehat{u}_j \pi_j((u_i, \bar{c}_i), \hat{\theta}_{-i}) \\ &+ \pi_i((u_i, \bar{c}_i), \hat{\theta}_{-i}) - \pi_i(u_i, \hat{c}_i), \hat{\theta}_{-i}) - u_i \bar{c}_i((u_i, \hat{c}_i), \hat{\theta}_{-i}) + \pi_i((u_i, \hat{c}_i), \hat{\theta}_{-i}) \end{aligned} \tag{15.9}$$

Comparing (15.8) and (15.9), we need to establish that

$$\begin{aligned} & - \sum_{j \neq i} \widehat{u}_j \pi_j((\hat{u}_i, \bar{c}_i), \hat{\theta}_{-i}) + \pi_i((\hat{u}_i, \bar{c}_i), \hat{\theta}_{-i}) - \pi_i(\hat{\theta}) - u_i \bar{c}_i(\hat{\theta}) \\ \leq & - \sum_{j \neq i} \widehat{u}_j \pi_j((u_i, \bar{c}_i), \hat{\theta}_{-i}) + \pi_i((u_i, \bar{c}_i), \hat{\theta}_{-i}) - \pi_i((u_i, \hat{c}_i), \hat{\theta}_{-i}) - u_i \bar{c}_i((u_i, \hat{c}_i), \hat{\theta}_{-i}) \end{aligned}$$

We note that π_i can be seen as a two-dimensional distribution from the following:

$$\pi_i(u, c) \geq 0,$$

$$\sum_{u=0}^{u_{\max}} \sum_{c=0}^{D} \pi_i(u, c) = L,$$

where L is some positive constant that depends on the specific production setting considered. Note from the definition of π in (15.1) that for any $u \leq 0$, $\pi_i(u, \cdot) = 0$ and similarly, for any $c \leq 0$, $\pi_i(\cdot, c) = 0$. Further, it is safe to assume that the capacities of the individual agents are upper bounded by D, as any agent capacity upwards of D does not impact the allocation π_i. It is also clear that the allocation π_i will be zero if the agent has considerably overstated its unit price and hence, there exists a $u_{\max}$, which acts a threshold for the unit price beyond which the agent's allocation π_i is zero.

Now, if one normalizes π_i using L, then the following

$$\pi_i((\hat{u}_i, \hat{c}_i), \hat{\theta}) - \pi_i((\hat{u}_i, \bar{c}_i), \hat{\theta}_{-i}) - \pi_i((u_i, \hat{c}_i), \hat{\theta}_{-i}) + \pi_i((u_i, \bar{c}_i), \hat{\theta}_{-i}) \geq 0, \tag{15.10}$$

holds because it is the probability of the rectangle with end points $(u_i, \bar{c}_i)$,$(u_i, \hat{c}_i)$,$(\hat{u}_i, \bar{c}_i)$ and $(\hat{u}_i, \hat{c}_i)$, calculated using π_i. Hence, we need to establish that

$$-\sum_{j \neq i} \widehat{u}_j \pi_j((\hat{u}_i, \bar{c}_i), \hat{\theta}_{-i}) - u_i \bar{c}_i(\hat{\theta}) \leq -\sum_{j \neq i} \widehat{u}_j \pi_j((u_i, \bar{c}_i), \hat{\theta}_{-i}) - u_i \bar{c}_i((u_i, \hat{c}_i), \hat{\theta}_{-i})$$

Or equivalently,

$$-\sum_{j \neq i} \widehat{u}_j \pi_j((\hat{u}_i, \bar{c}_i), \hat{\theta}_{-i}) - u_i \bar{c}_i(\hat{\theta}) \leq -\min_{y_j \leq \widehat{c}_j, y_i \leq \bar{c}_i} \left[\sum_{j \neq i} \widehat{u}_j y_j + u_i y_i \right].$$

The above inequality holds because of the ‘min’ in the RHS. Hence proved.

■

Proposition 15.6 $U_i((u_i, \hat{c}_i), \hat{\theta}_{-i}) \leq U_i(\theta_i, \hat{\theta}_{-i})$.

Proof: Using the definitions of utility function (15.3) and the transfer scheme (15.7), the LHS of the above claim can be simplified as follows:

$$\begin{aligned}
U_i((u_i, \hat{c}_i), \hat{\theta}_{-i}) = & \sum_{j \neq i} \widehat{u}_j \pi_{-i,j}(\hat{\theta}_{-i}) - \sum_{j \neq i} \widehat{u}_j \pi_j((u_i, \bar{c}_i), \hat{\theta}_{-i}) \\
& + \pi_i((u_i, \bar{c}_i), \hat{\theta}_{-i}) - \pi_i((u_i, \hat{c}_i), \hat{\theta}_{-i}) - u_i \bar{c}_i((u_i, \hat{c}_i), \hat{\theta}_{-i}) + \pi_i((u_i, \hat{c}_i), \hat{\theta}_{-i}) \\
= & \sum_{j \neq i} \widehat{u}_j \pi_{-i,j}(\hat{\theta}_{-i}) - \sum_{j \neq i} \widehat{u}_j \pi_j((u_i, \bar{c}_i), \hat{\theta}_{-i}) \\
& + \pi_i((u_i, \bar{c}_i), \hat{\theta}_{-i}) - u_i \bar{c}_i((u_i, \hat{c}_i), \hat{\theta}_{-i}).
\end{aligned} \tag{15.11}$$

In the above, note that $\bar{c}_i(\hat{\theta})$ denotes the achieved capacity of agent i. The term $u_i \bar{c}_i((u_i, \hat{c}_i), \hat{\theta}_{-i})$ is the cost to agent i and the utility U_i an agent derives, see (15.3), is the difference between the payment he receives and the cost he incurs, plus an additive term representing his allocation with the reported capacity (and true unit price).

The RHS of the claim in this proposition can be simplified as follows:

$$\begin{aligned}U_i(\theta_i,\hat{\theta}_{-i}) =& \sum_{j\neq i}\widehat{u}_j\pi_{-i,j}(\hat{\theta}_{-i}) - \sum_{j\neq i}\widehat{u}_j\pi_j((u_i,\bar{c}_i),\hat{\theta}_{-i})\\ &+ p_i(\theta_i,\hat{\theta}_{-i}) - u_i\bar{c}_i(\theta_i,\hat{\theta}_{-i}) + \pi_i(\theta_i,\hat{\theta}_{-i})\\ =& \sum_{j\neq i}\widehat{u}_j\pi_{-i,j}(\hat{\theta}_{-i}) - \sum_{j\neq i}\widehat{u}_j\pi_j((u_i,\bar{c}_i),\hat{\theta}_{-i})\\ &- u_i\bar{c}_i(\theta_i,\hat{\theta}_{-i}) + \pi_i(\theta_i,\hat{\theta}_{-i}). \end{aligned} \tag{15.12}$$

The last step holds because under true report, agent i's penalty term $p_i(\theta_i,\hat{\theta}_{-i})$ is 0.

Comparing (15.11) and (15.12), it is enough if we establish that,

$$\begin{aligned}&- u_i\bar{c}_i((u_i,\hat{c}_i),\hat{\theta}_{-i}) + \pi_i((u_i,\bar{c}_i),\hat{\theta}_{-i})\\ \leq &- u_i\bar{c}_i(\theta_i,\hat{\theta}_{-i}) + \pi_i(\theta_i,\hat{\theta}_{-i}).\end{aligned}$$

By applying Lemma 15.4, it can be seen that $\pi_i((u_i,\bar{c}_i),\hat{\theta}_{-i}) \leq \pi_i(\theta_i,\hat{\theta}_{-i})$ because $\bar{c}_i \leq c_i$ (the achieved capacity $\bar{c}_i$ of agent i can be at most his true capacity c_i). Further, the achieved capacity remains the same regardless of whether agent i overstated its capacity or reported it truly, i.e., $\bar{c}_i(\theta_i,\hat{\theta}_{-i}) = \bar{c}_i((u_i,\hat{c}_i),\hat{\theta}_{-i})$. The claim follows. ■

Theorem 15.7 *The mechanism $\mathcal{MC}$ is strategyproof.*

Proof: Follows from Propositions 15.5 and 15.6. ■

Theorem 15.8 *The mechanism $\mathcal{MC}$ is individually rational.*

Proof: Assuming that the utility derived by an agent by not participating is 0 and the fact that $\mathcal{MC}$ is strategyproof, we need to establish that $U_i(\theta) \geq 0$. This claim follows from the following:

- It can be seen in a similar manner as Proposition 3 of the mechanism proposed in Dash et al. [2007] that $t_i - u_i\pi_i(\theta) \geq 0$.
- Further, $\pi_i \geq 0$.

■

15.3 Summary

In this chapter, we presented a novel mechanism $\mathcal{MC}$ that enhances the static mechanism of Dash et al. [2007] to incorporate a variable penalty scheme. The penalty scheme in $\mathcal{MC}$ penalized a capacity-misreporting agent only to the extent of damages caused by his misreport. Unlike Dash et al. [2007], the utility function in our setting included an additive factor that captured the preference of an agent to harm others by overstating its capacity. We established the strategyproofness of $\mathcal{MC}$ in this enhanced setting.

In the next chapter, we extend the variable penalty idea to a dynamic setting, where agent types evolve with time and the agent's utility factors its preference to ruin other agents by overstating the capacity. We establish the incentive compatibility of our proposed dynamic mechanism.

Chapter 16

Dynamic mechanism design with capacity constraints

A natural and crucial extension of Dash et al. [2007] is to consider the private unit costs and capacities that change with time. Such settings naturally arise and are especially important when the center makes allocations to participating agents repeatedly. For example, service workers employed by a service provider would typically attend to thousands of service requests during their employment period. Each service worker has a limited amount of time (which keeps changing every day) and her estimate of the amount of time required to resolve service requests also changes with skill gain or decay. For an efficient allocation, i.e., for minimizing the total time required to resolve a set of service requests, it is not enough for the provider to consider the current types of service workers but also the future horizon. Significant results have been already established in this area, see Bergemann and Välimäki [2010], Cavallo [2008] and Cavallo et al. [2009].

We focus on the work of Bergemann and Välimäki [2010], that proposes a dynamic pivot mechanism that is truthful, efficient, and individually rational. The optimal allocation is computed as the expectation on the discounted sum of valuations of each agent with his payoff computed based on his marginal contribution. However the dynamic pivot mechanism considers a setting where a common resource is allocated to an agent at every time step and the agent capacities are assumed to be infinite. Hence the question of capacity misreports and the consequent need to penalize misreporting agents does not arise in their work.

Unfortunately, capacity constraints are nontrivial to accommodate in the design of truthful and socially efficient dynamic mechanisms. The main challenge is that in a given time step, the payoff to an agent cannot be determined until later when the actual number of units produced by an agent is known. If the allocation π to agent i to produce π_i units is made at time t_1 based on the expected discounted sum of current and future valuations of all agents, and the actual number of units $\bar{c}_i$ produced by agent i is known at a future point in time t_2 ($t_2 > t_1$), then the payoff to agent i for the allocation π cannot be made prior to t_2 because it depends on $\bar{c}_i$. However, because the valuations are discounted in the future, the marginal contribution computed at t_1 may not yield allocative efficiency at t_2. This is because, intuitively, a payment at t_1 is worth more than the same payment at t_2. The question then is what should the payoff to agent i be at time t_2 for allocation π such that the resulting mechanism is truthful in terms of unit costs as well as capacities and is socially efficient? Note that this question does not arise for the dynamic pivot mechanism because the payoffs can be made at t_1, i.e., the time of allocations.

Addressing the above question, we develop our dynamic mechanism $\mathcal{DMC}$ for a setting where agent type includes a unit cost and a capacity of production. The key novelty of $\mathcal{DMC}$ is a delayed transfer scheme that consists of a marginal contribution component and a penalty component. The marginal contribution component is similar to the transfer scheme in the dynamic pivot mechanism (Bergemann and Välimäki [2010]). However, we establish via a counterexample that providing the marginal contribution alone is not sufficient to ensure incentive compatibility of the mechanism. For this purpose, a penalty component is necessary in the transfer to the agents. The penalty scheme in $\mathcal{DMC}$ penalizes the agent only to the extent of the damage caused by his capacity misreport. We establish that $\mathcal{DMC}$ is within period ex-post incentive compatible, allocatively efficient, and individually rational.

16.1 The Dynamic Setting

Unlike $\mathcal{MC}$, here we consider a dynamic setting that involves an infinite time horizon over which the individual agent types evolve. The $\mathcal{DMC}$ that we develop in the following involves 1. a socially efficient allocation which differs from B&V (Bergemann and Välimäki [2010]) as the capacity constraints are included, and 2. a novel delayed transfer scheme that involves a penalty component apart from the payment.While the payment to an agent is his marginal contribution as in B&V, the penalty component that ensures true capacity reports is new. We illustrate the need for the delayed penalty based scheme via an example and also prove that $\mathcal{DMC}$ is incentive compatible in Theorem 16.6.

16.1.1 The Setting

Consider a setting where there are N agents (numbered $1, 2, \ldots, N$), capable of performing certain manufacturing tasks. A central controller receives new tasks from external sources. We use an infinite horizon discrete time-line $1, 2, \ldots, n, \ldots$ to indicate allocation of new tasks and also completion of tasks by individual agents. We allow for completion of a particular task by different agents to be at different instants. Upon arrival of a new task at current instant requiring manufacturing of $D \in \Theta_c$ number of units of commodity, the central controller performs allocation to all the agents based on the current capacity c_i and unit cost u_i of manufacturing for each agent i. Let $\theta_i = (u_i, c_i) \in \Theta_i$ be the cost-capacity type vector for agent i and $\theta = (D, \theta_1, \theta_2, \ldots, \theta_N) \in \Theta \overset{\triangle}{=} \Theta_c \times \Theta_1 \times \Theta_2 \times \cdots \times \Theta_N$ be the joint type vector. As before, we use sub-script $-i$ to denote all agents other than i. For instance, $\theta_{-i} = (D, \theta_1, \theta_2, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_N) \in \Theta_{-i} \overset{\triangle}{=} \Theta_c \times \Theta_1 \times \Theta_2 \times \ldots \times \Theta_{i-1} \times \Theta_{i+1} \times \cdots \times \Theta_N$.

We assume that θ evolves as a Markov process on the discrete-time horizon. Also, we assume that the Markov process thus formed is ergodic for all allocation scenarios from the central controller. Thus, we restrict ourselves to stationary Markovian allocation policies where allocation is given by $y : \Theta \to \mathbb{R}^N$, i.e., at a given instant if the Markov process is at θ, the allocation is given by $y(\theta)$. Let $y_i : \Theta \to \mathbb{R}$ denote the allocation to agent i. Let $\mathcal{Y}$ be the set of all admissible allocation scenarios. When necessary for clarification, we use super-script n to denote a quantity at time instant n. For example, $\theta^n \in \Theta$ represents the types at time instant n, D^n represents the number of units to be manufactured at time instant n, etc. Note that unlike the setting in the static mechanism $\mathcal{MC}$, this setting for dynamic mechanism allows for all parameters including the manufacturing capacity c_i of each agent i, to evolve or change with time instants.

Allocative Efficiency

Given the Markov process of θ, the central controller needs to allocate to all agents in a way that the total cost of manufacturing is minimized with no agent asked to manufacture beyond its capacity. Also, the central controller needs to ensure that the requirement of manufacturing of a particular D units for the task at each instant is met. Thus, the central controller performs optimal allocation according to the following optimization problem:

$$\left.\begin{array}{l} \text{Find } \pi \overset{\triangle}{=} \underset{y\in\mathcal{Y}}{\text{argmin}} \sum_{\theta\in\Theta} \sum_{i=1}^{N} V_i(\theta, y) \\ \text{s.t.} \\ 0 \le y_i(\theta) \le c_i, i = 1, 2, \ldots, N, \forall\theta \\ \sum_{i=1}^{N} y_i(\theta) = D, \forall\theta, \end{array}\right\} \tag{16.1}$$

where

$$V_i(\theta, y) = \mathrm{E}\left[\sum_{k=0}^{\infty} \gamma^k v_i(\theta^k, y_i) | \theta^0 = \theta, y\right]$$

represents the discounted cost infinite horizon value added to the system by agent i, where $v_i(\theta^k, y_i) = u_i^k y_i(\theta^k)$ represents the cost incurred at instant k by that agent and $0 < \gamma < 1$ is the discount factor. The expectation is over various sample paths $(\theta, \theta^1, \theta^2, \ldots)$ starting with θ. The expression for $V_i(\theta, y)$ could be rearranged using dynamic programming as follows:

$$V_i(\theta, y) = v_i(\theta, y_i) + \gamma \mathrm{E}\left[V_i(\theta', y) | \theta\right],$$

where the expectation is now over the next state $\theta' \in \Theta$. We assume that for all possible $\theta \in \Theta$, $\sum_{i=1}^{N} c_i \ge D$, so that there are feasible solutions available for the above optimization problem. In the optimization problem (16.1), we assume that only one optimal allocation exists which is defined to be $\pi \in \mathcal{Y}$. In cases where there are multiple optimal allocations possible, π is assumed to represent any one of them. We also formulate an auxiliary optimization problem which is for computing optimal allocation without an agent i:

$$\left.\begin{array}{l} \text{Find } \pi_{-i} \overset{\triangle}{=} \underset{y\in\mathcal{Y}_{-i}}{\text{argmin}} \sum_{\theta\in\Theta} \sum_{\substack{j=1 \\ j\neq i}}^{N} V_j(\theta, y) \\ \text{s.t.} \\ 0 \le y_j(\theta) \le c_j, j = 1, 2, \ldots, N, j \neq i, \forall\theta \\ \sum_{\substack{j=1 \\ j\neq i}}^{N} y_j(\theta) = D, \forall\theta. \end{array}\right\} \tag{16.2}$$

In the above, $\mathcal{Y}_{-i}$ denotes the set of admissible allocation policies of the form $y : \Theta_{-i} \to \mathbb{R}^{N-1}$. The optimal allocation thus obtained is denoted by $\pi_{-i} \in \mathcal{Y}_{-i}$. Also, $\pi_{-i,j} : \Theta_j \to \mathbb{R}$ represents efficient allocation to agent j without participation by agent i.

Given θ, the problem thus is simply to solve the optimization problem (16.1). However, in practical circumstances there are two potential issues: **(i)** θ_i is the private information of agent i. Thus, the controller needs to derive this information either directly or indirectly. In either case, the problem is to obtain the correct θ-value as otherwise the allocation via π may not make any useful sense; and **(ii)** The capacity, though assumed to be c_i by an agent i, would be subject to realization. Thus, the actual capacity attained by the end of manufacturing may be $\bar{c}_i$ which could be different from c_i. So, the problem here is that the allocation is done at the beginning of a task while the attained capacity ($\bar{c}_i$) is known only at the end of the task.

In order to address these issues, we will formulate in the next subsection, a dynamic mechanism (π, t) where π is an optimal allocation policy discussed previously and $t = (t^1, t^2, \ldots)$ is the payment policy for the entire time horizon. At each time instant, agents are asked to report their types, represented by $\hat{\theta} = \left(D, \hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_N\right)$, based on which a suitable payment t is given by the controller. Let $\bar{\theta}_i = (\hat{u}_i, \bar{c}_i)$ represent the type with achieved capacity of agent i. Then, the payment for each allocation corresponding to instant n is of the functional form, $t_i^n : \Theta_i \to \mathbb{R}$, i.e., $t_i^n(\bar{\theta}_i, \hat{\theta}_i, \hat{\theta}_{-i})$.

Remark 20 *As in the case of the static setting, the payment to agent i is made using the report type vector ($\hat{\theta}^k$) as well as its achieved capacity ($\bar{c}_i$). The need for using the achieved capacity of an agent (known only at a later instant, see Figure 16.1) is illustrated via Example 2.*

We assume that $\sigma_i : \Theta_i \to \Theta_i$, a stationary strategy, is used by agent i to report its type, i.e., $\hat{\theta}_i = \sigma_i(\theta_i), \forall i = 1, \ldots, N$. Henceforth, we use $\hat{\theta}_i$ and $\sigma_i(\theta_i)$ interchangeably. Let $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_N)$. Also, we use $V_i(\theta, \pi, \sigma)$ interchangeably with $V_i(\hat{\theta}, \pi)$.

We define the utility U_i realized by participation by agent i as follows.

Definition 16.1 *The utility U_i realized by agent i is given by*

$$U_i(\theta, \pi, \sigma_i) = E\left[\sum_{k=0}^{\infty} \gamma^k \tilde{U}_i(\theta_i^k, \pi_i, \hat{\theta}^k) | \theta^0 = \theta, \pi, \sigma_i\right], \tag{16.3}$$

where

$$\tilde{U}_i(\theta_i^k, \pi_i, \hat{\theta}^k) = t_i^k(\bar{\theta}_i^k, \hat{\theta}^k) - v_i(\theta_i^k, \pi_i) + \pi_i((u_i^k, \hat{c}_i^k), \hat{\theta}_{-i}^k) \tag{16.4}$$

is the quasi-linear single-stage utility of agent i at instant k. While the first component in $\tilde{U}_i(\cdot)$ above is the transfer of agent i, the second component is the cost and the third component is the allocation received under agent i's (possibly misreported) type. Note that the utility function formulation above is different from the one used in Bergemann and Välimäki [2010]. The difference is that, unlike Bergemann and Välimäki [2010], we have an extra additive term (the last term in (16.4)) that denotes the allocation to agent i at instant k, under true unit price report and a possibly overstated capacity report. As noted in the static setting, this factor is incorporated to capture the preference of an agent to harm others by obtaining a higher allocation. Further, as in the static setting, the reward function $f(\cdot)$ can be incorporated to convert from allocation units to money in the last term of (16.4).

Summing the individual components over the infinite horizon, one obtains the following form for the utility of agent i:

$$U_i(\theta, \pi, \sigma_i) = T_i(\hat{\theta}, \pi, \sigma_i) - V_i(\theta, \pi, \sigma_i) + \mathcal{A}_i(\tilde{\theta}, \pi, \sigma_i),$$

where

$$T_i(\hat{\theta}, \pi, \sigma_i) = E\left[\sum_{k=0}^{\infty} \gamma^{k+\delta_i(k)} t_i(\bar{\theta}_i^k, \hat{\theta}^k) | \theta^0 = \hat{\theta}, \pi, \sigma\right], \tag{16.5}$$

$$\mathcal{A}_i(\tilde{\theta}, \pi, \sigma_i) = E\left[\sum_{k=0}^{\infty} \gamma^{k} \pi_i(\tilde{\theta}^k) | \theta^0 = \tilde{\theta}, \pi, \sigma\right], \tag{16.6}$$

where $\tilde{\theta} = ((u_i, \hat{c}_i), \hat{\theta}_{-i})$ and $\tilde{\theta}^k = ((u_i^k, \hat{c}_i^k), \hat{\theta}_{-i}^k)$. In (16.5), $\delta_i(k)$ is the time taken by agent i to complete the task allocated at instant k. As discussed later, $t_i(\bar{\theta}_i^k, \hat{\theta}^k)$ is a payment made to agent i after completion of the task by him. So, in order to account for the delay, $\gamma^{\delta_i(k)}$ discount factor is used in the above definition of the total transfer to agent i.

We now define within-period ex-post Nash equilibrium, incentive compatibility and individual rationality in a similar manner as Cavallo [2008].

Definition 16.2 (WITHIN-PERIOD EX-POST NASH EQUILIBRIUM) *For a given dynamic mechanism (π, t), we say that a strategy profile σ constitutes a within-period ex-post Nash equilibrium if and only if for all agents $i = 1, 2, \ldots, N$, for all possible types $\theta_i \in \Theta_i$, and for all profiles σ_i',*

$$U_i(\theta, \pi, \sigma_i) \geq U_i(\theta, \pi, \sigma_i').$$

Definition 16.3 (WITHIN-PERIOD EX-POST INCENTIVE COMPATIBILITY) *We say that a dynamic mechanism* (π, t) *is within-period ex-post incentive compatible if and only if for all agents* $i = 1, 2, \ldots, N$*, for all possible types* $\theta_i \in \Theta_i$*, and for all* σ_i*,*

$$T_i(\theta, \pi) - V_i(\theta, \pi) \geq U_i(\theta, \pi, \sigma_i).$$

Definition 16.4 (WITHIN-PERIOD EX-POST INDIVIDUAL RATIONALITY) *We say that a dynamic mechanism* (π, t) *is within-period ex-post individually rational if and only if there exists a within-period ex-post Nash equilibrium strategy profile* σ *such that for all agents* $i = 1, 2, \ldots, N$*, for all possible types* $\theta_i \in \Theta_i$*,*

$$U_i(\theta, \pi, \sigma_i) \geq 0.$$

In the next section, we provide a dynamic mechanism $\mathcal{DMC}$ for this setting and prove that it is within period ex-post incentive compatible and individually rational.

16.1.2 Motivation

Capacity constraints are not considered previously in the context of dynamic mechanism design. For instance, B&V (Bergemann and Välimäki [2010]) propose a marginal contribution based transfer scheme for allocating a common resource. The transfer scheme proposed in B&V's work cannot be applied in our setting where agents have limited capacities. This is because the transfer scheme of B&V would not ensure true capacity reports from the agents. On the other hand, in $\mathcal{DMC}$ the same is ensured via a delayed penalty based transfer scheme.

Example 2 *We illustrate by this example that the transfer scheme from B&V's mechanism would not ensure incentive compatibility, when the setting involves capacity constraints. Consider a scenario where* $D^n = 150, n \geq 0$*. Let there be three agents with their types given by* $(u_1^n, c_1^n) = (1, 100)$*,* $(u_2^n, c_2^n) = (2, 50)$ *and* $(u_3^n, c_3^n) = (3, 100)$*, for all* $n \geq 0$*. Fix a time instant* n *and suppose that the reported types at* n *are given by* $(\hat{u}_1^n, \hat{c}_1^n) = (1, 125)$*,* $(\hat{u}_2^n, \hat{c}_2^n) = (2, 50)$ *and* $(\hat{u}_3^n, \hat{c}_3^n) = (3, 100)$*. Also, assume that the agents report truthfully for all time instants* $m > n$*. Hence, we have a scenario where agent types are static and agent* 1 *misreports his capacity at time instant* n*.*

From the fact that the future joint type of agents remain the same in the above setting *and also that the agents report truthfully in the future, we have that* $\pi_i(\theta^k) \equiv \pi_i$*, a constant allocation for all time instants. Hence, the efficient allocation to agents remains as* $(100, 50, 0)$ *for all time instants* $m > n$ *in the future. Further, the value function of agent* i *turns out as follows:*

$$V_i(\theta^m, \pi) = \sum_{k=m}^{\infty} \gamma^{k-m} u_i^k \pi_i(\theta^k) = u_i \pi_i \sum_{k=m}^{\infty} \gamma^{k-m} = \frac{u_i \pi_i}{1-\gamma}.$$

Assuming $\gamma = \frac{3}{4}$*, we obtain*

$$\begin{aligned} V(\theta^m, \pi) &= \sum_{i=1}^{3} V_i(\theta^m, \pi) \\ &= \frac{1 \times 100 + 2 \times 50 + 3 \times 0}{\frac{1}{4}} = 800. \end{aligned} \tag{16.7}$$

Also, it is easy to see that $V_1(\theta^m, \pi) = 400 = V_2(\theta^m, \pi)$ *and* $V_3(\theta^m, \pi) = 0, \forall m > n$*.*

We denote the payment made to agent i *at instant* n *using the transfer scheme from B&V's mechanism as* $\tilde{x}_i^n(\hat{\theta})$*, i.e.,*

$$\begin{aligned} \tilde{x}_i^n(\hat{\theta}) = V_{-i}(\hat{\theta}, \pi_{-i}) - \Bigg(& v_{-i}(\hat{\theta}_{-i}, \pi(\hat{\theta})) \\ & + \gamma \boldsymbol{E}_{\theta'} \left[V_{-i}(\theta', \pi_{-i}) | \hat{\theta}, \pi(\hat{\theta}) \right] \Bigg). \end{aligned}$$

For the given example, since types are constant for all time instants, this payment simplifies to

$$\tilde{x}_i^n(\hat{\theta}) = V_{-i}(\hat{\theta}, \pi_{-i}) - \left[v_{-i}(\hat{\theta}_{-i}, \pi(\hat{\theta})) + \gamma V_{-i}(\hat{\theta}, \pi_{-i}) \right].$$

We observe that for instant n*, the efficient allocation is* $\pi(\hat{\theta}^n) = (125, 25, 0)$ *and* $\pi_{-1}(\hat{\theta}_{-1}^n) = (50, 100)$*. Hence,* $V_{-1}(\hat{\theta}, \pi_{-1}) = \dfrac{(2 \times 50 + 3 \times 100)}{\frac{1}{4}} = 1600$ *and*

$$\tilde{x}_1^n(\hat{\theta}) = 1600 - (2 \times 25 + \frac{3}{4} \times 1600) = 350,$$

while with true report, the agent would have got,

$$x_1^n(\theta) = 1600 - (2 \times 50 + \frac{3}{4} \times 1600) = 300.$$

Hence, the mechanism of B&V Bergemann and Välimäki [2010] did not penalize agent

1 *for misreporting his capacity. In the* $\mathcal{DMC}$ *scheme that we present subsequently, we design a delayed transfer scheme with a penalty that compensates the loss caused by agent* 1*'s capacity misreport.*

Note that any mechanism which gives a penalty < 0 to an agent upon misreport of capacity by him will not necessarily be ex-post incentive compatible. For instance, we could simply extend the static mechanism of Dash et al. [2007], i.e., give a constant penalty $p < 0$ upon misreport regardless of the value of the misreported capacity. However, as observed in the static case, the utility derived by an agent i in our setting, i.e., according to (16.3), may be higher with a misreported capacity as compared to that with true capacity report. This is clear in the example setting above where the utility derived under true report by agent is $300 - 1 \times 100 + 100 = 300$, whereas with a capacity misreport (agent 1 overstating his capacity to be 125), the utility is $300 - \delta - 1 \times 100 + 125 = 325 - \delta$.

As we observed in the above example, an agent has an incentive to overstate his capacity. In order to design a mechanism that results in truthful reporting by the agents in the system, we devise a delayed transfer scheme that considers the achieved capacity of an agent and gives the marginal contribution of the agent as his payment. An important aspect of the mechanism worth noting here is that on completion of a production task by agent i, the allocation is re-computed with achieved capacity of agent i being used instead of the reported capacity. The policy that results from this allocation is used in deciding the transfer for agent i. We formalize these aspects of $\mathcal{DMC}$ below.

16.1.3 The Mechanism

We employ a direct dynamic mechanism in which the central controller solicits type information $\hat{\theta}$ from each agent i at every time instant n. Based on the reported types, the central controller then makes an efficient allocation. The payment, however, is not immediately disbursed. Instead, the agent continues to produce according to the allocation and on completion of the production task, the central controller observes the achieved capacity level of the agent and makes a payment which is the marginal contribution.

We use the notation as in Table 15.1 for denoting allocations with reported and achieved types and combinations thereof. The overall allocation process in the dynamic mechanism can be illustrated by the Figure 16.1. At instant n, an allocation is performed based on reported types $\hat{\theta}$ of all the agents. The task gets completed by agent i in a time span of $\delta_i(n)$. The achieved capacity of agent i is captured in $\bar{\theta}_i = (\hat{u}_i, \bar{c}_i)$ at completion instant $\bar{n}_i$.

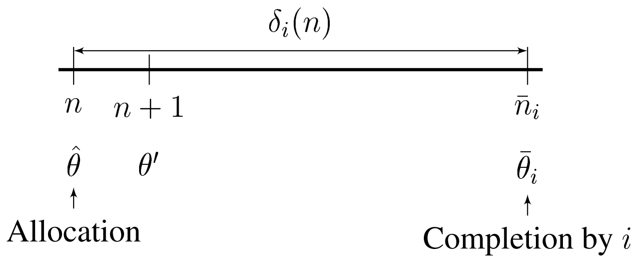


Figure 16.1: A portion of the time-line illustrating the process

The transfer $t_i(\bar{\theta}_i, \hat{\theta})$ to agent i for the task allocated at instant n is performed upon completion of the task by that agent. In order to account for this delay in payment, a compounding factor of $\frac{1}{\gamma^{\delta_i(n)}}$ is used in computing the transfer t_i^n.

$\mathcal{DMC}$ can be described step-by-step as follows:

1. Solicit the types, i.e., the unit price and capacity of each of the agents. The reported type vector is $\hat{\theta}$ as noted before.

2. Perform an allocation $\pi(\hat{\theta})$, by solving the system (16.1) using the reported unit prices and capacities of the agents.

3. The individual agents perform production tasks as per the above allocation.

4. Each agent i completes his part of the allocated task at instant $\bar{n}_i$. Wait till all agents finish their portions of the task.

5. Compute the transfer to each agent according to (16.8).

The above procedure is repeated in the infinite horizon.

Transfer scheme

The expression for t_i^n is as given below:

$$t_i(\bar{\theta}_i, \hat{\theta}) = \frac{1}{\gamma^{\delta_i(n)}} \left[x_i(\bar{\theta}_i, \hat{\theta}) + p_i(\bar{\theta}_i, \hat{\theta}) \right], \tag{16.8}$$

which is composed of two additive components: (i) $x_i(\bar{\theta}_i, \hat{\theta})$, the marginal gain brought into the process by agent i's participation at instant n, and (ii) $p_i(\bar{\theta}_i, \hat{\theta})$, the penalty imposed on agent i to cover the damage caused to the process by misreport of capacity by him. Let

$$W_{-i}(\theta) = v_{-i}(\theta_{-i}, \pi(\theta)) + \gamma\, \mathbf{E}_{\theta'}\left[V_{-i}(\theta', \pi_{-i})|\theta, \pi(\theta)\right],$$

where

$$v_{-i}(\theta_{-i}, \pi_{-i}) = \sum_{\substack{j=1\\ j\neq i}}^{N} v_j(\theta, \pi_{-i,j}), V_{-i}(\theta, \pi_{-i}) = \sum_{\substack{j=1\\ j\neq i}}^{N} V_j(\theta, \pi_{-i}).$$

In the above, the expectation is with respect to $F(\theta'|\theta, \pi(\theta))$, the conditional distribution of the next state θ' of the Markov process, given θ and the allocation policy π. The two components of transfer $t_i(\bar{\theta}_i, \hat{\theta})$ can be now written as

$$x_i(\bar{\theta}_i, \hat{\theta}) = V_{-i}(\hat{\theta}, \pi_{-i}) - W_{-i}(\bar{\theta}_i, \hat{\theta}_{-i}), \text{ and} \tag{16.9}$$

$$p_i(\bar{\theta}_i, \hat{\theta}) = \sum_{j\neq i} \pi_j(\hat{\theta}) - \sum_{j\neq i} \pi_j(\bar{\theta}_i, \hat{\theta}_{-i}). \tag{16.10}$$

The quantity $V_{-i}(\hat{\theta}, \pi_{-i})$ in (16.9) represents the total cost incurred by all agents other than i if agent i had not participated at all while the second term in the equation represents the total cost incurred by them if agent i did participate. Thus, the difference gives the marginal gain brought in by agent i into the setting at instant n. On the other hand, because of the misreporting of capacity, the possible loss caused by agent i is characterized in $p_i(\bar{\theta}_i, \hat{\theta})$ in equation (16.10). If not for capacity related constraints, $p_i(\bar{\theta}_i, \hat{\theta})$ would not be needed and the payment can simply be the marginal contribution $x_i(\bar{\theta}_i, \hat{\theta})$ which may have been given at time instant n itself without waiting for task to be completed. But as shown previously in an example, the marginal contribution alone given as payment would not be enough to ensure that revealing true capacity is incentive compatible. As will be shown later, the additive marginal compensation $p_i(\bar{\theta}_i, \hat{\theta})$ is sufficient to ensure this fact. In equation (16.10), the first term represents the total cost incurred by agents other than i when agent i misreports its capacity and the second term represents the total cost incurred by them when the achieved capacity by agent i is used instead of its reported capacity. Thus one can see that the difference gives the net loss created by the misreport of capacity by agent i. As noted before in the static setting, the penalty $p_i(\bar{\theta}_i, \hat{\theta})$ can be seen as equivalent to $\pi_i(\bar{\theta}_i, \hat{\theta}_{-i}) - \pi_i(\hat{\theta})$. Further, the reward function can be included in the penalty component in exactly the same manner as in the static setting (see Remark 19). Several remarks are in order.

Remark 21 *Comparing with the transfer scheme in Bergemann and Välimäki [2010], i.e., $\tilde{x}_i(\hat{\theta})$ and the payment in our scheme, i.e., $x_i(\bar{\theta}_i, \hat{\theta})$, we note that both are similar in form except that the former is computed without the achieved capacity and is paid at the allocation instant. However, as discussed previously in example 2, agent i will have incentive to misreport capacity in the former case.*

Remark 22 *In the example mentioned above, $\bar{c}_1^n = 100$ and hence, $\bar{\pi}(\bar{\theta}_i^n, \hat{\theta}_{-i}^n) = (100, 50, 0)$. Using the transfer scheme of $\mathcal{DMC}$ (16.8), we obtain:*

$$\begin{aligned} x_i^n(\bar{\theta}_i^n, \hat{\theta}^n) &= 1600 - (100 + \tfrac{3}{4} \times 1600) = 300, \\ p_i^n(\bar{\theta}_i^n, \hat{\theta}^n) &= 25 - 50 = -25 < 0. \end{aligned} \tag{16.11}$$

The utility derived by agent 1 with an overstated capacity of 125, can be calculated as $300 - 25 - 1 \times 100 + 125 = 300$. On the other hand, as noted before, the utility with true capacity report turns out to be 300. Hence, in comparison to the dynamic pivot mechanism's Bergemann and Välimäki [2010] transfer scheme, $\mathcal{DMC}$ penalizes the agent i to the extent of the damage his misreport caused. This is a unique feature of $\mathcal{DMC}$, which we subsequently use to prove the incentive compatibility property (see Theorem 2).

16.1.4 Incentive Compatibility of $\mathcal{DMC}$

We now provide a proof of the incentive compatibility of our mechanism $\mathcal{DMC}$. Let $T_i(\hat{\theta}) = X_i(\hat{\theta}) + P_i(\hat{\theta})$ where X_i and P_i denote the expected discounted sums of the x_i and p_i components of the transfer defined similar to (16.5). Thus,

$$\begin{aligned} X_i(\hat{\theta}, \pi, \sigma) &= E\left[\sum_{k=0}^{\infty} \gamma^k x_i(\hat{\theta}^k, \bar{\theta}_i) \middle| \theta^0 = \hat{\theta}, \pi, \sigma \right], \\ P_i(\hat{\theta}, \pi, \sigma) &= E\left[\sum_{k=0}^{\infty} \gamma^k p_i(\hat{\theta}^k, \bar{\theta}_i) \middle| \theta^0 = \hat{\theta}, \pi, \sigma \right]. \end{aligned}$$

Note that the multiplication factor $\dfrac{1}{\gamma^{\delta_i(n)}}$ is absorbed in the discounted sum.

Lemma 16.5 $X_i(\hat{\theta}_i, \theta_{-i}) = V_{-i}((\hat{\theta}_i, \theta_{-i}), \pi_{-i}) - V_{-i}((\hat{\theta}_i, \theta_{-i}), \pi)$.

Proof: Recall from (16.9) that

$$\begin{aligned} x_i((\hat{\theta}_i,\theta_{-i}),\bar{\theta}_i) =& V_{-i}((\hat{\theta}_i,\theta_{-i}),\pi_{-i}) - W_{-i}(\bar{\theta}_i,\theta_{-i}) \\ =& V_{-i}((\hat{\theta}_i,\theta_{-i}),\pi_{-i}) - \Bigg(v_{-i}(\theta_{-i},\pi(\bar{\theta}_i,\theta_{-i})) \\ & + \gamma\, \mathbf{E}_{\theta'}\left[V_{-i}(\theta',\pi_{-i})|(\bar{\theta}_i,\theta_{-i}),\pi(\bar{\theta}_i,\theta_{-i})\right]\Bigg). \end{aligned}$$

Now, in the RHS, performing discounted summation and taking the expectation over the second term, we obtain

$$\begin{aligned} &\mathbf{E}\left[\sum_{k=0}^{\infty}\gamma^k v_{-i}(\theta_{-i}^k,\pi(\bar{\theta}_i,\theta_{-i}))\right] = V_{-i}((\hat{\theta}_i,\theta_{-i}),\pi). \\ &\text{Combining the other two terms of } x_i\text{, we obtain} \\ &\mathbf{E}\left[\sum_{k=0}^{\infty}\gamma^k\left(V_{-i}((\hat{\theta}_i^k,\theta_{-i}^k),\pi_{-i}) - \gamma V_{-i}((\hat{\theta}_i^{k+1},\theta_{-i}^{k+1}),\pi_{-i})\right)\right] \\ &= V_{-i}((\hat{\theta}_i,\theta_{-i}),\pi_{-i}). \end{aligned} \tag{16.12}$$

The claim follows. ■

Theorem 16.6 *$\mathcal{DMC}$ is ex-post incentive compatible.*

Proof: Suppose there is a simplified mechanism where we know the achieved capacity of agent i at the instant of allocation. Hence, the payment can be made in each period without any delay. Let $\tilde{t}_i$ denote the payment in this simplified mechanism, i.e.,

$$\tilde{t}_i(\bar{\theta}_i,\hat{\theta}) = \left[x_i(\bar{\theta}_i,\hat{\theta}) + p_i(\bar{\theta}_i,\hat{\theta})\right],$$

which does not have any scaling factor with γ. Let $\tilde{T}_i(\hat{\theta})$ denote the total payment in the simplified mechanism. So,

$$\tilde{T}_i(\hat{\theta}) = E\left[\sum_{k=0}^{\infty}\gamma^k\tilde{t}_i(\hat{\theta}^k,\bar{\theta}_i)|\theta^0 = \hat{\theta},\pi,\sigma\right].$$

Note that $\tilde{T}_i(\hat{\theta}) = T_i(\hat{\theta})$ because

$$\begin{aligned}
&E\left[\sum_{k=0}^{\infty} \gamma^{k+\delta_i(k)} t_i(\hat{\theta}^k, \bar{\theta}_i) | \theta^0 = \hat{\theta}, \pi, \sigma\right] \\
=&E\left[\sum_{k=0}^{\infty} \gamma^{k} \tilde{t}_i(\hat{\theta}^k, \bar{\theta}_i) | \theta^0 = \hat{\theta}, \pi, \sigma\right].
\end{aligned}$$

So, we use $T_i(\hat{\theta})$ itself to represent the payment rather that $\tilde{T}_i(\hat{\theta})$ in the simplified mechanism. We now establish that this simplified mechanism is incentive compatible. Later, we prove that the expected discounted sum of transfers in our mechanism is equivalent to this simplified mechanism where transfers are made without any delay.

For the first part of the proof, i.e., to establish the incentive-compatibility of the simplified mechanism, we need to show that

$$\begin{aligned}
&T_i(\theta) - V_i(\theta, \pi) + \mathcal{A}_i(\theta, \pi) \\
\geq &T_i(\hat{\theta}_i, \theta_{-i}) - V_i((\hat{\theta}_i, \theta_{-i}), \pi) + \mathcal{A}_i(\tilde{\theta}, \pi),
\end{aligned} \tag{16.13}$$

i.e., we need to show that the utility that agent i derives under true report is better than the one derived under a misreport. Recall the $\tilde{\theta} = ((u_i, \hat{c}_i), \theta_{-i})$ is the joint type with true unit price and a possibly overstated capacity of agent i and true types of other agents.

The LHS of (16.13) can be simplified as follows:

$$\text{LHS} = V_{-i}(\theta, \pi_{-i}) - V(\theta, \pi) + \mathcal{A}_i(\theta, \pi). \tag{16.14}$$

Note that we have used the fact that under true type report, the penalty component of the transfer vanishes and the above then follows from Lemma 16.5. Now the RHS of (16.13) satisfies

$$\begin{aligned}
\text{RHS} &= X_i(\hat{\theta}_i, \theta_{-i}) + P_i(\hat{\theta}_i, \theta_{-i}) - V_i((\hat{\theta}_i, \theta_{-i}), \pi) + \mathcal{A}_i(\tilde{\theta}, \pi) \\
&= V_{-i}((\hat{\theta}_i, \theta_{-i}), \pi_{-i}) - V_{-i}((\hat{\theta}_i, \theta_{-i}), \pi) + P_i(\hat{\theta}_i, \theta_{-i}) - V_i((\hat{\theta}_i, \theta_{-i}), \pi) + \mathcal{A}_i(\tilde{\theta}, \pi) \quad (16.15) \\
&= V_{-i}((\hat{\theta}_i, \theta_{-i}), \pi_{-i}) - V((\hat{\theta}_i, \theta_{-i}), \pi) + P_i(\hat{\theta}_i, \theta_{-i}) + \mathcal{A}_i(\tilde{\theta}, \pi). \quad (16.16)
\end{aligned}$$

The equality (16.16) follows by combining the second and fourth term in (16.15). Now, we combine the last two terms in (16.16) with $\mathcal{A}_i(\tilde{\theta}, \pi)$ on the LHS and simplify as follows:

$$\begin{aligned}&\mathcal{A}_i(\theta, \pi) - P_i(\hat{\theta}_i, \theta_{-i}) - \mathcal{A}_i(\tilde{\theta}, \pi)\\ =&E\left[\sum_{k=0}^{\infty} \gamma^k \left(\pi_i(\theta^k) - p_i(\hat{\theta}^k, \bar{\theta}_i) - \pi_i((u_i^k, \hat{c}_i^k), \theta_{-i}^k)\right) \middle| \theta^0 = \hat{\theta}, \pi, \sigma\right] \geq 0\end{aligned}$$

The inequality above holds because each of term inside the infinite summation (see below) can be seen to non-negative by using an argument similar to the one used in Proposition 15.5 (see (15.10)).

$$\begin{aligned}&\pi_i(\theta^k) - p_i(\hat{\theta}^k, \bar{\theta}_i) - \pi_i((u_i^k, \hat{c}_i^k), \theta_{-i}^k)\\ =&\pi_i(\theta^k) - \pi_i(\bar{\theta}_i^k, \theta_{-i}^k) + \pi(\hat{\theta}_i^k, \theta_{-i}^k) - \pi_i((u_i^k, \hat{c}_i^k), \theta_{-i}^k) \geq 0\end{aligned}$$

Using the above simplifications, it is enough if we show that

$$\begin{aligned}&V_{-i}(\theta, \pi_{-i}) - V(\theta, \pi)\\ \geq &V_{-i}((\hat{\theta}_i, \theta_{-i}), \pi_{-i}) - V((\hat{\theta}_i, \theta_{-i}), \pi).\end{aligned}$$

Note that $V_{-i}(\theta, \pi_{-i}) = V_{-i}((\hat{\theta}_i, \theta_{-i}), \pi_{-i})$ as we are concerned only with the value function V_{-i} and allocation π_{-i} of agents other than i, which turn out to be the same on both sides. Finally, note that

$$V(\theta, \pi) \leq V((\hat{\theta}_i, \theta_{-i}), \pi),$$

holds from the optimality of π. The claim follows. ■

Theorem 16.7 *$\mathcal{DMC}$ is ex-post individually rational.*

Proof: By optimality of π, $V_{-i}(\theta^n, \pi_{-i}) \geq V(\theta^n, \pi)$. Further, $\mathcal{A}_i(\theta, \pi) \geq 0$. Using these facts and also (16.14) in the proof above, it follows that the utility U_i of agent i is non-negative. ■

16.2 Conclusions and Future Work

We presented two novel mechanisms with progressively realistic assumptions about agent types aimed at economic scenarios where agents have limited capacities. For the simplest case where agent types consist of a unit cost of production and a capacity that does not change with time, we proposed a novel mechanism $\mathcal{MC}$ that extends the work of Dash et al. [2007] with a novel variable penalty based transfer scheme. We established the strategyproofness of this mechanism. Next, we proposed $\mathcal{DMC}$ that accommodates agents having dynamic types in $\mathcal{MC}$. A penalty scheme is needed for achieving truthful reporting of capacities. However, penalties cannot be determined until the actual number of units produced by agents are known. We showed the non-triviality of extending the current dynamic frameworks in determining the payoffs in the case of limited capacities. In $\mathcal{DMC}$, this was achieved by adjusting the payoffs upwards based on a compounded future discount factor. We showed that $\mathcal{DMC}$ possesses the desired properties of ex-post incentive compatibility, individual rationality, and allocative efficiency.

Our proposed mechanisms, especially the dynamic variant is applicable to several real-world settings involving capacity constraints. To name a few, work dispatch in service systems, purchase of electricity from power plants, resource allocation for computational grids and sensor networks are some application scenarios where our proposed dynamic mechanism can be effective, unlike the dynamic pivot mechanism (Bergemann and Välimäki [2010]) which does not guarantee truthful capacity reports in the limited capacity setting.

Although we established the usefulness of these mechanisms theoretically, it is important to validate them via simulation experiments followed by actual pilots in real economic setups. We plan to leverage service worker and service provider environments to perform the needed validations. Further, there are real scenarios exhibiting additional aspects of agent types that our mechanisms may not readily accommodate. We plan to further study such scenarios and either extend or establish afresh the relevant theoretical results.

Extension to type as a function of allocation A practical extension of our setting is where unit cost and capacity depend on allocation. For example, in a certain manufacturing setup, upto 100 units of manufacturing might be with a unit price of 10 while beyond that it might be priced lower, say 5, per unit. On similar lines, one could consider a case where the capacity changes with ranges of allocation. Such scenarios could be modelled by considering the unit price u and capacity c to be functions of the allocation π and in the mechanism, agents be asked to report these functions rather than a pair of numbers. For the static setting, the optimization problem could be written as follows:

$$\begin{aligned}
&\text{Find } \pi = \operatorname*{argmin}_{\pi\in\Pi} \sum_{i=1}^{n} u_i(y_i)y_i \\
&\text{s.t. } y_i \le c_i(y_i), i = 1, 2, \ldots, n, \\
&\text{and } \sum_{i=1}^{n} y_i = D.
\end{aligned} \tag{16.17}$$

A simple example illustrating the functions $u(y_i)$ and $c(y_i)$ is as follows: Consider a scenario of two agents where

$$\begin{aligned}
u(y_i) &= \begin{cases} 10 \text{ if } y_i \le 10, \\ 5 \text{ if } y_i > 10. \end{cases} \\
c(y_i) &= \begin{cases} 10 \text{ if } y_i \le 10, \\ 100 \text{ if } y_i > 10. \end{cases}
\end{aligned} \tag{16.18}$$

It is clear that the allocation (16.17) is different in comparison to the allocation of (15.1). However, the transfer scheme of the mechanism is still the same in this modified setting. Similarly, the dynamic mechanism $\mathcal{DMC}$ could also be extended to handle such types. We plan to further study such scenarios and either extend or establish afresh the relevant theoretical results.

Chapter 17

Conclusions and Future Directions

In this thesis, we studied problems of resource allocation under various uncertainties. The problems investigated encompass complex application domains such as vehicular traffic control, service systems, wireless sensor networks and mechanism design. In the first three studies, the uncertainty arose as a result of the inherent stochastic dynamic environment, whereas in the mechanism design application, it was due to the private information of the individual agents. Further, the algorithms had to grapple with the high-dimensional state and action spaces and computational efficiency was a critical requirement in this regard. The aim of the resource allocation algorithms was to search for the 'best' solution in different sequential decision making problems:

Vehicular traffic control

The aim here was to optimize the 'green time' resource of the individual lanes in a road network that maximized a certain long performance objective. We considered both discounted as well as average cost objectives. We formulated this problem using the MDP framework and proposed several reinforcement learning based algorithms for its solution. All the algorithms incorporated function approximation and hence, novel feature selection schemes to handle the curse of dimensionality associated with large road networks. Further, they worked with coarse estimates of the congestion levels on the individual lanes of the road networks - an advantage considering the fact that it is hard to obtain precise queue length estimates in practice. The congestion levels on the lanes were estimated by means of thresholds.

The next sub-problem considered was to develop an algorithm to tune the threshold values used in any thresholds based TLC algorithm to the optimum. This is motivated by the fact that the threshold values were fixed in all the traffic light control (TLC) algorithms and this may not be optimal for all choices of road network settings. We proposed a threshold tuning algorithm based on SPSA with deterministic perturbations. This algorithm is online, provably convergent and works with any threshold-based TLC algorithm to find the optimal values for the thresholds. The empirical usefulness of this algorithm was established in various road network settings and in conjunction with three different TLC algorithms, that were all thresholds based.

The final sub-problem we considered in this application context is of optimal feature selection. Note that all the TLC algorithms proposed here incorporated feature based representations and the choice of features in each algorithm was dependent on our understanding of 'what captures the system state effectively via features'. A problem of interest is to find a way to adapt features so as to tune to the optimal choice. We partially answer this question by developing a novel feature adaptation method and establishing its empirical usefulness by studying it in conjunction with the Q-learning based TLC algorithm.

Service Systems

The aim here was to optimize the 'workforce', the critical resource of any service system. However, adapting the staffing levels to the workloads in such systems is nontrivial as the queue stability and aggregate SLA constraints have to be complied with. We formulated this problem as a parameterized Markov process with (a) a discrete worker parameter that specifies the number of workers across shifts and skill levels, (b) hidden or unobserved state component, (c) a novel single stage cost function that balances the conflicting objectives of utilizing workers better and attaining the target SLAs, and (d) the aggregate SLA and queue stability constraints. The aim was to find the optimum worker parameter from a discrete high-dimensional parameter set, that minimizes the long run average of the single-stage cost function, while adhering to the constraints relating to queue stability and SLA compliance. Another difficulty in finding the optimum (constrained) worker parameter is that the single stage cost and constraint functions can be estimated only via simulation.

We propose novel simultaneous perturbation based simulation optimization algorithms for solving the above problem. The algorithms include both first order as well as second order methods and incorporate SPSA based gradient estimates in the primal, with dual ascent for the Lagrange multipliers. All the algorithms that we propose are online, incremental and easy to implement. Further, they involve a certain generalized smooth projection operator, which is essential to project the continuous-valued worker parameter tuned by SASOC algorithms onto the discrete set. The smoothness is necessary to ensure that the underlying transition dynamics of the constrained Markov cost process is itself smooth (as a function of the continuous-valued parameter) - a critical requirement to prove the convergence of all SASOC algorithms. We validated our algorithms on five real-life service systems and compared them with a state-of-the-art optimization tool-kit OptQuest. Being 25 times faster than OptQuest, our algorithms are particularly suitable for adaptive labor staffing. Also, we observe that our algorithms guarantee convergence and find better solutions than OptQuest in many cases.

Sensor Networks

We studied the problem of tracking a single object moving randomly through a dense network of wireless sensors such that the entire sensing region has no discontinuities. However, we wish to conserve energy by optimizing the sleep times of the sensors in a manner that does not compromise on the tracking quality. We remove the waking channel assumption, i.e., a sleeping sensor can be communicated with. In this context, the resource considered is the ‘sleep time’ of the individual sensors and the objective is to manage the sleep times of the sensors efficiently, while keeping the tracking error to a minimum.

We model the sleep–wake scheduling problem as a partially-observable Markov decision process (POMDP). We propose novel RL-based algorithms - with both long-run discounted and average cost objectives - for solving this problem. All our algorithms incorporate function approximation and feature-based representations to handle the curse of dimensionality. Further, the feature selection scheme used in each of the proposed algorithms intelligently manages the energy cost and tracking cost factors, which in turn, assists the search for the optimal sleeping policy. The results from the simulation experiments suggest that our proposed algorithms perform better than a recently proposed algorithm from Fuemmeler and Veeravalli [2008].

Mechanism Design

From a single agent in a stochastic dynamic environment, we progressed to multiple self-interested agents sharing a resource in this study. In addition to the inherent stochasticity of the system, the setting also involved uncertainty in the form of private information of the agents. The aim of the resource allocator here then is to be allocatively efficient and also maximize the social welfare via the ‘right’ transfer payments. In other words, the problem was to find an incentive compatible transfer scheme following a socially efficient allocation.

We presented two novel mechanisms with progressively realistic assumptions about agent types aimed at economic scenarios where agents have limited capacities. For the simplest case where agent types consist of a unit cost of production and a capacity that does not change with time, we proposed a novel mechanism $\mathcal{MC}$ that extends the work of Dash et al. [2007] with a novel variable penalty based transfer scheme. We established the strategyproofness of this mechanism. Next, we proposed $\mathcal{DMC}$ that accommodates agents having dynamic types in $\mathcal{MC}$. A penalty scheme is needed for achieving truthful reporting of capacities. However, penalties cannot be determined until the actual number of units produced by agents are known. We showed the non-triviality of extending the current dynamic frameworks in determining the payoffs in the case of limited capacities. In $\mathcal{DMC}$, this was achieved by adjusting the payoffs upwards based on a compounded future discount factor. We showed that $\mathcal{DMC}$ possesses the desired properties of ex-post incentive compatibility, individual rationality, and allocative efficiency.

17.0.1 Future Work

We briefly summarize four possible research directions geared towards optimal resource allocation under uncertainty using techniques from reinforcement learning and its allied areas:

- The problem of feature adaptation has been receiving a lot of research attention in the RL community. An algorithm that combines both exploration and exploitation in trying to find the optimal features was proposed in Bhatnagar et al. [2012a]. It would be interesting to continue work in this direction, for instance, by combining feature adaptation algorithms with well-known prediction algorithms such as LSTD and LSPE and also, developing actor-critic or LSPI like algorithms in conjunction with a feature adaptation scheme for the problem of control. Also challenging in this context is establishing the convergence of feature adaptation algorithms to the optimal choice of features.

- In the context of vehicular traffic control, it would be interesting to consider modeling of driver behaviour, ambulance movement, VIP passage, accidents, etc and study the impact of such exogenous events on allocating the "green time". Further, it is worth studying adaptive traffic light control in a setting where a car-following model is incorporated. A car-following model captures a driver's response to the current traffic conditions.

- In the context of service systems, one could incorporate enhancements into the single-stage cost function where even monetary costs are considered apart from staff utilization and SLA attainment factors. An orthogonal direction of future work in this context is to develop skill updation algorithms, i.e., derive novel work dispatch policies that improve the skills of the workers beyond their current levels by way of assigning work of a higher complexity. However, the setting is still constrained and the SLAs would need to be met while improving the skill levels of the workers. The skill updation scheme could then be combined with the SASOC algorithms presented in Chapter 10 to optimize the staffing levels on a slower timescale.

- The problem of decision making under uncertainty is further complicated in a dynamic setting involving multiple agents who are selfish and hold private information. This falls into the purview of dynamic mechanism design, where the agent types evolve as an MDP. A novel mechanism for a capacity-constrained setting was proposed in Prashanth et al. [2012a]. However, if we assume no model information in this context, it is non-trivial to design RL-like algorithms that converge to a desirable equilibrium point. Another challenge here is that of computation, i.e., of finding a socially optimal policy in a manner that is computationally efficient. It is indeed interesting to note that this problem has reductions to a multi-armed bandit setting.

- Exploration-exploitation dilemma has been studied in various contexts and multi-armed bandits probably present the simplest, nevertheless a challenging setting for this problem. This is a decision making problem involving several factors of uncertainty and it would be interesting to develop novel solution approaches and apply them in an MDP/RL setting to find the optimal strategy for choosing actions.

Bibliography

B. Abdulhai, R. Pringle, and G.J. Karakoulas. Reinforcement learning for true adaptive traffic signal control. *Journal of Transportation Engineering*, 129:278, 2003.

J. Abounadi, D. Bertsekas, and V.S. Borkar. Learning algorithms for Markov decision processes with average cost. *SIAM Journal on Control and Optimization*, 40(3):681–698, 2002. ISSN 0363-0129.

G. Abu-Lebdeh and R.F. Benekohal. Design and evaluation of dynamic traffic management strategies for congested conditions. *Transportation Research Part A: Policy and Practice*, 37(2):109–127, 2003.

S. Alter. Service system fundamentals: Work system, value chain, and life cycle. *IBM Systems Journal*, 47(1):71–85, 2008.

G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella. Energy conservation in wireless sensor networks: A survey. *Ad Hoc Networks*, 7(3):537–568, 2009.

I. Arel, C. Liu, T. Urbanik, and AG Kohls. Reinforcement learning-based multi-agent system for network traffic signal control. *Intelligent Transport Systems, IET*, 4(2):128–135, 2010.

Susan Athey and Ilya Segal. An efficient dynamic mechanism. Technical report, UCLA Department of Economics, March 2007.

G. Atia, J. Fuemmeler, and V. Veeravalli. Sensor scheduling for energy-efficient target tracking in sensor networks. In *Signals, Systems and Computers (ASILOMAR), 2010 Conference Record of the Forty Fourth Asilomar Conference on*, pages 1903–1907. IEEE, 2010.

L. C. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 30–37, 1995.

B. Bakker, M. Steingrover, R. Schouten, E. Nijhuis, and L. Kester. Cooperative multi-agent reinforcement learning of traffic lights. In *Proceedings of the Workshop on Cooperative Multi-Agent Learning, European Conference on Machine Learning, ECML*, volume 5, page 65, 2005.

Dipyaman Banerjee, Nirmit Desai, and Gargi Dasgupta. Simulation-based evaluation of dispatching policies in service systems. In *Winter simulation conference*, 2011.

J.S̃. Baras and V.S̃. Borkar. 2000. A learning algorithm for Markov decision processes with adaptive state aggregation", *Proceedings of the 39th IEEE Conference on Decision and Control, Dec. 12 – 15,* , vol.4, Sydney, Australia: 3351 - 3356.

V. S. Barman, K. and Borkar. pages 784–786, 2008. A note on linear function approximation using random projections", *Systems and Control Letters*, 57(9):.

M. Beccuti, D. Codetta-Raiteri, and G. Franceschinis. Multiple abstraction levels in performance analysis of wsn monitoring systems. In *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*, page 73, 2009.

D. Bergemann and J. Välimäki. The dynamic pivot mechanism. *Econometrica*, 78(2): 771–789, 2010.

D.P̃. Bertsekas. *Dynamic Programming and Optimal Control, Vol*, note = I (3rd ed.), athena scientific, belmont, ma. 2005.

D.P̃. Bertsekas. *Dynamic Programming and Optimal Control, Vol*, note = II (3rd ed.), athena scientific, belmont, ma. 2007.

D.P̃. Bertsekas. Approximate dynamic programming. 2011. (Online) Chapter 6 of *Dynamic Programming and Optimal Control Vol.II, (3rd ed.).*
URL: http://web.mit.edu/dimitrib/www/dpchapter.html.

D.P̃. Bertsekas and J.Ñ. Tsitsiklis. *Neuro-Dynamic Programming*, athena scientific, belmont, ma. 1996a.

D.P̃. Bertsekas and H. Yu. Projected equation methods for approximate solution of large linear systems. *Journal of Computational and Applied Mathematics*, pages 27–50, 2009. 227:.

Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control, vol. I and II*. Athena Scientific, 1995. ISBN 1886529132.

Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming (Optimization and Neural Computation Series, 3)*. Athena Scientific, May 1996b. ISBN 1886529108.

V.S̃. Bertsekas, D. P.; Borkar and A. Nedic. Improved temporal difference methods with linear function approximation. *Handbook of Learning and Approximate Dynamic Programming by A.Barto, W.Powell, J.Si (Eds.)*, pages 231–255, 2004. pp.IEEE Press.

S. Bhatnagar. Adaptive multivariate three-timescale stochastic approximation algorithms for simulation based optimization. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 15(1):74–107, 2005.

S. Bhatnagar. Adaptive Newton-based multivariate smoothed functional algorithms for simulation optimization. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 18(1):1–35, 2007.

S. Bhatnagar. An actor-critic algorithm with function approximation for discounted cost constrained Markov decision processes. *Systems & Control Letters*, 2010.

S. Bhatnagar and K. Lakshmanan. An Online Convergent Q-learning Algorithm with Linear Function Approximation. Technical report, Stochastic Systems Lab, IISc, 2012. URL http://stochastic.csa.iisc.ernet.in/www/research/files/IISc-CSA-SSL-TR-2012-3.pdf.

S. Bhatnagar, M.C. Fu, S.I. Marcus, and I. Wang. Two-timescale simultaneous perturbation stochastic approximation using deterministic perturbation sequences. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 13(2):180–209, 2003. ISSN 1049-3301.

S. Bhatnagar, R.S̃. Sutton, M. Ghavamzadeh, and M. Lee. pages 2471–2482, 2009a. Natural actor-critic algorithms", *Automatica*, 45:.

S. Bhatnagar, R.S. Sutton, M. Ghavamzadeh, and M. Lee. Natural actor-critic algorithms. *Automatica*, 45(11):2471–2482, 2009b.

S. Bhatnagar, N. Hemachandra, and V.K. Mishra. Stochastic approximation algorithms for constrained optimization via simulation. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 21(3):15, 2011a.

S. Bhatnagar, V. Mishra, and N. Hemachandra. Stochastic algorithms for discrete parameter simulation optimization. *IEEE Transactions on Automation Science and Engineering*, 8 (4):780 –793, 2011b.

S. Bhatnagar, V.S. Borkar, and L.A. Prashanth. Adaptive feature pursuit: Online adaptation of features in reinforcement learning. *Reinforcement Learning and Approximate Dynamic Programming for Feedback Control (Ed. F. Lewis and D. Liu), IEEE Press Computational Intelligence Series (to appear)*, 2012a.

S Bhatnagar, H.L. Prasad, and L.A. Prashanth. *Stochastic Recursive Algorithms for Optimization: Simultaneous Perturbation Methods*. Lecture Notes in Control and Information Sciences Series, Springer (Accepted), 2012b.

S. Bhulai, G. Koole, and A. Pot. Simple methods for shift scheduling in multi-skill call centers. *Manufacturing and Service Operations Management*, 10(3), 2008.

F. Boillot, JM Blosseville, JB Lesort, V. Motyka, M. Papageorgiou, and S. Sellam. Optimal signal control of urban traffic networks. In *Proc. 6th IEE Int. Conf. Road Traffic Monitoring and Control*, pages 75–79, 1992.

V. S. Borkar. *Probability Theory: An Advanced Course*. Springer, New York, 1995.

VS Borkar. An actor-critic algorithm for constrained markov decision processes. *Systems & control letters*, 54(3):207–213, 2005.

V.S. Borkar. *Stochastic approximation: a dynamical systems viewpoint*. Cambridge University Press, 2008.

V.S. Borkar and S.P. Meyn. The ODE method for convergence of stochastic approximation and reinforcement learning. *SIAM Journal on Control and Optimization*, 38(2):447–469, 2000.

J. A. Boyan. Least-squares temporal difference learning. *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 49–56, 1999. pagesMorgan Kaufmann, San Francisco, CA.

S. J. Bradtke and A. G. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, pages 33–57, 1996. 22:.

C. Brickner, D. Indrawan, D. Williams, and S.R. Chakravarthy. Simulation of a stochastic model for a service system. In *Winter Simulation Conference (WSC), Proceedings of the 2010*, pages 1636 –1647, dec. 2010. doi: 10.1109/WSC.2010.5678905.

R. Cavallo. Efficiency and redistribution in dynamic mechanism design. In *Proceedings of the 9th ACM Conference on Electronic Commerce*, pages 220–229, 2008.

R. Cavallo, D.C. Parkes, and S. Singh. Efficient mechanisms with dynamic populations and dynamic types. *Unpublished manuscript, Harvard University*, 2009.

Ruggiero Cavallo. Mechanism design for dynamic settings. *SIGecom Exch.*, 8:7:1–7:5, December 2009.

M.T. Cezik and P. L'Ecuyer. Staffing multiskill call centers via linear programming and simulation. *Management Science*, 54(2):310–323, 2008.

W.K.V. Chan. An analysis of emerging behaviors in large-scale queueing-based service systems using agent-based simulation. In *Winter Simulation Conference*, pages 872–878, 2008.

HF Chen, T.E. Duncan, and B. Pasik-Duncan. A kiefer-wolfowitz algorithm with randomized differences. *IEEE Transactions on Automatic Control*, 44(3):442–453, 1999.

D.C. Chin, J.C. Spall, and R.H. Smith. Evaluation of system-wide traffic signal control using stochastic optimization and neural networks. In *Proceedings of the American Control Conference*, volume 3, pages 2188–2194, 1999.

S.B. Cools, C. Gershenson, and B. DHooghe. Self-organizing traffic lights: A realistic simulation. *Advances in Applied Self-organizing Systems*, pages 41–50, 2008.

R.K. Dash, P. Vytelingum, A. Rogers, E. David, and N.R. Jennings. Market-based task allocation mechanisms for limited-capacity suppliers. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 37(3):391–405, 2007.

D. de Oliveira, A.L.C. Bazzan, B.C. da Silva, E.W. Basso, L. Nunes, R. Rossetti, E. de Oliveira, R. da Silva, and L. Lamb. Reinforcement learning based control of traffic lights in non-stationary environments: a case study in a microscopic simulator. In

Proceedings of the 4th European Workshop on Multi-Agent Systems (EUMAS06), pages 31–42, 2006.

L.B. De Oliveira and E. Camponogara. Multi-agent model predictive control of signaling split in urban traffic networks. *Transportation Research Part C: Emerging Technologies*, 18(1):120–139, 2010.

Rahul Deb. Optimal contracting of new experience goods. MPRA Paper 9880, University Library of Munich, Germany, August 2008.

M. Di and E.M. Joo. A survey of machine learning in wireless sensor netoworks from networking and application perspectives. In *Information, Communications & Signal Processing, 2007 6th International Conference on*, pages 1–5. IEEE, 2007.

D. Di Castro and S. Mannor. Adaptive bases for reinforcement learning. pages 20–24, 2010. *Machine Learning and Knowledge Discovery in Databases, Proceedings of ECML PKDD Barcelona, Spain, September2010, Part I*, José Luis Balcźar, Francesco Bonchi, Aristides Gionis and Michèle Sebag (eds.), Lecture Notes in Computer Science Volume 6321: 312-327.

O. Dousse, C. Tavoularis, and P. Thiran. Delay of intrusion detection in wireless sensor networks. In *Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing*, pages 155–165. ACM, 2006.

J.A. Fuemmeler and V.V. Veeravalli. Smart sleeping policies for energy efficient tracking in sensor networks. *IEEE Transactions on Signal Processing*, 56(5):2091–2101, 2008.

J.A. Fuemmeler, G.K. Atia, and V.V. Veeravalli. Sleep control for tracking in sensor networks. *Signal Processing, IEEE Transactions on*, 59(9):4354–4366, 2011.

N.H. Gartner. Opac: A demand-responsive strategy for traffic signal control. *Transportation Research Record*, (906), 1983.

Enrico Gerding, Sebastian Stein, Kate Larson, Alex Rogers, and Nicholas R. Jennings. Scalable mechanism design for the procurement of services with uncertain durations. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, pages 649–656, 2010.

M. Girianna and R.F. Benekohal. Using genetic algorithms to design signal coordination for oversaturated networks. *Journal of Intelligent Transportation Systems*, 8(2):117–129, 2004.

B.P. Gokulan and D. Srinivasan. Distributed geometric fuzzy multiagent urban traffic signal control. *IEEE Transactions on Intelligent Transportation Systems*, 11(3):714–727, 2010.

C. Gui and P. Mohapatra. Power conservation and quality of surveillance in target tracking sensor networks. In *Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 129–143. ACM, 2004.

SG Henderson and BL Nelson. Handbook of simulation, 2006.

J.J. Henry, J.L. Farges, and J. Tuffal. The PRODYN real time traffic algorithm. In *Int. Fed. of Aut. Control (IFAC) Conf*, 1984.

M. W. Hirsch. Convergent activation dynamics in continuous time networks. *Neural Networks*, 2:331–349, 1989.

J. Hong, J. Cao, Y. Zeng, S. Lu, D. Chen, and Z. Li. A location-free prediction-based sleep scheduling protocol for object tracking in sensor networks. In *Network Protocols, 2009. ICNP 2009. 17th IEEE International Conference on*, pages 63–72. IEEE, 2009.

D. Huang, W. Chen, P. Mehta, S. Meyn, and A. Surana. Feature selection for neuro-dynamic programming. *Reinforcement Learning and Approximate Dynamic Programming for Feedback Control (Ed. F.L.Lewis and D.Liu), IEEE Press Computational Intelligence Series, Chapter 24, this volume.*, 2011.

B. Jiang and B. Ravindran. Completely distributed particle filters for target tracking in sensor networks. In *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 334–344. IEEE, 2011.

B. Jiang, K. Han, B. Ravindran, and H. Cho. Energy efficient sleep scheduling based on moving directions in target tracking sensor network. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–10. IEEE, 2008.

G. Jin, X. Lu, and M.S. Park. Dynamic clustering for object tracking in wireless sensor networks. *Ubiquitous Computing Systems*, pages 200–209, 2006.

L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning: A survey. *Arxiv preprint cs/9605103*, 1996.

Sham M. Kakade, Ilan Lobel, and Hamid Nazerzadeh. Optimal dynamic mechanism design via a virtual vcg mechanism. *SIGecom Exch.*, 10:27–30, March 2011.

S. Keller, P. W.; Mannor and D. Precup. Automatic basis function construction for approximate dynamic programming and reinforcement learning. *Proceedings of the 23rd International Conference on Machine Learning, June 25 – 29, 2006*, 2006. Pittsburgh, PA.

V. R. Konda and J. N. Tsitsiklis. On actor–critic algorithms. SIAM Journal on Control and Optimization:1143–1166, 2003. 42(4):.

E.B. Kosmatopoulos and A. Kouvelas. Large scale nonlinear control system fine-tuning through learning. *IEEE Transactions on Neural Networks*, 20(6):1009–1023, 2009.

E.B. Kosmatopoulos, M. Papageorgiou, A. Vakouli, and A. Kouvelas. Adaptive fine-tuning of nonlinear control systems with application to the urban traffic control strategy TUC. *IEEE Transactions on Control Systems Technology*, 15(6):991–1002, 2007.

A. Kouvelas, K. Aboudolas, E.B. Kosmatopoulos, and M. Papageorgiou. Adaptive performance optimization for large-scale traffic control systems. *Intelligent Transportation Systems, IEEE Transactions on*, (99):1–12, 2011.

Harold J. Kushner and Dean S. Clark. *Stochastic approximation methods for constrained and unconstrained systems*. Springer-Verlag, 1978a. ISBN 0-387-90341-0.

Harold Joseph Kushner and George Yin. *Stochastic approximation and recursive algorithms and applications*. Springer, 1997. ISBN 0-387-00894-2.

H.J. Kushner and D.S. Clark. *Stochastic approximation methods for constrained and unconstrained systems*, volume 6. Springer-Verlag New York, 1978b.

K. Kwong, R. Kavler, R. Rajagopal, and P. Varaiya. Real-time measurement of link vehicle count and travel time in a road network. *IEEE Transactions on Intelligent Transportation Systems*, (accepted), 2010. doi: 10.1109/TITS.2010.2050881. URL http://dx.doi.org/10.1109/TITS.2010.2050881.

M. Laguna. Optimization of complex systems with optquest. *OptQuest for Crystal Ball User Manual, Decisioneering*, 1998.

W. Lai and I.C. Paschalidis. Optimally balancing energy consumption versus latency in sensor network routing. *ACM Transactions on Sensor Networks (TOSN)*, 4(4):21, 2008.

J. P. Lasalle and S. Le fschetz. *Stability by Liapunov's Direct Method with Applications*. Academic Press, New York., 1961.

J. Li, Q.S. Jia, X. Guan, and X. Chen. Tracking a moving object via a sensor network with a partial information broadcasting scheme. *Information Sciences*, 2011.

T. Li, D. Zhao, and J. Yi. Adaptive dynamic programming for multi-intersections traffic signal intelligent control. In *11th International IEEE Conference on Intelligent Transportation Systems, 2008. ITSC 2008*, pages 286–291, 2008.

W.H. Lin and C. Wang. An enhanced 0-1 mixed-integer LP formulation for traffic signal control. *IEEE Transactions on Intelligent Transportation Systems*, 5(4):238–245, 2004.

Z. Liu and I. Elhanany. Rl-mac: a qos-aware reinforcement learning based mac protocol for wireless sensor networks. In *Networking, Sensing and Control, 2006. ICNSC'06. Proceedings of the 2006 IEEE International Conference on*, pages 768–773. IEEE, 2006.

S. Mahadevan and B. Liu. Basis construction from power series expansions of value functions. *Proceedings of Advances in Neural Information Processing Systems*, 2010. Vancouver, B.C., Canada.

P. Marbach and J.N. Tsitsiklis. Simulation-based optimization of markov reward processes. *IEEE Transactions on Automatic Control*, 46(2):191–209, 2001.

A. Mas-Colell, M.D. Whinston, and J.R. Green. *Microeconomic theory*. Oxford University Press, 1995. ISBN 9780195073409. URL `http://books.google.com/books?id=KGtegVXqD8wC`.

F. Melo and M. Ribeiro. Q-learning with linear function approximation. *Learning Theory*, pages 308–322, 2007.

S. Menache, I.; Mannor and N. Shimkin. Basis function adaptation in temporal difference reinforcement learning. *Annals of Operations Research*, pages 215–238, 2005. 134:.

K Mierendorff. Optimal dynamic mechanism design with deadlines. *Unpublished manuscript, University of Bonn*, 2009.

M. Mihaylov, Y.A. Le Borgne, K. Tuyls, and A. Nowé. Decentralised reinforcement learning for energy-efficient scheduling in wireless sensor networks. *International Journal of Communication Networks and Distributed Systems*, 6, 2011.

M. Naderan, M. Dehghan, and H. Pedram. Mobile object tracking techniques in wireless sensor networks. In *Ultra Modern Telecommunications & Workshops, 2009. ICUMT'09. International Conference on*, pages 1–8. IEEE, 2009.

Swaprava Nath, Onno Zoeter, Yadati Narahari, and Christopher Dance. Dynamic mechanism design for markets with strategic resources. In *Proceedings of the Twenty-Seventh Conference Annual Conference on Uncertainty in Artificial Intelligence*, pages 539–546, 2011.

A. Nedic and D.P̃. Bertsekas. Least-squares policy evaluation algorithms with linear function approximation. *Journal of Discrete Event Systems*, pages 79–110, 2003. 13:.

J. Niu. Self-learning scheduling approach for wireless sensor network. In *Future Computer and Communication (ICFCC), 2010 2nd International Conference on*, volume 3, pages V3–253. IEEE, 2010.

N. Ouferhat and A. Mellouk. A qos scheduler packets for wireless sensor networks. In *Computer Systems and Applications, 2007. AICCSA'07. IEEE/ACS International Conference on*, pages 211–216. IEEE, 2007.

Y. Pal, LK Awasthi, and AJ Singh. Maximize the lifetime of object tracking sensor network with node-to-node activation scheme. In *IEEE International Advance Computing Conference, 2009. IACC 2009*, pages 1200–1205. IEEE, 2009.

M. Papageorgiou, C. Diakaki, V. Dinopoulou, A. Kotsialos, and Y. Wang. Review of road traffic control strategies. *Proceedings of the IEEE*, 91(12):2043–2067, 2003.

M. Papageorgiou, M. Ben-Akiva, J. Bottom, P.H.L. Bovy, SP Hoogendoorn, N.B. Hounsell, A. Kotsialos, and M. McDonald. ITS and traffic management. *Handbooks in Operations Research and Management Science*, 14:715–774, 2007.

R. Parr, C. Painter-Wakefield, L. Li, and M. Littman. Analyzing feature generation for value-function approximation. *Proceedings of the 24th International Conference on Machine Learning, June 20 – 24, 2007*, 2007. Corvallis, OR.

H. Prasad, L.A. Prashanth, N. Desai, and S. Bhatnagar. Adaptive Smoothed Functional Algorithms for Optimizing Staffing Levels in Service Systems. *Service Science (Under review)*, 2012.

L. Prashanth, H. Prasad, N. Desai, S. Bhatnagar, and G. Dasgupta. Stochastic optimization for adaptive labor staffing in service systems. *Service-Oriented Computing*, pages 487–494, 2011a.

L. A. Prashanth and S. Bhatnagar. Reinforcement learning with function approximation for traffic signal control. *IEEE Transactions on Intelligent Transportation Systems*, 12(2): 412 – 421, 2011a.

L.A. Prashanth and S. Bhatnagar. Reinforcement Learning With Function Approximation for Traffic Signal Control. *IEEE Transactions on Intelligent Transportation Systems*, 12 (2):412 – 421, 2011b.

L.A. Prashanth and S. Bhatnagar. Reinforcement learning with average cost for adaptive control of traffic lights at intersections. In *14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1640–1645. IEEE, 2011c.

L.A. Prashanth and S. Bhatnagar. Threshold Tuning using Stochastic Optimization for Graded Signal Control. *IEEE Transactions on Vehicular Technology (Accepted)*, 2012.

L.A. Prashanth, H. Prasad, N. Desai, S. Bhatnagar, and G. Dasgupta. Stochastic Optimization for Adaptive Labor Staffing in Service Systems. *Service-Oriented Computing*, pages 487–494, 2011b.

L.A. Prashanth, H. Prasad, N. Desai, and S. Bhatnagar. Dynamic Mechanism Design with Capacity Constraints. In *IEEE Transactions on Automation Science and Engineering (Under review)*, 2012a.

L.A. Prashanth, H. Prasad, N. Desai, S. Bhatnagar, and G. Dasgupta. Simultaneous Perturbation methods for Adaptive Labor Staffing in Service Systems. *ACM TOMACS (Under Review)*, 2012b.

K. Premkumar and A. Kumar. Optimal sleep–wake scheduling for quickest intrusion detection using sensor networks. *IEEE INFOCOM, Arizona, USA*, 2008.

M.L̃. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. 1994. John Wiley, New York.

L. Ramaswamy and G. Banavar. A formal model of service delivery. In *2008 IEEE International Conference on Services Computing*, pages 517–520. IEEE, 2008.

P. Ravikumar. Area traffic control system for heterogeneous traffic having limited lane discipline. Technical report, 2009.

T.R. Robbins and T.P. Harrison. A simulation based scheduling model for call centers with uncertain arrival rates. In *Winter Simulation Conference*, pages 2884–2890, 2008.

D.I. Robertson. *TRANSYT: a traffic network study tool.* Road Research Laboratory Crowthorne, 1969.

DI Robertson and RD Bretherton. Optimizing networks of traffic signals in real time-the SCOOT method. *IEEE Transactions on Vehicular Technology*, 40(1, Part 2):11–15, 1991.

Maher Said. Auctions with dynamic populations: Efficiency and revenue maximization. MPRA Paper 11456, University Library of Munich, Germany, July 2010.

A. Salkham, R. Cunningham, A. Garg, and V. Cahill. A collaborative reinforcement learning approach to urban traffic control optimization. In *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, volume 2, pages 560–566, 2008.

J.J. Sanchez-Medina, M.J. Galan-Moreno, and E. Rubiyo-Royo. Traffic signal optimization in “La Almozara” district in Saragossa under congestion conditions, using genetic algorithms, traffic microsimulation, and cluster computing. *IEEE Transactions on Intelligent Transportation Systems*, 11(1):132–141, 2010.

S. Sathiya Keerthi and B. Ravindran. A tutorial survey of reinforcement learning. *Sadhana*, 19(6):851–889, 1994.

P. J. Schweitzer. Perturbation theory and finite Markov chains. *Journal of Applied Probability*, 5:401–413, 1968.

S. Sen and K.L. Head. Controlled optimization of phases at an intersection. *Transportation science*, 31(1):5–17, 1997.

K. Shah and M. Kumar. Distributed independent reinforcement learning (dirl) approach to resource management in wireless sensor networks. In *Mobile Adhoc and Sensor Systems, 2007. MASS 2007. IEEE Internatonal Conference on*, pages 1–9. IEEE, 2007.

A.G. Sims and KW Dobinson. The Sydney coordinated adaptive traffic (SCAT) system philosophy and benefits. *IEEE Transactions on Vehicular Technology*, 29(2):130–137, 1980.

R.H. Smith and D.C. Chin. Evaluation of an adaptive traffic control technique with underlying system changes. In *Proceedings of the Winter Simulation Conference*, pages 1124–1130, 1995.

J.C. Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, 37(3):332–341, 1992. ISSN 0018-9286.

J.C. Spall. A one-measurement form of simultaneous perturbation stochastic approximation. *Automatica*, 33(1):109–112, 1997. ISSN 0005-1098.

J.C. Spall. Adaptive stochastic approximation by the simultaneous perturbation method. *IEEE Transactions on Automatic Control*, 45(10):1839–1853, 2000.

J.C. Spall and D.C. Chin. Traffic-responsive signal timing for system-wide traffic control. *Transportation Research Part C*, 5(3-4):153–163, 1997.

J. Spohrer, P.P. Maglio, J. Bailey, and D. Gruhl. Steps toward a science of service systems. *Computer*, 40(1):71–77, 2007.

P. Sridhar, T. Nanayakkara, A.M. Madni, and M. Jamshidi. Dynamic power management of an embedded sensor network based on actor-critic reinforcement based learning. In *Information and Automation for Sustainability, 2007. ICIAFS 2007. Third International Conference on*, pages 76–81. IEEE, 2007.

D. Srinivasan, M.C. Choy, and R.L. Cheu. Neural networks for real-time traffic signal control. *IEEE Transactions on Intelligent Transportation Systems*, 7(3):261–272, 2006.

Y. Sun, F. Gomez, M.and Ring, and J. Schmidhuber. Incremental basis construction from temporal difference error. *Proceedings of the Twenty Eighth International Conference on Machine Learning*, 2011. Bellevue, WA, USA.

R. S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, pages 9–44, 1988. 3:.

R. S. Sutton and A. Barto. *Reinforcement Learning: An Introduction*, mit press, cambridge, ma. 1998a.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, March 1998b. ISBN 0262193981.

R.S. Sutton and A.G. Barto. *Reinforcement learning: An introduction*, volume 1. Cambridge Univ Press, 1998c.

J. Tsitsikis and B. Van Roy. Average cost temporal-difference learning. Automatica:1799–1808, 1999. 35:.

J. N. Tsitsiklis and B. Van Roy. An analysis of temporal difference learning with function approximation. IEEE Transactions on Automatic Control:674–690, 1997. 42(5):.

J.N. Tsitsiklis. Asynchronous stochastic approximation and Q-learning. *Machine Learning*, 16(3):185–202, 1994a.

J.N. Tsitsiklis. Asynchronous stochastic approximation and q-learning. *Machine Learning*, 16(3):185–202, 1994b.

A. Verma, N.V. Desai, A. Bhamidipaty, A.N. Jain, J. Nallacherry, S. Roy, and S. Barnes. Automated optimal dispatching of service requests. *SRII Global Conference*, 2011.

S. Wasserkrug, S. Taub, S. Zeltyn, D. Gilat, V. Lipets, Z. Feldman, and A. Mandelbaum. Creating operational shift schedules for third-level it support: challenges, models and case study. *International Journal of Services Operations and Informatics*, 3(3):242–257, 2008.

Christopher J. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992a. doi: 10.1007/BF00992698. URL http://dx.doi.org/10.1007/BF00992698.

C.J.C.H. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992b.

J. Wei, A. Wang, and N. Du. Study of self-organizing control of traffic signals in an urban network based on cellular automata. *IEEE Transactions on Vehicular Technology*, 54(2): 744–748, 2005.

M. Wiering, J. Vreeken, J. van Veenen, and A. Koopman. Simulation and optimization of traffic in a city. In *IEEE Intelligent Vehicles Symposium*, pages 453–458, June 2004.

H. Yu and D. P. Bertsekas. Basis function adaptation methods for cost approximation in mdp. *Proceedings of the IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning, March 30 – April 2, 2009*, 2009. Nashville, TN, USA.

X.H. Yu and WW Recker. Stochastic adaptive control model for traffic signal systems. *Transportation Research Part C: Emerging Technologies*, 14(4):263–282, 2006.

I. Yun and B.B. Park. Application of stochastic optimization method for an urban corridor. In *Proceedings of the 38th Winter Simulation Conference*, page 1499, 2006.