Abstraction Using Symmetries in Markov Decision Processes

A THESIS

submitted by

SHRAVAN MATTHUR NARAYANAMURTHY

for the award of the degree

of

MASTER OF SCIENCE
(by Research)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, MADRAS.

October 2007

THESIS CERTIFICATE

This is to certify that the thesis entitled **Abstraction Using Symmetries in Markov Decision Processes**, submitted by **Shravan Matthur Narayanamurthy**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Science (by Research)**, is a bona fide record of the research work carried out by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. B. Ravindran
Research Guide
Assistant Professor
Dept. of Computer Science and Engineering
IIT-Madras, 600 036

Place: Chennai

Date:

ACKNOWLEDGEMENTS

First, I would like to thank my advisor, Dr. B. Ravindran for his guidance and support. The freedom and unconstrained atmosphere that you provided has been a great help. I have always enjoyed discussing with you and amidst all the turbulence, those chats I had with you have helped me set my bearing right. Thank you Sir!

I would also like to thank Dr. N. S. Narayanaswamy and Dr. Shankar Balachandran who have supported and advised me in times of uncertainty. They have helped me gain clearer thoughts that enabled me to take better decisions. Thanks a lot sirs.

I am also thankful to the members of my general test committee, Prof. P. Sreenivasa Kumar and Dr. Venkatesh Balasubramanian. They have always been forthcoming with their advice and help on my research and career.

I would like to express my deepest sense of gratitude to my family for their unceasing support and understanding. Thank you for being there for me when I needed it the most.

I wish to thank my dearest friends, Aditya, Sunando, Mishra, Rohith, Srini, Vimal, Anoop and Hari who have been an integral part of my life at IIT Madras. They have supported me through each one of my struggles. I consider myself very lucky to have made such friends. Thank you very much guys. I will always

cherish your company.

I will always yearn for the deep discussions that I had with Hari and Vimal in the “Bermuda Quadrangle”. We have spent so many fun-filled hours there that it makes me nostalgic. Thanks guys. I have learnt a lot from you two!

I wish to thank all the members of RISE lab for creating a vibrant atmosphere! I would also like to thank the CSE department for providing me with excellent facilities to complete my research. Thanks to all the administrative, faculty and student community of the department.

Finally, I wish to thank the institute for these wonderful two years

ABSTRACT

KEYWORDS: Markov Decision Processes, Symmetries, Abstraction

Current approaches to solving Markov Decision Processes (MDPs), the de-facto standard for modeling stochastic sequential decision problems, scale poorly with the size of the MDP. When used to model real-world problems though, MDPs exhibit considerable implicit redundancy, especially in the form of symmetries. However, existing model minimization approaches do not efficiently exploit this redundancy due to symmetries. These approaches involve constructing a reduced model first and then solving them. Hence we term these as "explicit minimization" approaches.

In the first part of this work, we address the question of finding symmetries of a given MDP. We show that the problem is *Isomorphism Complete*, that is, the problem is polynomially equivalent to verifying whether two graphs are isomorphic. Apart from the theoretical importance of this result it has an important practical application. The reduction presented can be used together with any off-the-shelf Graph Isomorphism solver, which performs well in the average case, to find symmetries of an MDP. In fact, we present results of using NAutY (the best Graph Isomorphism solver currently available), to find symmetries of MDPs. We next address the issue of exploiting the symmetries of a given MDP. We propose the use of an explicit model minimization algorithm called the $\mathcal{G}$-reduced image algorithm

that exploits symmetries in a time efficient manner. We present an analysis of the algorithm and corroborate it with empirical results on a probabilistic GridWorld domain and a single player GridWorld Soccer domain. We also present results of integrating the symmetry finding with the explicit model minimization approach to illustrate an end-to-end approach for "Abstraction using Symmetries in MDPs".

We then note some of the problems associated with this explicit scheme and as a solution present an approach wherein we integrate the symmetries into the solution technique implicitly. However, we should select a suitable solution technique so that the overheads due to integration do not outweigh the improvements. We validate this approach by modifying the Real Time Dynamic Programming (RTDP) algorithm and empirically demonstrate significantly faster learning and reduced overall execution time on several domains.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

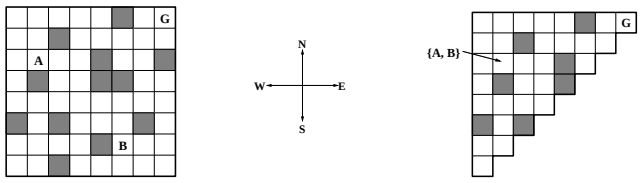
CHAPTER 1

Introduction

Markov Decision Processes (MDPs) [Puterman, 1994] have become the de-facto standard for modeling and solving stochastic sequential decision problems. Many real world problems can be easily modeled using MDPs. However, due to their large size, they usually do not yield readily to the current solution techniques as most of the solution techniques scale poorly with the size of the MDP. Nevertheless, models of real world problems exhibit much redundancy that can be eliminated, reducing the size of the problem.

One way of handling redundancy is to form abstractions, as we humans do, by ignoring details not needed for performing the immediate task at hand. While driving, for example, we ignore details regarding clothing and the state of our hair. On the other hand, while preparing to attend a ball, we would want to pay special attention to our clothing and hair. Researchers in artificial intelligence (AI), in particular machine learning (ML), have long recognized the utility of being able to form abstractions.

Researchers in many fields, ranging from various branches of mathematics to social network analysis, also recognize the utility of abstractions. They have tried to answer some important questions on the usefulness of abstractions and on modeling abstractions. Informally, one can define a good abstraction to be a function of the observable features of a task such that it is a "sufficient statistic" and the notion of sufficiency varies with the goal.



(a) (b)

Figure 1.1: **(a)** A symmetric GridWorld problem. The goal state is G and there are four deterministic actions. State-action pairs (A, E) and (B, N) are equivalent in the sense described in the text. **(b)** A reduced model of the GridWorld in (a). The state-action pairs (A, E) and (B, N) in the original problem both correspond to the pair $(\{A, B\}, E)$ in the reduced problem. A solution to this reduced GridWorld can be used to derive a solution to the full problem.

Determining sufficiency and providing ways of modeling abstractions are well studied problems in AI (e.g., Amarel [1968]; Popplestone and Grupen [2000]; Dean *et al.* [1997]; Knoblock [1990]; Dean and Lin [1995]). Specifically, for MDPs we use the MDP homomorphisms framework proposed by Ravindran [2004] as it is generic and can accommodate different notions of abstractions.

The approach to abstraction we use in this work belongs to the class of model minimization methods. The goal of model minimization is to derive a reduced model representation in which some key property of the original model is preserved. In the case of MDPs, we require that transition and reward dynamics of the MDP is preserved in the reduced model.

To illustrate the concept of minimization, consider the simple GridWorld shown in Figure 1.1(a). The goal state is labeled G. Taking action E in state A is equivalent to taking action N in state B, in the sense that they go to equivalent states that are both one step closer to the goal. One can say that the state-action pairs (A, E) and (B, N) are equivalent. One can exploit this notion of equivalence to construct

a smaller model of the GridWorld (Figure 1.1(b)) that can be used to solve the original problem.

Figure 1.1 also illustrates a situation in which the symmetry in the problem is exploited in the abstraction. Symmetries of a structure are characterized traditionally by the symmetry group of the structure. This is the group of mappings of the structure onto itself, such that some structural property is preserved. For example, in the GridWorld in Figure 1.1(a), such a mapping is given by reflecting the states about the NE-SW diagonal and flipping actions N and E, and actions S and W. This leaves the transition structure of the GridWorld unaltered. The MDP homomorphism framework incorporates this traditional group-theoretic definition to model symmetries of MDPs. Identifying symmetrically equivalent situations frequently results in useful abstraction. The model minimization approach to this is to find symmetries and derive the corresponding minimal image. We explore the following aspects of using symmetries for abstraction:

1. Complexity of the problem of finding symmetries
2. Efficient approaches to deriving the minimal model and application of symmetries in solving the original MDP

1.1 Outline of the Thesis

We establish notation for the thesis, provide some background and discuss related work in Chapter 2. We then take up the problem of finding symmetries in Chapter 3. We prove that it is equivalent to finding whether two graphs are isomorphic and discuss the significance of the result. The constructive proof helps us to use heuristic algorithms available for finding graph isomorphisms to find symmetries

for MDPs. Next, we propose the use of a polynomial time algorithm [Ravindran, 2004] to explicitly construct the $\mathcal{G}$-reduced image of an MDP given the symmetry group, $\mathcal{G}$. We prove the correctness of the algorithm and demonstrate its efficiency on a couple of domains. This addresses the second part of the goal. Due to the explicit nature of the construction and representation of MDPs, we categorize them as "Explicit Model Minimization". We also present results on the time taken for Explicit Model Minimization using Value Iteration.

We then identify some problems associated with this explicit scheme of things in Chapter 4. As a solution, we propose an idea for "Implicit Model Minimization". We note that the idea cannot be implemented with all solution techniques. However, used with the right solution technique and representation, we observe notable reduction in execution times and significantly faster learning. We demonstrate the results of using a modified version of the Real Time Dynamic Programming (RTDP) [Barto *et al.*, 1995] algorithm on multiple domains.

Finally, we summarize and present some directions for future work in Chapter 5.

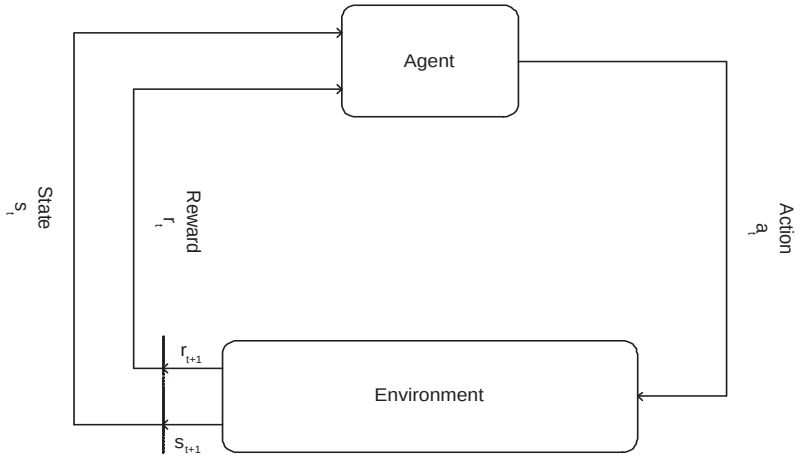
CHAPTER 2

Background and Related Work

In this chapter we introduce some notation that we will use in the thesis. We provide some background on the Reinforcement Learning problem and introduce in a limited fashion the necessary Dynamic Programming (DP) solutions. We also introduce the MDP homomorphism framework and provide an account of the related literature. With this background we formally define the problem and bound the improvements possible if the problem is solved.

2.1 Reinforcement Learning Problem

This section is adapted from Sutton and Barto [1998], Puterman [1994] and Ravindran [2004]. The reinforcement learning problem is a formulation of the problem



of learning from interaction to achieve a goal. The learner is called the agent. It

interacts continually with, what is called, the environment by selecting actions. The environment responds to those actions by presenting new situations to the agent. The environment also gives rise to rewards, special numerical values that the agent tries to maximize over time. An instance of the problem is called a task and it involves a complete specification of an environment. A reinforcement learning task that satisfies the Markov property is called a Markov Decision Process, or MDP. If the state and action spaces are finite, then it is called a finite Markov Decision Process (finite MDP). Finite MDPs are particularly important to the theory of reinforcement learning. We provide a limited introduction to MDPs next.

2.1.1 Markov Decision Processes

A finite *Markov decision process* is a tuple $\langle S, A, \Psi, P, R \rangle$, where S is the set of states, A is the set of actions, $\Psi \subseteq S \times A$ is the set of admissible state-action pairs, $P : \Psi \times S \to [0, 1]$ is the transition probability function with $P(s, a, s')$ being the probability of transition from state s to state s' under action a, and $R : \Psi \to \mathbb{R}$ is the expected reward function, with $R(s, a)$ being the expected reward for performing action a in state s. We assume that the rewards are bounded. Let $A_s = \{a \mid (s, a) \in \Psi\} \subseteq A$ denote the set of actions admissible in state s. We assume that for all $s \in S$, A_s is non-empty. In this work we assume that the set of states and set of actions are finite, but the language of homomorphisms we employ extends to infinite spaces with little work.

A *stochastic policy* π is a mapping from Ψ to the real interval $[0, 1]$ s.t. $\sum_{a \in A_s} \pi(s, a) = 1$ for all $s \in S$. For any $(s, a) \in \Psi$, $\pi(s, a)$ gives the probability of picking action a in state s. The *value* of state s under policy π is the expected value of the discounted

sum of future rewards starting from state s and following policy π thereafter. The *value function* V^π corresponding to a policy π is the mapping from states to their values under π. It can be shown (e. g., Bertsekas [1987]) that V^π satisfies the *Bellman equation*:

$$V^\pi(s) = \sum_{a \in A_s} \pi(s,a) \left[R(s,a) + \gamma \sum_{s' \in S} P(s,a,s') V^\pi(s') \right],$$

where $0 \leq \gamma < 1$ is a discount factor. This formulation is known as the discounted sum of rewards criterion.

Similarly, the value of a state-action pair (s,a) under policy π is the expected value of the discounted sum of future rewards starting from state s, taking action a, and following π thereafter. The *action value function* Q^π corresponding to a policy π is the mapping from state-action pairs to their values and satisfies:

$$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s,a,s') V^\pi(s'),$$

where $0 \leq \gamma < 1$ is a discount factor.

The solution of an MDP is an *optimal policy* $\pi^\star$ that uniformly dominates all other possible policies for that MDP. In other words, $V^{\pi^\star}(s) \geq V^\pi(s)$ for all s in S and for all possible π. It can be shown [Bertsekas, 1987] that the value function for all optimal policies is the same. We denote this *optimal value function* by $V^\star$. It satisfies the *Bellman optimality equation*:

$$V^\star(s) = \max_{a \in A_s} \left[R(s,a) + \gamma \sum_{s' \in S} P(s,a,s') V^\star(s') \right].$$

Similarly the *optimal action value function* $Q^\star$ satisfies:

$$Q^\star(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s,a,s') \max_{a' \in A_{s'}} Q^\star(s',a').$$

These two optimal value functions are related by $V^\star(s) = \max_a Q^\star(s,a)$. Typically MDPs are solved by approximating the solution to the Bellman optimality equations (e. g., Bertsekas, 1987; Sutton and Barto, 1998). Given the optimal action value function, an optimal policy is given by

$$\begin{aligned} \pi^\star(s,a) &\geq 0 \quad \text{if } Q^\star(s,a) = \max_{a' \in A_s} Q^\star(s,a') \\ &= 0 \quad \text{otherwise.} \end{aligned}$$

Next, we discuss a class of methods for solving the reinforcement learning problem, formulated as a MDP, called Dynamic Programming (DP).

2.1.2 Dynamic Programming

The term dynamic programming (DP) [Bellman, 1957] refers to a collection of algorithms used to compute optimal policies given a model of the environment as a Markov decision process (MDP).

The key idea of DP, and of reinforcement learning generally, is the use of value functions to organize and structure the search for good policies. Next we present two algorithms for solving the reinforcement learning problem. We only provide the algorithm and its complexity for completeness. The details can be found in [Sutton and Barto, 1998] and [Puterman, 1994]. Complexity results can also be found in [Littman, Dean, and Kaelbling, 1995].

Value Iteration

Algorithm 1 Value Iteration with action value functions

Given $\mathcal{M} = \langle S, A, \Psi, P, R \rangle$ and $\mathcal{G} \leq Aut\mathcal{M}$,
Hashtables $Q_0, Q_1 \leftarrow Nil$ are the action value functions for the previous and current iterations respectively
$|S|$ dimension vectors v_0, v_1 are the state value functions for the previous and current iterations respectively
$v_0 \leftarrow \bar{\mathbf{0}}$
repeat
  for each $s \in S$ **do**
    for each $a \in A_s$ **do**
      if $(s, a) \notin Q_1$ **then**
        add (s, a) to Q_1
        $Q_1(s, a) \leftarrow 0$
      end if

$$Q_1(s, a) \leftarrow R(s, a) + \gamma \sum_{s' \in S'} P(s, a, s') \max_{a' \in A_{s'}} Q_0(s', a')$$

    end for
  end for
  $\forall s \in S, v_1(s) = \max_{a \in A_s} Q_1(s, a)$
  $\Delta \leftarrow \max(abs(v_1 - v_0))$
  $Q_0 \leftarrow Q_1$
  $v_0 \leftarrow v_1$
until $\Delta < \frac{\epsilon(1-\gamma)}{2\gamma}$

Value Iteration is a simple iterative algorithm that uses the bellman optimality equation as the update equation to find successive approximations to the optimal value function. The algorithm is presented in Algorithm 1.

The computational complexity per iteration of the value-iteration algorithm with full backups is quadratic in the number of states and linear in the number of actions. Commonly, the transition probabilities P(s,a,s') are sparse. If there are a constant number of next states with non-zero probability then the cost per iteration is linear in the number of states and linear in the number of actions. The

number of iterations required to reach the optimal policy is polynomial in the number of states and the magnitude of the largest reward if the discount factor is held constant [Littman *et al.*, 1995]. However, in the tests that we run, we only need an ϵ-optimal policy.[1] So we fix ϵ and use the Bellman error magnitude to decide when the current value function is ϵ-optimal. So, effectively, we can write the running time of value iteration as $O(|\Psi|.|S|.p_\epsilon)$, where p_ϵ denotes the number of iterations for converging to an ϵ-optimal policy with a fixed ϵ, which can be considered independent of the size of the state and action spaces; a fact that we use to simplify analysis.

Real Time Dynamic Programming (RTDP)

Algorithm 2 Real Time Dynamic Programming algorithm

Given $\mathcal{M} = \langle S, A, \Psi, P, R \rangle$ and $\mathcal{G} \leq Aut\mathcal{M}$,
Hashtable $Q \leftarrow Nil$ is the action value function.
for each episode **do**
  Initialize s and $S' \leftarrow \{s\}$
  Choose a from s using policy derived from Q (e.g. ϵ-greedy policy)
  for each step of the episode **do**
    Take action a and observe reward r and next state s'
    Choose a' from s' using policy derived from Q (e.g. ϵ-greedy policy)
    if $(s, a) \notin Q$ **then**
      add (s, a) to Q
      $Q(s, a) \leftarrow 0$
    end if

$$Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in S} P(s,a,s') \max_{a' \in A_{s'}} Q(s',a')$$

    $s \leftarrow s'; a \leftarrow a'$
  end for
end for

[1] An ϵ-optimal policy π is such that $Q^\star(s,a) \leq Q^\pi(s,a) + \epsilon$ for all $(s,a) \in \Psi$.

A major drawback to value iteration that we discussed above is that they involve operations over the entire state set of the MDP, that is, they require sweeps of the state set. If the state set is very large, then even a single sweep can be very expensive. For example, the game of backgammon has over 10^{20} states. Even at the rate of a million state values backed up per second, value iteration would take over a thousand years to complete a single sweep.

Asynchronous DP algorithms are in-place iterative DP algorithms that do not have a fixed order for backing up the values of states. The values of some states may be backed up several times before the values of others are backed up once. However for convergence, the values of all states must be backed up and no state can be ignored after some point of time. Asynchronous DP algorithms allow great flexibility in selecting states to which backup operations are applied.

RTDP is an asynchronous DP algorithm where the agent performs asynchronous DP concurrently with the process of executing actions, i.e., learning and control occur simultaneously. The interaction is as follows:

1. Control decisions are based on the most up-to-date information from the DP computation, and
2. The control decisions influence the selection of states to which the DP backup is applied

As a consequence of this interaction, the DP computation can *focus* on regions of the state set that are most relevant for control as revealed in the system's behavior. However, specific conditions on the interaction must hold for the algorithm to be RTDP [Barto *et al.*, 1995]. These conditions are necessary to ensure the convergence properties of the algorithm. Another consequence of these interactions is that the

number of iterations required to converge to an ϵ-optimal policy is dependent on the state-action space and topology.

A specific implementation is presented in Algorithm 2. In spite of the term "real-time" being used, DP and control can be carried out in *simulation mode*, where the model is used as a surrogate for the actual system.

2.2 Homomorphisms and Symmetry Groups

This section has been adapted from [Ravindran and Barto, 2002].

Let B be a partition of a set X. For any $x \in X, [x]_B$ denotes the block of B to which x belongs. Any function f from a set X to a set Y induces a partition (or equivalence relation) on X, with $[x]_f = [x']_f$ if and only if $f(x) = f(x')$ and x, x' are f-equivalent written $x \equiv_f x'$. Let B be a partition of $Z \subseteq X \times Y$, where X and Y are arbitrary sets. The projection of B onto X is the partition $B|X$ of X such that for any $x, x' \in X, [x]_{B|X} = [x']_{B|X}$ if and only if every block containing a pair in which x is a component also contains a pair in which x' is a component or every block containing a pair in which x' is a component also contains a pair in which x is a component.

Definition 1. An *MDP homomorphism* h from an MDP $\mathcal{M} = \langle S, A, \Psi, P, R \rangle$ to an MDP $\mathcal{M}' = \langle S', A', \Psi', P', R' \rangle$ is a surjection from Ψ to Ψ', defined by a tuple of surjections $< f, \{g_s | s \in S\} >$, with $h((s,a)) = (f(s), g_s(a))$, where $f : S \rightarrow S'$ and

$g_s : A_s \to A'_{f(s)}$ for $s \in S$, such that: $\forall s, s' \in S, a \in A_s$

$$P'(f(s), g_s(a), f(s')) = \sum_{s'' \in [s']_f} P(s, a, s'') \tag{2.1}$$

$$R'(f(s), g_s(a)) = R(s, a) \tag{2.2}$$

We use the shorthand $h(s, a)$ for $h((s, a))$. Often for convenience, we use $< f, \{g_s\} >$ to denote $< f, \{g_s | s \in S\} >$.

Definition 2. Let $\mathcal{M}'$ be an image of the MDP $\mathcal{M}$ under homomorphism $h =< f, \{g_s\} >$. For any $s \in S, g_s^{-1}(a')$ denotes the set of actions that have the same image $a' \in A'_{f(s)}$ under g_s. Let π' be a stochastic policy in $\mathcal{M}'$. Then π' lifted to $\mathcal{M}$ is the policy $\pi_{\mathcal{M}'}$ such that for any $a \in g_s^{-1}(a'), \pi'_{\mathcal{M}}(s, a) = \pi'(f(s), a')/|g_s^{-1}(a')|$

Definition 3. An MDP homomorphism $h =< f, \{g_s\} >$ from MDP $\mathcal{M} = \langle S, A, \Psi, P, R \rangle$ to MDP $\mathcal{M}' = \langle S', A', \Psi', P', R' \rangle$ is an MDP *isomorphism* from $\mathcal{M}$ to $\mathcal{M}'$ if and only if f and g_s, are bijective. $\mathcal{M}$ is said to be *isomorphic* to $\mathcal{M}'$ and vice versa. An MDP isomorphism from MDP $\mathcal{M}$ to itself is called an *automorphism* of $\mathcal{M}$.

Definition 4. The set of all automorphisms of an MDP $\mathcal{M}$, denoted by $Aut\mathcal{M}$, forms a group under composition of homomorphisms. This group is the *symmetry group* of $\mathcal{M}$.

Let $\mathcal{G}$ be a subgroup of $Aut\mathcal{M}$. The subgroup $\mathcal{G}$ induces a partition $B_{\mathcal{G}}$ of Ψ: $[(s_1, a_1)]_{B_{\mathcal{G}}} \equiv [(s_2, a_2)]_{B_{\mathcal{G}}}$ if and only if there exists $h \in \mathcal{G}$ such that $h(s_1, a_1) = (s_2, a_2)$ and $(s_1, a_1), (s_2, a_2)$ are said to be $\mathcal{G}$ equivalent written $(s_1, a_1) \equiv_{\mathcal{G}} (s_2, a_2)$. Further if $s_1 \equiv_{B_{\mathcal{G}}|S} s_2$ then we write as shorthand $s_1 \equiv_{\mathcal{G}|S} s_2$. It can be proved that there exists a homomorphism $h^{\mathcal{G}}$ from $\mathcal{M}$ to some $\mathcal{M}'$, such that the partition induced by $h^{\mathcal{G}}$,

$B_{h^{\mathcal{G}}}$, is the same as $B_{\mathcal{G}}$. The image of $\mathcal{M}$ under $h^{\mathcal{G}}$ is called the *$\mathcal{G}$-reduced image* of $\mathcal{M}$.

Definition 5. An MDP $\mathcal{M}'$ is said to be a *reduced model* of an MDP $\mathcal{M}$, iff there exists an MDP homomorphism $h : \mathcal{M} \rightarrow \mathcal{M}'$.

2.2.1 Structured MDPs

A structured MDP is described by the tuple $\langle S, A, \Psi, P, R \rangle$ where the state set S is given by a set of M features, that is, $S \subseteq \prod_{i=1}^{M} S_i$, where S_i is the set of permissible values for the feature i. So a state represents a unique assignment to the state variables s_i.[2]

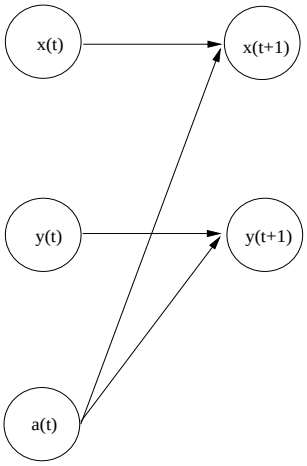
The transition probabilities P are often described by a two-slice *Temporal Bayesian Network* (2-TBN). The state transition probabilities can be factored as:

$$P(s, a, s') \quad = \quad \prod_{i=1}^{M} Prob(s'_i | Pre(s'_i, a)) \tag{2.3}$$

where $Pre(s'_i, a)$ denotes the parents of node s'_i in the 2-TBN corresponding to action a and each of the probabilities $Prob(s'_i | Pre(s'_i, a))$ is given by a conditional probability table (CPT) associated with node s'_i. The reward function may be similarly represented.

An example 2-TBN is shown in Figure 2.1 which represents transition probabilities of the usual GridWorld domain. The co-ordinates of the grid act as features that are denoted as nodes. The CPT for the changes corresponding to state transitions factored on the x-co-ordinate are also shown in the figure.

[2]We do not consider structure in the action space in this work.



x(t)	a(t)	x(t+1)	P
x	N	x	0.9
x	E	x + 1	0.9
x	W	x	0.9
x	S	x + 1	0.9

Figure 2.1: Factored representation of a Grid-World domain where (x,y) are the co-ordinates of a grid

2.3 Related Work

The use of symmetries as a tool for simplification and elegance is widespread in almost all fields of science. For example, a simple application of symmetry is in integration where the problem is often simplified by using symmetry of the function. Another application is for lowering the order of an Ordinary Differential Equation (ODE) or to reduce the number of variables of a Partial Differential Equation (PDE) [Wolf, 1995]. In chemistry, point group symmetry provides a useful classification scheme for simple molecules [Dunitz, 1996].

Not surprisingly, symmetries have been used, heavily, even in the field of computer science in the areas of Automata, Planning and Constraint Satisfaction and Model Checking. The goal is generally to find a smaller model which is equivalent to the original model or to prune the search space by symmetry breaking to improve search efficiency. We first discuss the model minimization approach for MDPs which extends notions of minimization in automata literature. We then discuss symmetry breaking in Constraint Satisfaction.

MDP Minimization is a well studied problem. As stated earlier, in the model minimization approach, a reduced MDP that that preserves some key properties as the original MDP is found by combining "equivalent" states. The reduced MDP found depends on the notion of equivalence between states used in the aggregation. The notion of equivalence chosen will be fundamental in designing and analyzing algorithms for reducing MDPs. In [Dean and Givan, 1997] a minimization algorithm is proposed based on the notion of *stochastic bi-simulation homogeneity*. Informally, a partition of the state space for an MDP is said to be homogenous if for each action, states in the same block have the same probability of transitioning to each other block. They also provide an algorithm for finding the coarsest homogenous refinement of any partition of the state space of an MDP. The algorithm starts with an initial partition P_0 and iteratively refines it by splitting the blocks until the coarsest homogenous refinement of P_0 is obtained. A notion of stability of a block with respect to another is defined and unstable blocks are split till all blocks of the partition are stable. The complexity of the algorithm is expressed in terms of the partition manipulation operations. Hence, the actual complexity depends on the underlying partition representation and manipulation algorithms. Givan *et al.* [2003] discuss the application of the algorithm to solving factored MDP problems. Enumerating the state space is avoided by describing large blocks of equivalent states in factored form with the block descriptions being inferred directly from the original factored representation.

Ravindran [2004] proposes a more generic framework based on the notion of MDP homomorphisms with *state-dependent action recoding* as introduced in Section 2.2. This allows a greater reduction in problem size and aids in modeling many other notions of equivalence like symmetries. A polynomial time algorithm to find

the reduced model under the notion of MDP homomorphisms is also proposed by extending the algorithm proposed by Givan *et al.* [2003] and Lee and Yannakakis [1992]. Again, the algorithm is polynomial in the number of block operations, the stability criterion is modified to suit the equivalence notion and the same process of iterative splitting is used. The notion of stability used is called the *stochastic substitution property*, which is an extension of the *substitution property* for finite state machines [Hartmanis, 1966].

However, literature on MDP minimization using symmetries is sparse. Zinkevich and Balch [2001] define symmetries based on state-action equivalence but do not make any connections to group-theoretic concepts or minimization algorithms. So we look at literature from the area of Constraint Satisfaction.

In artificial intelligence and operations research, constraint satisfaction is the process of finding a solution to a set of constraints. Such constraints express allowed values for variables. A solution is therefore an evaluation of these variables that satisfies all constraints. Formally, a constraint satisfaction problem is defined as a triple $< X, D, C >$, where X is a set of variables, D is a set of domain values and C is a set of constraints. Every constraint is in turn a pair $< t, R >$, where t is a tuple of variables and R is a set of tuples of values; all these tuples having the same number of elements; as a result R is a relation. An evaluation of the variables is a function from $v : X \to D$. Such an evaluation satisfies a constraint $< (x_1, x_2, \ldots, x_n), R >$ if $(v(x_1), v(x_2), \ldots, v(x_n)) \in R$. A solution is an evaluation that satisfies all constraints in C.

A problem contains symmetry when any of its structure can be permuted to give an equivalent problem. Symmetry in a Constraint Satisfaction Problem (CSP)

can be defined as solution preserving or constraint preserving. Solution symmetry is a permutation of the variables and values that preserve the set of solutions to the problem. A constraint symmetry is an automorphism of the micro-structure of the problem.

Symmetries occur frequently in constraint satisfaction problems leading to redundant search when traditional backtracking methods are used. Their removal will thus simplify the problem space. Benhamou [1994] define solution preserving symmetries and point out that solution preserving symmetries are not practically useful as the solution has to be found for identifying symmetries. Hence they define constraint symmetry as a permutation on the domain values that does not alter membership of the value tuples to the constraint relation, when the permutation is applied. This naturally induces permutations on the relations themselves. They derive a set of necessary conditions for values to be symmetric and use these conditions for symmetry breaking.

Another approach to symmetry breaking checks the current partial assignment against previous no-goods. Pearson [2004] uses graph isomorphism to check if a partial assignment is symmetrically equivalent to a previously found no-good. They use the **NAutY** [McKay, 1981] system to store canonical representations of the no-good graphs and compare the canonical graph of the partial assignment with no-good canonical to check symmetrical equivalence.

Another dimension to analyze the literature is the approach to symmetry finding. Two main approaches exist:

1. To derive a set of necessary conditions for elements to be symmetric
2. Prove Isomorphism Completeness and use a graph isomorphism finding system

Intuitively symmetries seem easier to identify than homomorphisms and we tried the first approach to find a polynomial time algorithm for symmetry finding, along the lines of the MDP homomorphism finding, with the motivation of finding better algorithms for MDP minimization. The MDP homomorphism definition allows for deriving this easily because, two state action pairs $(s_1, a_1), (s_2, a_2)$ are homomorphically equivalent if

$$\begin{aligned} h(s_1, a_1) &= h(s_2, a_2) \\ \text{So, } P'(f(s_1), g_{s_1}(a_1), f(s')) &= P'(f(s_2), g_{s_2}(a_2), f(s'))\forall s' \in S \\ \text{Hence, } T(s_1, a_1, [s']_{B_h|S}) &= T(s_2, a_2, [s']_{B_h|S})\forall s' \in S \end{aligned}$$

This is the stochastic substitution property and it allows us to deal just with blocks without worrying about the actual functions. However, a similar attempt for symmetries still needs the symmetry f in the necessary condition as below:

$$\begin{aligned} h(s_1, a_1) &= (s_2, a_2) \\ \text{So, } P(f(s_1), g_{s_1}(a_1), f(s')) &= P(s_2, a_2, f(s'))\forall s' \in S \\ \text{and } P(s_1, a_1, s') &= P(s_2, a_2, f(s'))\forall s' \in S \end{aligned}$$

Flener *et al.* [2002] and Crawford [1992] point that symmetry finding for CSPs in general is Isomorphism Complete. However, there also exist results showing that symmetry finding is NP-complete (in case of geometric automorphism of graphs [Manning, 1990]). So we were still unclear whether symmetry finding for MDPs is Isomorphism Complete or NP-complete due to the presence of factorially many action recoding functions. A better understanding of the use of symmetries for abstraction in MDPs is the motivation for this work.

2.4 Problem Definition

To exploit the power of abstraction using symmetries, we identify them and construct a reduced model by abstracting away the symmetric portions. As the reduced model can be significantly smaller, it can be easier to solve. We use the notion of automorphisms to model symmetries. So formally, given an MDP $\mathcal{M}$,

1. Find the automorphism group, $Aut\mathcal{M}$ and
2. Given the automorphism group, $Aut\mathcal{M}$ find the corresponding reduced model, the $Aut\mathcal{M}$-Reduced Image

2.5 Bounds on improvement

By definition, a reduced model preserves the dynamics of the system. Ravindran and Barto [2001] show that optimal value functions and policies are also preserved by reduced models. So, reduced models are functionally equivalent to the original model but are significantly smaller. Hence running times of solution techniques can be reduced by following the model minimization approach:

1. Construct the functionally equivalent reduced model.
2. Solve the reduced model.
3. *Lift* the solution to the original model.

To estimate the improvements obtained let us look at value iteration. The results below apply to stationary deterministic markov policies.

2.5.1 Value Iteration

Let $\mathcal{M}' =< S', A', \Psi', P', R' >$ be a reduced model of $\mathcal{M} =< S, A, \Psi, P, R >$.

Let $|\Psi'| = |\Psi|/k_1$ and $|S'| = |S|/k_2$.

As seen in Section 2.1.2, to solve $\mathcal{M}'$, that is, finding an ϵ-optimal policy using Value Iteration takes $n_1.|\Psi'|.|S'|.p_\epsilon$ time where, p_ϵ is the number of iterations taken to converge to an ϵ-optimal value function, which is independent of $|\Psi'|$ or $|S'|$ and $n_1 \in \mathbb{N}$.

Also, lifting the policy takes $n_2.|S|$ where, $n_2 \in \mathbb{N}$.

$$
\begin{aligned}
n_1.|\Psi'|.|S'|.p_\epsilon + n_2.|S| &\leq n.(\frac{|\Psi|.|S|.p_\epsilon}{k_1.k_2} + |S|) \text{ where } n = max(n_1, n_2) \\
&\leq n.(\frac{|\Psi|.|S|.p_\epsilon + k^2.|S|}{k^2}) \text{ where } k = max(k_1, k_2) \\
&\leq n.(\frac{|\Psi|.|S|.(p_\epsilon + 1)}{k^2}) \\
&\approx n.(\frac{|\Psi|.|S|.p_\epsilon}{k^2})
\end{aligned}
$$

Hence, if we are already given a reduced model, then we can achieve a speed-up factor, which is at most the square of the reduction in the state space of the reduced model. But, it should be noted that this is the best possible scenario because in practice, the number of states that one can transition to, from a given state, is constant and reduces the speed-up to nearly as much as the reduction in state-action space. This is demonstrated in Figure 2.2 on the probabilistic grid-world domain.

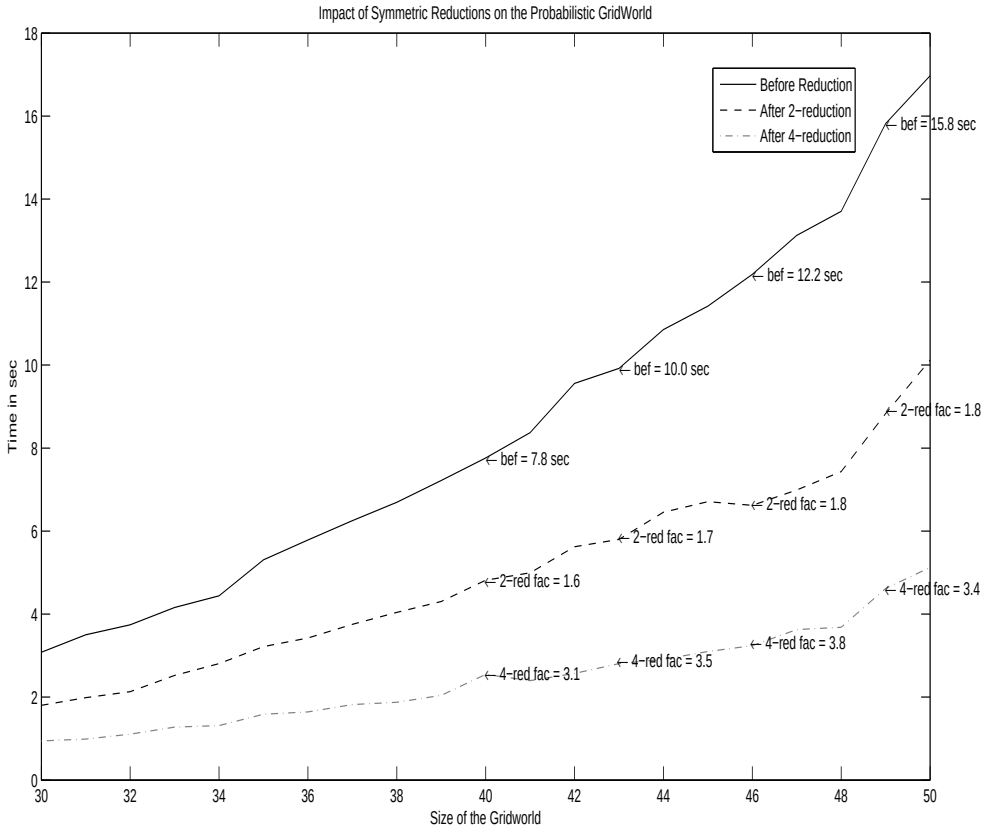


Figure 2.2: Running times of the Value Iteration algorithm on the Probabilistic GridWorld domain plotted against the size of the GridWorld with various degrees of symmetry (0, 2 and 4)

With the above background, we address the problem defined in Section 2.4 in the next chapter.

CHAPTER 3

Explicit Model Minimization

3.1 Finding Symmetries

3.1.1 Problem Simplification

Let us consider the first part of our problem, i.e., given an MDP $\mathcal{M}$, find the automorphism group of $\mathcal{M}$, $Aut\mathcal{M}$. We know that a group can be specified using its generators. So we simplify the problem to finding the generators of $Aut\mathcal{M}$. Let $AMGEN(\mathcal{M})$ denote the problem of finding the generators of $Aut\mathcal{M}$. We write $A \propto B$ if a problem A is polynomially reducible to B. We say that problems A and B are polynomially equivalent iff $A \propto B$ and $B \propto A$. We denote polynomial equivalence by $\equiv_{\propto}$.

Definition 6. A problem A is *Isomorphism Complete* iff A is polynomially equivalent to finding whether two graphs are isomorphic.

Let G_1, G_2 be two simple graphs unless otherwise mentioned. The following is a list of relevant *Isomorphism Complete* problems [Booth and Colbourn, 1977] on graphs:

- $ISO(G_1, G_2)$: Isomorphism recognition for G_1 and G_2
- $IMAP(G_1, G_2)$: Isomorphism Map from G_1 to G_2(if it exists),
- $AGEN(G_1)$: Generators of the automorphism group, $AutG_1$

- *DGEN*(*G*): Generators of the automorphism group, *Aut*G, where G is a weighted digraph

From [Mathon, 1979], [Read and Corneil, 1977], [Miller, 1977] we have,

$DGEN(G) \equiv_{\propto} AGEN(G) \equiv_{\propto} IMAP(G_1, G_2) \equiv_{\propto} ISO(G_1, G_2)$.

We intend to prove that $AMGEN(\mathcal{M})$ is *Isomorphism Complete*. We are done if we prove that $AMGEN(\mathcal{M}) \equiv_{\propto} DGEN(G_{\mathcal{M}})$, where $G_{\mathcal{M}}$ is a weighted graph constructed in polynomial time from $\mathcal{M}$, that is, $AMGEN(\mathcal{M}) \propto DGEN(G_{\mathcal{M}})$ and $DGEN(G_{\mathcal{M}}) \propto AMGEN(\mathcal{M})$. It is easy to see that $DGEN(G_{\mathcal{M}}) \propto AMGEN(\mathcal{M})$ is true because we can always construct a degenerate MDP from a digraph. So we need to prove that $AMGEN(\mathcal{M}) \propto DGEN(G_{\mathcal{M}})$.

3.1.2 *Isomorphism Completeness* of the problem

An MDP $\mathcal{M}$ can be considered as a pseudograph with states acting as vertices and actions acting as edges. Since there can be more than one action affecting the transition between 2 states, we need to represent this as a pseudograph. The transition probabilities and rewards can be thought of as weight functions. Next, we formally pose $AMGEN(\mathcal{M})$ as a problem on a weighted pseudograph.

Let $G_{\mathcal{M}} =< \Sigma_a, V, E, W_P, W_R >$ be the pseudograph corresponding to $\mathcal{M}$, where

$$
\begin{aligned}
\Sigma_a &: \text{Alphabet for labelling corresponding to actions} \\
V &: \text{Set of vertices corresponding to states} \\
E &: \text{Set of edges, where each edge is a triple} \\
&\quad (u, a, v) \text{ where, } u, v \in V \text{ and } a \in \Sigma_a \\
&\quad \text{corresponding to state transitions} \\
W_P &: E \to \mathbb{R} \text{ corresponding to transition probabilities} \\
W_R &: E \to \mathbb{R} \text{ corresponding to rewards with} \\
&\quad W_R(u, a, v) = W_R(u, a, v') \ \forall\ (u, a, v), (u, a, v') \in E \\
\text{Note, } E &= \bigcup_{u,v \in V} E_{uv} \text{ where, } E_{uv} = \{\, (u', a, v') \in E \mid u' = u \text{ and } v' = v \,\}
\end{aligned}
$$

$AMGEN(\mathcal{M})$ can be formulated as finding the generators of the group of bijections $h : V \times \Sigma_a \to V \times \Sigma_a$. h is defined by $h(u, a) = (f(u), g_u(a))$, where

$$
\begin{aligned}
f &: V \to V \text{ and} \\
g_u &: \Sigma_a \to \Sigma_a \text{ defined for each } u \in V \text{ are bijections s. t.} \\
W_P(f(u), g_u(a), f(v)) &= W_P(u, a, v) \text{ and} \\
W_R(f(u), g_u(a), f(v)) &= W_R(u, a, v) \ \forall\ (u, a, v) \in E
\end{aligned}
$$

These two components of each generator can be interpreted as follows:

1. f is a function that permutes the states/vertices
2. The set of functions $\{g_u\}$ defined for each state/vertex permutes the actions/edge labels. These are called the State-Dependent Action Recoding (SDAR) functions.

Set Bijections

Let us assume, for a moment, that we have a procedure that constructs a weighted digraph $WD_{\mathcal{M}}$ from $G_{\mathcal{M}}$. Now, solving $DGEN(WD_{\mathcal{M}})$ gives the generators of $WD_{\mathcal{M}}$. Even if these were somehow same as the fs we are looking for, we still need a way to find the SDAR functions. To achieve this, we define the notion of a *set bijection* which represents a set of bijections very compactly. In the worst case, for each f, there can be factorially many SDAR functions. So a normal explicit representation cannot be used. We also define the operations of intersection between two *set bijections* to find the bijections that are common to both *set bijections*, composition between two *set bijections* and an inverse of a *set bijection*. All these operations can be done in time polynomial of the number of elements in the domain of a bijection belonging to the *set bijection*.

Definition 7. Consider two finite sets A and B. Let $U_A = \{U_{A_1}, U_{A_2}, \ldots, U_{A_k}\}$ and $U_B = \{U_{B_1}, U_{B_2}, \ldots, U_{B_k}\}$ be partitions of A and B respectively. U_A and U_B are said to be *similar* iff $|U_A| = |U_B|$ and for each $U_{A_i} \in U_A$ there exists a unique $U_{B_j} \in U_B$ such that $|U_{A_i}| = |U_{B_j}|$. We denote it by $U_A \sim U_B$.

Note that, by definition the sets A and B will be of the same size.

Definition 8. Let A and B be two finite sets and $U_A = \{U_{A_1}, U_{A_2}, \ldots, U_{A_k}\}$ and $U_B = \{U_{B_1}, U_{B_2}, \ldots, U_{B_k}\}$ be partitions of A and B respectively such that $U_A \sim U_B$. A bijective map $X : U_A \to U_B$ where $X(U_{A_i}) = U_{B_j}$ implies $|U_{A_i}| = |U_{B_j}|$ for all $U_{A_i} \in U_A$ is called a *set bijection*.

Informally, a *set bijection* can be interpreted as representing a set of bijections from A to B. $X(U_{A_i}) = U_{B_j}$ represents all possible bijective mappings from elements

in U_{A_i} to elements in U_{B_j}. A bijection from A to B in the set of bijections that represent the *set bijection*, can be formed by collating mappings from each $X(U_{A_i}) = U_{B_j}$. The *set bijection* represents all mappings that can be formed by such collations. To formalize this notion, we define the *interpretation function* next.

Let $X_{AB} \triangleq \{$all bijections $X : U_A \to U_B$ such that U_A and U_B are similar partitions of A and B respectively $\}$ be the set of all *set bijections*. Let $2^{S_{|V|}}$ be the powerset set of all permutations from $A \to B$. Define, $\hat{I} : X_{AB} \to 2^{S_{|V|}}$ such that $\hat{I}(X : U_A \to U_B) = \{$ all bijections $l : A \to B \mid l(x \in U_{A_i}) \in X(U_{A_i}) \ \forall U_{A_i} \in U_A\}$. Evidently, $\hat{I}$ is only injective and not surjective as there exist sets of $2^{S_{|V|}}$ that cannot be represented by a *set bijection*. For example, consider the set of bijections, between $\{a, b, c\}$ and $\{1, 2, 3\}$, $L = \{(a \to 1, b \to 2, c \to 3), (a \to 2, b \to 1, c \to 3), (a \to 2, b \to 3, c \to 1)\}$. Clearly there does not exist an $X : U_A \to U_B$ such that $\hat{I}(X) = L$. All we can say is that there exists an X such that $L \subset \hat{I}(X)$. To get a bijective interpretation function, we define, $I : X_{AB} \to image(\hat{I})$ such that $I(X : U_A \to U_B) = \hat{I}(X : U_A \to U_B)$. Clearly I is a bijection and we call this the interpretation function.

Example 1. Consider $A = \{1, 2, 3, 4\}$ and $B = \{N, E, W, S\}$. Let $U_A^1 = \{\{1, 2\}, \{3, 4\}\}$ and $U_B^1 = \{\{N, E\}, \{W, S\}\}$. Consider the following *set bijection*:

$$\begin{aligned} X_1(\{1, 2\}) &= \{N, E\} \\ X_1(\{3, 4\}) &= \{W, S\} \end{aligned}$$

$X_1(\{1,2\}) = \{N, E\}$ represents the following bijections:

$$1 \to N, \quad 2 \to E$$

$$1 \to E, \quad 2 \to N$$

$X_1(\{3,4\}) = \{W, S\}$ represents the following bijections:

$$3 \to W, \quad 4 \to S$$

$$3 \to S, \quad 4 \to W$$

So the *set bijection* X_1 represents the bijections in Table 3.1, i.e., $I(X_1)$ is the set of bijections in Table 3.1.

1 →N,	2 →E,	3 →W,	4 →S
1 →N,	2 →E,	3 →S,	4 →W
1 →E,	2 →N,	3 →W,	4 →S
1 →E,	2 →N,	3 →S,	4 →W

Table 3.1: Bijections in the interpretation, $I(X_1)$, of the *set bijection* of Example 1

Example 2. Consider $A = \{1,2,3,4\}$ and $B = \{N, E, W, S\}$. Let $U_A^2 = \{\{3\}, \{1,2,4\}\}$ and $U_B^2 = \{\{N, E, S\}, \{W\}\}$. Consider the following *set bijection*:

$$X_2(\{3\}) = \{W\}$$

$$X_2(\{1,2,4\}) = \{N, E, S\}$$

$X_2(\{1,2,4\}) = \{N, E, S\}$ represents the bijections in Table 3.1.

So $I(X_2)$ is the set of bijections in Table 3.3.

$1 \rightarrow$N,	$2 \rightarrow$E,	$4 \rightarrow$S
$1 \rightarrow$N,	$2 \rightarrow$S,	$4 \rightarrow$E
$1 \rightarrow$E,	$2 \rightarrow$N,	$4 \rightarrow$S
$1 \rightarrow$E,	$2 \rightarrow$S,	$4 \rightarrow$N
$1 \rightarrow$S,	$2 \rightarrow$N,	$4 \rightarrow$E
$1 \rightarrow$S,	$2 \rightarrow$E,	$4 \rightarrow$N

Table 3.2: Bijections that $X_2(\{1,3,4\}) = \{N, E, S\}$ of Example 2 represents

$1 \rightarrow$N,	$2 \rightarrow$E,	$3 \rightarrow$W,	$4 \rightarrow$S
$1 \rightarrow$N,	$2 \rightarrow$S,	$3 \rightarrow$W,	$4 \rightarrow$E
$1 \rightarrow$E,	$2 \rightarrow$N,	$3 \rightarrow$W,	$4 \rightarrow$S
$1 \rightarrow$E,	$2 \rightarrow$S,	$3 \rightarrow$W,	$4 \rightarrow$N
$1 \rightarrow$S,	$2 \rightarrow$N,	$3 \rightarrow$W,	$4 \rightarrow$E
$1 \rightarrow$S,	$2 \rightarrow$E,	$3 \rightarrow$W,	$4 \rightarrow$N

Table 3.3: Bijections in the interpretation, $I(X_2)$, of the *set bijection* of Example 2

Definition 9. Let A be a finite set and let $U_A^1 = \{U_{A_1}^1, U_{A_2}^1, \ldots, U_{A_k}^1\}$, $U_A^2 = \{U_{A_1}^2, U_{A_2}^2, \ldots, U_{A_k}^2\}$ be two partitions of A such that, $U_A^1 \sim U_A^2$. We define the intersection of two similar partitions of a finite set as $U_A^1 \cap U_A^2 = \{U_{A_i}^1 \cap U_{A_j}^2 \mid U_{A_i}^1 \in U_A^1, U_{A_j}^2 \in U_A^2$ *and* $U_{A_i}^1 \cap U_{A_j}^2 \neq \emptyset\}$.

Definition 10. Let A and B be two finite sets and $U_A^1 = \{U_{A_1}^1, U_{A_2}^1, \ldots, U_{A_k}^1\}$, $U_A^2 = \{U_{A_1}^2, U_{A_2}^2, \ldots, U_{A_k}^2\}$ be two partitions of A and $U_B^1 = \{U_{B_1}^1, U_{B_2}^1, \ldots, U_{B_k}^1\}$, $U_B^2 = \{U_{B_1}^2, U_{B_2}^2, \ldots, U_{B_k}^2\}$ be two partitions of B. Also let $U_A^1 \sim U_B^1$ and $U_A^2 \sim U_B^2$. Let two *set bijections* X_1 and X_2 be defined from U_A^1 to U_B^1 and from U_A^2 to U_B^2 respectively. If $(U_A^1 \cap U_A^2) \sim (U_B^1 \cap U_B^2)$, we define the intersection between the two set bijections $X = X_1 \cap X_2$ as follows: $\forall U_{A_i}^1 \in U_A^1, U_{A_j}^2 \in U_A^2$ such that $U_{A_i}^1 \cap U_{A_j}^2 \neq \emptyset$, $X(U_{A_i}^1 \cap U_{A_j}^2) = X_1(U_{A_i}^1) \cap X_2(U_{A_j}^2)$. Note that, $X : U_A^1 \cap U_A^2 \rightarrow U_B^1 \cap U_B^2$ and it can be shown that $I(X) = I(X_1) \cap I(X_2)$.

Example 3. Consider the *set bijections* X_1 and X_2 of Example 1 and Example 2 respectively. $U_A^1 \cap U_A^2 = \{\{1,2\},\{3\},\{4\}\}$. $U_B^1 \cap U_B^2 = \{\{N,E\},\{W\},\{S\}\}$. Since, $U_A^1 \cap U_A^2$

is similar to $U_B^1 \cap U_B^2$, $X = X_1 \cap X_2 : U_A^1 \cap U_A^2 \rightarrow U_B^1 \cap U_B^2$ is:

$$\begin{aligned} X(\{1,2\} \cap \{1,2,4\}) &= X_1(\{1,2\}) \cap X_2(\{1,2,4\}) \\ X(\{1,2\}) &= \{N,E\} \cap \{N,E,S\} \\ X(\{1,2\}) &= \{N,E\} \end{aligned}$$

Similarly,

$$\begin{aligned} X(\{3\}) &= \{W\} \\ X(\{4\}) &= \{S\} \end{aligned}$$

So, $I(X)$ is the set of bijections in Table 3.4:

1 →N,	2 →E,	3 →W,	4 →S
1 →E,	2 →N,	3 →W,	4 →S

Table 3.4: Bijections $I(X)$ which, as claimed, is the set of bijections common to both $I(X_1)$ and $I(X_2)$.

Definition 11. Let A be a finite set. Let $U_A^1 = \{U_{A_1}^1, U_{A_2}^1, \ldots, U_{A_k}^1\}$, $U_A^2 = \{U_{A_1}^2, U_{A_2}^2, \ldots, U_{A_k}^2\}$ be two *similar* partitions of A. Let X be a *set bijection* defined from U_A^1 to U_A^2. We define the *inverse* of X as $X^{-1} : U_A^2 \rightarrow U_A^1$ such that $X^{-1}(U_{A_i}^2) = U_{A_j}^1$ iff $X(U_{A_j}^1) = U_{A_i}^2$.

Definition 12. Let A, B and C be three finite sets and $U_A = \{U_{A_1}, U_{A_2}, \ldots, U_{A_k}\}$, $U_B = \{U_{B_1}, U_{B_2}, \ldots, U_{B_k}\}$ and $U_C = \{U_{C_1}, U_{C_2}, \ldots, U_{C_k}\}$ be partitions of A, B and C respectively . Also let U_A, U_B and U_C be pairwise similar to each other. Let two *set bijections* X_1 and X_2 be defined from U_B to U_C and U_A to U_B respectively. We define the composition of X_1 and X_2, $X = X_1 \odot X_2$ as the *set bijection* from U_A to U_C defined by $X(U_{A_i}) = X_1(X_2(U_{A_i}))$, for each $U_{A_i} \in U_A$. It can be shown that for each $l \in I(X)$ there exist, $l_1 \in I(X_1)$ and $l_2 \in I(X_2)$ such that $l = l_1 \circ l_2$ where $\circ$ denotes normal function composition.

Example 4. Let $U_A^3 = \{\{N, W\}, \{E, S\}\}$. Consider the *set bijection* X_1 of Example 1 and the *set bijection* $X_2 : U_A^1 \to U_A^3$ defined by,

$$\begin{aligned} X_2(\{1,2\}) &= \{E, S\} \\ X_2(\{3,4\}) &= \{N, W\} \end{aligned}$$

Then X_1^{-1} is given by,

$$\begin{aligned} X_1^{-1}(\{N, E\}) &= \{1, 2\} \\ X_1^{-1}(\{W, S\}) &= \{3, 4\} \end{aligned}$$

If $X = X_2 \odot X_1^{-1}$, then X is

$$\begin{aligned} X(\{N, E\}) &= X_2(X_1^{-1}(\{N, E\})) \\ X(\{N, E\}) &= X_2(\{1, 2\}) \\ X(\{N, E\}) &= \{E, S\} \end{aligned}$$

Similarly,

$$X(\{W, S\}) = \{N, W\}$$

Vector-Weighted Digraph

We assume that Σ_a can be ordered and let O be such an ordering.

Without loss of generality, we can assume that $|E_{uv}| = k, \forall u, v \in V$ because, we can always take $max_{u,v \in V}|E_{uv}| = k$ and if $\exists u, v \in V$ such that $(u, a, v) \in E$ for some $a \in \Sigma_a$ and $|E_{uv}| < k$, then add dummy labels (chosen from the remaining labels in

Σ_a) and zero weights to make $|E_{uv}| = k$. This corresponds to the general assumption in MDPs that $|A_s| = k$, $\forall s \in S$.

Let $< a_1, a_2, \ldots, a_k >$ ordered as per O be the k-tuple representing the label of each edge in E_{uv}. This being the same for all edges, we leave out labeling from the graph definition.

Now we define the vector-weighted digraph corresponding to $\mathcal{M}$, $VWG_{\mathcal{M}} =< V, E', \mathbf{W_P}, \mathbf{W_R} >$, as follows:

$$
\begin{aligned}
E' &= \{(u,v) \mid \exists a \in \Sigma_a \text{ and } (u,a,v) \in E\} \\
\mathbf{W_P} &: E' \rightarrow \mathbb{R}^k \text{ defined by} \\
&\quad \mathbf{W_P}(\mathbf{u},\mathbf{v}) =< W_P(u,a_1,v), \ldots, W_P(u,a_k,v) > \\
\mathbf{W_R} &: E' \rightarrow \mathbb{R}^k \text{ defined by} \\
&\quad \mathbf{W_R}(\mathbf{u},\mathbf{v}) =< W_R(u,a_1,v), \ldots, W_R(u,a_k,v) >
\end{aligned}
$$

where $a_1, a_2, \ldots, a_k$ are ordered as per O.

Sorted Vector-Weighted Digraph

We define the sorted vector-weighted digraph, $SVWG_{\mathcal{M}} =< V, E', \mathbf{W_{P_s}}, \mathbf{W_{R_s}} >$, as follows:

$\mathbf{W_{P_s}}$: $E' \rightarrow \mathbb{R}^k$ defined by

$\mathbf{W_{P_s}}(\mathbf{u}, \mathbf{v}) =< W_P(u, p_{uv}(1), v), \ldots, W_P(u, p_{uv}(k), v) >$

where, $p_{uv} : \mathbb{N}_k \rightarrow \Sigma_a$ such that

$W_P(u, p_{uv}(1), v) \leq W_P(u, p_{uv}(2), v) \leq \ldots \leq W_P(u, p_{uv}(k), v)$

$\mathbf{W_{R_s}}$: $E' \rightarrow \mathbb{R}^k$ defined by

$\mathbf{W_{R_s}}(\mathbf{u}, \mathbf{v}) =< W_R(u, r_{uv}(1), v), \ldots, W_R(u, r_{uv}(k), v) >$

where, $r_{uv} : \mathbb{N}_k \rightarrow \Sigma_a$ such that

$W_R(u, r_{uv}(1), v) \leq W_R(u, r_{uv}(2), v) \leq \ldots \leq W_R(u, r_{uv}(k), v)$

Note that, p_{uv} and r_{uv} are not unique. So we choose them such that the order O is preserved.

***Set Bijections* that sort the vector-weights**

Here we show that there exists a *set bijection* whose interpretation is the set of permutations that sort the vector-weights. Let $\mathbb{N}_k$ be the set of first k natural numbers. Let $D^P_{uv} \triangleq \{$ all permutations $l : \mathbb{N}_k \rightarrow \Sigma_a \mid l$ sorts $\mathbf{W_P}(\mathbf{u}, \mathbf{v})\}$ be defined for each $(u, v) \in E'$. So, $\mathbf{W_{P_s}}(\mathbf{u}, \mathbf{v}) =< W_P(u, l(1), v), \ldots, W_P(u, l(k), v) >$ and $W_P(u, l(1), v) \leq W_P(u, l(2), v) \leq \ldots \leq W_P(u, l(k), v)$. Clearly, $\mathbb{N}_k$ can be partitioned into $U^{uv}_{\mathbb{N}_k} = \{\mathbb{N}^1_k, \mathbb{N}^2_k, \ldots, \mathbb{N}^j_k\}$ such that, $\forall t \in \mathbb{N}^y_k$, $W_P(u, l(t), v)$ has the same value for each $y = 1, 2, \ldots, j$ and if $t \in \mathbb{N}^y_k$ and $t' \in \mathbb{N}^{y+1}_k$ then $W_P(u, l(t), v) < W_P(u, l(t'), v)$. This partition induces a corresponding partition $U^{uv}_{\Sigma_a} = \{\Sigma^1_a, \Sigma^2_a, \ldots, \Sigma^j_a\}$ where $\Sigma^i_a = \{l(t) | t \in \mathbb{N}^i_k\}$. Since, each l sorts $\mathbf{W_P}(\mathbf{u}, \mathbf{v})$, they satisfy the property that $l(x \in \mathbb{N}^i_k) \in \Sigma^i_a$. Therefore, there exists a set bijection $Q^P_{uv} : U^{uv}_{\mathbb{N}_k} \rightarrow U^{uv}_{\Sigma_a}$ such that, $I(Q^P_{uv}) = D^P_{uv}$.

Using a similar procedure, we can show that there exists set bijection Q^R_{uv} : $U^{uv}_{\mathbb{N}_k} \rightarrow U^{uv}_{\Sigma_a}$ whose interpretation is the set of permutations that sort $\mathbf{W_R}(\mathbf{u}, \mathbf{v})$.

Let $Q_{uv} = Q^P_{uv} \cap Q^R_{uv}$. If $Q_{uv} = \emptyset$, then there doesn't exist an automorphism for the MDP $\mathcal{M}$.

Weighted Digraph

Now we define the weighted digraph $WG_{\mathcal{M}} =< V, E', W' >$ as follows:

$$W' \quad : \quad E' \rightarrow \mathbb{R} \text{ such that } W'(u,v) = m(\mathbf{W_{P_s}}(u,v).\mathbf{W_{R_s}}(u,v))$$

where m is a bijection from $\mathbb{R}^{2k} \rightarrow \mathbb{R}$

and . denotes concatenation

Construction

The procedure for finding symmetries of an MDP $\mathcal{M}$ is given in Algorithm 3.

The complexity of the algorithm is as follows. The construction steps in lines 3 to 5, are at most polynomial in $|E|$. Using a constant access time data structure like a hash-table, Q^P_{uv} and Q^R_{uv} can be constructed in $O(|E_{uv}|)$ time. The intersection takes $O(|E_{uv}|^2)$ time. Since this runs for $|E'|$ iterations, computation of Q_{uv} is at most polynomial in $|E|$. Since m is known, the construction of weighted digraph in line 13, is polynomial in $|E|$. With the use of procedures that return at most $|V|$ automorphisms of $AutWG_{\mathcal{M}}$ [Mathon, 1979], the construction of G_u for each f, from lines 15 to 26, runs for at most $|V|$ iterations. The most expensive part of the loop from lines 20 to 26 is the computation of $|V|^2$ intersections. But this is

Algorithm 3 Construction

Given $\mathcal{M} = \langle S, A, \Psi, P, R \rangle$
Let *SOLN* be an empty set
Construct the pseudograph $G_{\mathcal{M}} =< \Sigma_a, V, E, W_P, W_R >$ as defined on page 25
Construct the vector-weighted digraph $VWG_{\mathcal{M}} =< V, E', \mathbf{W_P}, \mathbf{W_R} >$ as defined on page 31
Construct the sorted vector-weighted digraph $SVWG_{\mathcal{M}} =< V, E', \mathbf{W_{P_s}}, \mathbf{W_{R_s}} >$ as defined on page 32
for each $(u, v) \in E'$ **do**
    Compute Q^P_{uv} and Q^R_{uv} by finding the partition of $\mathbb{N}_k$ as described on page 33
    $Q_{uv} \leftarrow Q^P_{uv} \cap Q^R_{uv}$
    if $Q^P_{uv} \cap Q^R_{uv}$ does not exist **then**
        exit
    end if
end for
Construct the weighted digraph $WG_{\mathcal{M}} =< V, E', W' >$ using m as described on page 34
$F \leftarrow DGEN(WG_{\mathcal{M}})$ where F is the set of generators of $AutWG_{\mathcal{M}}$
for each $f \in F$ **do**
    for each $(u, v) \in E'$ **do**
        $G_{uv} \leftarrow Q_{f(u)f(v)} \odot Q^{-1}_{uv}$
    end for
    Let $\hat{H}^f$ be an empty set
    for each $u \in V$ **do**
        $G_u \leftarrow G_{uv}$ from some $v \in V$
        for each $v \in V$ **do**
            $G_u \leftarrow G_u \cap G_{uv}$
        end for
        Add G_u to $\hat{H}^f$
    end for
    Add $< f, \hat{H}^f >$ to *SOLN*
end for

still polynomial in $|V||E|$ time. Hence the algorithm takes polynomially more time than the solution time of *DGEN*. Also to extract a solution from *SOLN*, we need to extract $|V|$ SDAR functions from $\hat{H}^f$ for each f, which takes $|E_{uv}|$ time if we use a constant access time data structure. So extraction of a solution takes $O(|V|^2|E|)$ which is still polynomial in $|V||E|$. Next we prove the correctness of the algorithm.

Correctness of construction

Let f belong to $AutWG_{\mathcal{M}}$.

Let $u, v \in V$ be such that $(u, v) \in E'$

Let $G_{uv} \triangleq Q_{f(u)f(v)} \odot Q_{uv}^{-1}$

Let $G_u \triangleq \bigcap_{v \in V} G_{uv}$

If the intersection exists, define,

$$\chi_{\mathcal{M}} = \{ \text{ all functions } l : V \rightarrow \bigcup_{u \in V} I(G_u) \mid l(u) \in I(G_u) \}$$

Else, the properties of *set bijection* intersection ensure that

there does not exist a $< f, \{g_u\} > \in Aut\mathcal{M}$

Note that, $\chi_{\mathcal{M}} \triangleq \prod_{u \in V} I(G_u)$

Let $\overline{AutWG_{\mathcal{M}}} \triangleq \{f \in AutWG_{\mathcal{M}} \mid < f, \{g_u\} > \notin Aut\mathcal{M} \text{ for any } \{g_u\}\}$

For each l let $g_u^l \triangleq l(u) \ \forall u \in V$

Since $l(u) \in I(G_u)$ $l(u) : \Sigma_a \rightarrow \Sigma_a$

Lemma 1. The set $H^f = \{< f, \{g_u^l\} >,\ \forall l \in \chi_{\mathcal{M}}\}$ is a set of automorphisms of $Aut\mathcal{M}$.

Proof.

$$
\begin{aligned}
W'(f(u), f(v)) &= W'(u, v) \\
\implies m^{-1}(W'(f(u), f(v))) &= m^{-1}(W'(u, v)) \\
\implies \mathbf{W_{P_s}}(f(u), f(v)).\mathbf{W_{R_s}}(f(u), f(v)) &= \mathbf{W_{P_s}}(\mathbf{u}, \mathbf{v}).\mathbf{W_{R_s}}(\mathbf{u}, \mathbf{v}) \ \because \text{ m is injective} \\
\implies W_P(f(u), q_{f(u)f(v)}(i), f(v)) &= W_P(u, q_{uv}(i), v)\ \forall\ i = 1, \ldots, k \text{ and} \\
W_R(f(u), q_{f(u)f(v)}(i), f(v)) &= W_R(u, q_{uv}(i), v)\ \forall\ i = 1, \ldots, k \\
&\quad \text{where } q_{uv} \in I(Q_{uv}) \text{ and} \\
&\quad q_{f(u)f(v)} \in I(Q_{f(u)f(v)}) \qquad (3.1)
\end{aligned}
$$

$$
\begin{aligned}
\text{Also, } Q_{f(u)f(v)} &: U_{\mathbb{N}_k}^{f(u)f(v)} \to U_{\Sigma_a}^{f(u)f(v)}, \\
Q_{uv}^{-1} &: U_{\Sigma_a}^{uv} \to U_{\mathbb{N}_k}^{uv} \text{ and} \\
U_{\mathbb{N}_k}^{f(u)f(v)} &= U_{\mathbb{N}_k}^{uv} \text{ (from 3.1)} \\
\text{Hence, } G_{uv} &= Q_{f(u)f(v)} \odot Q_{uv}^{-1} \text{ is well defined} \\
\text{and } Q_{f(u)f(v)} &= G_{uv} \odot Q_{uv}
\end{aligned}
$$

$$
\begin{aligned}
\forall g_{uv} \in I(G_{uv}),\ q_{f(u)f(v)}(i) &= g_{uv}(q_{uv}(i))\ \forall\ i = 1, \ldots, k, \\
&\quad \text{For some, } q_{f(u)f(v)} \in I(Q_{f(u)f(v)}) \\
&\quad \text{and } q_{uv} \in I(Q_{uv}) \\
\therefore W_P(f(u), g_{uv}(a), f(v)) &= W_P(u, a, v),\ \forall\ (u, a, v) \in E \text{ and} \\
W_R(f(u), g_{uv}(a), f(v)) &= W_R(u, a, v),\ \forall\ (u, a, v) \in E
\end{aligned}
$$

$$\text{If } \bigcap_{v\in V} U_{\Sigma_a}^{uv} \nsim \bigcap_{f(v)\in V} U_{\Sigma_a}^{f(u)f(v)}$$

There does not exist an automorphism

corresponding to f in $Aut\mathcal{M}$

$$\begin{aligned}
\because G_u &= \bigcap_{v\in V} G_{uv} \\
g_u &: \Sigma_a \to \Sigma_a \qquad (3.2) \\
\therefore W_P(f(u), g_u(a), f(v)) &= W_P(u,a,v),\ \forall\ (u,a,v) \in E \text{ and} \\
W_R(f(u), g_u(a), f(v)) &= W_R(u,a,v),\ \forall\ (u,a,v) \in E \qquad (3.3) \\
f &: V \to V \qquad (3.4)
\end{aligned}$$

From equations 3.2- 3.4, it is clear that, $< f, \{g_u\} >$ where $g_u \in I(G_u)$, is an automorphism of $\mathcal{M}$. Since, u is arbitrary, we can pick from any G_u and hence their cartesian product should be considered. So, it is quite evident that $H^f = \{< f, \{g_u^l\} >,\ | \ l \in \chi_{\mathcal{M}}\}$ is a set of automorphisms of $Aut\mathcal{M}$. □

Lemma 2.

If $< f, \{g_u\} > \in Aut\mathcal{M}$, $< f, \{g_u\} > \in H^f$

Proof.

$$\begin{aligned}
f &: V \to V \\
g_u &: \Sigma_a \to \Sigma_a \\
W_P(f(u), g_u(a), f(v)) &= W_P(u,a,v),\ \forall\ (u,a,v) \in E \text{ and} \\
W_R(f(u), g_u(a), f(v)) &= W_R(u,a,v),\ \forall\ (u,a,v) \in E
\end{aligned}$$

$$
\begin{aligned}
\text{Let } q_{f(u)f(v)} &\in I(Q_{f(u)f(v)}) \text{ for some } v \in V \\
\text{Let } q_u &= g_u^{-1} \odot q_{f(u)f(v)} \\
\because q_u &: \mathbb{N}_k \to \Sigma_a \text{ and is a bijection} \\
W_P(f(u), g_u(q_u(i)), f(v)) &= W_P(u, q_u(i), v) \; \forall \; i = 1, \ldots, k \text{ and} \\
W_R(f(u), g_u(q_u(i)), f(v)) &= W_R(u, q_u(i), v) \; \forall \; i = 1, \ldots, k
\end{aligned}
$$

$$
\begin{aligned}
\because \; g_u(q_u(i)) &= q_{f(u)f(v)}(i) \; \forall \; i = 1, \ldots, k \\
W_P(f(u), q_{f(u)f(v)}(i), f(v)) &= W_P(u, q_u(i), v) \; \forall \; i = 1, \ldots, k \text{ and} \\
W_R(f(u), q_{f(u)f(v)}(i), f(v)) &= W_R(u, q_u(i), v) \; \forall \; i = 1, \ldots, k \qquad (3.5) \\
\text{If } i &\leq j \text{ then} \\
W_P(f(u), q_{f(u)f(v)}(i), f(v)) &\leq W_P(f(u), q_{f(u)f(v)}(j), f(v)) \text{ and} \\
W_R(f(u), q_{f(u)f(v)}(i), f(v)) &\leq W_R(f(u), q_{f(u)f(v)}(j), f(v)) \\
&\quad \because q_{f(u)f(v)} \in I(Q_{f(u)f(v)})
\end{aligned}
$$

From Equation 3.5 we have

$$
\begin{aligned}
\text{If } i &\leq j \text{ then} \\
W_P(u, q_u(i), v) &\leq W_P(u, q_u(j), v) \text{ and} \\
W_P(u, q_u(i), v) &\leq W_P(u, q_u(j), v) \\
\because \; v \text{ is arbitrary} \quad q_u &\in I(Q_{uv}) \; \forall v \in V \\
\implies g_u &\in I(G_u) \qquad (3.6)
\end{aligned}
$$

From Equation 3.5 we also have

$$
W'(f(u), f(v)) = W'(u, v) \quad \because m \text{ is bijective} \qquad (3.7)
$$

From equations 3.6 and 3.7 we have that $< f, \{g_u\} > \in H^f$. □

Corollary 1.

From lemmas 1 and 2, we have

$$\bigcup_{f \in AutWG_{\mathcal{M}} - \overline{AutWG_{\mathcal{M}}}} H^f \subseteq Aut\mathcal{M} \text{ and } Aut\mathcal{M} \subseteq \bigcup_{f \in AutWG_{\mathcal{M}} - \overline{AutWG_{\mathcal{M}}}} H^f$$

$$\implies Aut\mathcal{M} = \bigcup_{f \in AutWG_{\mathcal{M}} - \overline{AutWG_{\mathcal{M}}}} H^f$$

Lemma 3. Let $\phi : Aut\mathcal{M} \to AutWG$ be defined by $\phi(< f, \{g_u\} >) = f$. Then ϕ is a group homomorphism.

Proof.

$$\begin{aligned} \phi(< f_1, \{g_u^{f_1}\} > . < f_2, \{g_u^{f_2}\} >) &= \phi(< f_1.f_2, \{g_{f_2(u)}^{f_1} . g_u^{f_2}\} >) \\ &= f_1.f_2 \\ &= \phi(< f_1, \{g_u^{f_1}\} >).\phi(< f_2, \{g_u^{f_2}\} >) \quad \square \end{aligned}$$

Corollary 2. $Aut\mathcal{M}/ker(\phi) \cong im(\phi)$

Proof. It is a basic result from Group Theory that if $\phi : G_1 \to G_2$ is a group homomorphism, then $ker(\phi)$ is a normal subgroup and the quotient group, $G_1/ker(\phi)$ is isomorphic to $im(\phi)$ which is a subgroup of G_2. $\square$

Lemma 4. $Aut\mathcal{M}$ partitioned as per Corollary 1 is the set of all left cosets of the kernel of ϕ, $ker(\phi)$.

Proof. By definition of ϕ, it is easy to see that the set of automorphisms in H^e, where e is the identity permutation, forms $ker(\phi)$. So we need to prove that $H^f.H^e = H^f$

for some arbitrary f. Since $ker(\phi)$ has the identity element, $< f, \{g_u^l\} > \in H^f \implies < f, \{g_u^l\} > \in H^f.H^e$. So we only need to prove that $< f, \{g_u^l\} > \in H^f.H^e \implies < f, \{g_u^l\} > \in H^f$.

$$\begin{aligned} < f, \{g_u^l\} > .ker(\phi) &= < f, \{g_u^l\} > .H^e \\ &= \{< f.e, \{g_{e(u)}^l . g_u^{l'}\} > \forall g_u^{l'} \mid < e, \{g_u^{l'}\} > \in H^e\} \\ &= \{< f, \{g_u\} >, g_u = g_u^l . g_u^{l'}, \forall g_u^{l'} \mid < e, \{g_u^{l'}\} > \in H^e\} \end{aligned}$$

Since $< f, \{g_u\} >$ is an automorphism, from Lemma 2, $< f, \{g_u\} > \in H^f$. Hence, the result. □

Lemma 5.

Let $F \triangleq \{f_1, f_2, \ldots, f_n\}$ be the generators of $AutWG_{\mathcal{M}}$.

So $AutWG_{\mathcal{M}} = \{f_1^{\alpha_1} f_2^{\alpha_2} \ldots f_m^{\alpha_m} \mid$ for each $i, f_i \in F, \alpha_i \in \mathbb{Z}, f_i \neq f_{i+1}$ and $m \in \mathbb{Z}^+\}$

Let $\bar{F} \triangleq F \cap \overline{AutWG_{\mathcal{M}}}$

Then $Aut\mathcal{M} = \ll \bigcup_{f \in F - \bar{F}} H^f \gg$

where, $\ll \{a, b, c\} \gg$ stands for the group generated by the set $\{a, b, c\}$.

Proof.

$$\begin{aligned} f \in \ll F - \bar{F} \gg &\implies f \in im(\phi) \\ \because f_i \in F - \bar{F} &\implies f_i \in im(\phi) \text{ by defn.} \\ &\implies \prod_{i=1}^{m} f_i^{\alpha_i} \in im(\phi) \text{ as } im(\phi) \text{ is a subgroup of } AutWG_{\mathcal{M}} \\ \therefore \ll F - \bar{F} \gg &\subseteq im(\phi) \end{aligned} \tag{3.8}$$

$$
\begin{aligned}
f \notin \ll F - \bar{F} \gg \quad &\implies \quad \exists i \text{ such that } f_i \in \bar{F}, \alpha_i \neq 0 \text{ and } \prod_{i=1}^{m} f_i^{\alpha_i} = f \\
&\implies \quad f \notin im(\phi) \\
&\quad \because (f \notin im(\phi)) \implies fg \notin im(\phi) \ \forall g \in im(\phi) \\
\therefore f \in im(\phi) \quad &\implies \quad f \in \ll F - \bar{F} \gg \\
\text{Hence, } im(\phi) \quad &\subseteq \quad \ll F - \bar{F} \gg
\end{aligned} \tag{3.9}
$$

From equations 3.8 and 3.9, we have,

$$
\begin{aligned}
im(\phi) &= \ll F - \bar{F} \gg \\
Aut\mathcal{M}/ker(\phi) &= \ll \{\pi^{-1}(f) \mid f \in F - \bar{F}\} \gg \\
&\quad \text{where } \pi \text{ is the isomorphism} \\
&\quad \text{from } Aut\mathcal{M}/ker(\phi) \text{ to } im(\phi) \\
Aut\mathcal{M}/ker(\phi) &= \ll \bigcup_{f \in F - \bar{F}} H^f \gg \\
Aut\mathcal{M} &= \ll \bigcup_{f \in F - \bar{F}} H^f \gg \text{ by Corollary 1} \quad \square
\end{aligned}
$$

Since $\hat{H}^f$, found by the algorithm, can be interpreted as representing H^f, the algorithm finds the generators and as discussed takes time polynomial in $|V||E|$. Hence $AMGEN(\mathcal{M})$ is *Isomorphism Complete*.

3.1.3 Significance

The above result is significant both theoretically and practically. Practically speaking, the reduction to Graph Isomorphism allows us to use any of the numerous off-the-shelf Graph Isomorphism solvers to find symmetries on MDPs.

In fact, we use NAutY - No Automorphisms, Yes?, the best Graph Isomorphism solver currently available [Skiena, 1997] to find out symmetries in MDPs. NAutY solves $AGEN(G)$. It uses backtracking and a refinement procedure to find the canonical labeling. If two different labelings lead to the same graph, then an automorphism can be found using these labelings [McKay, 1981]. In the worst case it can take exponential time. So it allows the use of a variety of vertex invariants, which act like heuristics, to solve harder problems. However, for random graphs with n vertices and edge probability 0.5, average execution times for large n are about n^2 nanosecs.

We use NAutY in the fourteenth line in the construction, where we need to solve $DGEN(G)$. We first convert the weighted digraph into an unweighted digraph using standard procedure. We then use NAutY to find the generators of the automorphism group of the so found digraph. From these we extract generators of $AutWG$ as per the above procedure. We present some results in Section 3.3.

3.2 Exploiting Symmetries

We address the second part of our problem here. We suggest a way to exploit abstractions resulting from symmetries. We make the assumption that the abstractions are known before applying any of these ideas. This assumption is reasonable because:

1. In a large family of tasks, abstractions are known beforehand or can be specified by the designer through a superficial examination of the problem.
2. We can always find them using the approach outlined in Section 3.1. However, it might need some tuning in order to find the right vertex invariant.

However, even with the availability of such information, it is not a trivial task to efficiently use them. For e.g., to utilize the symmetry information presented as the *symmetry group*, $\mathcal{G}$ of an MDP, a straightforward way by explicit enumeration, takes time proportional to $|\Psi| \times |\mathcal{G}|$.

3.2.1 $\mathcal{G}$ -Reduced Image Algorithm

We use an efficient incremental algorithm, proposed in [Ravindran, 2004], for building the reduced MDP given a symmetry group or subgroup. This is an adaptation of an algorithm proposed by Emerson and Sistla [1996] for constructing reduced models for concurrent systems. The algorithm is presented in Algorithm 4 for completeness.

The algorithm does a breadth-first enumeration of states skipping states and state-action pairs that are equivalent to those already visited. On encountering a state-action pair not equivalent to one already visited, it examines the states reachable from it to compute the image MDP parameters. Infinite paths are detected by maintaining already visited nodes.

For example, consider the MDP shown in Figure 3.1. It is easy to see that the MDP exhibits rotation symmetry. Hence the Symmetry Group $\mathcal{G}$ consists of $\{f_1(1) = 2, f_1(2) = 3, f_1(3) = 1, \{g_s(A_1) = A_1, g_s(A_2) = A_2, \forall s \in S\}\}$, $\{f_2(1) = 3, f_2(2) = 1, f_2(3) = 2, \{g_s(A_1) = A_1, g_s(A_2) = A_2, \forall s \in S\}\}$, $\{f_3(1) = 1, f_3(2) = 2, f_3(3) = 3, \{g_s(A_1) = A_1, g_s(A_2) = A_2, \forall s \in S\}\}$.

Algorithm 4 Incremental algorithm for constructing the $\mathcal{G}$ -reduced image given MDP $\mathcal{M}$ and some $\mathcal{G} \leq Aut\mathcal{M}$.

Given $\mathcal{M} = \langle S, A, \Psi, P, R \rangle$ and $\mathcal{G} \leq \text{Aut}\mathcal{M}$,
Construct $\mathcal{M}/B_{\mathcal{G}} = \langle S', A', \Psi', P', R' \rangle$.
Set Que to some initial state $\{s_0\}$, $S' \leftarrow \{s_0\}$
while Que is non-empty **do**
  s = dequeue$\{Que\}$
  for each $a \in A_s$ **do**
    if $(s, a) \not\equiv_{\mathcal{G}} (s', a')$ for any $(s', a') \in \Psi'$, **then**
      $\Psi' \leftarrow \Psi' \cup (s, a)$
      $A' \leftarrow A' \cup a$
      $R'(s, a) = R(s, a)$
      for each $t \in S$ such that $P(s, a, t) > 0$ **do**
        if $t \equiv_{\mathcal{G}|S} s'$, for some $s' \in S'$, **then**
          $P'(s, a, s') \leftarrow P'(s, a, s') + P(s, a, t)$
        else
          $S' \leftarrow S' \cup t$
          $P'(s, a, t) = P(s, a, t)$
          add t to Que
        end if
      end for
    end if
  end for
end while

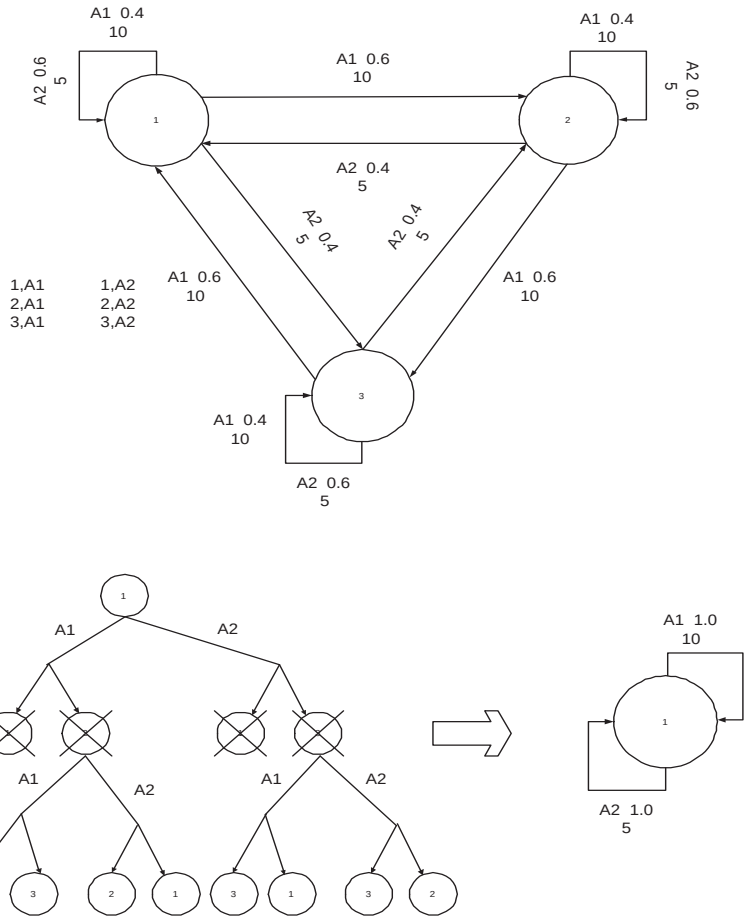


Figure 3.1: An example MDP and the tree derived due to a breadth first enumeration of states of the MDP shown above with the crosses indicating the pruning of the branch and the reduced MDP got by using the $\mathcal{G}$ - Reduced Image Algorithm in Algorithm 4

The following is a description of how the algorithm works on this MDP.

1. State 1 is added to the queue and to S'.
2. State 1 is dequeued from the queue and state-action pair $(1, A_1)$ is added to Ψ'.
3. Action A_1 is added to A'.
4. $R'(1, A_1) = 10$.
5. States 1 and 2 are reachable by taking action A_1.
6. As both 1 and 2 are equivalent to 1, $P'(1, A_1, 1) = 0.4 + 0.6$ and nothing is added to the queue.
7. Now state-action pair $(1, A_2)$ is considered.
8. Action A_2 is added to A'.
9. $R'(1, A_2) = 5$.
10. States 1 and 3 are reachable by taking action A_2.
11. As both 1 and 3 are equivalent to 1, $P'(1, A_2, 1) = 0.6 + 0.4$ and nothing is added to the queue.
12. The algorithm stops because all actions have been exhausted and the queue is empty.

As a result we have,

$$\mathcal{M}' =< \{1\}, \{A_1, A_2\}, \{(1, A_1), (1, A_2)\}, P'(1, A_1, 1) = 1.0, P'(1, A_2, 1) = 1.0,$$
$$R'(1, A_1) = 10, R'(1, A_2) = 5 >$$

as the reduced MDP, which is indicated in Figure 3.1.

From Lemma 6, it is evident that the algorithm terminates when at least one representative from each equivalence class of $\mathcal{G}$ has been examined.

Lemma 6. If the branch rooted at (s', a') is pruned because it is equivalent to (s, a), then states and state-action pairs occurring in the pruned branch are equivalent to that occurring in the branch rooted at (s, a).

Proof.

$$
\begin{aligned}
s \equiv_{\mathcal{G}|S} s' \quad &\implies \quad \exists \text{ an automorphism } \; < f, \{g_s\} > \; \text{ such that } \; f(s) = s' \\
&\implies \quad P(f(s), g_s(a), f(t)) = P(s', g_s(a), f(t)) \;\; \forall \, t \in S \\
&\implies \quad P(s, a, t) = P(s', g_s(a), f(t)) \;\; \forall \, t \in S \qquad (3.10) \\
&\qquad\qquad \text{by definition of automorphisms}
\end{aligned}
$$

$$
\begin{aligned}
(s, a) \equiv_{\mathcal{G}} (s', a') \quad &\implies \quad \exists \text{ an automorphism } \; < f, \{g_s\} > \\
&\qquad\qquad \text{such that } \; f(s) = s' \; \text{ and } \; g_s(a) = a' \\
&\implies \quad P(f(s), g_s(a), f(t)) = P(s', a', f(t)) \;\; \forall \, t \in S \\
&\implies \quad P(s, a, t) = P(s', a', f(t)) \;\; \forall \, t \in S \qquad (3.11) \\
&\qquad\qquad \text{by definition of automorphisms}
\end{aligned}
$$

From Equation 3.10, it is clear that if two states are equivalent then there will be equivalent actions to choose from and from Equation 3.11 that if two state-action pairs are equivalent then there exist equivalent states to which, they transition to.

Since the states are arbitrary, by induction the lemma holds. □

Lemma 7. The transition probabilities computed for the reduced model are correct.

Proof. Let $B_{\mathcal{G}}$ be the partition induced by $\mathcal{G}$. Let $\rho(s,a) = \{t \in S \mid P(s,a,t) > 0\}$ denote the set of states reachable from state s by taking action a.

Since, there exists a homomorphism from $\mathcal{M} \to \mathcal{M}/B_{\mathcal{G}}$, for some arbitrary $[s,a]_{B_{\mathcal{G}}}$ and $[s']_{B_{\mathcal{G}|S}}$,

$$\begin{aligned} P'([s,a]_{B_{\mathcal{G}}}, [s']_{B_{\mathcal{G}|S}}) &= \sum_{s'' \in [s']_{B_{\mathcal{G}|S}}} P(s,a,s'') \\ &= \sum_{s'' \in ([s']_{B_{\mathcal{G}|S}} \cap \rho(s,a))} P(s,a,s'') \end{aligned}$$

If we assume that S and Ψ can be ordered and that a block written $[s']_{B_{\mathcal{G}|S}}$ means that s' occurs before all other $s'' \in [s']_{B_{\mathcal{G}|S}}$, then it can be seen that lines 11 to 19 compute

$$P'(s,a,s') = \sum_{s'' \in ([s']_{B_{\mathcal{G}|S}} \cap \rho(s,a))} P(s,a,s'')$$

and represents $P'([s,a]_{B_{\mathcal{G}}}, [s']_{B_{\mathcal{G}|S}})$.

Since, $[s,a]_{B_{\mathcal{G}}}$ and $[s']_{B_{\mathcal{G}|S}}$ are arbitrary, from Lemma 6 all $(s,a) \in \Psi'$ are covered and line 11 makes sure that no $s' \in S$ is left out, the lemma holds. □

From Lemma 7 it is clear that the transition probabilities actually represent those for the reduced image. The algorithm as presented assumes that all states are reachable from the initial state. It is easy, however, to modify the algorithm suitably. Assuming an explicit representation for the symmetry group and that table look-up takes constant time, the algorithm will run in time proportional to

$|\Psi'|.max_{(s,a)\in\Psi}|\rho(s,a)|.|\mathcal{G}|$. In the worst case, this can be as large as $|\Psi'|.|S|.|\mathcal{G}|$. However, most real-world domains exhibit considerable sparseness in their transition matrices and most times, for a particular domain, $max_{(s,a)\in\Psi}|\rho(s,a)|$ is a constant and the algorithm takes $O(|\Psi'|.|\mathcal{G}|)$ time. Also as $\mathcal{G}$ is just a subgroup, the algorithm can work with whatever little symmetry information the designer might have.

For an end-to-end approach to abstraction using symmetries, we can first find the symmetries using the method outlined in Section 3.1. Then, we can use the symmetry group so found to construct the reduced model and follow the explicit model minimization approach. We present some results next.

3.3 Results

The experiments were run on the following two domains. We describe results per domain.

3.3.1 Probabilistic GridWorld

The domain is an $N \times N$ GridWorld with four probabilistic actions of going UP, DOWN, RIGHT and LEFT having a 90% success probability. The initial state was (0,0) and the goal states were $\{(0, N-1), (N-1, 0)\}$. We used Algorithm 3 to find the symmetries with NAutY being used as the *DGEN* solver. We then used the symmetries to find the partition of Ψ. We were able to find the partition corresponding to the symmetry group, that is, for a grid of size $M \times N$, states (x,y), (y,x), (M-1-x,N-1-y) and (N-1,M-1-x) are equivalent. We present the time taken by the algorithm for GridWorlds of different sizes.

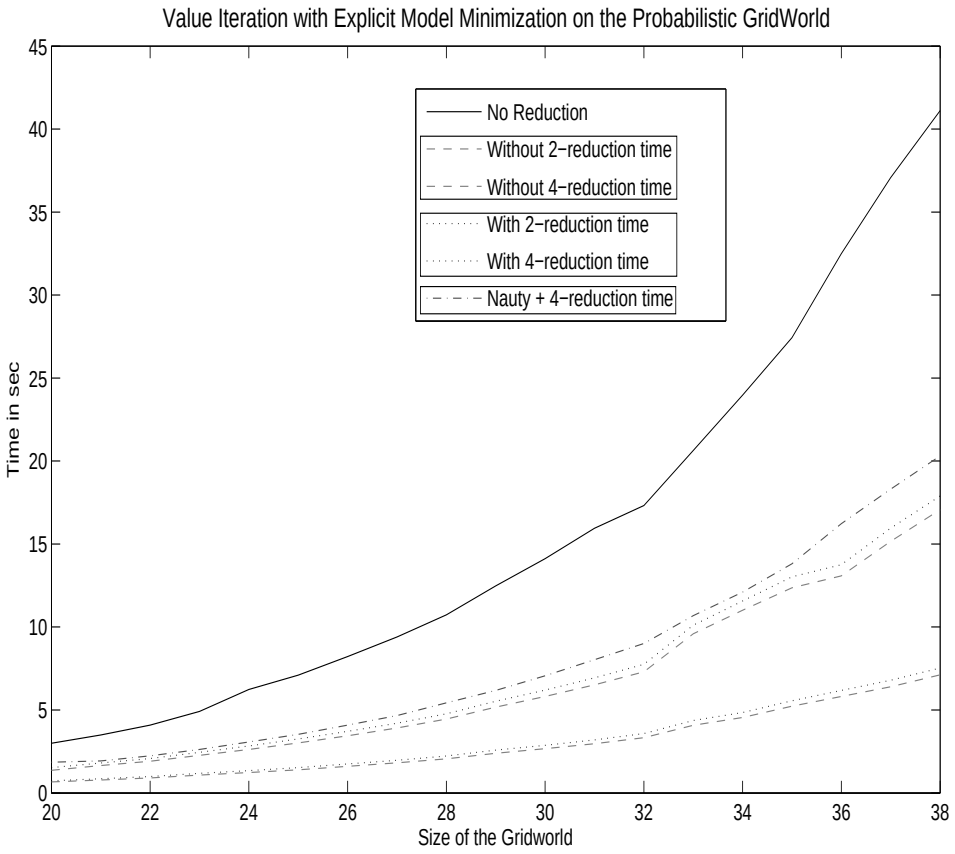


Figure 3.2: Average running times of the value iteration algorithm with explicit model minimization on Probabilistic GridWorld vs size of the GridWorld. Each of the 3 sets should be compared with the graph for no reduction. Curves in a set represent different degrees of symmetry. Each set shows the time reduction with reduced model usage. First one discounts the time taken to find symmetries and for reduction. The next set includes the time for reduction but discounts time taken to find symmetries. The last one includes both the time taken to find symmetries using NAutY and time for reduction.

To complete the end-to-end approach, we ran the $\mathcal{G}$-reduced image algorithm, presented in Section 3.2.1, to find the reduced image and ran the Value Iteration algorithm, presented in Section 2.1.2, on the reduced image. To show the efficiency of reduction, we show the time taken for reduction and solution separately. We also present the case of a handcrafted 2-folded symmetry which is used with the $\mathcal{G}$-reduced image algorithm and reduced model is used with Value Iteration.

From Figure 3.2 it is evident that the reduced model construction is efficient and adds little overhead. However, the results of the end-to-end approach show a significant overhead due to symmetry finding. It cuts the saving by almost half. Still the results are significant because they double the size of the largest GridWorld that can be solved in some given time.

3.3.2 GridWorld Soccer

The domain is a soccer-inspired grid domain. It is a slightly modified version of that described in [Bowling, 2003]. We first describe the original version of the domain and then state the modification.

It is basically an $M \times N$ grid with two agents. One is denoted the attacker (**A**) who holds the ball and the other as the defender (**B**) who tries to snatch the ball from the attacker. The center lines/grids(depending on whether M and N are even or odd) for both x-axis and y-axis are chosen naturally. The state is defined by the non-identical positions of the attacker and the defender. This defines the state space with $(MN)^2 - (MN)$ states. The actions are movements in the four compass directions: **N**, **E**, **W**, **S** and the hold action **H**. It is a single player game, in that, only the attacker chooses actions deliberately while the defender executes random actions. The action chosen by the attacker and the random action of the defender are executed in random order, which determines the next state. However if the defender tries to move into the attacker's location then the state is unchanged and if the attacker tries to move into the defender's location, the game is reset to the initial state which is shown in Figure 3.3. The right hand section of the grid is the attacker's half and the left hand section that of the defender. The goal is chosen

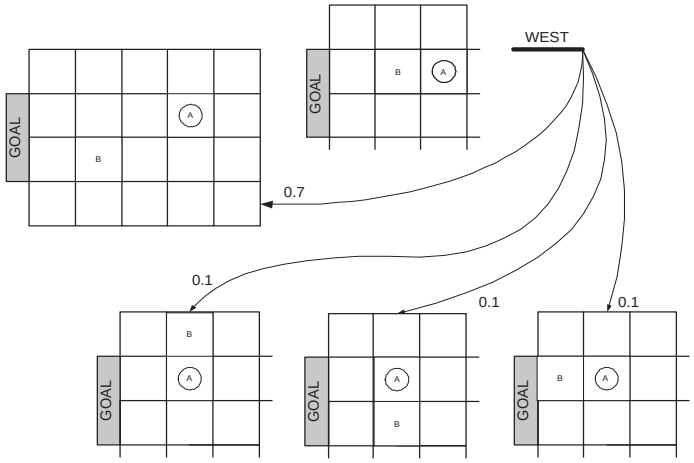


Figure 3.3: Single Player grid soccer where agent **B** selects it actions randomly. The initial state is shown on the left and an example of transitions and associated probabilities are given for a particular state and action on the right. Notice that fifty percent of the time **A**'s actions are executed first causing it to lose the ball and the game reset to the initial state. In addition, if **B** selects **H** or **E** it does not move and so **A** still loses the ball and returns to the initial state. The other outcomes are equiprobable.

to be situated beyond the first column of grids occupying one grid on each side of the y-axis central line/grid. A **W** action from the squares in front of the goal state leads to a goal with a reward of 1 and to the end of an episode. Everywhere else the reward is 0. A 5×4 domain is shown in Figure 3.3.

Intuitively, the domain seems symmetric around the y-axis center line. However, the results of using Algorithm 3 on this domain showed us that the domain is not symmetric due to the existence of the reset action when the attacker tries to move into the defender's position. So we modified the domain to have symmetric reset, that is, reset happens to the initial state and its symmetric state around the y-axis center line with equal probability. This makes the domain symmetric as per intuition, which the algorithm confirms.

Interestingly, the algorithm also finds that the existence of the hold action adds further symmetry. The grids along the border of the domain act as walls.

For example, the northern wall stops the **N** action leaving the state unchanged which is the same result if the agent were to execute a **H** action. These additional symmetries which we did not think of before running algorithm were found by the algorithm. This suggests that there might exist complicated symmetries that will be discovered by the algorithm, which are hard to find, even upon a close examination. Also in many cases, symmetries are size invariant. So we can use the algorithm on a relatively smaller version of the domain and find symmetries which might still hold on the larger version.

We present the time taken by the algorithm for different sizes. An increment of one here means an increase of one along both axes. The presence of two agents, blows up the state space very rapidly and we hit the limit on the order of the graph imposed by NAutY very soon (for a 11×10 grid).

To present similar graphs as in the probabilistic GridWorld case, we use the explicit model minimization approach with Value Iteration. The results are presented in Figure 3.4.

In this case, we find that the overheads due to the construction and the $\mathcal{G}$-reduced image algorithm is negligible. Though efficiency of the $\mathcal{G}$-reduced image algorithm is expected, the efficiency of the construction can be possibly because of the structure of the domain yielding an easy graph to find automorphisms on.

In this chapter, we have provided a constructive proof for the *Isomorphism Completeness* of the problem of finding symmetries. We have also proposed the use of this constructive proof along with an efficient minimization algorithm to solve an MDP using symmetries. We have also demonstrated the approach empirically.

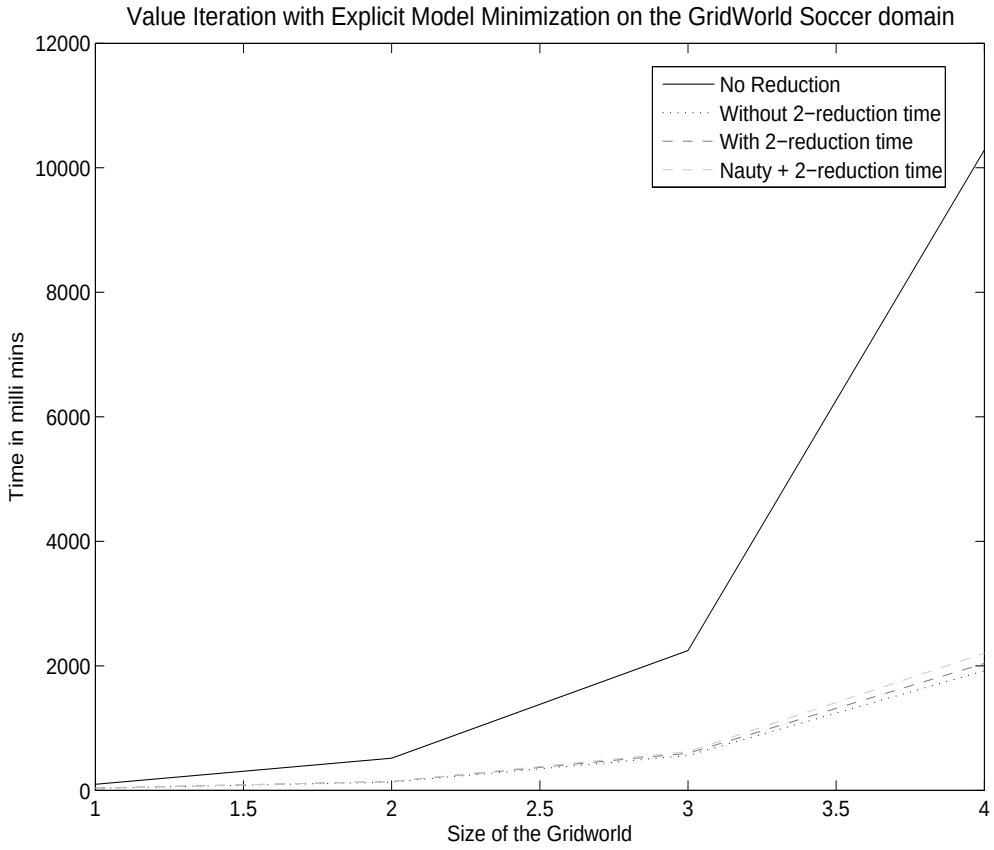


Figure 3.4: Average running times of the value iteration algorithm with explicit model minimization on GridWorld Soccer domain vs size of the domain. Size of one represents the 5×4 grid. Thereafter an increment of one means an increment of one along both axes. Each graph should be compared with the graph for no reduction. The other graphs show the time reduction with reduced model usage. First one discounts the time taken to find symmetries and for reduction. The next one includes the time for reduction but discounts time taken to find symmetries. The last one includes both the time taken to find symmetries using NAutY and time for reduction.

CHAPTER 4

Implicit Model Minimization

In this chapter, we note some problems associated with Explicit Model Minimization. Most of them stem from the explicit construction of a reduced model. We also present some solutions addressing each issue separately. We then present the Implicit Model Minimization approach, which resolves all these issues.

4.1 Problems with Explicit Model Minimization

4.1.1 Exorbitant memory requirements

An explicitly specified symmetry group consumes $O(|\mathcal{G}| \times |\Psi|)$ space. So to address this, we need to specify the symmetry group implicitly. One approach to do this is to use factored representations and structured morphisms. For e.g., the NE-SW symmetry in a GridWorld can be succinctly represented as follows:

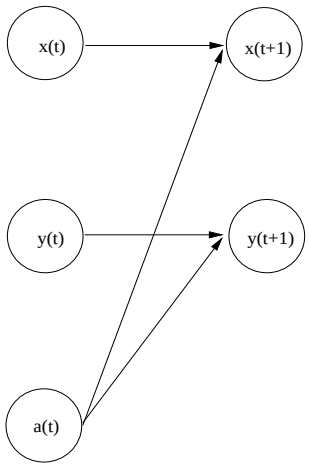
$$\begin{aligned} (x, y) - N &\equiv (y, x) - E \\ (x, y) - S &\equiv (y, x) - W \end{aligned}$$

The advantage here is that the morphisms forming the symmetry group need not be stored explicitly as they are defined on the features instead of states. However, we incur some overhead in the running time. For example, let us consider the

case of permutation automorphisms. To check whether $(s,a) \equiv_{\mathcal{G}} (s',a')$, we need to generate $|\mathcal{G}|$ states that are equivalent to (s',a') by applying each $h \in \mathcal{G}$. Each application of h incurs a time linear in the number of features. Thus in this case the time complexity of the algorithm presented is of the order of $|\Psi|'.|\mathcal{G}|.M$, where M is the number of features whereas no space is needed for storing $\mathcal{G}$ explicitly.

Thus by restricting the class of automorphisms to functions that are defined on features instead of states, we only incur additional time, which is a function of the number of features (significantly less than the number of states) along with a drastic decrease in the space complexity. The use of factored representations leads to further reduction in space needed for storing the transition probabilities and the reward function, thereby making the $\mathcal{G}$ -Reduced Image algorithm more effective than its use in the generic case.

4.1.2 Non-preservation of structure



x(t)	a(t)	x(t+1)	P
x	N	x	0.9
x	E	x + 1	0.9
x	W	x	0.9
x	S	x + 1	0.9

Figure 4.1: Factored representation of a Grid-World domain where (x,y) are the co-ordinates of a grid

Due to the explicit nature of the algorithm wherein the state-action spaces, transition probabilities and rewards are stored explicitly, the structure present in the transition probabilities and the reward function are not preserved in the reduced model. Hence, the reduced model might take up considerably more space than the original model.

For example, consider the factored representation of a 4×4 Grid-World domain as shown in Figure 4.1. The number of entries in the transition probability tables is $2 * 4 * 4 * 4 = 128$.

Now consider the reduced model constructed by the $\mathcal{G}$ -Reduced Image algorithm using a 2-fold symmetry along the NE-SW diagonal. The number of entries in the transition probability matrix is $|\Psi'| \times |S'| = 64/2 * (16 - 6) = 320$.

4.1.3 Redundant operations of reduced model construction and policy lifting

The goal of the model minimization approach is to solve the given MDP, though taking lesser time. Hence, the explicit construction of a reduced model and lifting the policy in a reduced model to the original model seem redundant. The previous problem and this one can be resolved by forgoing the explicit construction and integrating the minimization approach with the solution methods. We call this Implicit Model Minimization and present it in Section 4.2.

4.2 Approach

To resolve some of the issues mentioned in the previous section, we propose the Implicit Model Minimization approach where we modify a solution technique to integrate the symmetries. This is achieved by including the reduced model construction as a part of the solution technique. This enables us to work in the reduced state-action space of a compact original model without converting the compact model into an explicit one. This will also aid to reduce the overheads for constructing a reduced model and policy lifting.

A solution technique like value iteration, involves multiple sweeps through the state-action space with each sweep generating a better approximation to the optimal value function. The idea of implicit model minimization is to restrict the sweeps only to the state-action space of the reduced MDP and compute transition probabilities and reward parameters for the reduced MDP before computing the update equation. Since we are dealing with symmetries, the reduced MDP is actually a portion of the original MDP. Because of this we can restrict the sweeps to this portion of the given MDP and reap the benefits of model minimization as each sweep takes lesser time. Also as the parameters of the reduced MDP are used in the update equation, the value function is computed only for the reduced MDP and the solution is correct as long as we stay in the corresponding portion of the original MDP. We apply this idea to Value Iteration next.

4.3 Reduced Value Iteration

We use the $\mathcal{G}$ -Reduced Image algorithm as a basis to present the reduced value iteration algorithm in Algorithm 5.

Following is an analysis of the time taken by the algorithm.

Let $\mathcal{M}' =< S', A', \Psi', P', R' >$ be a reduced model of $\mathcal{M} =< S, A, \Psi, P, R >$.

Let $|\Psi'| = |\Psi|/k_1$ and $|S'| = |S|/k_2$.

Let p_ϵ be the number of iterations taken to converge to the ϵ-optimal value function.

Algorithm 5 computes the value function only for the reduced model but in each iteration there is a $|\mathcal{G}| = O(k_1)$ overhead for comparing equivalence of state-action pairs and a $O(k_2)$ overhead for comparing equivalence of states while computing the transition probabilities of the reduced model. So, the time taken by the algorithm is:

$$\begin{aligned} n_1.|\Psi'|.|\mathcal{G}|.|S'|.O(k_2).p_\epsilon &= n_1.\frac{|\Psi|.n_2.k_1}{k_1}.\frac{|S|.n_3.k_2}{k_2}.p_\epsilon \\ &= n_1'.|\Psi|.|S|.p_\epsilon \end{aligned} \tag{4.1}$$

Usually, the number of operations involved is more in this algorithm, that is, the value of the constant involved is more than that in normal Value Iteration. Hence, it will take more time than normal Value Iteration, that is, the overheads outweigh the reduction. This is verified on the Grid-World domain and the results

Algorithm 5 Reduced Value Iteration with integrated symmetries

Given $\mathcal{M} = \langle S, A, \Psi, P, R \rangle$ and $\mathcal{G} \leq Aut\mathcal{M}$,
Hashtables $Q_0, Q_1 \leftarrow Nil$ are the action value functions for the previous and current iterations respectively
$|S|$ dimension vectors v_0, v_1 are the state value functions for the previous and current iterations respectively
$v_0 \leftarrow \bar{\mathbf{0}}$
repeat
  Set *Que* to some initial state $\{s_0\}$
  while *Que* is non-empty **do**
    s = dequeue{*Que*}
    for each $a \in A_s$ **do**
      if $(s,a) \not\equiv_{\mathcal{G}} (s'',a'')$ for any $(s'',a'') \in Q_0$, **then**
        Hashtable $S' \leftarrow Nil$ is for checking equivalent states
        for each $t \in S$ such that $P(s,a,t) > 0$ **do**
          if $t \equiv_{\mathcal{G}|S} s'$, for some $s' \in S'$, **then**
            $P'(s,a,s') \leftarrow P'(s,a,s') + P(s,a,t)$
          else
            $S' \leftarrow S' \cup t$
            $P'(s,a,t) = P(s,a,t)$
            add t to *Que*
          end if
        end for
        if $(s,a) \notin Q_1$ **then**
          add (s,a) to Q_1
          $Q_1(s,a) \leftarrow 0$
        end if

$$Q_1(s,a) \leftarrow R(s,a) + \gamma \sum_{s'' \in S'} P'(s,a,s'') \max_{a'' \in A_{s''}} Q_0(s'',a'')$$

      end if
    end for
  end while
  $\forall s \in S, v_1(s) = \max_{a \in A_s} Q_1(s,a)$
  $\Delta \leftarrow \max(abs(v_1 - v_0))$
  $Q_0 \leftarrow Q_1$
  $v_0 \leftarrow v_1$
until $\Delta < \frac{\epsilon(1-\gamma)}{2\gamma}$

are shown in Figure 4.2. As is evident, the amount of symmetry does not matter and the running time of the Reduced Value Iteration algorithm is more than that of normal Value Iteration by a constant factor.

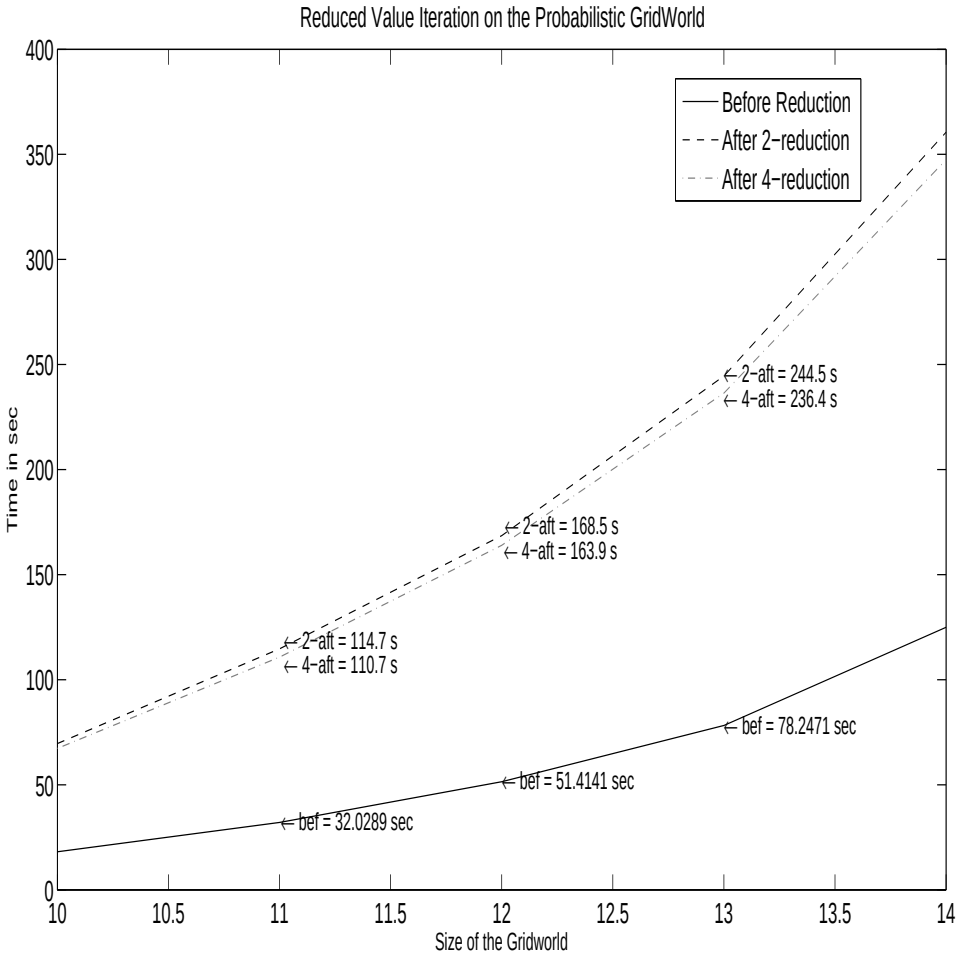


Figure 4.2: Average running times of the RVI algorithm on the Probabilistic GridWorld domain plotted against the size of the GridWorld with various degrees of symmetry (0, 2 and 4). Irrespective of the amount of symmetry, the running times of RVI have increased by nearly equal amounts from that of the normal Value Iteration algorithm. This corroborates the analysis.

From the above analysis it is clear that the equivalence comparisons act as overheads and nullify any reductions possible in the state and state-action spaces. This also means that if we can find a solution technique, unlike the Value Iteration algorithm, for which the number of iterations required for convergence depends on the state-action space, we can integrate the ideas of implicit model minimization to achieve reductions in the time taken. Since the Real Time Dynamic Programming (RTDP), as seen in Section 2.1.2, has this property, we modify RTDP to integrate the symmetry information as in the case of the Reduced Value Iteration algorithm. We present the Reduced RTDP algorithm next and demonstrate the significant gains over the normal RTDP case empirically in Section 4.5.

4.4 Reduced Real Time Dynamic Programming

Clearly, we are presented with two contrasting goals here:

1. Achieving gains over normal solution techniques by exploiting symmetries efficiently.
2. Doing so with the preservation of any structure present in the transition probabilities and the reward function.

To achieve the best of both worlds, we apply the implicit model minimization idea to the RTDP algorithm. The number of iterations required for convergence of RTDP depends on:

1. Topology of the domain
2. Size of the state-action space

The topology helps RTDP to use the agent's experience to focus on the relevant sections of the state space. This saves the time spent on building a reduced model

of the irrelevant sections of the state-space. Also, the time spent in random walks during the early phases of exploration by the Reduced RTDP algorithm is lesser because there is only a reduced amount of space to explore, which is because it works only on the state space corresponding to the reduced model.

As the number of iterations required for convergence depends on the size of the state-action space, Reduced RTDP takes lesser number of iterations and hence lesser time than normal RTDP because the reduced version of the algorithm only acts on the portion of the state space that corresponds to the reduced model. We present the algorithm in Algorithm 6.

4.4.1 Convergence of Reduced RTDP

The algorithm is a modification of the RTDP algorithm with steps from the previous algorithm integrated into lines 7 to 20. If we assume that we have the reduced MDP M', then leaving out lines 7 to 10 and lines 13 to 20 leaves us with the normal RTDP algorithm being run on the reduced image since as explained below, for all $(s,a) \in \Psi', R'(s,a) = R(s,a)$. Due to the equivalence tests done at lines 7 and 14, the algorithm maintains a policy for and considers only the reduced state space. From lemmas 6 and 7, lines 13 to 20 compute the transition probabilities for the reduced image. From Equation 2.2, $R(s,a)$ is the expected reward under the reduced image. So for all $(s,a) \in \Psi', R'(s,a) = R(s,a)$. Thus the update equation in line 21 can be rewritten as,

$$Q(s,a) = R'(s,a) + \sum_{s'' \in S'} \gamma . P'(s,a,s'') . \max_{a'' \in A_{s''}} Q(s'',a'') \tag{4.2}$$

Algorithm 6 RTDP algorithm with integrated symmetries, which computes the Action Value function for the reduced MDP without explicitly constructing it.

Given $\mathcal{M} = \langle S, A, \Psi, P, R \rangle$ and $\mathcal{G} \leq Aut\mathcal{M}$,
Hashtable $Q \leftarrow Nil$ is the action value function.
for each episode **do**
  Initialize s and $S' \leftarrow \{s\}$
  Choose a from s using policy derived from Q (e.g. ϵ-greedy policy)
  for each step of the episode **do**
    if $(s,a) \equiv_{\mathcal{G}} (s'',a'')$ for some $(s'',a'') \in Q$ where $(s'',a'') \neq (s,a)$ **then**
      $s \leftarrow s''; a \leftarrow a''$
      continue
    end if
    Take action a and observe reward r and next state s'
    Choose a' from s' using policy derived from Q (e.g. ϵ-greedy policy)
    for each t such that $P(s,a,t) > 0$ **do**
      if $t \equiv_{\mathcal{G}|S} s''$, for some $s'' \in S'$, **then**
        $P'(s,a,s'') \leftarrow P'(s,a,s'') + P(s,a,t)$
      else
        $S' \leftarrow S' \cup t$
        $P'(s,a,t) = P(s,a,t)$
      end if
    end for
    if $(s,a) \notin Q$ **then**
      add (s,a) to Q
      $Q(s,a) \leftarrow 0$
    end if

$$Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s'' \in S'} P'(s,a,s'') \max_{a'' \in A_{s''}} Q(s'',a'')$$

    $s \leftarrow s'; a \leftarrow a'$
  end for
end for

which is nothing but the update equation for the reduced image. Thus it is exactly similar to running normal RTDP on the reduced image. As normal RTDP converges to an optimal action value function [Barto *et al.*, 1995], reduced RTDP also converges, as long as it continues to back up all states in the reduced image.

4.5 Results

An exact analysis of the Reduced RTDP algorithm is beyond the scope of this thesis. Hence we present empirical evidence to corroborate the above made claims.

Experiments were done on three domains, Deterministic GridWorld, Probabilistic GridWorld and GridWorld Soccer. The latter two domains were explained in Chapter 3, Section 3.3. The Deterministic GridWorld is similar to the Probabilistic GridWorld. The only difference is that, the actions are deterministic. We consider the effect of the degree of symmetry by presenting graphs for multiple symmetries wherever applicable. We compare the reduced RTDP algorithm using multiple degrees of symmetry with the normal RTDP algorithm[1].

We present learning curves representing the decrease in the number of steps taken to finish each episode. We plot the running times of the reduced RTDP algorithm with multiple symmetries and the normal RTDP algorithm against the size of the domain. We ran 200 episodes of each domain and the times were averaged over 25 runs. All the algorithms used a discount factor, $\gamma = 0.9$. An epsilon greedy policy with $\epsilon = 0.1$ was used to choose the actions at each step. We present one learning graph per domain.

[1]The normal RTDP algorithm is sometimes referred in this work as the no or zero symmetry case.

One observation contrary to the graphs presented is that when reduced RTDP algorithms are used for very small domains, the overhead involved in checking equivalence of states outweighs the benefit from the reduction due to symmetry because the topology is not complicated enough to cause an explosion in the space to be explored, which does not affect the number of iterations too much.

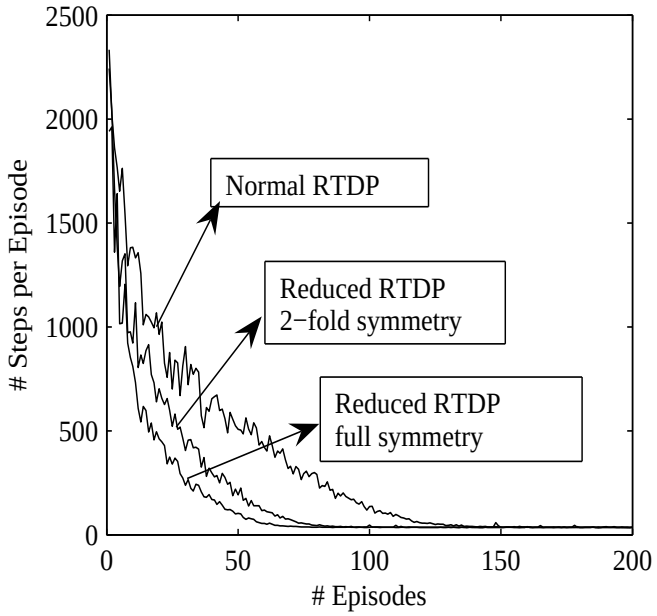


Figure 4.3: Learning curves for the Deterministic Grid World(25x25 grid) showing the decrease in the number of steps taken per episode

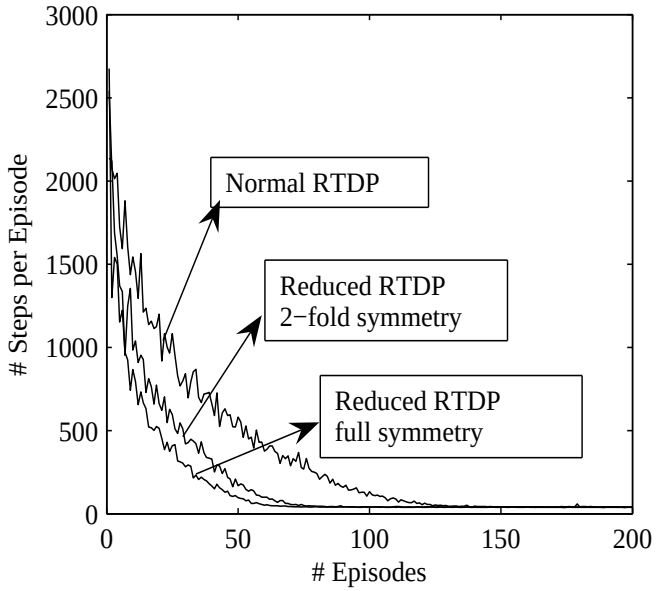


Figure 4.4: Learning curves for the Probabilistic Grid World(25x25 grid) showing the decrease in the number of steps taken per episode

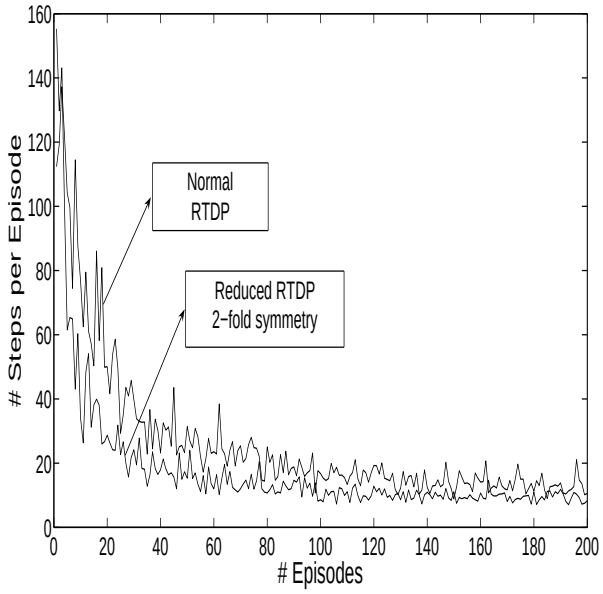


Figure 4.5: Learning curves for the GridWorld(5x4 grid) soccer domain showing the decrease in the number of steps taken per episode

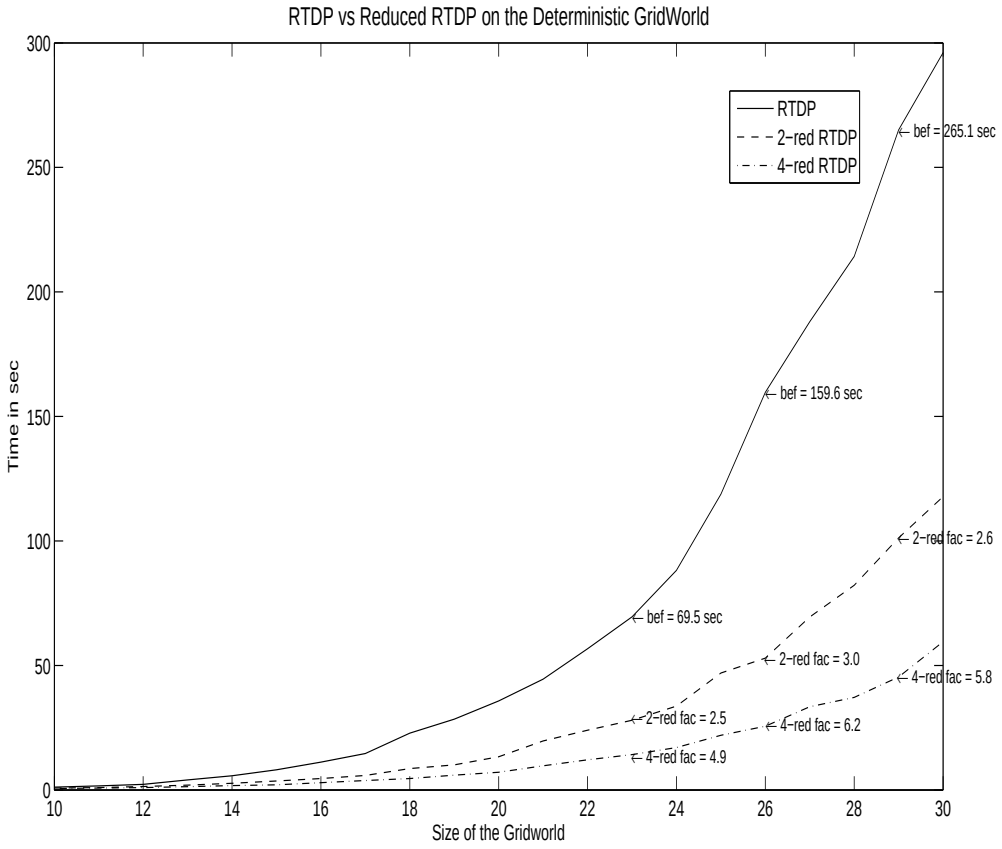


Figure 4.6: Running times of the Reduced RTDP algorithm on the Deterministic GridWorld domain plotted against the size of the GridWorld with various degrees of symmetry(0,2 and 4)

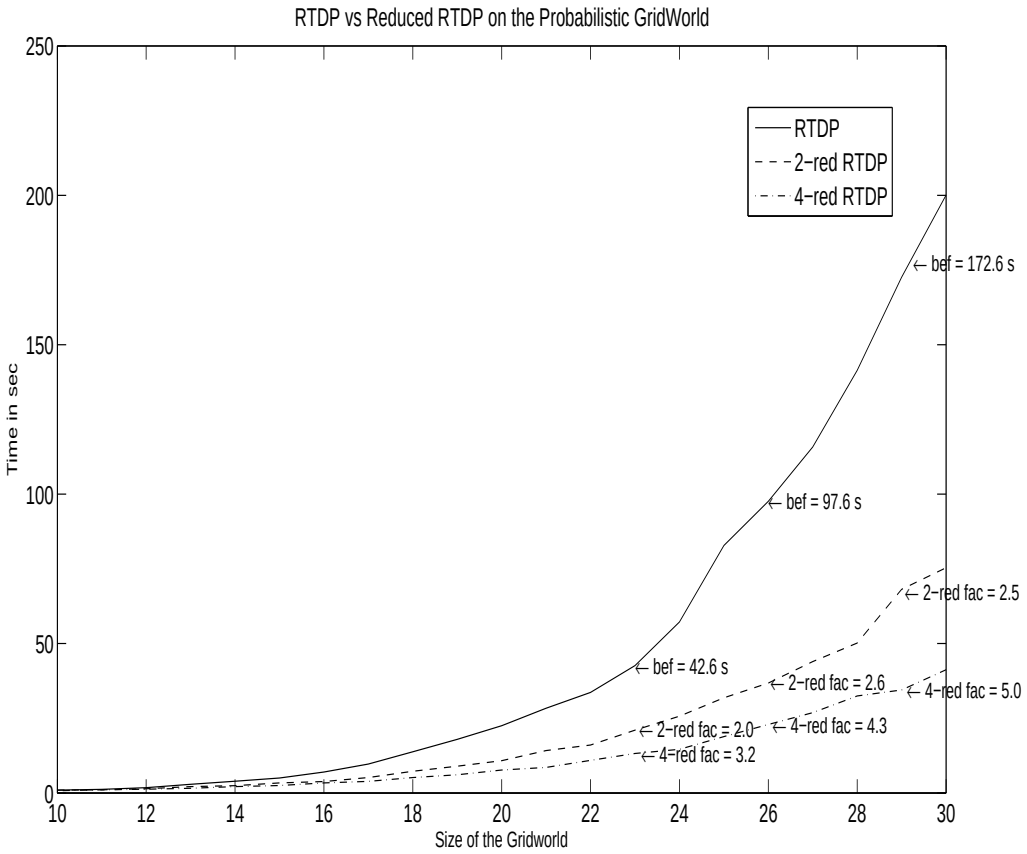


Figure 4.7: Running times of the Reduced RTDP algorithm on the Probabilistic GridWorld domain plotted against the size of the GridWorld with various degrees of symmetry(0,2 and 4)

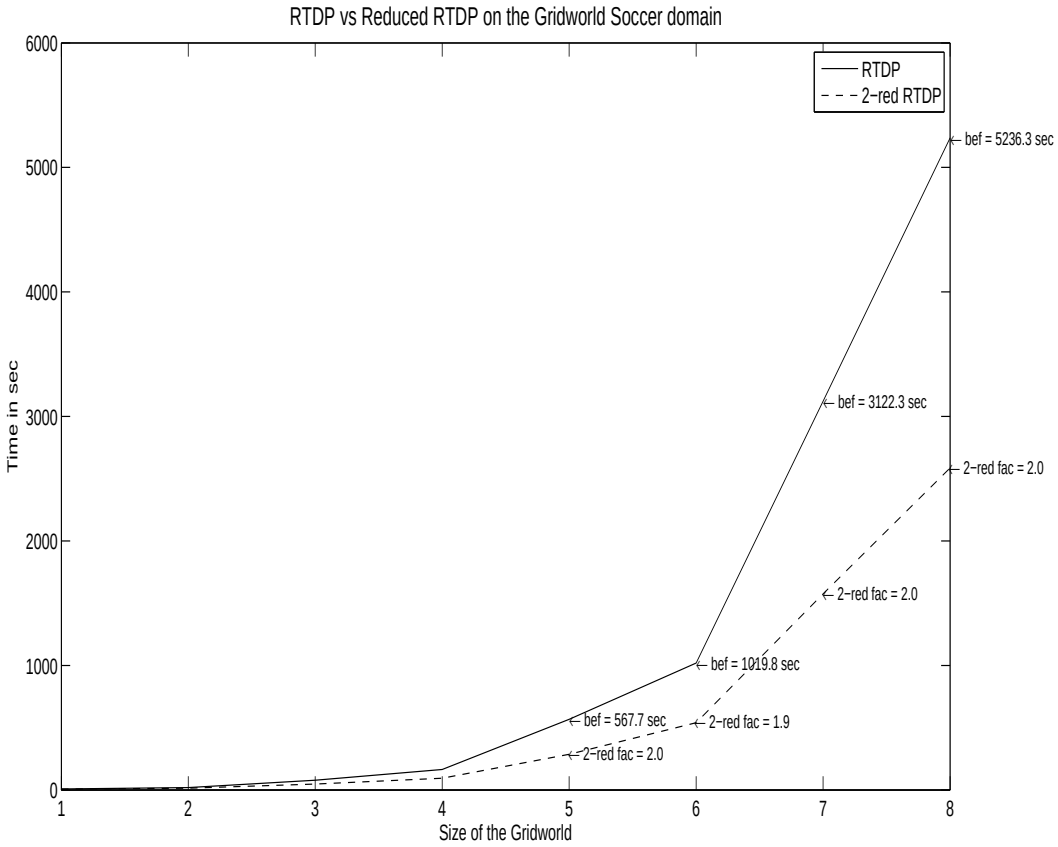


Figure 4.8: Running times of the Reduced RTDP algorithm on the GridWorld soccer domain plotted against the size of the GridWorld with 0 and 2-fold symmetry

CHAPTER 5

Conclusions and Future Work

The primary motivation for this work was to gain a better understanding of the use of symmetries for abstraction in MDPs. In this regard, first we have addressed the problem of finding symmetries, which we have shown is *Isomorphism Complete*. This is an interesting result because even with all the additional complexities involved in the formulation of an MDP, it turns out that finding symmetries on MDPs is no harder than finding graph isomorphisms. An important practical application is that, it now allows the use of existing Graph Isomorphism solvers and heuristics to our advantage. Next, we proposed the use of an efficient minimization algorithm to perform abstraction using symmetries in an end-to-end manner. Not surprisingly, our tests on various domains indicate that we can achieve the practical bounds on improvement that we set out in Section 2.5.1 if we can use good heuristics for symmetry finding.

We then identified certain problems with the explicit approach and looked at ways of minimization without actually constructing a reduced image. Combined with a suitable technique and representation, this results in enormous savings in space and possibly time.

There are two aspects which need further attention. One is the use of approximate notions of symmetry because exact symmetries might not exist in real world domains. Though the approach works on functions that are not exactly symmetries, the solutions found need not be optimal. We need to find bounds on the

amount of deviation from the optimal policy. Ravindran and Barto [2004] discuss the notion of approximate homomorphisms for non-exact minimization of MDPs and derive bounds on the deviation from the optimal policy. This should be a good starting point.

Another aspect is to tailor algorithms to particular representations. For e.g., in factored representations, finding symmetries might be easier due to the tree structure of the transition probabilities and reward function. Though on a generic graph structure we have shown that finding symmetries is a hard problem, one might find representations for which it turns out easier.

Publication

1. S. M. Narayanamurthy and B. Ravindran (2007). Efficiently Exploiting Symmetries in Real Time Dynamic Programming. *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 2556–2561.

REFERENCES

Amarel, S., On representations of problems of reasoning about actions. *In* **D. Michie** (ed.), *Machine Intelligence 3*, volume 3. Elsevier/North-Holland, Amsterdam, London, New York, 1968, 131–171. Amarel, S.

Barto, A. G., **S. J. Bradtke**, and **S. P. Singh** (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, **72**, 81–138.

Bellman, R. E., *Dynamic Programming*. Princeton University Press, 1957.

Benhamou, B., Study of symmetry in constraint satisfaction problems. *In* **A. Borning** (ed.), *PPCP'94: Second International Workshop on Principles and Practice of Constraint Programming*. Orcas Island, Seattle, USA, 1994. URL `citeseer.ist.psu.edu/benhamou94study.html`.

Bertsekas, D. P., *Dynamic programming: deterministic and stochastic models*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1987. ISBN 0132215810.

Booth, K. S. and **C. J. Colbourn** (1977). Problems polynomially equivalent to graph isomorphism. Technical report, University of Waterloo.

Bowling, M. (2003). *Multiagent Learning in the Presence of Agents with Limitations*. Ph.D. thesis, Carnegie Mellon University.

Crawford, J. (1992). A theoretical analysis of reasoning by symmetry in first-order logic. URL `citeseer.ist.psu.edu/crawford92theoretical.html`.

Dean, T. and **R. Givan**, Model minimization in markov decision processes. *In AAAI/IAAI*. 1997. URL `citeseer.ist.psu.edu/dean97model.html`.

Dean, T., **R. Givan**, and **S. Leach**, Model reduction techniques for computing approximately optimal solutions for Markov decision processes. *In Proceedings of UAI-97*. 1997. URL `citeseer.ist.psu.edu/article/dean97model.html`.

Dean, T. and **S.-H. Lin**, Decomposition techniques for planning in stochastic domains. *In Proceedings of the 1995 International Joint Conference on Artificial Intelligence*. 1995. URL `citeseer.ist.psu.edu/article/dean95decomposition.html`.

Dunitz, J. D., Symmetry arguments in chemistry. *In Proceedings of the National Academy of Sciences, USA*, volume 93. 1996.

Emerson, F. A. and **A. P. Sistla** (1996). Symmetry and model checking. *Formal Methods in System Design: An International Journal*, **9**(1/2), 105–131. URL `citeseer.ist.psu.edu/emerson94symmetry.html`.

Flener, P., **A. Frisch**, **B. Hnich**, **Z. Kiziltan**, **I. Miguel**, **J. Pearson**, and **T. Walsh** (2002). Breaking row and column symmetries in matrix models. URL `citeseer.ist.psu.edu/flener02breaking.html`.

Givan, R., **T. Dean**, and **M. Greig** (2003). Equivalence notions and model minimization in markov decision processes. *Artificial Intelligence*, **147**(1-2), 163–223.

Hartmanis, J., *Algebraic structure theory of sequential machines (Prentice-Hall international series in applied mathematics)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1966. ISBN B0006BNWTE.

Knoblock, C. A., Learning abstraction hierarchies for problem solving. *In* **T. Dietterich** and **W. Swartout** (eds.), *Proceedings of the Eighth National Conference on Artificial Intelligence*. AAAI Press, Menlo Park, California, 1990. URL `citeseer.ist.psu.edu/knoblock90learning.html`.

Lee, D. and **M. Yannakakis**, Online minimization of transition systems (extended abstract). *In STOC '92: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*. ACM Press, New York, NY, USA, 1992. ISBN 0-89791-511-9.

Littman, M. L., **T. L. Dean**, and **L. P. Kaelbling**, On the complexity of solving Markov decision problems. *In Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI–95)*. Montreal, Québec, Canada, 1995. URL `citeseer.ist.psu.edu/littman95complexity.html`.

Manning, J. B. (1990). *Geometric symmetry in graphs*. Ph.D. thesis, Purdue University.

Mathon, R. (1979). A note on the graph isomorphism counting problem. *Information Processing Letters*, **8**(3), 131–132. ISSN 0020-0190.

McKay, B. D. (1981). Practical graph isomorphism. *Congressus Numerantium*, **30**, 45–87.

Miller, G. L., Graph isomorphism, general remarks. *In STOC '77: Proceedings of the ninth annual ACM symposium on Theory of computing*. ACM Press, New York, NY, USA, 1977.

Pearson, J., Symmetry breaking in constraint satisfaction with graph-isomorphism: Comma-free codes. *In AI&M 1-2004, Eighth International Symposium on Artificial Intelligence and Mathematics, January 4-6, 2004, Fort Lauderdale, Florida, USA*. 2004.

Popplestone, R. J. and **R. A. Grupen**, Symmetries in world geometry and adaptive system behaviour. *In AFPAC '00: Proceedings of the Second International Workshop on Algebraic Frames for the Perception-Action Cycle*. Springer-Verlag, London, UK, 2000. ISBN 3-540-41013-9.

Puterman, M. L., *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1994. ISBN 0471619779.

Ravindran, B. (2004). *An Algebraic Approach to Abstraction in Reinforcement Learning*. Ph.D. thesis, Department of Computer Science, University of Massachusetts Amherst.

Ravindran, B. and **A. G. Barto** (2001). Symmetries and model minimization of markov decision processes. Technical report, University of Massachusetts, Amherst.

Ravindran, B. and **A. G. Barto** (2002). Model minimization in hierarchical reinforcement learning. *Lecture Notes on Computer Science*, **2371**, 196–211. ISSN 0302-9743. URL http://link.springer-ny.com/link/service/series/0558/bibs/2371/23710196.htm;http://link.springer-ny.com/link/service/series/0558/papers/2371/23710196.pdf.

Ravindran, B. and **A. G. Barto**, Approximate homomorphisms: A framework for non-exact minimization in markov decision processes. *In In the Proceedings of the Fifth International Conference on Knowledge Based Computer Systems (KBCS 04)*. 2004.

Read, R. C. and **D. G. Corneil** (1977). The graph isomorphism disease. *Journal of Graph Theory I*, 339–363.

Skiena, S. (1997). The stony brook algorithm repository. URL http://www.cs.sunysb.edu/~algorith/implement/nauty/implement.shtml.

Sutton, R. and **A. Barto**, *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998. URL citeseer.ist.psu.edu/sutton98reinforcement.html.

Wolf, T. (1995). The program crack for solving pdes in general relativity.

Zinkevich, M. and **T. Balch**, Symmetry in markov decision processes and its implications for single agent and multiagent learning. *In Proceedings of the ICML-01*. Morgan Kaufmann, 2001. ISBN 1-55860-778-1.