A Log Space Algorithm for Reachability in Planar Acyclic Digraphs with Few Sources

Princy Lunawat

CS6840: Advanced Complexity Theory

31^{st} March, 2012

Outline I

Outline II

- Reachability in a Source Tree
- SMPD Algorithm
- The Contracted Graph
- Coin Crawl Game

3. Non Deterministic Search
   - Explored Region
   - Initializing the Explored Region
   - Expanding the Explored Region
   - Moving the Explored Region

4. Removing the Non Determinism
   - Bounding the Depth of an Accepting Path
   - Structure of the Irreducible Path

Outline III

Directed Graphs

- Given a graph $G = (V, E)$ and an ordered pair of vertices (u, v), v is said to be reachable from u if there exists a directed path from u to$v$.
- General Directed Graph reachability is the problem of determining if v is reachable from u.
- It is known to be NL - complete.

Why Planar Directed Graphs?

- Planarity is an important restriction on general digraphs when looking for reachability:
- Planar Reachability:
  - Is L- hard under AC_0 reductions.
  - If proved to be in L, then it will be L- complete under AC_0 reductions.
  - Is not known to be complete for NL.

The Class UL

- A language L is in UL if and only if there exists a nondeterministic log-space machine M accepting L such that, for every instance x, M has at most one accepting computation on input x. Hence, by definition $UL \subseteq NL$.
- Planar Reachability is in the class UL. (unambiguous log space)
- If planar reachability is proved to be NL- complete, then $NL = UL$ thereby, all non deterministic log space computations can be made unambiguous.

Planar DAGs

- The space complexity for reachability problem over planar graphs is not settled.
- A natural question is to ask for a class of planar graphs that admit log space reachability.
- We now talk about the class of planar directed graphs which are acyclic.

Problem Statement:

- Given a planar, directed, acyclic graph $G = (V, E)$ and an ordered pair of vertices (u, v), determine if v is reachable from u.

Background:

- Jacoby et al. [2006], proved reachability for series parallel graphs (a special case of single source single sink planar digraphs) is in L.
- Reingold [2008], proved that undirected graph reachability is in log space.
- Allender et al.[2009], have extended Jacoby's result to show that the reachability problem for planar DAGs with at most two sources and multiple sinks is in L.

Author's Contribution:

- Main Theorem: *The reachability problem for planar directed acyclic graphs with m sources and n vertices is decidable in* $O(m + logn)$ *space.*
- Corollary: *The reachability problem for planar directed acyclic graphs with* $O(logn)$ *sources is in* L.

Algorithm Outline

- **Forest Decomposition:** Decomposing G into a forest, with each tree rooted at a source, and two more trees rooted at u and v. Each Tree is a single source multiple sink instance, for which reachability is in log space.
- **Contraction of Graph G:** A Multigraph is obtained by contracting each source tree of G into a node. This graph can have loops and parallel edges.
- **Coin Crawl Game:** A high level description of the non deterministic algorithm to determine reachability using the contracted graph.
- Description of the non deterministic algorithm.
- Elimination of non determinism using space linear in the number of sources.

Forest Decomposition

- **Observation:** A reverse walk starting at any vertex ends at a source.
- **The Technique:**
  - Pick any arbitrary incoming edge from each vertex $x \in V$.
  - Assume u to be a source, since its incoming edges cannot contribute to a $u - v$ path.
  - Do no pick any incoming edge for v and leave it isolated.
- **Result:** A forest F with $m + 2$ sources, $u, s_1, s_2, \ldots s_m, v$. These are called source trees, denoted by T_x where x is the root. The edges in a source tree are called tree edges.

Structural Relations Among Vertices

Some Terminology:

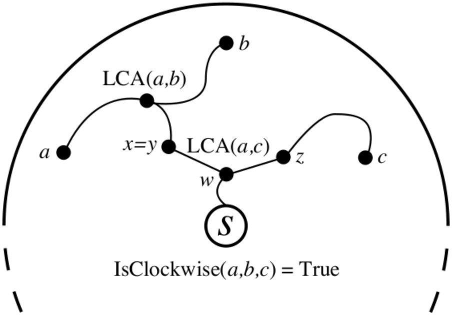
- **Ancestors of a vertex:** The source tree of a vertex a, denoted by $T_{s(a)}$can be found following the incoming edges starting at a until $s(a)$ is reached. Every vertex along this path is an ancestor of a.
- **LCA(a,b) :** For two vertices lying in the same source tree T, the least common ancestor of a and b is defined as a vertex x in T which is an ancestor of both a and b and maximizes the number of edges between x and $s(a)$.
- **Combinatorial Embedding :** An embedded planar graph uniquely defines a cyclic order of edges incident on every vertex. This is called the combinatorial embedding of the graph.

IsClockwise(a, b, c)

The boolean function IsClockwise(a, b, c) :

- Returns true if the vertices a, b, c respect obey the cyclic order in clockwise direction.
- Find $LCA(a, c) = w$ (say). If w is not an ancestor of b, then the function of course returns false.
- Otherwise, let x, y, z be ancestors of a, b, c on the path to w such that $(w, x), (w, y)$ and (w, z) are tree edges. If x, y, z appear in a clockwise order in the combinatorial embedding of w, then the function returns true.

Illustration of IsClockwise(a, b, c)



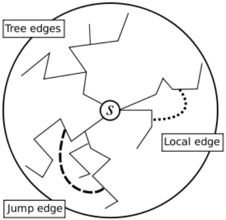
Edge Classification

The topological properties of the embedding along with the forest decomposition lead to a classification of the edges in the graph, using the following sub-structures:

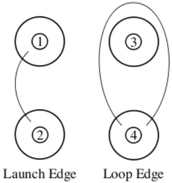
- **Tree Path:** A tree path between two vertices a and b in a tree T, is the unique undirected path formed by the path from a to $LCA(a, b)$ and then from $LCA(a, b)$ to b.
- **Tree Cycle:** If there is an edge between a and b, then the tree path along with this edge, forms a tree cycle.

- A tree cycle partitions the plane into two regions, this partition is *trivial* if the number of vertices in any region is zero. This can be determined in log space.
- This classifies the edges into the following types:
  - **Tree edges** are the chosen incoming edges used to define the forest F.
  - **Local Edges** are edges so that the tree cycle partitions the vertices trivially.
  - **Jump Edges** are edges so that the tree cycle partitions the vertices within a source tree non-trivially, but the sources, u and v trivially.
  - **Launch Edges** are edges between different source trees.
  - **Loop Edges** are edges which partition the sources non-trivially.

Classification of Edges



(a) Tree, Local, and Jump edges.



(b) Launch and Loop edges.

Reachability in a Source Tree

- A vertex y can be reachable from a vertex x in a source tree using two types of paths:
  - **Local Paths:** Using only tree and local edges.
  - **Jump Paths:** Can use jump edges also.
- Allender et al. have shown that **Single Source Multiple Sink Planar DAG reachability** (using jump paths) is in log space.

SMPD Algorithm

Aim: Given a source tree T, and a vertex x, find TreeRight(x) and TreeLeft(x) which represent how far can jump paths travel from x in clockwise and counter-clockwise directions.

Steps:

- Find LocalLeft(x) $= l$ and LocalRight(x)$= r$ which are the most counter-clockwise and clockwise vertices from x via local paths.
- (l, r) define an explored region containing every vertex z in T for which isClockwise(l, z, r) is true.
- Consider a jump edge yz with y lying in the explored region and z lying outside the explored region, but closest to it among all jump edges.
- Expand the explored region by setting l to LocalLeft(z) and r to LocalRight(z).
- Repeat until the process stabilizes and there are no new jump edges.
- TreeLeft(x) $= l$ and TreeRight(x) $= r$.

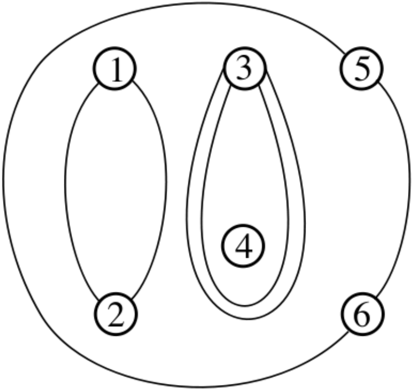
The Contracted Graph

- H is the contracted graph of G: It is the multigraph obtained by contracting the source tree T_s into the root s.
- H consists of launch and loop edges.
- **Topological Equivalence** among the edges:
  - Consider edges e_1, and e_2 with common end points in H.
  - e_1 and e_2 partition the vertices into to disjoint subsets. If any one of the subsets is empty, then e_1 and e_2 are topologically equivalent.

The above relation partitions the edges of the contracted graph into topological equivalence classes.

- **Lemma:** *The number of topological equivalence classes in a planar multigraph of n vertices is at most $3n - 6$.*
- As a consequence of the above lemma, the no. of equivalence classes in the contracted graph H can be at most $3(m + 2) - 6 = 3m$.

Topoligically Equivalent Edges



Coin Crawl Game

- The game provides a high level description of the nondeterministic search algorithm.
- The board for the game is the contracted graph H and its embedding.
- Each vertex is represented as a unit circle.
- Each equivalence class of edges is represented as a single edge.
- The game piece is a unit radius coin with an arrow drawn on it.

The game proceeds as follows:

- **Initial Position:** The coin is is placed initially at the vertex T_u with its arrow pointing an edge going out of T_u.
- **Objective:** To make the coin reach the vertex v.
- Three kinds of moves are present in the game:
  - **Left:** The player rotates the coin left until another edge leaving T is found.
  - **Right:** The player rotates the coin right until another edge leaving T is found.
  - **Cross:** The player crosses the current edge e, reaches tree T_i at the other end and rotates the coin to so that the the arrow points back to e.

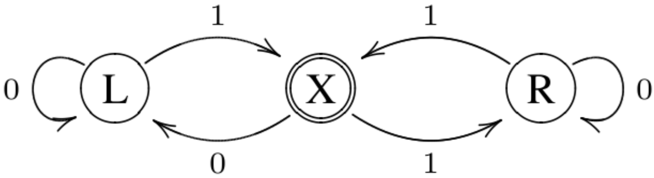
- An oracle confirms whether each move is legal based on the knowledge of connectivity in the graph.
- The non deterministic player proceeds by taking the above moves one by one.
- If a move is illegal, the oracle prevents it and player admits failure, else if the player guesses the right path(if it exists), she wins.

Promises To the Player

- Reversing the coin direction is not necessary.
- Crossing a crossed edge again in not necessary.
- Forbidden Zones: The portion of a circle vertex that has been visited by the player, need not be visited again.

The above promises lead to the an automata $M(x)$ which determines the next move of the player where the non-deterministic bit x is chosen by the player.

Automata Describing the Moves



L = Left   X = Cross   R = Right

Non Deterministic Search: Detailed Algorithm

- What does the coin represent?
- What happens when the coin moves?
- What does each coin move signify in the underlying graph?

Explored Region

- log space data structure representing the coin and its placement in H.
- A five-tuple $C = (A_L, A_R, e_c, B_L, B_R)$.
- e_c is an edge representing the current class of edges that the coin points to, the class has endpoints in source trees A and B.
- A_L represents $TreeLeft(e_c^A)$ and A_R represents $TreeRight(e_c^A)$. Similarly for B.
- The explored region is given by each vertex x such that either isClockwise(A_L, x, A_R) is true or isClockwise(B_L, x, B_R) is true.

Initializing the Explored Region:The initial Coin Position

- e_c is any launch edge leaving T_u. The source tree on the other end is A.
- $A_L =$ TreeLeft(e_c^A) and $A_R =$TreeRight(e_c^A).
- B_L and B_R are initialized to NULL.
- The automata M is initialized to Cross (X) State.
- The oracle NextClass determines if the move is legal, and returns the next edge class e_n at the end of the move.

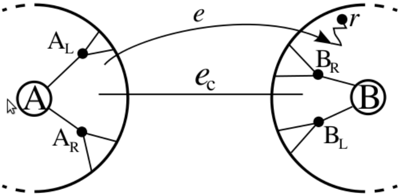
Expanding the Explored Region: When the Coin crosses an edge

A-B iteration:

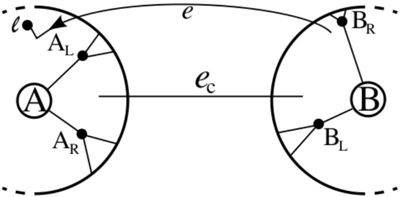
- Consider all edges in edge class e_c starting from source tree A. Let the end point (head) of edge be denoted by e^2.
- If B is null, update B_L to TreeLeft(e^2), and B_R to TreeRight(e^2).
- If not, then update B_L to TreeLeft(e^2) if TreeLeft(e^2) is further counterclockwise from e than B_L. Similarly for B_R.

B_A iteration is symmetric.

Alternating Expansion of Region



(a) An A–B iteration.



Moving the Explored Region: Legality of Moves

The current explored region C is examined by the oracle NextClass, to determine if a move is legal as follows:

- **Right Move:**
  - The coin is rotated Right until the next edge class e_n is found.
  - NextClass determines if the first edge encountered in this class has a tail in C_A. If so, the move is permitted. In effect:
    - Update current edge class to $e_c = e_n$.
    - Set B_L and B_R to null.
- **Left Move:** The argument is symmetric.

- **Cross Move:** A cross move along an edge class $[e_n]$ is legal iff there exists an edge e in the class with its tail in C_A. If so, the oracle allows the move. In effect:
  - Swap A_L with B_L and A_R with B_R.
  - Update edge class $e_c = e_n$.
  - Expand the region as before.

Observations

At each move, we maintain the following invariants:

- At least one end point of $[e_c]$ is reachable from u. Let this be z.
- A_L, A_R, B_L, B_R (if not null) are reachable from z via edges from $[e_c]$.
- Any launch edge xy with x in C has y reachable from z.

Bounding the Depth of an Accepting Path

Forbidden Zones: The coin never revisits a part of any unit circle, thus bounding the no. of moves. But to how much?

Irreducible Path:

- Let $P = u, x_1, x_2, \ldots x_k, v$ be a directed path in G.
- P is said to be irreducible if for every $x_i, x_j, i < j$ that have an ancestor descendant relationship in the forest F, The path P between $x_i and x_j$ consists of the tree path between them.
- Any path P' which is not reducible, can be converted to a reducible path.

Structure of the Irreducible Path

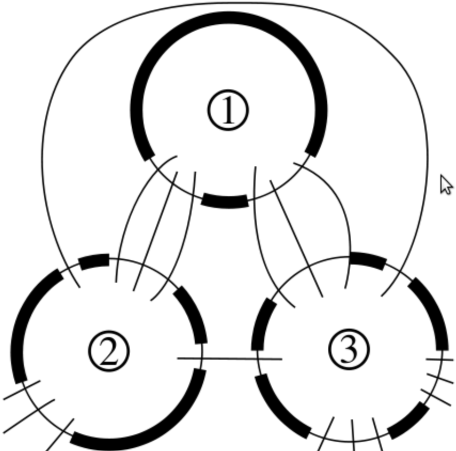
- A sequence representing jump paths.
- Followed by a one or more launch edges.
- Again followed by jump paths, and so on.
- For two consecutive launch edges belonging to different equivalent classes in P, there exists a jump path. Due to acyclic nature of the graph and irreducibility, this path is not revisited.

P cannot revisit a vertex in the jump path between two consecutive launch edges of different edge classes.

What happens in the Coin-Crawl World?

- The equivalence class of edges partitions the source trees into two types of regions:
  - Vertices between tree paths from equivalent launch edges to their sources.
  - Vertices between tree paths from non- equivalent launch edges to their sources.
- One rotation of the coin, corresponds to the path P traversing between two launch edges , since P cannot revisit this again, this rotation marks a forbidden region.
- Each class of launch edges has two such regions, one on each side.
  - One side is used to to reach this class.
  - Another side is used to leave this class.

Patitions of a Source Tree



- Hence, the class can be visited at most twice by a move string.
- Since the no. of classes is at most $3m$, the number of rotation moves can be at most $6m$.
- Each Cross move has to be followed by a rotation move, thus bounding the no. of cross moves to $6m$.

The Bound

An irreducible u-v path induces a list of moves of length at most $12m$.

The Deterministic Algorithm

- For each possible $12m$ bit string x, find a $12m$ length sequence of moves by using $M(x)$.
- If the coin reaches the vertex v using any of these move sequences, output *Yes* (meaning v is reachable from u), otherwise output *No*.

Hence reachability can be decided in $O(m + logn)$ space. If the no. of sources is $m = O(logn)$, then reachability can be decided in $O(logn)$ space.

Finding Cycles

- For correct execution of the algorithm, it is essential that the planar graph is acyclic.
- By Forest Decomposition, the SMPD algorithm by Allender et al., can check for cycles in log space.
- If a cycle exists using a launch edge (x, y) then there exists an irreducible path from y to x which can be found using the algorithm
- A cycle can be detected by iterating over all launch edges.

Future Work

- An intelligent choice of incoming edges, instead of choosing an arbitrary one, may lead to sublinear no. of moves.
- The technique of Forest Decomposition and Topological Equivalence can have useful applications to other problems.

Thank You