A Polynomial-time Algorithm for Non-optimal Multi-Agent Pathfinding

Mokhtar M. Khorshid
Computing Science Department
University of Alberta
Edmonton, AB, Canada T6G 2E8
mokhtar@ualberta.ca

Robert C. Holte
Computing Science Department
University of Alberta
Edmonton, AB, Canada T6G 2E8
holte@cs.ualberta.ca

Nathan Sturtevant
Computer Science Department
University of Denver
Denver, CO, USA
sturtevant@cs.du.edu

Abstract

Multi-agent pathfinding, where multiple agents must travel to their goal locations without getting stuck, has been studied in both theoretical and practical contexts, with a variety of both optimal and sub-optimal algorithms proposed for solving problems. Recent work has shown that there is a linear-time check for whether a multi-agent pathfinding problem can be solved in a tree, however this was not used to actually produce solutions. In this paper we provide a constructive proof of how to solve multi-agent pathfinding problems in a tree that culminates in a novel approach that we call the tree-based agent swapping strategy (TASS). Experimental results showed that TASS can find solutions to the multi-agent pathfinding problem on a highly crowded tree with 1000 nodes and 996 agents in less than 8 seconds. These results are far more efficient and general than existing work, suggesting that TASS is a productive line of study for multi-agent pathfinding.

Introduction

In multi-agent pathfinding there are a number of agents situated on a graph, each with their own goal location. The agents must move to their goal locations, but cannot move into nodes occupied by other agents, and cannot simultaneously cross edges at the same time as another agent.

Multi-agent pathfinding has been studied in both theoretical and practical contexts, with a variety of both optimal and sub-optimal algorithms proposed for solving problems. Recent work (Masehian and Nejad 2009) has shown that there is a linear-time check for whether a multi-agent pathfinding problems can be solved in a tree, however this was not used to actually produce solutions. In this paper, we provide a constructive proof of how to solve multi-agent pathfinding problems in a tree in polynomial time using a novel tree-based agent swapping strategy (TASS). Experimental results showed that TASS can find solutions to the multi-agent pathfinding problem on a highly crowded tree with 1000 nodes and 996 agents in less than 8 seconds. These results are far more efficient and general than existing work, suggesting that TASS is a productive line of study for multi-agent pathfinding. Our approach also allows us to quickly identify, for a given tree, the maximum number of agents for which we are guaranteed to find complete solutions given any valid configuration of such agents.

Background and Related Work

In this paper we study movement on a tree, but we are eventually interested in decomposing graphs into trees in order to efficiently find solutions. Some problems, like mazes, are already represented as trees. For other problems, we have devised Graph-to-Tree Decomposition (GTD) algorithms that can induce trees from general graphs (see Slidable Induced Trees below). We are confident that a large number of multi-agent pathfinding problems can be represented and solved as trees, hence the goal of this paper, which is to provide a foundation for studying this approach.

Multi-agent pathfinding has been studied in a range of contexts, including agent simulations for interactive entertainment (Silver 2005) and robotics (Ryan 2008). A variety of algorithms have been developed for both optimal and sub-optimal solutions. We highlight a few here that define the necessary context for this work.

Finding optimal solutions in the general multi-agent pathfinding problem is NP-complete (it reduces to the sliding-tile puzzle (Ratner and Warmuth 1986)), which is probably why there has not been extensive work on optimal solutions. One notable optimal approach (Standley 2010) used the idea of operator decomposition to simplify the problem and extend the range of problems that can be solved optimally.

Optimal approaches require centralized planning, although centralized planners can also be used to find suboptimal solutions. Ryan (2010), for instance, used a constraint-based model to find solutions more quickly, at the cost of losing optimality. In this paper, like Ryan, we take a centralized approach that results in sub-optimal solutions.

Most other approaches to the problems are distributed. Silver (2005) developed a hierarchy of algorithms culminating in Windowed Hierarchical Cooperative A* (WHCA*). In this approach each agent plans cooperatively over a sliding window. Wang and Botea (2008; 2009) have provided multiple improvements on this approach. Most importantly, they introduced a class of problems called SLIDEABLE, which can be solved in low polynomial time. While they showed that minor relaxations to this problem could still be solved, we will show that TASS can handle problems which are not slideable, we will also provide an algorithm, GTD-SLIDABLE, that converts SLIDEABLE problems into solvable trees.

One piece of work which is orthogonal to these approaches is (Surynek 2009) which looks at methods for improving the quality of existing solutions. The solutions produced by TASS are inefficient in general. The focus of this paper is on how to actually solve multi-agent pathfinding problems on trees while guaranteeing completeness within the domain of solvable trees; improving solution quality is a matter of future work.

Our work in this paper is primarily based on that of Masehian and Nejad (2009). Their notion of Solvable Trees and detecting solvable trees is central to our work. Our simplified solvability criteria follows directly from their work, but we have provided our own complete constructive proof of solvability that not only answers the question whether a solution exists, but also constructs this solution. Therefore, the focus of this paper is on constructing this algorithm and analyzing its performance.

Finally, (Luna and Bekris 2011) in parallel to our work, had a similar algorithm to solve multiagent problems on general graphs. Their solving approach shares with ours the idea of swaping agents at specific junctions but the overall solution as well as the scope differs. GTD and TASS have a narrower scope defined by more strict requirements on the topology of the graphs we can handle so far, however, TASS has a lower asymptotic runtime and seems to be orders of magnitude faster. In a quick independent collaborative experiment with the authors, we found out that on trees, TASS performed more than 2 orders of magnitude faster and produced results that are half as long as their Push and Swap algorithm[1].

Solving Multi-Agent Path Planning on a Tree

We will begin by making a few key definitions; the first two of which are restated from (Masehian and Nejad 2009). In all of these definitions and lemmas there is some fixed (given) undirected tree T in which each node is either a "hole" or contains an "agent". All agents are distinct. An agent that is adjacent to a hole can swap places with it. The number of nodes in T is n, the number of agents is m, and the number of holes is $H = n - m$. Agents have goal locations that they are trying to reach; each goal location must be distinct for a problem to be solvable.

Definition 1 *(Masehian and Nejad 2009) A junction is any node in T with 3 or more neighbours.*

Definition 2 *(Masehian and Nejad 2009) Two junctions are "near" if there is no junction on the path between them.*

Tree Solvability Conditions

Based on the work in (Masehian and Nejad 2009) we have adapted the following three sufficient conditions for a tree, T, to be solvable for any configuration of agents:

1. T contains at least one junction.
2. There are at most $H - 1$ edges on the path between any node and the junction nearest to it.
3. The path connecting any two junctions that are near contains at most $H - 2$ edges.

Definition 3 *Let v be any node in T and S any set of nodes in T. $Trees(v, S)$ is defined as the set of trees that result when v is removed from T, excluding any such tree that contains one or more nodes in S.*

Definition 4 *Let v be any node in T and S any set of nodes in T. $Holes(v, S)$ is defined to be the total number of holes in $Trees(v, S)$.*

Definition 5 *Let u and v be any two nodes in T. $E(u, v)$ is the number of edges on the path from u to v in T.*

Note: If $u \neq v$ then $E(u, v)$ is also the number of nodes on the path if you include one endpoint and exclude the other. That's how we are going to think of it in most of what follows.

Definition 6 *Let u and v be any two nodes in T. $H(u, v)$ is the number of holes on the path from u to v in T excluding u.*

Note: $H(u, v)$ and $H(v, u)$ will differ by one if one of the endpoints is a hole and the other one isn't.

Definition 7 *Let u and v be any two nodes in T. $A(u, v)$ is the number of "agents" on the path from u to v excluding u. This is equal to $E(u, v) - H(u, v)$.*

Lemma 1 *Let x and y be any two adjacent nodes in T, and J any junction nearer to y than to x such that $Holes(J, \{x, y\}) \geq A(y, J) + 2$. Then there exists a sequence of moves that, when completed, swaps the contents of x and y and leaves the contents of all other nodes unchanged.*

Proof. Figure 1 depicts the general situation, where y and J are distinct. There are $E(y, J)$ nodes between y and J (including J but not y) and J has at least two neighbours, which we will refer to as "ports," A and B, in addition to the node on the path between y and J. Let T' be the tree rooted at J excluding the subtree that contains x and y. By the premise of the lemma, T' contains at least $A(y, J) + 2$ holes. This is enough holes to clear all the nodes between y and J (including J but not y) and to clear A and B as well,

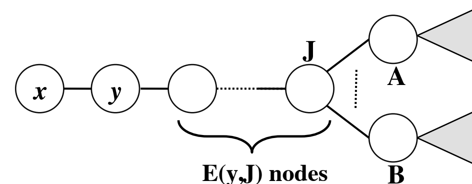


Figure 1: This arrangement of holes allows x and y to be swapped (*cf.* Lemma 1). Any node not labelled with a letter inside it is considered to be a hole. There may be more nodes to the left of x; they are irrelevant so are not shown.

[1]Please note that this experiment was conducted in two different environments on systems with different processing powers and so the results are not conclusive. It is also worth noting that their results were comparable to our initial implementation before our algorithm was refined to produce shorter results.

since in the starting configuration there are $A(y, J)$ agents on the path from y to J (including J but not y). A sequence of moves S that establishes the configuration in Figure 1 is constructed as follows. Let A and B be any two neighbours of J. Move holes within T' as necessary so that there are at least $A(y, J) + 2$ holes in the A and B subtrees combined and at least one hole in each. Now move these holes into the nodes on the path from y to J, starting with the node closest to y and finishing with J, being sure to always leave at least one hole in each of the A and B subtrees. Finally, move a hole from A's subtree into A and a hole from B's subtree into B. All this can be done without disturbing x and y since the holes are all being drawn from and moved to the part of the tree on the opposite side of y from x. This establishes the configuration shown in Figure 1. Let S be the sequence of moves executed thus far. Now move y into A and x into B, and then move y to the node in which x began and then x to the node in which y began. x and y have swapped positions. All that remains is to reverse sequence S to get all the nodes that have been disturbed by S back into their initial positions. S did not disturb x and y when it was executed, so reversing S will not disturb them either.

When $y = J$ the above reasoning does not apply since the sequence S may move y out of J. This case divides into three subcases.

1. If J has two neighbours (not counting x), A and B, whose subtrees each contain at least one hole then, just as we did above, we can clear A and B using those holes without disturbing x and y (let S' be the sequence that does this), move y into A and x into B, and then move y to the node in which x began and then x into J. x and y have swapped positions, and A and B are clear. We can now reverse S' to restore the A and B subtrees to their original configuration without disturbing x and y.

2. If there is at least one hole on the opposite side of x from y, it can be moved to J. Let S' be the sequence that does this. This keeps x and y adjacent and does not decrease $Holes(J, \{x, y\})$, but y is no longer in J so we can apply the general case to swap x and y. When this is finished J will be clear and when we apply S' in reverse x will move into J, y will move into the node that originally contained x and the other nodes that were disturbed to get this hole into J will be restored to their original configuration.

3. If neither of the above is possible it means all H holes are in one subtree rooted at a neighbour of J other than x. This subtree must contain a junction because it contains at least H nodes and T has the property that there are at most $H - 1$ edges on the path between any node and the junction nearest to it. Let J' be the junction in this subtree that is nearest to J. J' is obviously nearer to y than x and does not contain y so if we can establish that $Holes(J', \{x, y\}) \geq A(y, J') + 2$, then we can use this junction and the method in the general case to swap x and y. T has the property that the path connecting any two junctions that are near contains at most $H - 2$ edges. J and J' are near, so this property ensure that $E(y, J') \leq H - 2$, and therefore $A(y, J') + 2 \leq H - H(y, J')$. But $H - H(y, J')$ is equal to $Holes(J', \{x, y\})$ because all H holes are in this subtree, so $H - H(y, J')$ of them are in $Trees(J', \{x, y\})$. Thus we have established that $Holes(J', \{x, y\}) \geq A(y, J') + 2$ and can use J' to swap x and y.

□

Definition 8 *Let u and v be any two adjacent nodes in T. $J(u, v)$ is the junction in T, if it exists, that is closest to u and closer to u than to v.*

Lemma 2 *Let u and v be any two adjacent nodes in T. Then at least one of $J(u, v)$ and $J(v, u)$ exists.*

Proof. Let J be the junction closest to u. This must exist because T contains at least one junction. If J is closer to u than to v then $J(u, v)$ exists (because J satisfies the definition of $J(u, v)$). If J is closer to v than to u then $J(u, v)$ does not exist but $J(v, u)$ does (because J satisfies the definition of $J(v, u)$). □

Theorem 3 *Let u and v be any two adjacent nodes in T. Then there exists a junction J such that at least one of the following holds:*

1. $Holes(J, \{u, v\}) \geq A(u, J) + 2$
2. $Holes(J, \{u, v\}) \geq A(v, J) + 2$

Proof. From Lemma 2 we know that at least one of $J_u = J(u, v)$ and $J_v = J(v, u)$ exists. We will show that at least one of these satisfies the requirements for J in the Theorem statement.

We will begin with the general case, shown in Figure 2, where both J_u and J_v exist, $u \neq J_u$, and $v \neq J_v$. We simplify our notation to make the following proof simpler. Thus K_1, H_1, K_2, H_2, W_1, and W_2 in the figure are $E(u, J_u), H(u, J_u), E(v, J_v), H(v, J_v), Holes(J_u, \{u, v\})$ and $Holes(J_v, \{v, u\})$ respectively. So to prove the theorem is to prove that either $W_1 \geq (K_1 - H_1 + 2)$ or $W_2 \geq (K_2 - H_2 + 2)$.

J_u and J_v are near junctions, so there are at most $H - 2$ edges on the path connecting them, *i.e.,* at most $H - 1$ nodes including both J_u and J_v. Hence $K_1 + K_2 + 2 \leq H - 1$, *i.e.,* $K_1 + K_2 + 3 \leq H = H_1 + H_2 + W_1 + W_2$. Hence $3 + (K_1 - H_1) + (K_2 - H_2) \leq W_1 + W_2$.

Now suppose that the theorem is false, *i.e.,* that $W_1 \leq (K_1 - H_1 + 1)$ and $W_2 \leq (K_2 - H_2 + 1)$. This implies $W_1 + W_2 \leq (K_1 - H_1 + 1) + (K_2 - H_2 + 1) = 2 + (K_1 - H_1) + (K_2 - H_2)$, contradicting the fact we just derived, $W_1 + W_2 \geq 3 + (K_1 - H_1) + (K_2 - H_2)$.

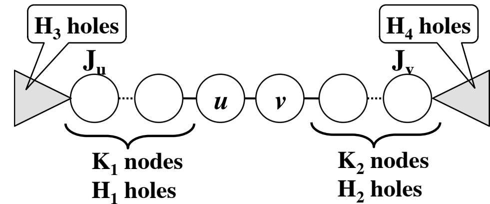


Figure 2: The general case analyzed in Theorem 3.

The same reasoning applies when either $u = J_u$ or $v = J_v$ (or both) since these are the cases $K_1 = H1 = 0$ and $K_2 = H_2 = 0$ respectively.

The final case to consider is when one of the junctions, say J_u does not exist. In this case we have $W_1 = 0$ and a total of $K_1 + K_2 + 2$ nodes from the "leftmost" node on this chain, X (which can be visualized as J_u in Figure 2 with the understanding that the subtree to its left is empty), to J_v (including X, J_v, u and v) which means $K_1 + K_2 + 1$ edges. We know the number of edges on the path from X to J_v (the junction nearest to X in this case) cannot exceed $H - 1$ (this is one of the properties we have assumed of T), so $K_1 + K_2 + 2 \leq H = H_1 + H_2 + W_2$. This implies that $W_2 \geq 2 + (K_1 - H_1) + (K_2 - H_2)$. Since $(K_1 - H_1) \geq 0$ we are guaranteed that $W_2 \geq 2 + (K_2 - H_2)$ as required by the theorem. □

Corollary 4 *Let u and v be any two adjacent nodes in T. Then there exists a sequence of moves that, when completed, swaps the contents of u and v and leaves the contents of all other nodes unchanged.*

Proof. Use Theorem 3 to obtain a junction J. If case (1) of Theorem 3 holds, invoke Lemma 1 with J, $x = v$ and $y = u$; otherwise invoke Lemma 1 with J, $x = u$ and $y = v$. □

Lemma 5 *The number of moves described in Corollary 4 is polynomially bounded.*

Proof. Finding the correct closest junction, as required by Theorem 3 and Lemma 1 requires looking at no more than n nodes. Swapping elements, as described in the general case of Lemma 1, can require moving at most n holes distances n before and after the swap, or $2n^2$ total moves. The other cases of Lemma 1 have the same bound. Swapping x and y can take no more than $4n$ moves. Thus, the number of moves required to swap two agents is bounded by $O(n^2)$. □

Theorem 6 *If we have the ability to swap any two agents in polynomial time, we can solve any multi-agent problem in polynomial time.*

Proof. There is a path of at most length n between any agent's start and goal location. Therefore, any individual agent can be moved from its start to goal location with no more than n swaps with a neighbor. There are m total agents, with $m < n$, so any problem can then be solved in $O(n^4)$ time. □

This bound also applies to the maximum solution length, meaning that solutions will be no larger than $O(n^4)$ moves long. For completeness, the full algorithm follows:

- Starting at the leaves of the tree and working inward, find the first goal state that is not occupied by its target agent.
- Move this agent to its goal by swapping it with any agents along the path to the goal.
- Repeat until all agents are on their goal states.

The swapping algorithm is:

- Find the nearest junction that has enough holes to allow the agents to swap at that junction.
- Move all holes towards the agent until the junction can be used for the swap.
- Swap the agents.
- Return the holes to their original positions, restoring all other agents as well.

Worked Example

To illustrate how TASS works, we will go over the example shown in Figure 3. Suppose we want to swap agents u and v on the original tree (a). The first step would be to identify a junction that meets the conditions of Theorem 3. The first such junction we have is the one currently occupied by u. This meets the conditions of the theorem, however, it is not safe to do the swap as it falls under the third category of Lemma 1. Following the reasoning in the proof, we use another junction that is safe, as labelled in Figure 3(a), and use it to swap the agents instead.

The first step of TASS is to clear the junction and two of its ports[2]. To do this TASS moves agents 1, 2, and 3 out of the way and end up with the configuration in Figure 3(b). TASS then clears the path between u and the junction by importing the hole in the lower left of the graph[3] and we get the configuration in Figure 3(c). Now that the junction, the path to it, and two of its ports are clear we can do the swap. As shown in Figure 3(d) we swap by moving u into one port, v into the other, and then pull them out in reverse order to obtain the configuration shown in Figure 3(e). The final step is to return the other agents to their original locations from the beginning of the procedure by reversing their actions. This achieves the final state in Figure 3(f).

Slideable Induced Trees

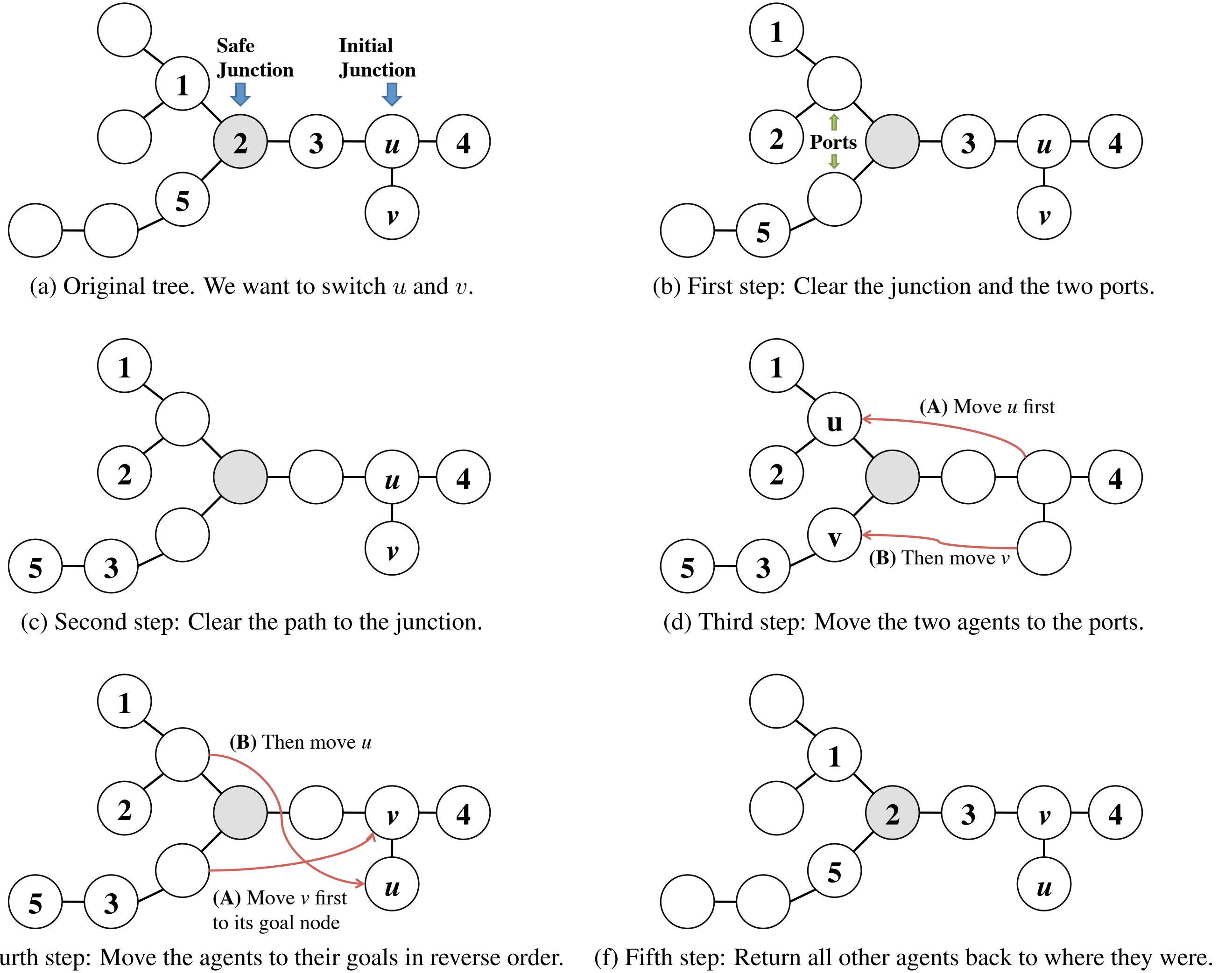
The long term goal of our work is to solve multi-agent problems on as general graphs as possible. Our first step in this direction is solving a class of problems that (Wang and Botea 2009) called SLIDEABLE. This class of problems was shown to be solvable in polynomial time. The original definition for SLIDEABLE problems given in (Wang and Botea 2009) can be rewritten as follows:

Definition 9 *A problem is* SLIDEABLE *if the following conditions hold:*

1. *A full path, call it P_i, exists for each agent i from its starting location to its destination.*
2. *For each three consecutive steps a, b, and c on the path P_i, an alternative path, Ω_{iac}, exists from a to c that does not pass through b.*
3. *The first step on P_i is vacant.*
4. *No destination for any agent will be on any P_i where i is a different agent.*
5. *No destination for any agent will be on any of the alternative paths for any of the agents.*

[2]The exact order of which to clear first is not significant, since holes can be easily moved around the graph.

[3]Note that when we move a hole, agents on its path occupy the same nodes before and after the move, with the exception of the node where the hole started its movement and the node where the hole ended.



(a) Original tree. We want to switch u and v.
(b) First step: Clear the junction and the two ports.
(c) Second step: Clear the path to the junction.
(d) Third step: Move the two agents to the ports.
(e) Fourth step: Move the agents to their goals in reverse order.
(f) Fifth step: Return all other agents back to where they were.

Figure 3: Example run of TASS

Lemma 7 *We can always decompose a* SLIDEABLE *problem into an induced tree that is guaranteed to be solvable.*

Proof. The GTD-SLIDABLE algorithm creates trees as follows. Create the induced tree by adding all the nodes and edges on P_i for $i = 1 \to m$ and breaking any cycles that are created arbitrarily. We will prove the solvability for the general case where we have at least 3 junctions or more. If we have less than 3 junctions, then exhaustive analysis has shown that edges on the main path can be swapped with edges from the alternative paths to either create new junctions or reduce distances to existing junctions to make the problem solvable.

Let ℓ be the number of leaves. By the definition of SLIDEABLE and the GTD-SLIDABLE algorithm every goal node must be on a leaf which is initially vacant, and at least one extra node is vacant[4]. This implies that $\ell \geq m$, so $H = n - m \geq n - \ell$. Now, consider the three conditions required for tree solvability:

Condition 1: (*T contains at least one junction.*) This is satisfied trivially since we are assuming we have 3 junctions.

Condition 2: (*There are at most $H - 1$ edges on the path between any node and the junction nearest to it.*) Pick any node q and near junction J. $E(J, q) \leq n - \ell + 1 - 3$. On the right-hand side of the equation we add 1 because q can be a leaf node. We subtract 2 for the two junctions that cannot be on the path and an additional 1 because the left-hand side is expressed in edges instead of nodes. Since $H \geq n - \ell$, $E(J, q) \leq n - \ell - 2 \leq H - 2$, which is stronger than necessary.

Condition 3: (*The path connecting any two junctions that are near contains at most $H - 2$ edges.*) Because we have at least three junctions in total, $\ell \geq 5$. For all near junctions J_a and J_b, $E(J_a, J_b) \leq n - \ell - 1 - 1$. This is because

[4]An exception is if every agent is adjacent to its goal node, in which case the problem is trivially solved by moving every agent directly to its goal.

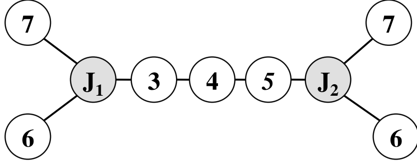


Figure 4: A problem which is not slideable, but can be solved by TASS. Agents at J_1 and J_2 need to swap positions.

(1) leaves cannot be on the path between junctions (2) at least one other junction will not be on the path between J_a and J_b and (3) $E(J_a, J_b)$ counts edges which is one more than nodes. We have already established that $H \geq n - \ell$ so $E(J_1, J_2) \leq n - \ell - 2 \leq H - 2$. This establishes that the distance between any two near junctions is less than or equal $H - 2$. $\square$

Note that the SLIDEABLE class of problems is a strict subset of the problems that our thee algorithm solves. Figure 4 shows an example that is not SLIDEABLE because there are no alternate paths, yet this can easily be solved by TASS.

Experimental Results

We performed a number of experiments to better understand the practical performance of our algorithm. Our experiments were primarily performed on binary and ternary trees, as shown in Figure 5. These trees have branching factors of 3 and 4 respectively, except for the leaves and the root of the tree. They grow exponentially in the depth, so they are different from the trees that could be induced from grid maps; however, 4-connected grid maps will have a branching factor similar to these trees, as each state in a grid has at most four neighbors.

Our initial experiments were very promising on their own as we were able to solve problems with trees of 1000 nodes in 50 minutes which is far better than any of the existing approaches. However, after performing a first round of optimizations on our algorithm we achieved more than 300 times faster performance and solution lengths were divided in half. These results further strengthens the applicability of TASS to real world applications.

In this section, we begin with scalability experiments and then look at the distribution of solution lengths on fixed size trees.

Scaling Experiments

A number of experiments were conducted to measure the scalability of TASS. In each experiment trees of increasing sizes were systematically populated with agents with which they can still be solvable. The experiments were run on a 2.66 GHz Q8400 processor.

To make the experiments easily replicable, we have incrementally built binary and ternary trees starting with trees of size 6 all the way up to 1000 in increments of one node. For a tree of size n, there are $m = n - 4$ agents in the tree.

The tree nodes are labelled starting from 0 at the root all the way to TreeSize-1. We increment the node labels by the level; so the first level only has the 0 node which is connected to nodes 1 and 2 on the second level. Then on the third level we have 3 and 4 which are connected to node 1 and 5 and 6 to node 2, and so on.

The agents start on nodes TreeSize-1 through node 4 and their destinations span the nodes from 0 to TreeSize-5. For example, in the binary tree in Figure 5, there would be three agents that start on locations 6, 5 and 4 with respective goals of 0, 1 and 2.

In the results below, for binary and ternary trees, two pairs of graphs are presented to show the relationships between the number of nodes vs. computation time and between the number of nodes vs. solution length.

Binary Trees Results on binary trees are in Figure 6 which shows planning time (left) and overall plan length (right). In this experiment we could solve a tree of 267 nodes in less than a second. Our largest tree of size 1000 nodes was solved in a bit below 22 seconds. It is important to note that these trees are almost completely filled with agents, conditions under which most, if not all, existing algorithms would fail.

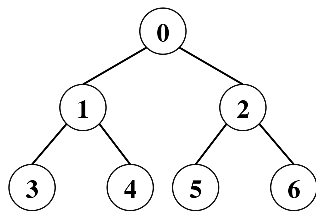
The solution sizes are, as expected, quite large. These problems are hard in nature and only some attempts have been made to optimize the solution lengths. In future work, we intend to further improve the suboptimiality of the solutions. In the smallest tree, our algorithm found the optimal solution of length 3, while the largest tree of 1000 nodes had a solution plan of 663,056 moves. As it could take up to $O(n^2)$ actions just to swap two agents, this is clearly much better than the worst possible case.

Ternary Trees The results for the ternary trees, shown in Figure 7, share the same characteristics as the results of binary trees, except that solution lengths and runtimes are lower. The solution lengths are shorter because with the increased branching factor all the nodes are closer together.

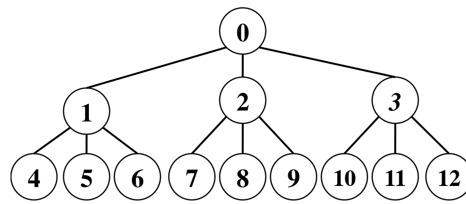
Our largest ternary tree also had 1000 nodes and 996 agents. This took about 8 seconds and 378,744 moves to solve, which is approximately three times faster and with half the solution length required to solve the 1000-node binary tree.

Plan length distribution

Since it is possible that our experiment problems were particularly easy or hard, we wanted to get a better understanding of the distribution of the solution lengths over a fixed tree size. To do this, we fixed the size of two ternary search trees, one with 14 nodes and the other with 40 nodes. On the smaller tree we ran TASS on all permutations of destinations with 10 agents on the tree. On the larger tree, we ran TASS on 2,024,401 problems with 36 of the 40 nodes occupied. Results are in Figure 8. The x-axis is the plan length, while the y-axis is the frequency of paths of that length in thousands.

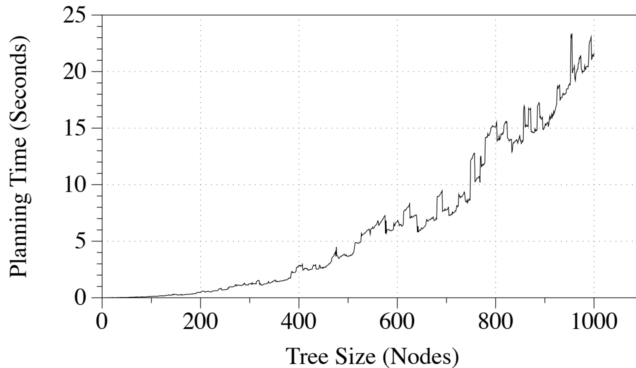


Binary Tree



Ternary Tree

Figure 5: Sample of the trees used in the experiments.



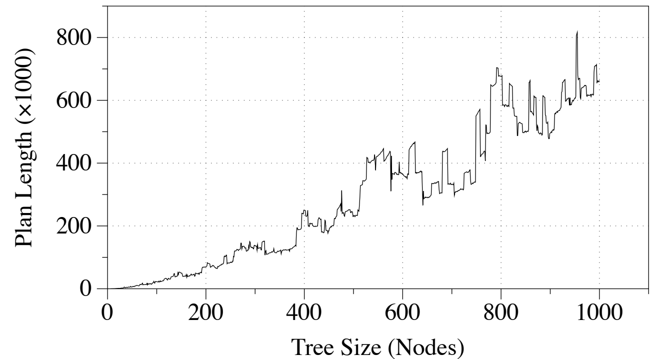


Figure 6: Timing and solution-length results on binary trees.

The distributions appears to be heavy-tailed. It is an open question as to whether this is the result of the optimal solutions being skewed, or the result of how TASS handles different problem instances. Certain aspects of our implementation hint to the latter, as some portions of the algorithm are still unoptimized and would negatively affect the solution lengths in certain problem instances.

Conclusions and Future Work

In this paper we have presented a new algorithm which can solve multi-agent pathfinding problems on trees. We have demonstrated that this algorithm runs in polynomial time, and shown that in practice it can scale up to thousands of agents.

This work is just one step in classifying problems which can be solved in polynomial time. The next step would be expanding the family of GTD algorithms to decompose more general graphs into trees and optimize the trees generated for shorter solutions. This will help classify what graph properties make multi-agent pathfinding problems truly difficult. Another future improvement is to refine our approach to make use of edges that were removed through the decomposition in a controlled manner to significantly improve solutions on graph problem. Furthermore, we intend to study the optimality of the solutions created by TASS on smaller problems for which optimal solutions can be computed.

References

Ryan Luna and Kostas E. Bekris. Push and swap: Fast cooperative path-finding with completeness guarantees. In *International Joint Conference on Artificial Intelligence*, 2011.

Ellips Masehian and Azadeh Hassan Nejad. Solvability of multi robot motion planning problems on trees. In *IROS*, pages 5936–5941, 2009.

D. Ratner and M. K. Warmuth. Finding a shortest solution for the N $\times$ N extension of the 15-puzzle is intractable. In *National Conference on Artificial Intelligence (AAAI-86)*, pages 168–172, 1986.

Malcolm R. K. Ryan. Exploiting subgraph structure in multi-robot path planning. *J. Artif. Int. Res.*, 31:497–542, March 2008.

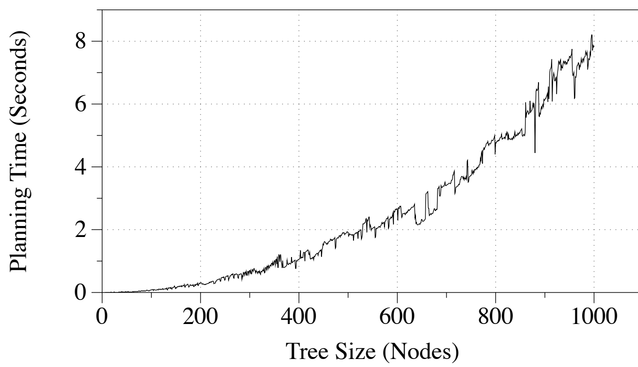
Malcolm Ryan. Constraint-based multi-robot path planning. In *ICRA*, pages 922–928, 2010.

David Silver. Cooperative pathfinding. In *AIIDE*, pages 117–122, 2005.

T. S. Standley. Finding optimal solutions to cooperative pathfinding problems. In *AAAI*, 2010.

Pavel Surynek. Making solutions of multi-robot path planning problems shorter using weak transpositions and critical path parallelism. In *Proceedings of the 2009 International Symposium on Combinatorial Search*, 2009.

Ko-Hsin Cindy Wang and Adi Botea. Fast and memory-



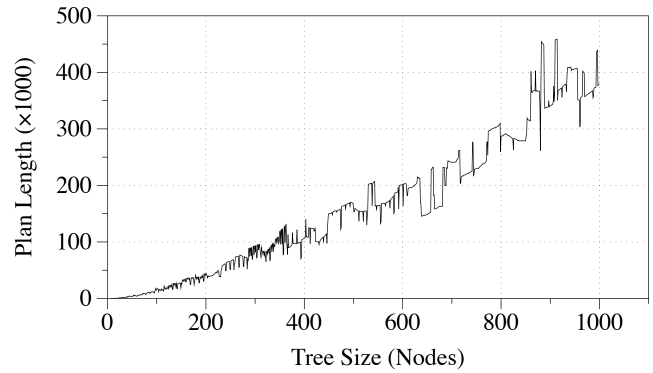
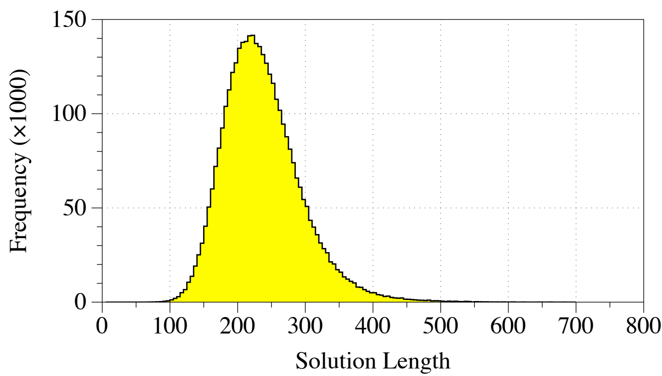
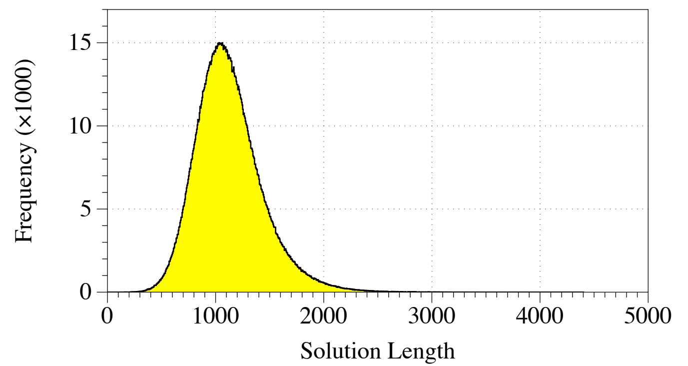


Figure 7: Timing and solution-length results on ternary trees.



Solution length distribution over all 14 node trees with 10 agents.



Solution length distribution over 2,024,401 random 40 node trees with 36 agents.

Figure 8: Distribution of solution lengths on 14- and 40-node trees.

efficient multi-agent pathfinding. In *ICAPS*, pages 380–387, 2008.

Ko-Hsin Cindy Wang and Adi Botea. Tractable multi-agent path planning on grid maps. In *IJCAI'09: Proceedings of the 21st international Joint conference on Artificial intelligence*, pages 1870–1875, San Francisco, CA, USA, 2009.