

Path Planning for Altruistically Negotiating Systems: The Near-sighted Tarzan Algorithm

Arthur W. Mahoney
Dept. of Computer Science
Utah State University, Logan UT, 84322
awmahoney@cc.usu.edu

Daniel W. Watson
Dept. of Computer Science
Utah State University, Logan UT, 84322
dan.watson@usu.edu

Abstract—This paper introduces a variant of the Rapidly-exploring Random Leafy Tree, called the Near-sighted Tarzan Algorithm, that performs path planning (for message routing) in a discrete graph simulating the communication network of an Altruistically Negotiating System. The Near-sighted Tarzan Algorithm is designed to operate in environments with similar characteristics as an ad hoc network. These characteristics include an unordered topology, large quantities of agents, network connectivity, and limited knowledge of the network by any given agent. Comparison tests with Dijkstra’s algorithm as a benchmark show the Tarzan algorithm to be superior when searching for paths between large distances. The Tarzan algorithm can be easily run in parallel making it suitable for use in an Altruistically Negotiating System. This paper introduces a variant of the Rapidly-exploring Random Leafy Tree, called the Near-sighted Tarzan Algorithm, that performs path planning (for message routing) in a discrete graph simulating the communication network of an Altruistically Negotiating System. The Near-sighted Tarzan Algorithm is designed to operate in environments with similar characteristics as an ad hoc network. These characteristics include an unordered topology, large quantities of agents, network connectivity, and limited knowledge of the network by any given agent. Comparison tests with Dijkstra’s algorithm as a benchmark show the Tarzan algorithm to be superior when searching for paths between large distances. The Tarzan algorithm can be easily run in parallel making it suitable for use in an Altruistically Negotiating System.

I. INTRODUCTION

Rapid development of the Unmanned Autonomous Vehicle (UAV) has made them increasingly prevalent in the workplaces and battlefields of today [1]. Because multiple UAV’s are often required to operate together in the same environment, considerable study is being focused on multi-robot systems [2]. Of particular interest is the Altruistically Negotiating System (ANS). An ANS is considered a collective of robots or software agents that behave in a manner that is in the best interest of the system while executing their individual tasks [6]. A common form of the ANS, is a network of UAV’s who strive to achieve a common objective by sharing available resources. These resources are often in the form of idle computational resources. Robots are frequently called upon to operate in extremely dynamic, high-dimensional, and hostile environments. In order for a group of robots to participate in an ANS, communication is of the essence. If the members of an ANS are equipped with gear only enabling line-of-sight communication, obstacles in the hostile environment (e.g., buildings and landscape) can

cause the breakdown of effective communication. Maintaining a network of communication between members of the system can be a challenge, not only because of the mere presence of obstacles, but also because each member cannot be expected to know the location of every one of its peers.

This paper proposes a new path planning algorithm, based on the Rapidly-exploring Random Leafy Tree, for the communication network of an Altruistically Negotiating System. The algorithm, dubbed the Near-sighted Tarzan Algorithm¹, locates a path of agents through which a message from a source can be forwarded to its destination. The Near-sighted Tarzan Algorithm performs with the stipulation that each agent of the system has minimal knowledge of the communication network. By reducing the amount of information needed at each iteration of the algorithm to localized knowledge, the result is an easily parallelized search algorithm that determines paths in ad hoc networks. The objective of this study is to introduce the Tarzan algorithm and show its capability for near-optimal path planning in undirected graphs with similar restrictions that may be imposed while operating in an ad hoc network. This paper does not address the development of a communication protocol associated with the Tarzan algorithm. The rest of this paper is organized in the following manner: section 2 provides a brief introduction to the Rapidly-exploring Random Tree as well as the Rapidly-exploring Random Leafy Tree. Section 3 introduces the Near-sighted Tarzan algorithm. A series of experiments illustrating the Near-sighted Tarzan Algorithm’s potential is presented in section 4, and concluding remarks are given in section 5.

II. BACKGROUND

The ANS communication network can be simulated with a randomized discrete graph whose vertices are agents in the system and whose edges are represented by an established line of communication between two agents. The edges are discrete, with equal weights, due to the fact that the distance between two agents is irrelevant so long as messages can be passed between them. There exist many path planning algorithms capable of optimal and near-optimal solutions in discrete spaces [8]. Two common methods are Dijkstra’s Algorithm and the A* algorithm [7], [8].

¹Pseudo Rapidly-exploring Random Leafy Tree was deemed too cumbersome.

Recently, a new sampling-based method, known as the Rapidly-exploring Random Tree (RRT), has been proposed and adapted to discrete spaces. The RRT has been found to be useful for planning in highly dimensional, continuous configuration spaces with both global and differential constraints as well as planning and control of hybrid-systems [3]. The RRT is a stochastic data structure that explores the configuration space by randomly sampling points from the space and pulling the tree towards them in an iterative manner. RRTs have been adapted for use in a discrete configuration space in the form of the Rapidly-exploring Random Leafy Tree (RRLT) [4], [5].

The growth of an RRT begins with the initial configuration, q_{init} . At each iteration a configuration, q_{rand} , is randomly sampled from the given configuration space. Using a metric, q_{near} , the nearest configuration already on the tree to q_{rand} is found. A new configuration, q_{new} , is then created such that q_{new} minimizes the distance between q_{near} and q_{rand} and it does not violate any given global nor differential constraints. The configuration, q_{new} , is then added as a vertex to the tree along with an edge that is defined by q_{near} and q_{new} . This process will be referred to as ‘expansion’ and it continues until a given goal state has been reached, or the size of the tree exceeds a given limit.

The RRT has two distinct advantages over existing randomized sampling based planners such as the Probabilistic Road Map technique (also known as the Monte-Carlo method). The first notable advantage is the fact that the RRT is naturally biased to expand towards the unexplored configuration space [3], [4], [5]. This becomes clear when examining the probability that a given configuration already on the tree will be selected as q_{near} during expansion. If each configuration on the tree is assigned a Voronoi region, those with larger regions will be more likely to contain q_{rand} and thus be selected as q_{near} . During growth, the largest Voronoi regions are found on the outside areas of the tree that contain large amounts of unexplored space. Therefore, the tree is more likely to expand outwards in the direction of the empty space [5].

The second advantage of the RRT over existing methods is its tendency to search the configuration space in a uniform and consistent manner [3]. When a vertex is added to the tree, the Voronoi region of q_{near} is divided between itself and q_{new} . During growth, the largest Voronoi regions are repeatedly divided into smaller ones [3]. As the tree increases in size, the large Voronoi regions are decreased until they approximately cover the same area. When the mean of the configuration’s Voronoi regions converges, then the RRT has uniformly covered the space [3]. At this point, the probability of any one vertex on the tree being selected as q_{near} becomes the same.

The RRT algorithm has been adapted for use in a discretized configuration space in the form of the Rapidly-expanding Random Leafy Tree (RRLT). In a discrete space, the RRT method, without modification, experiences setback during expansion. Due to the discrete space, there is potential that q_{new} cannot be created on a line between q_{near} and q_{rand} . This can yield a q_{new} that is, in fact, farther away from q_{rand}

than is q_{near} . Therefore, there is no guarantee that the RRT will grow in the most efficient manner possible [4], [5]. The Rapidly-expanding Random Leafy Tree mitigates this issue. The RRLT keeps a list of configurations, known as ‘leafs,’ that can be reached in only one step from the existing tree. After q_{rand} is sampled, the closest leaf, q_{new} , is found in terms of the given metric and is then added to the tree. Naturally, following expansion the list of leafs must be updated to incorporate q_{new} ’s leafs and q_{new} must be removed. In Discrete spaces, the RRLT has been shown to outperform the RRT in terms of speed and accuracy [4].

Both the RRT and RRLT algorithms search an entirely known configuration space where a distance metric can be easily defined. The RRT/RRLT cannot be used for path planning in the graph simulating the communication network of an ANS. Because the topology of the network is often unordered, it is difficult to define a metric where distance can be easily found. Also, when growing a tree, the process has full knowledge of the configuration space. While an agent is performing path planning in a communication network, it may be unreasonable to expect it to know the location of all other agents on the network.

III. THE NEAR-SIGHTED TARZAN ALGORITHM

Path planning will be explained in general terms as the search for a continuous path of configuration states through a metric space, X , from an initial configuration, $q_{init} \in X$, to a final set of configurations, $q_{goal} \subset X$. The path is defined in terms of an ordered list of inputs to a state transition equation, $x' = f(x, u)$. The transition equation maps an initial configuration, $x \in X$, and an action u , to a final configuration, $x' \in X$. For example, in a configuration space defined as a discrete grid, $f(x, u)$ would map a given cell (x) and an action (u) to another cell (x') in the grid. The action, u , could consist of moving one space above/below or to the left/right of x . The resulting configuration, x' , would be an immediately adjacent cell to x .

For this study, the communication network of an ANS will be simulated by a configuration space, X , that is defined by an unordered, random, bidirectional graph where distance is defined as the least number of edges separating two vertices. A vertex on the graph will represent an agent, while an edge will be considered a viable line of communication between two nodes. The existence of an edge between two vertices implies the ability of two-way communication between their respective agents. The set, $N(x)$ will contain all vertices connected to vertex $x \in X$. To simplify the problem, the weight of each edge will be discrete and all edges will have equal weight. The state transition equation will be defined by $x' = f(x, N(x))$ where $x, x' \in X$. The limited knowledge of each agent in the network will be simulated by banning the measurement of distance between nodes on the graph whose representative agents are unaware of each other. An existing infrastructure is assumed to exist which enables agents awareness of peer agents near themselves. Through this infrastructure, an agent will also have data regarding the distance, in edge traversals,

to each node of which it is aware. The infrastructure can be similar in form to the routing tables of DSDV [11], though such a brute force method is not needed. For the simulating graph, $U(x)$ is the set containing all nodes of which information can be obtained by node x via the infrastructure.

The construction of the Near-sighted Tarzan tree is executed in similar manner to the RRLT. The growth of the tree begins with the initial configuration, q_{init} . Expansion is performed by first sampling a random configuration. The random configuration, q_{rand} , is selected from the set $U(q_{tree})$ where the configuration, q_{tree} , is selected at random from configurations already on the tree. Following the selection of q_{rand} , from the lists of configurations in X that are located only one edge traversal from the existing tree, a 'leaf' configuration, q_{leaf} , is found that minimizes the distance between itself and q_{rand} . Finally, q_{leaf} is added to the tree. The tree is iteratively expanded until the goal configurations have been found.

The Tarzan tree differs from the RRLT only by the space from which q_{rand} is randomly sampled. The RRLT selects from the entire configuration space. This enables the bias towards unexplored regions due to the large Voronoi regions on the outside of the tree. The Tarzan tree, on the other hand, selects q_{rand} from $U(q_{tree})$. Because $U(q_{tree})$ represents the agents a given agent is aware of, it can be expected that its size will be much less than the entire configuration space. This reduces dependence on Voronoi regions biasing the search, and thus, decreases the tree's tendency to grow towards unexplored regions. However, by sampling from the region defined by $U(q_{tree})$ the problem of searching for q_{leaf} , the nearest leaf to q_{rand} is localized to the configuration space already known by the configurations on the tree. This guarantees at least one configuration on the tree (q_{tree}) will be aware of q_{rand} . Implicitly, q_{rand} will also be aware of q_{tree} and therefore, the task of locating q_{leaf} can be distributed to q_{rand} . Because finding q_{leaf} can be distributed to the configuration, q_{rand} , the tree can be parallelized without the OR paradigm method or the embarrassingly parallel approach [9].

IV. EXPERIMENTS AND RESULTS

In order to demonstrate the capabilities of the Near-sighted Tarzan Algorithm in the communication network of an ANS, experimentation was performed with several random graphs. As a benchmark for accuracy and speed, the Tarzan algorithm was compared against Dijkstra's shortest path algorithm [7]. Several characteristics of the random graphs were considered during experimentation. Because the communication network of an ANS can be rather large [10], the size of each graph, measured in vertices, is an important factor. To demonstrate the Tarzan algorithm's scalability, the algorithm was tested in graphs ranging in size from 100 to 1000 vertices. The sparseness of the network is an important consideration as well. ANS often operate in hostile environments where obstacles may disrupt network communication; therefore, the connectivity of the network may vary. The sparsity (which is equivalent to connectivity) of the random graphs are measured

in the average number of edges attached to each vertex. A low average number of edges per vertex denotes a very sparse graph, while a high average represents a very dense graph. The graphs in which the Tarzan algorithm was tested range in connectivity from an average of 3 to 11 edges per vertex. For all random graphs used in this experiment, a histogram of the connectivity is an approximately normal distribution. The final characteristic is the amount of data distributed across the network with the given information infrastructure. The amount of data provided by the infrastructure was denoted as $U(x)$ in the previous section. The Tarzan algorithm was tested with $U(x)$ ranging in size from including all vertices within 1 edge traversal to 3 edge traversals from x . In figures 1, 2 and 3, each simulated agent is aware of all agents within 3 edge traversals of itself.

To eliminate specificity, speed is measured in vertices visited by each algorithm rather than actual runtime. The number of vertices visited is proportional to the amount of overhead in a communication network while performing path planning with the Near-sighted Tarzan tree. Dijkstra's algorithm cannot be executed within the environment of the communication network (because all information must be known *a priori*); however, it is used as a simple benchmark to compare the Tarzan tree with similar breadth-first and wavefront methods. Figure 1 relates the number of vertices visited by the Tarzan and Dijkstra algorithms when searching for a path to the actual distance between an arbitrary source and destination. The graph used in Figure 1 is composed of 500 vertices with a connectivity of 5 edges per vertex. Because the Tarzan algorithm is stochastic, the vertices visited for any actual distance is seen as the average of 25 runs.

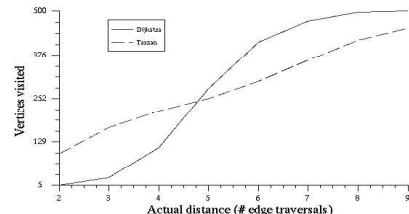


Fig. 1. Comparison of the nodes visited by the Tarzan algorithm and Dijkstra's algorithm.

The connectivity of the graph has an important affect on the performance of the Tarzan algorithm. Figure 2 shows the performance of the algorithm in terms of actual distance across several graphs each with a different connectivity. All graphs used for Figure 2 have 500 vertices with connectivity ranging from 5 to 13 edges per vertex. An increase in connectivity has an adverse effect on the algorithm's performance. As the average number of edges per vertex rises, the vertices visited does so as well.

The accuracy of a path found by the Tarzan algorithm is determined by comparing the number of edge traversals in the calculated path to that found by Dijkstra's algorithm. Dijkstra's algorithm is an excellent benchmark because it is guaranteed to find the shortest path between two vertices so

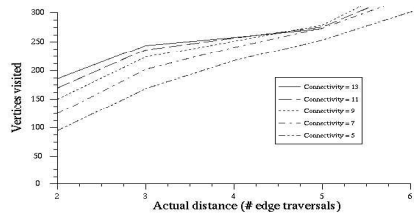


Fig. 2. Comparison of the nodes visited by the Tarzan algorithm with varying connectivity.

long as a path exists [7]. Figure 3 relates the actual distance, measured in edge traversals, between any two vertices and the average number of edge traversals in paths found by the Tarzan algorithm. The average is computed from the results of 25 runs. The graph used contains 1000 vertices with a connectivity of 5 edges per vertex. The difference in accuracy between Dijkstra’s algorithm and the Tarzan algorithm is approximately constant.

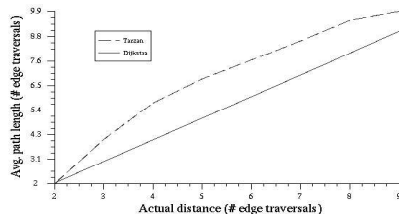


Fig. 3. Accuracy of the Tarzan algorithm.

Varying the amount of information known by each vertex (the sampling space described by $U(x)$) has a peculiar affect on the performance of the Tarzan algorithm. When $U(x)$ contains only vertices within 1 edge traversal from x , the algorithm behaves much like a random walk [12]. As the sampling space grows, the algorithm behaves in a similar manner to a RRLT. Figure 4 shows the performance, in terms of vertices visited, of the Tarzan algorithm operating with $U(x)$ containing all vertices within 1 edge traversal and 3 traversals from x . The connectivity of the graph used in fig. 4 has an average of 11 edges per vertex. Figure 4 demonstrates that increasing the amount of information known improves performance of the Near-sighted Tarzan Algorithm for large distances in high connectivity environments. However, when operating in low connectivity graphs, performance is hampered with increased knowledge. In graphs with low connectivity, the Tarzan algorithm performs no better than a random walk.

V. CONCLUSION

This paper introduced a new variant of the RRLT, dubbed the Near-sighted Tarzan Algorithm, that performs path planning in discrete graphs with similar restrictions that may be imposed while operating in the communication network of an Altruistically Negotiating System. A comparison study indicates, for large actual distances, the Tarzan algorithm performs better than Dijkstra’s algorithm in terms of the number of

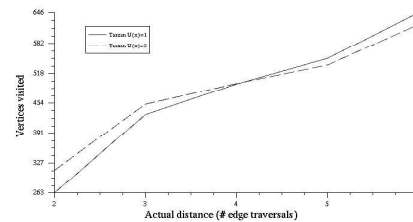


Fig. 4. Comparison of performance when varying information known.

vertices visited. Though for low connectivity environments the Tarzan algorithm performs similarly to a random walk, for environments with high connectivity it outperforms both Dijkstra’s algorithm and the random walk for large distances. Because the Tarzan algorithm operates within localized information, it can be easily parallelized, thus utilizing the possibly idle resources of peer agents participating in the ANS.

REFERENCES

- [1] P. Lerner. “Robots Go To War.” *Popular Science*, vol. 268, no. 1, pp. 42-96, 2006.
- [2] D. Guzzoni, et al. “Many robots make short work.” *AI Mag.*, vol. 18, no. 1, pp. 55-64, 1997.
- [3] S.M. LaValle. “Rapidly exploring random trees: A new tool for path planning.” TR 98-11, Comp. Sci. Dept., Iowa State University, Oct. 1998.
- [4] M.S. Branicky, M.M. Curtiss, J. Lavine, and S. Morgan. “Sampling-Based Planning and Control.” *Proc. 12th Yale Workshop on Adaptive and Learning Systems*, May 2003.
- [5] M.S. Branicky and S.B. Morgan. “Sampling-Based Planning for Discrete Spaces”. *Proc. IEEE/RSJ Int’l Conf. on Intelligent Robots and Systems*, Oct. 2004.
- [6] D. Devalarazu and D.W. Watson. “Path Planning for Altruistically Negotiating Processes.” *Proc. Int’l Symp. on Collaborative Technologies and Systems*, May 2005.
- [7] E.W. Dijkstra. “A note on two problems in connexion with graphs.” *Numerische Mathematik*. 1 (1959), pp. 269-271.
- [8] S.M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, 2006.
- [9] S. Carpin and E. Pagello. “On parallel RRTs for multi-robot systems.” *Proc. 8th Conf. of the Italian Assoc. for Artificial Intelligence*, Sept. 2002.
- [10] V.O.K. Li and Z. Lu. “Ad Hoc Network Routing.” *Proc. IEEE Int’l Conf. on Networking, Sensing, and Control*, March 2004.
- [11] C.E. Perkins and P. Bhagwat. “Routing over Multi-hop Wireless Network of Mobile Computers.” *SIGCOMM ’94: Computer Comm. Review*, Oct. 1994.
- [12] G. Barnes and U. Feige. “Short random walks on graphs.” *Proc. 25th ACM Symp. on Theory of Computing*, 1993.