2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
September 24–28, 2017, Vancouver, BC, Canada

Online RRT* and Online FMT*: Rapid Replanning with Dynamic Cost

Bryant Chandler[1] and Michael A. Goodrich[2]

Abstract— **Traditional path-planning involves (1) choosing start and goal points, (2) calculating a path, and (3) following that path. There are, however, many real world scenarios where an agent might need to change its goal and replan, which frequently includes expensively calculating a new path from scratch. We propose an adaptation to RRT* that locally rewires the RRT* tree as the robot moves and path costs change. Local rewiring takes advantage of information already existing in the tree and makes small adaptations that accommodate changes. Rewiring adds computational overhead during robot travel, but allows replanning in real time with approximately constant overhead. Empirical studies demonstrate that computational costs are lower than alternative replanners in 2D worlds with moderate obstacle density, and the resulting paths approach optimality as more time is allowed to replan.**

I. INTRODUCTION

There are many scenarios in which a robot might need to change goals in the middle of a task or adapt to changes in its environment. The naive approach would be to plan an entirely new path from scratch, but this is computationally expensive and does not take advantage of the information already learned about the configuration space. Replanners exist that can adapt for unexpected obstacles [11], [10], [7], [1], but they do not adapt to changing cost functions, and do not support replanning to new end points. Another replanner [4] improves the path as it travels, but it does not support changing cost functions or replanning to new end points. We propose two algorithms, Online RRT* (ORRT*) and Online FMT* (OFMT*), that adjust online as the environment and robot positions change. The algorithms facilitate (a) rapid replanning when goals change, (b) adapting paths when the cost function or environment changes, and (c) planning for multiple objectives; all while maintaining memory efficiency.

For this paper our distance unit will be the diameter of the robot. We assume that a robot moves at a constant rate of 0.15 units per timestep. A map is 4900 square units with randomly generated rectangular (non-overlapping) obstacles. Obstacle sizes range from 0.01 square units to almost as large as the map. We do not allow obstacles that take up the entire map, because it would become infeasible to plan paths. An example map can be seen in Figure 7. Note that we use rectangular obstacles for simulation, but our algorithms can function on an arbitrary occupancy grid.

ORRT* and OFMT* make two key additions to RRT*: (1) the location of the RRT* root changes to match the robot's location when the robot moves, and (2) new nodes are sampled up to a predefined density, and after that point *online sampling* only samples and rewires without adding new nodes.

We empirically validate the algorithms by comparing computation efficiency to FMT*, A* on a visibility graph, and A* on a grid. Additionally, we empirically verify that the algorithms can adapt to time-varying cost functions by comparing the resulting path cost to an approximately optimal "ground truth" path computed using PRM*.

II. RELATED WORK

This section describes three areas of related research: discrete-based planning, sampling-based planning, and replanning.

Discrete Planning. There are many algorithms in the literature for discrete multi-objective path planning, one of the most notable being A* [12] which is an alternative to Dijkstra's algorithm [2] when the goal is unknown. These algorithms build a graph in a configuration space that has been discretized into a grid with known available transitions between grid cells. They then perform search on those graphs. This allows the algorithms to operate quickly, but discretization reduces the likelihood of finding a truly optimal solution; the algorithms might not find paths through narrow passages and the location of grid cells might keep it from finding an optimal path. D* [11], AD* [10], and D* Lite [7] expand on Dijkstra and A* to get dynamic graphs that can handle unexpected obstacles, but they suffer from the same discretization issues.

Sampling-Based Planning. Sampling-based path planning addresses some of the problems created by discretizing the configuration space. Algorithms like RRT [8] and PRM [6] randomly sample a continuous configuration space, allowing for more path options with better performance than would be gained by simply increasing the resolution of discretization. RRT builds a rapidly exploring tree from random sampling and a cost function that can combine the multiple objectives. PRM builds a graph from randomly sampled points, and then paths are queried using another algorithm such as A*. The time complexity for constructing search trees is the same for both RRT and PRM, but RRT is $O(n)$ in query and space complexity whereas PRM is $O(nlog(n))$ in both query and space complexity; the reduced query time and memory requirements generally make RRT the favorite of the two. Additionally, RRT includes the ability to plan kinodynamically [9] (accounting for vehicle dynamics and velocity).

RRT has been expanded to RRT* [5], which guarantees asymptotic optimality as more points are sampled by

*This work has been funded by the Center for Unmanned Aircraft Systems (C-UAS), a National Science Foundation-sponsored industry/university cooperative research center (I/UCRC) under NSF Award No. IIP-1161036 along with significant contributions from C-UAS industry members.

rewiring the existing tree to get path cost reductions. An advantage of RRT* is that a sub-optimal path can be found quickly with a sparse tree. For the problems considered in this paper, a robot may be traversing a long distance and therefore the algorithm must be capable of running for a long time. Consequently, quickly generating a dense (and therefore closer to optimal) tree is more important than finding sub-optimal solutions on a sparse tree. FMT* [3] was designed to generate dense trees; FMT* constructs a tree with the same structure and optimality guarantees as RRT*, but samples all node locations before beginning, and builds out from the start point densely.

Replanning. Several solutions have been developed to allow RRT* to function in an online/anytime fashion. Anytime RRT* [4] takes advantage of the time it takes for a robot to travel a path by improving the path during travel. It does this by pruning the part of the tree that has already been traversed and continuing to sample. Anytime RRT* does not meet the requirements of this paper because it destroys part of the tree as the robot moves and our application benefits from reusing information that could be destroyed. Another notable solution is RRT^X [1], which switches the start and end points, so that the end point is now fixed and finding optimal paths to all other points in the space. Thus, when the start point moves, it can simply use one of the other paths. When obstacle conditions change, a cascading rewire is performed to update the tree. This is a reasonable approach to being online, but it still requires one of the points to be fixed, which is not the case for our application. We need to be able to move both the start and end point as the UAV moves, and we are less concerned about unexpected obstacles.

III. APPROACH

This section details the Online RRT* (ORRT*) and Online FMT* (OFMT*) algorithms. A tree-based path planning algorithm might plan a path as seen in Figure 1. The robot starts at the red square, and plans a path to the purple circle, but when it reaches the green circle the robot realizes that it actually wants to get to the blue point. The naive solution is to make a completely new plan to get to the new goal, but that is computationally expensive. If the robot can update the graph as it goes to take advantage of existing information, it will afford a shorter replanning time when the goal changes.

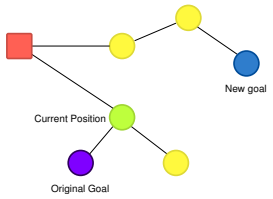


Fig. 1: Need to replan to new goal after the robot has moved.

A. Algorithm

ORRT* and OFMT* adapt to start and end point changes without needing to expensively build an entirely new tree. Instead, ORRT* and OFMT* reuse the existing tree with minimal new memory allocation and manageable processor utilization. In order to accomplish this, they depend on the asymptotic optimality guarantee of RRT*, which guarantees that the paths in the tree will approach optimality as the number of sampled nodes increases. The guarantee is possible, because every time a new node is added to the tree its neighbors are rewired, thus improving the tree to better fit the cost function. If we can rewire and improve the tree without linearly increasing the number of nodes, then we can adapt to a changing environment and changing cost functions while maintaining constant memory usage. A side benefit of using these tree-based algorithms is that the optimal path from the start point to every other point in the configuration space is embedded in the tree, so we can rapidly replan to new goals.

The problem is solved in three parts. The first part is *online sampling*, which varies slightly between ORRT* and OFMT* during initialization. ORRT* (see Algorithm 1) adds nodes to the tree until a "node add threshold" is reached. OFMT* (see Algorithm 2) samples nodes up to the "node add threshold", and then densely builds a tree with the sampled nodes. Once tree building is complete, both algorithms begin *online rewiring* by sampling and rewiring without adding nodes (see Algorithm 3). In both ORRT* and OFMT*, the online sampling and online rewiring approach allows the tree to re-optimize when the start point moves, or the cost function changes.

The second algorithm is *start-point moving* (see Algorithm 4), which (a) adds a node at the new start point and then (b) rewires appropriate neighbors to that point.

The third algorithm is *online pruning*, which balances for the newly added node by removing a leaf node in the vicinity of the new root.

The next three subsections discuss start-point moving, online sampling and rewiring, and online pruning, respectively.

Algorithm 1 ORRT* Online Sampling

```
1: procedure SAMPLEONLINE
2:     if number of nodes in tree < nodeAddThresh then
3:         sample and add a node as per RRT*
4:     else
5:         REWIREONLINE()
```

Algorithm 2 OFMT* Online Sampling

```
1: procedure SAMPLEONLINE
2:     if tree is not initialized then
3:         randomly generate nodeAddThresh nodes
4:         run FMT* to completion on sampled nodes
5:     else
6:         REWIREONLINE()
```

B. Start-Point Moving

RRT* and FMT* are able to handle changing the end point of the path, because they find the asymptotically optimal path to all of the points in the configuration space. What

Algorithm 3 Online Rewiring

```
1: procedure REWIREONLINE
2:     randPoint ← randomly sample a point
3:     neighbors ← all nodes in radius of randPoint
4:     nbr* ← best neighbor ∈ neighbors
5:     update cost from nbr* to its parent
6:     update cost for all children of nbr*
7:     for each nbr ∈ neighbors do
8:         potCost ← nbr*.tCost + cost(nbr*, nbr)
9:         if potCost < nbr.tCost then
10:            rewire nbr to nbr*
11:            update cost for all children of nbr
```

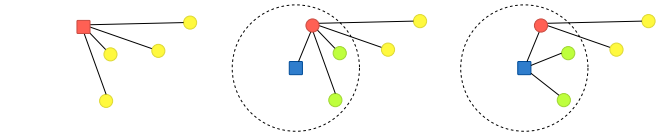
Algorithm 4 Move Start

```
1: procedure MOVESTART
2:     newRt ← new node at newStartPoint
3:     wire oldRoot to newRt
4:     update cost for all children of oldRoot
5:     neighbors ← all nodes within a radius of newRt
6:     for each nbr ∈ neighbors do
7:         potCost = newRt.tCost + cost(newRt, nbr)
8:         if potCost < neighbor.tCost then
9:             rewire neighbor to newRoot
10:            update weights for all children of neighbor
11:    for each nbr ∈ neighbors do
12:        if numNodes > nodeAddThreshold then
13:            if dist(newRt, nbr) < pruneDist then
14:                if nbr.isLeaf then
15:                    remove nbr
16:        else
17:            RETURN
```

they are not able to do is change the goal after the robot has started traveling; the end point has to remain fixed. Because a robot's goal can change after the robot starts traveling, the replanning algorithm must be able to move the start point and update the tree. As illustrated in Figure 2, ORRT* and OFMT* create a new node at the current location of the robot and make that the parent of the original start node. All nodes in the neighborhood of the new node are rewired subject to the constraint that a new edge does not intersect an obstacle. Empirical results have been omitted in the interest of space. These omitted empirical results are intuitive; the tree can be rewired quickly enough as long as the robot moves slowly (not too far between time samples) and continuously. Jumps larger than the rewire neighborhood would destroy the integrity of the tree.

C. Online Sampling and Rewiring

In order to re-optimize the tree after structure changes, the RRT* and FMT* algorithms need to continue sampling points and rewiring the tree. Naively continuing to sample new points in the configuration space does not work because the size of the tree continues to grow to the point that the algorithm becomes intractable in space and time. ORRT*



(a) Original tree with red square as start point.

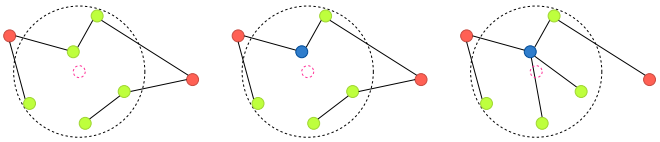
(b) New blue start node as parent of old red start node. Neighbors in green.

(c) Rewire green neighbors to blue new start point.

Fig. 2: Moving Start Point.

and OFMT* solve this problem by continuing to sample and rewire after we reach a chosen level of saturation (the total number of nodes required to "cover" a configuration space), but not adding new nodes to the tree. Instead of adding a new node to the tree, online rewiring is performed. A new sample is used as the center of a nearest neighborhood search; nodes within the neighborhood are rewired as shown in Figure 3. The lowest cost node from the neighbors near the new sample is selected, and all other nodes in the neighborhood are rewired to that best node as their parent, subject to the constraint that the new edge does not intersect an obstacle. This keeps the execution time per iteration approximately the same, if not less than it was when the algorithm reached the node add threshold. Memory utilization stays fixed at the threshold level since no new nodes are being added, and each iteration the tree improves to better fit the current cost function.

Online rewiring can refine the tree as long as the cost function is fixed because it will always try to reduce cost. If the cost function is dynamic, however, the cost along a path might need to increase to match the cost function. To support dynamic cost functions, we add 2 steps once the best neighbor of the sampled point has been found, but before rewiring other neighbors. First, we update the cost between the best neighbor and its parent. Second, we recursively propagate the new cost to all children of the best neighbor. This does not immediately make the whole tree match the new cost function, but it does shift the overall tree a little closer.



(a) Neighborhood of sampled point.

(b) Selecting best neighbor.

(c) Rewire other neighbors to best neighbor if it reduces their cost.

Fig. 3: Online Rewiring.

D. Online Pruning

By continually adding a new node when the robot moves, the start-point moving algorithm creates a problem, as illustrated in Figure 4. *Both ORRT* and OFMT* grow search*

trees based on a fixed number of nodes. Adding a new node each time the robot moves means that the number of nodes will increase linearly forever. Eventually, the number of nodes will become intractable in time and space. We could solve this problem by occasionally pruning the tree, but that would have an effect similar to how a garbage collector functions in software; it would have to pause execution occasionally in order to prune extra nodes. Instead of a "pause and prune" approach, we prune a single leaf node that is very close to the new root so that distant parts of the tree are not affected. This keeps the tree at a constant number of nodes without leaving holes (which would occur if we pruned branches or nodes far from the root). Because of the short distance, there is a reasonable chance that the leaf node removed will be the old root, but that is not required.

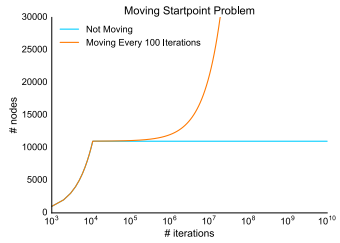


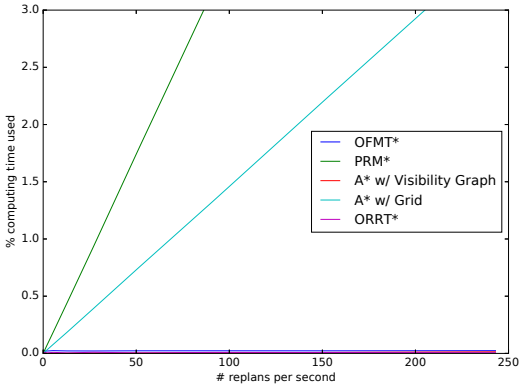
Fig. 4: Moving the start point adds nodes, which eventually becomes a significant problem.

IV. VALIDATION

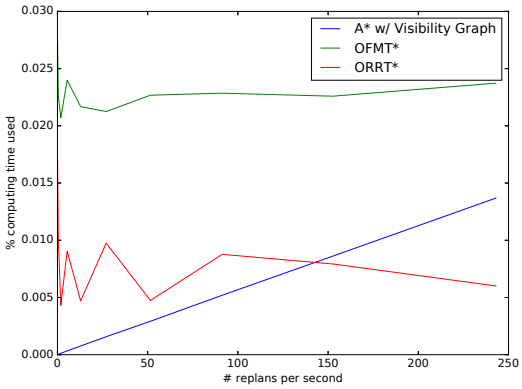
A. Computational Efficiency: Algorithm Comparison

This section empirically compares the computational efficiency of ORRT* and OFMT* to A* over a visibility graph, PRM*, and A* over a grid discretization. Results are shown in Figure 5 for all algorithms, as well as for the top three algorithms, namely ORRT*, OFMT*, and A* on a visibility graph. Results are computed for a 30 second simulation with frequent replanning over 50 randomly generated maps. Results use a shortest-path problem, and the optimal path is generated using A* running on a visibility graph (which is guaranteed to find the true shortest path). Each algorithm was simulated with the start point moving at 30 Hz, and replanning frequencies were varied from 1 to 250 Hz. The time for each algorithm to compute a new path was computed. Results are reported as the percentage of the 30 seconds available (the duration of the simulation) that was used for planning; higher values indicate that the algorithm is using more of the available time. Algorithms were allowed to use more than the theoretically available time, that is, they could use more than 1/*replan frequency* seconds each time they replanned, to demonstrate any inefficiency. All simulations were performed on a desktop workstation with an Intel Core i7-6700 3.4 GHz CPU and 15.6 GB DDR4 RAM.

As can be observed in Figure 5, PRM* and A* with a grid quickly began to perform very poorly with relatively low replan frequencies and reached far over 100% of the available time. A* with a visibility graph performed the best, despite the fact that it grows linearly with replans



(a) All tested algorithms



(b) Three best algorithms

Fig. 5: Percentage of 30 s run time used for path computation

per second. Note, however, that A* over a visibility graph only works for shortest-path problems; additionally, as the map becomes more complex the visibility graph becomes more complex and therefore A*, which has an exponential worst-case computational complexity, becomes less efficient. ORRT* and OFMT* had relatively constant results with a low percentage of compute time being used. They continue to take the same amount of computation time as replanning becomes very frequent, because they adjust the tree as the start point moves, and the optimal path to any new end point is embedded in the tree.

B. Memory Efficiency

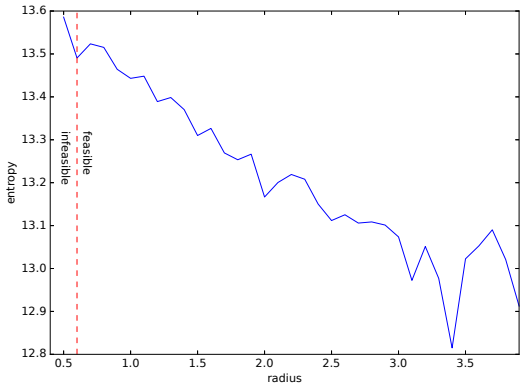
The memory efficiency of ORRT* and OFMT* is best understood by treating a node as a unit of memory. RRT* would add a new node every iteration, which equates to memory usage increasing linearly. ORRT*, by contrast, adds one node per iteration until the threshold is reached. Similarly, OFMT* builds a tree to a predetermined number of nodes. At that point, the number of nodes remains constant at the threshold for both algorithms as long as the pruning radius is set appropriately.

C. Online Pruning

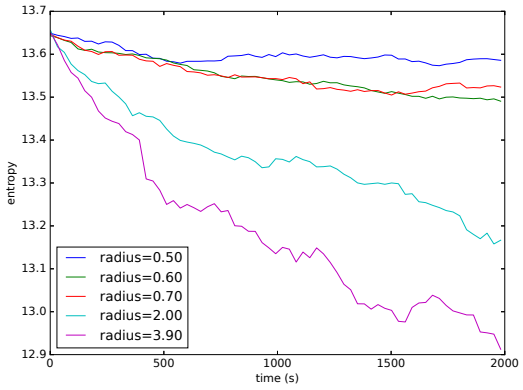
This section provides evidence that the moving start point algorithm allows a robot to move while still finding optimal

paths. The algorithm was executed for 33 minutes and measured the particle entropy of the nodes in our tree. Particle entropy is a measure that helps determine the uniformity of the distribution of nodes. A high entropy value indicates that the distribution is uniform, and therefore the asymptotic optimality guarantees of RRT* hold. We performed the experiment for several different pruning radii as seen in Figure 6, and found that 0.7 units was the best radius in our scenario. It can be observed that a radius of 0.5 units had a higher entropy, but that radius does not prune well enough to keep the number of nodes from growing.

Future work should consider how best to select the pruning radius. One potentially important relationship is the ratio of velocity to prune radius, which determines whether or not the old root will fall within the prune radius. The ratio in the simulations above was empirically and subjectively set to $3/14$, but better ratios might exist for other environments and other algorithm parameters. A second important parameter which might have influence on the ideal pruning radius is the sampling density. Sampling density can affect the ideal pruning radius because it influences the number of nodes that might fall within a given radius.



(a) Entropy as it relates to pruning radius.



(b) Entropy over time with various pruning radii.

Fig. 6: Online Pruning

D. Support for Multiple Objectives

Figure 5 illustrated that the only algorithm which might be faster than ORRT* and OFMT* for rapid replanning is A* on a visibility graph. Importantly, A* on a visibility graph only works when the objective function is shortest path. Many scenarios require a more complex cost function, and therefore need to use a different algorithm. When the objective functions have structures that can be exploited, similar to how a visibility graph extracts shortest path structures, then objective-specific efficient algorithms may be possible, but it we claim that ORRT* and OFMT* can work for a wide range of objectives.

To provide evidence for this claim, Figure 7 presents an an example of OFMT* using a cost function that seeks to avoid being "seen" by the pink and blue circles in the world. The selected path is obviously not the shortest path, and the tree shows a lot of curvature as it tries to avoid the pink and blue circles. The path also tends to maximize the time that the robot is hidden behind obstacles and, when not hidden behind obstacles, try to be far from the pink and blue circles to make the probability of being seen smaller. Space does not allow a full presentation of how many objectives are compatible with the algorithms, but note that simulation results generate subjectively acceptable paths for many convex blends of the shortest path objective and the "stealth" objective of avoiding being seen.

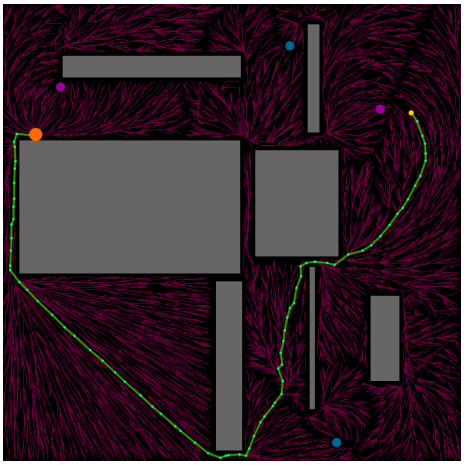


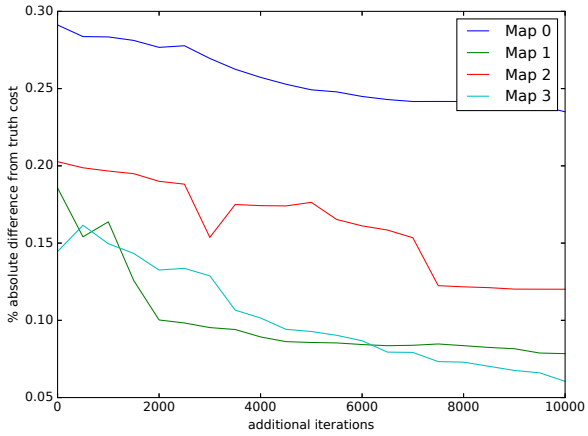
Fig. 7: A tree and path created using a cost function that attempts to avoid the pink and blue circles.

E. Time-Varying Cost

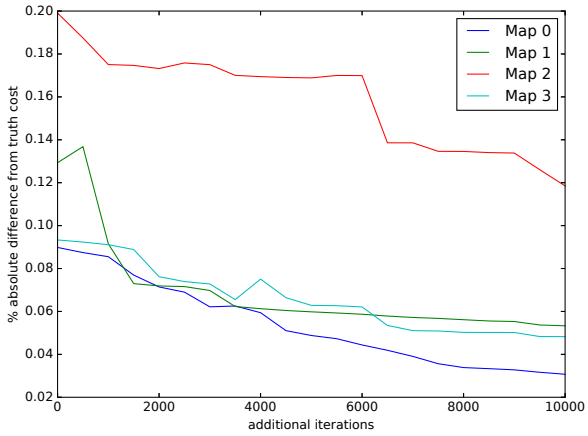
In many real-world scenarios, assuming a fixed cost function is unrealistic. ORRT* and ORMT* are capable of adjusting to changes in the cost function as long as those changes are gradual. Future work needs to characterize what constitutes a "gradual" change. To provide evidence in support of the claim that the algorithms adjust to gradual changes in cost functions, we simulated with fixed start and end points on 4 different maps. The cost function was based on the "stealth" objective of hiding from moving enemies. Cost is high for points where enemies can see the robot and are close to it. We allow the enemies to move over time, which gives us a time-varying cost function.

For 5 randomly generated scenarios, ground truth optimal paths at 14 time steps over a 30 second trial were found using

PRM*. For ORRT* and OFMT*, we paused the simulation at the same 14 time steps and ran the algorithms for an additional 10000 iterations. Figure 8 illustrates how ORRT* and OFMT* adapt their paths in such a way that path costs approach the optimal cost after the cost functions change. The trend toward decreasing absolute difference between the true optimal and the adapting path illustrate that ORRT* and OFMT* can adjust for time-varying cost and be used in an anytime fashion; given faster processors they would be able to compute fully optimal paths in real-time. Further research is required to assess how the algorithms perform with different cost functions as computation of cost can be a significant computational load.



(a) ORRT*



(b) OFMT*

Fig. 8: Anytime percent absolution difference from truth path cost with time-varying cost function

V. SUMMARY AND FUTURE WORK

This paper illustrates that the ORRT* and OFMT* algorithms rapidly replan in dynamic environments. Replanning to new end points occurs in real-time, and computational efficiency is good enough to scale to rapid replan frequencies. Additionally, the algorithms can adjust to gradual time-varying objective functions in an anytime fashion. This allows for scenarios, for example, where the cost depends on the positions of other moving agents.

The strength of ORRT* is to rapidly replan for new end points and adapting to changing cost functions, but that might not be its only application. We plan an extension to make ORRT* work for very large worlds by applying a sliding window to the planned tree. This approach would delete nodes that fall outside of the window due to movement, and add new nodes when new areas are exposed. The same framework would allow for handling moving obstacles.

All results in this paper were gathered assuming 2D worlds, but the same algorithms theoretically work for 3D configuration spaces. A possible limitation is that adding another dimension would increase the number of nodes required and increase the amount of time required to compute cost functions. Future work should evaluate how well these algorithms work in 3D.

Further future work needs to be done with the online pruning radius. The simulations showed that the ideal pruning radius depends on decisions about other algorithm parameters such as maximum segment length and UAV travel velocity. Future work would involve running many more simulations in this combinatorial space, and analyzing the relationship between the various parameters to establish rules of thumb.

REFERENCES

[1] Joshua Bialkowski, Michael Otte, Sertac Karaman, Emilio Frazzoli, Michael Otte, Emilio Frazzoli, Michael Otte, Nikolaus Correll, Michael Otte, Scott Richardson, et al. Efficient collision checking in sampling-based motion planning via safety certificates. *The International Journal of Robotics Research*, 26:212–240, 2010.

[2] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[3] Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International journal of robotics research*, page 0278364915577958, 2015.

[4] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller. Anytime motion planning using the rrt*. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1478–1483, May 2011.

[5] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[6] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[7] Sven Koenig and Maxim Likhachev. D* lite. In *AAAI/IAAI*, pages 476–483, 2002.

[8] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[9] Steven M LaValle and James J Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

[10] Maxim Likhachev, Dave Ferguson, Geoff Gordon, Anthony Stentz, and Sebastian Thrun. Anytime search in dynamic graphs. *Artificial Intelligence*, 172(14):1613–1643, 2008.

[11] Anthony Stentz. Optimal and efficient path planning for partially-known environments. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 3310–3317. IEEE, 1994.

[12] Wikipedia. A* search algorithm — wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=A*_search_algorithm&oldid=719205928`, 2016. [Online; accessed 18-May-2016].