Improving Collaborative Pathfinding Using Map Abstraction

Nathan Sturtevant and Michael Buro
Department of Computing Science, University of Alberta
Edmonton, Alberta, Canada T6G 2E8
{nathanst,mburo}@cs.ualberta.ca

Abstract

In this paper we combine recent pathfinding research on spatial abstractions, partial refinement, and space-time reservations to construct new collaborative pathfinding algorithms. We first present an enhanced version of WHCA* and then show how the ideas from WHCA* can be combined with PRA* to form CPRA*. These algorithms are shown to effectively plan trajectories for many objects simultaneously while avoiding collisions, as the original WHCA* does. These new algorithms are not only faster than WHCA* but also use less memory.

Introduction

Path planning is a fundamental operation in robotics and many video games. Some problems faced in these domains include navigating a point object through known terrain and pathfinding groups of objects. Additional tasks in adversarial pathfinding settings (e.g. real-time strategy games) include blocking paths, surrounding opponent objects and maintaining beliefs about other unit locations.

In this paper we present new research on collaborative pathfinding, where multiple units would like to share information about their expected paths in order allow behavior such as adaptive collision avoidance.

Current games often exhibit poor pathfinding behavior because they do not consider paths planned by other units, even when the units are on the same team. In general this occurs because the problem of finding a path through both space and time (spatial-temporal pathfinding) is reduced into a more simple spatial pathfinding problem, where all other units are considered static obstacles. If we wish to exhibit cooperative behavior we must consider both where units are planning to travel and when they will arrive at any given location along their path.

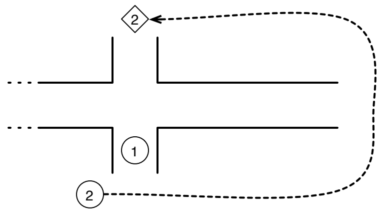

Figure 1: A simple collaborative pathfinding problem.

Consider a simple example in Figure 1 with two units. The first unit is stopped, blocking the hallway, and the second unit needs to walk to the other side of the hallway. The most efficient solution to this problem is for the first unit to move out of the way to allow the second unit to pass. But, algorithms which plan strictly using two-dimensional spatial pathfinding cannot find such solutions. Instead, the second unit will either stop behind the first unit, or find a longer route to the goal, depending on the implementation.

Disregarding computing resources, the best solution to this problem is to do simultaneous planning for all units in the world. But, this approach is infeasible in practice (PSPACE-hard (Hopcroft, Schwartz, & Sharir 1984)).

In the next section we will discuss a variety of approaches for cooperative pathfinding and speeding pathfinding using abstraction. We will then combine these ideas into new algorithms for cooperative pathfinding which are faster and more efficient that previous methods.

Related Work

A* (Hart, Nilsson, & Raphael 1968) is one of the most commonly used algorithms for pathfinding. A* maintains a closed list of nodes for which optimal paths are known, and an open list of nodes which are candidates for search.

But the traditional formulation of the pathfinding problem as a single-agent two-dimensional search problem is inadequate if we are interested in the full multi-agent pathfinding problem (Erdmann & Lozano-Perez 1987). The approaches to this problem have been explored and categorized by several authors. A good overview of this work can be found in (Silver 2005).

Windowed Hierarchical Cooperative A* (WHCA*) (Silver 2005) combines several different ideas to achieve more efficient cooperative pathfinding. First, units plan independently, but share a reservation table which holds their intended routes. Second, units construct an exact solution to the spatial pathfinding problem as a heuristic estimate to the cost of solving the spatial-temporal pathfinding problem. Finally, because of the accuracy of the constructed heuristic, units can truncate their planning to a fixed window size to reduce the cost of pathfinding.

Partial-Refinement A* (PRA*) (Sturtevant & Buro 2005) uses automatic state-space abstraction to speed traditional (non-temporal) pathfinding. PRA* uses a hierarchy of map abstractions; it computes abstract paths within this hierarchy, and then refines them into paths which can be executed.

This paper has three main contributions. First, we implement and test WHCA* in a domain where we have eight directions of movement available, as opposed to the four (NSEW) that were used in the previous implementation of WHCA*. Our implementation also supports variable speed units. We then introduce two new algorithms based on the combination of WHCA* and PRA*.

Previous Techniques

We introduce two previous algorithms here in some detail, as our new work draws heavily on the ideas introduced within these algorithms, and an understanding of these techniques is important for understanding our new work.

WHCA*

WHCA* combines three main ideas for efficient collaborative pathfinding.

Space-time reservation table. A space time reservation table is a three-dimensional table indexed by both an x/y location in the world and a time, t. The value of an entry (x, y, t), if it exists, is the unit which will be occupying the given cell at that point in time. It is relatively simple to incorporate a space-time reservation table into a search algorithm like A*. When expanding the neighbors of a given node you must simply check to see if any of these nodes is already in the reservation table. If so, these nodes are skipped, because they cannot be reached.

We demonstrate a space-time reservation table in Figure 2. Because of efficiency concerns, time is discretized in the table. Some units will be able to move through multiple tiles within a single time-step in the table depending on their speed. If there is a wide disparity in unit speeds, this can cause efficiency problems for the reservation table.

Reverse A* Heuristic. There is significant cost associated with moving from a two-dimensional spatial pathfinding problem to a three-dimensional temporal-spatial problem. The cost of an inaccurate heuristic in a two-dimensional problem is only the area in which the heuristic is inaccurate. In a three-dimensional problem the cost is the volume where the heuristic is inaccurate. Thus, it is very important to have an accurate heuristic.

In the case of cooperative pathfinding, we abstract the problem by building an accurate heuristic for just the spatial pathfinding problem, ignoring temporal constraints. This works well because the time to travel a path is highly correlated with the distance traveled; given no additional units in the world this would be a perfect heuristic.

An accurate spatial heuristic can be built by performing a reverse A* search from the goal node to the start node and saving the open and closed lists from the search. By following this A* search backwards, the exact spatial cost from not only the start node, but also every other node in the closed list can be quickly computed. Thus, a reverse A* search provides a heuristic for a wide range of nodes along the search path. If the cost of a node not on the closed list is needed, the reverse A* search can be resumed to add more nodes to the closed list.

Windowing. Given a reservation table and a reverse A* heuristic, cooperative pathfinding works well. But, as more units interact within the world, the initial cost of resolving all conflicts can become prohibitively large. This cost, however, can be reduced by windowing the search. If, for example, WHCA* uses a window size w, the first w steps in the path will be computed with full cooperation. The remaining portion of the path will assume there are no other units in the world. Because the reverse A* heuristic is an exact spatial heuristic, there is no danger of being lead into a local minimum from which a unit cannot escape. Since only the initial window of a path is actually reserved, a new path must be recomputed and reserved before the initial window is fully executed.

Windowing the search is not without drawbacks. We illustrate a difficult problem in Figure 3 which is made even more difficult when windowing is used. In this example the units are circles, and their goals are diamonds. If the search window for each unit isn't large enough, the units will take turns pushing each other off their goals instead of finding a stable solution.

Summary. Given these techniques, there are significant memory and search costs associated with WHCA*. WHCA* must store the results from the reverse A* search in memory on a per-unit basis. In the worst-case scenario,

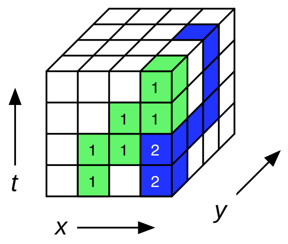

Figure 2: A visualization of the 3-dimension reservation table.

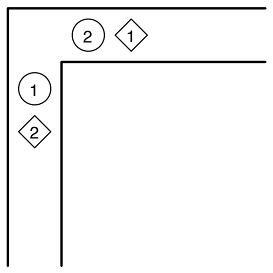

Figure 3: Windowing deadlock. Units 1 and 2 will successively push each other out of the way.

this means every unit will store a copy of the entire map. In this cost will be determined by the complexity of the map, but these rise linearly as units are added to the world. Additionally, we note that WHCA* must perform a complete A* search followed by a windowed search with reservations before WHCA* units can start moving. These costs can be significant, especially as the map size increases.

PRA*

Partial-Refinement A* (PRA*) (Sturtevant & Buro 2005) combines the ideas of spatial abstraction with partial path refinement to speed up path planning considerably while still producing high-quality paths. The main idea behind spatial abstraction is to build a lower-resolution version of the map which can be efficiently reasoned over. Then, the results of low-resolution planning can be refined into executable paths using path refinement. The strength of PRA* is its ability to efficiently interleave planning with execution. We describe the most important ideas from PRA* here.

Abstraction. PRA* relies on the existence of an abstraction hierarchy. Each level in the hierarchy is a successively more abstract version of the map. We show the general concept in the upper portion of Figures 4. In our implementation we represent the world as a graph, as shown in the bottom of this figure, and do the actual abstraction by reducing cliques in the original graph into single nodes in the abstract graph. The nodes in group B are fully connected, so they become a single node after abstraction. The nodes in group A are likewise abstracted. Because there is a single node that is only connected to group A it is folded into A. Groups C is also abstracted into a single node. Because all neighbors of node D are already abstracted, node D is preserved after abstraction. Edges between groups in the original graph become edges between nodes in the abstract graph.

Each abstract node is given a virtual location, which is the average of the location of its children. Edges then have cost equal to the straight-line distance between the nodes they connect. This cost may underestimate or overestimate the actual cost between two abstract nodes. Assigning all

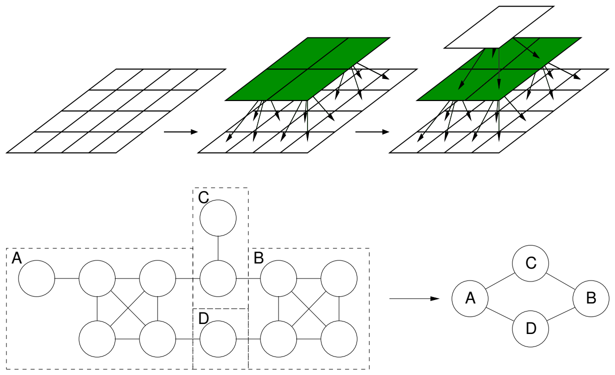

Figure 4: Tile-based abstraction in PRA*. In each level up to four fully connected traversable tiles in addition to hanging off degree one nodes are merged.

edges cost 1 guarantees that the cost of an abstract path is an admissible heuristic for (doesn't overestimate) the cost of the underlying path. There are trade-offs and arguments for both approaches.

Abstractions can be pre-computed or built at runtime as terrain is explored (Bulitko, Sturtevant, & Kazakevich 2005) with relatively low overhead costs. We have found the clique-based abstraction method effective in practice, but there are many other ways that abstractions can be built.

Besides their use for pathfinding, abstractions have other uses as well. For instance, we can annotate nodes with occupancy information. Then, we can quickly summarize information across wide areas of the map by looking at the values stored in the abstract nodes. While a proof is outside the scope of this paper, this information can be updated in constant time, as the cost is amortized across movement.

Path Refinement. Given that we have an abstraction hierarchy, we can use algorithms like A* to compute paths at any level of the hierarchy. These abstract paths can either be used to improve the heuristic at lower levels of the abstraction hierarchy (Holte *et al.* 1996), or can be used as a guide for pathfinding at these levels.

We demonstrate this in Figure 5. If we want a path between A and B at the bottom level, we first find the parents of nodes A and B (A' and B') at an abstract level in the graph. We can compute a path between A' and B' using A*. Because we are using abstraction the cost of computing this path is much smaller than the cost of computing a path between A and B. Given a path between A' and B' at an abstract level, we then define a corridor at the next lowest level, which is the set of all nodes below the path between A' and B'. This corridor can optionally be widened to encompass more nodes. When computing a path at the next level of abstraction, we restrict the ourselves to only expanding nodes within this corridor.

If we have an accurate heuristic, the process of building an abstract path and refining it is slightly more expensive than just building the regular path in the world. But, when our heuristic is poor, this process gives significant savings, because we are prevented from exploring areas of the map which do not lead to the goal.

Partial-Path Refinement. Searches with WHCA* can be windowed because the reverse A* heuristic is accurate. Oth-

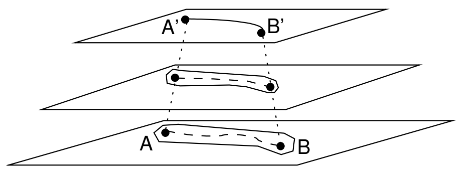

Figure 5: Abstraction and path refinement in PRA*. To find a path from A to B, PRA* first finds a path from A' to B', the respective parents of A and B. At the next lower level a new path is computed which stays in a corridor defined by the original path. This procedure is repeated until the bottom level is reached.

erwise there would be a risk of never reaching the goal. Searches in PRA* can also be windowed, but for a different reason. PRA* always computes a complete path from the start to the goal at some level of abstraction. Thus, as long as that abstract path is followed, a unit is guaranteed to eventually reach its goal. So, PRA* is windowed by truncating the corridor that is used when refining abstract paths and introducing intermediate goals at the end of each of these corridors.

Summary. PRA* is the combination of abstraction and refinement as described above. The use of abstraction in this manner does not guarantee optimal paths, but the over 90% of paths computed by this method are optimal in practice (Sturtevant & Buro 2005), and they are computed significantly faster than A*.

New Methods

We propose and analyze here two simple combinations of abstraction and WHCA*. The first idea is to compute the reverse A* heuristic in abstract space instead of the full map. The second idea is to use WHCA* inside the last step of a PRA* search.

WHCA*(w, a)

WHCA* uses values from the reverse A* search as an estimate of the full space-time search cost. We propose replacing this A* search with an A* search in an abstract space. This gives us WHCA* with two parameters, w, the window size used for reservations and a the abstraction level used for the heuristic. This simple addition not only reduces the size of the A* search that must be cached, but also speeds the initial A* search.

There are two ways we can use the abstract search to generate a heuristic. Given a set of paths through abstract space on the A* closed list we need an associated heuristic value for each path. If optimality is a concern, we should assume that the edges in this path have cost 1 each. But, since we have already lost optimality by planning units independently, this is not a great concern. So, instead we use the abstract edge cost (defined in the discussion of abstractions in the previous section). Empirical analysis has shown that this value is much more accurate, but tends to slightly overestimate the actual cost.

This approach has direct trade-offs between the value of a, the memory used by WHCA*(w, a) and the number of nodes expanded. The larger the value of a, the less memory we need, which initially reduces the number of nodes expanded in the first step. But, larger values of a result in a less accurate heuristic, which in turn results in more nodes expanded during the windowed search.

Combining WHCA* and PRA*

PRA* performs an A* search at each level of the abstraction hierarchy, restricting search to a corridor defined by the previous search results. Thus, the most simple way to combine PRA* and WHCA* is to simply replace the last A* search at the bottom of the abstraction hierarchy with a WHCA* search. In this approach the cost and the scope of the WHCA* search is limited by the PRA* corridor. Because the PRA* target location is always moved to the end of the current search window, no extra data needs to be cached like in WHCA*. We call this new algorithm Cooperative PRA* (CPRA*). CPRA*(k) takes a single parameter k which is the length of the path which is partially refined at each step.

The cost of a CPRA* search will depend greatly on the window size that is used for search. Unlike WHCA* which has a high initial cost when the A* heuristic is being computed, and then much lower costs after that, CPRA* will have roughly the same cost at each re-planning stage.

One advantage of using CPRA* is that the window size, and thus the pathfinding cost, can be dynamically adjusted, if needed. If, for instance, many units are simultaneously asked to path across the map, a smaller window size might be used for the first movement computation. While the window size in WHCA* can also be dynamically adjusted, there is no way to avoid building the reverse A* heuristic in the first step of execution.

Experiments

We now compare these algorithms across various sized maps and numbers of units. Our implementation of WHCA* has a few differences between the implementation in (Silver 2005). First, units can move to any of the eight adjacent tiles (including diagonals) on any move, instead of just the four adjacent tiles considered in prior implementations. Next, we re-plan paths at a fixed distance (3 steps) from the end of our path instead of at the half-way through the reserved path. Finally, we don't measure collisions. When a collision occurred in (Silver 2005) the colliding units were removed from the world. We instead left all units in the world after they collided, forcing them to replan.

We run experiments in two scenarios, S1 and S2. The maps for these scenarios are in Figure 6. In these maps black areas are out of bounds. The lighter areas are the potential start and end locations. For each experiment we placed a given number of units randomly on the left side of the map and asked them to path to random locations on the right side. These maps were scaled in size from 32×32 to 256×256.

In S1 the Manhattan distance heuristic is quite poor initially, but once units have crossed the middle of the map, the Manhattan distance is very accurate. While this isn't the

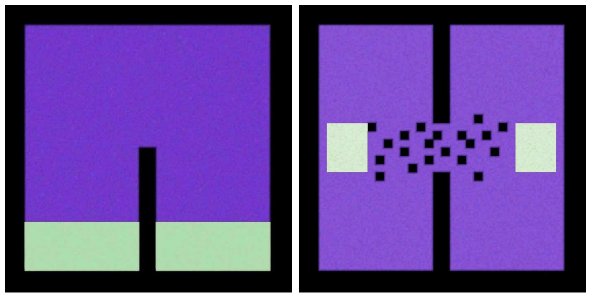
Figure 6: The maps for scenarios S1 (left) and S2 (right) we used in our experiments.

W,H	32	64	96	128	160	192	224	256
S1	376	1246	2894	4726	7744	10724	14887	18646
S2	418	1246	2962	4574	7713	10445	12960	17886

Table 1: PRA* abstraction sizes for scenarios S1 and S2 measured in nodes.

worst case heuristic, it is much better than S2, where the initial heuristic is fairly accurate, except for the obstacles in the way.

We considered the following algorithms in each experiment. WHCA*(w, a): When $a = 0$, we are using WHCA* without any spatial abstraction in the heuristic. We varied a between 0 and 3, and also varied w between 8 and 20. CPRA*(k): A corridor size of k for CPRA* is roughly equivalent to a window size of $2k$ for WHCA*. So, for CPRA* we varied k between 4 and 10. We also included PRA*(k) for some experiments which does not make any reservations; it just uses local repair.

Experimental results are measured in terms of nodes expanded, but for reference we have measured node generation speeds on a 1.5Ghz Powerbook G4 laptop with 512MB of RAM compiled with gcc 4.0.1 using -O3 for optimization. CPRA* expands 80-150k nodes/sec. WHCA(*, 0) expands 50-70k nodes per second on average. WHCA*(*, 1/2/3) all expand 40-50k nodes per second on average. Because we have implemented these algorithms in a framework designed for maximum flexibility and experimentation, we have not devoted significant effort to optimizing our code. It would not be surprising if a custom implementation of these algorithms was significantly faster.

The graphs in Figures 7 to 10 present important runtime statistics of the described algorithms in form of the maximum computed from a dataset of 100 samples for a fixed set of algorithms and varying start and goal locations.

In each plot presented here, algorithms are sorted by their performance. The top name in each legend corresponds to the highest values, and so on. For each algorithm we plotted the window settings with the best performance. When two algorithms have equal performance we plot a single curve and indicate that they are equal in the legend.

We use these experiments to illustrate four points. First, the memory savings when using abstraction. Second, the drop in initial planning costs from using abstraction. Third, the steady-state cost of using each algorithm, and finally the resulting path quality.

We illustrate memory savings in Figure 7. In this figure we plot the size of the open and closed lists for the WHCA*'s reverse A* heuristic at their largest point in the run. Figure 7A contains the results on running on the map S1 as we vary the map size. We see that increasing the abstraction level from 0 to 1 decreases the memory used by about a factor of 4. In Figure 7B we show how memory usage scales as we add more units to map S2. The total memory required increases linearly with the number of units. Using a single layer of abstraction decreases the memory usage by a factor of 4. Further layers of abstraction reduce memory usage, but by smaller margins.

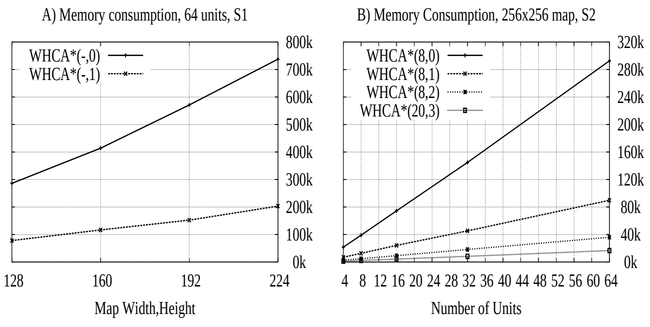

Figure 7: Memory consumption: maximum of the maximal number of nodes in open/closed lists.

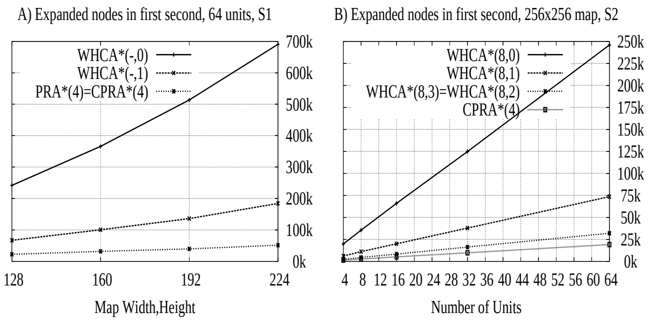

Figure 8: Initial planning time: maximum of the number of nodes expanded during the first second.

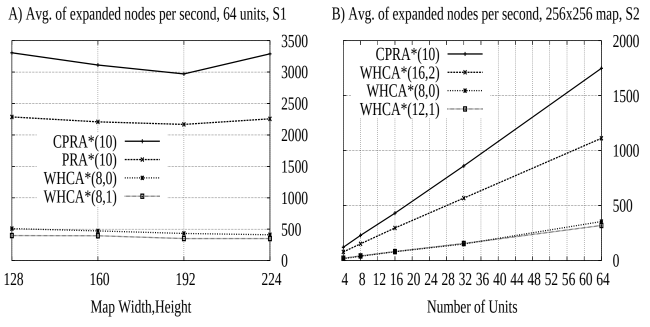

Figure 9: Follow-up planning time: maximum of the average number of nodes expanded in subsequent seconds.

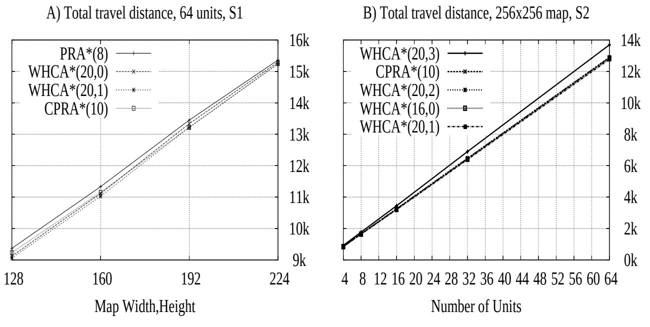

Figure 10: Path quality: maximum of the total travel distance.

In Table 1 we show the memory required to build an abstraction for a given map dimension. On a 256x256 map we need approximately 20,000 nodes to build a full abstraction hierarchy. CPRA* units can share a single abstraction hierarchy, so this cost would be shared by all CPRA* units. WHCA* only needs a single abstract map, not a full abstraction hierarchy, so this additional cost would be less for WHCA* than for CPRA*.

In Figure 8 we plot the first-move computation cost summed over all units. This is the number of nodes that all units must expand to begin acting in the world. Again, using an abstract map with WHCA* reduces the first-move cost significantly on both S1 (Figure 8A) and S2 (Figure 8B). All the CPRA* algorithms have the same performance in both of these figures. We also plot the PRA* (non-cooperative pathfinding cost) in part A, and it has the same performance as CPRA*.

We consider the steady-state cost of pathing all units after the first move in Figure 9. For S1 in 9A we can see that CPRA* has a higher steady-state cost than WHCA*. This is expected, because WHCA* has a very high-quality heuristic after the first move. It is interesting to note that the pathfinding cost stays constant despite the size of the map increasing. This occurs because neither algorithm is dependent on the size of the map after the first step. For S2 in 9B we measure the steady-state cost as the number of units increases. Adding units linearly increases the cost of pathfinding. We point out that the higher abstractions used by WHCA* eventually degrade performance, with WHCA*(16, 2) performing more slowly than WHCA*(8, 0).

We note that in Figure 9 the best CPRA* window size is 10, while in Figure 8 the best window size is 4. This suggests we should CPRA* should use a window size of 4 on the first move and 10 on remaining moves.

Finally, we show the total distance moved by all units while moving from the start to their goal in Figure 10. We see that all algorithms have very similar performance. So, the improvements suggested in this work do not have a meaningful effect on the quality of paths, just on the memory and speed of their computation.

Conclusion and Future Work

In this paper we have presented two ways in which state abstraction can be used to improve WHCA*'s performance. The first of these methods, building the reverse A* heuristic in abstract space, reduces the per unit memory needed for WHCA*. CPRA* uses an abstraction hierarchy to reduce the first move delay, although it is more expensive than WHCA* after the first step.

The practical choice of which of these algorithms to use will depend on the application in question and the problem constraints. If there are many units and memory is constrained, CPRA* is probably a better approach. If users can tolerate slower initial costs and memory is not a problem, WHCA* provides better performance once the initial computations are completed.

There are many interesting extensions which might be added to this work. For instance, we can use an abstraction hierarchy to store information about things like congestion, and this information can be used to help avoid congested routes. Similarly, we could use this information to dynamically vary the window size with CPRA*, ensuring we don't step into a congested region unless we can pass completely through. We plan to build on this work by exploring such ideas.

Finally, we note that while we have focused on improving spatial-temporal pathfinding, these exact techniques are more general and can be applied to three-dimensional pathfinding with or without an extra temporal dimension, or other more difficult pathfinding problems.

References

Bulitko, V.; Sturtevant, N.; and Kazakevich, M. 2005. Speeding up learning in real-time search via automatic state abstraction. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1349–1354.

Erdmann, M., and Lozano-Perez, T. 1987. On multiple moving objects. *Algorithmica* 2:477–521.

Hart, P.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. on Systems Science and Cybern.* 4:100–107.

Holte, R. C.; Mkadmi, T.; Zimmer, R. M.; and MacDonald, A. J. 1996. Speeding up problem solving by abstraction: A graph oriented approach. *Artificial Intelligence* 85(1-2):321–361.

Hopcroft, J.; Schwartz, J.; and Sharir, M. 1984. On the complexity of motion planning for multiple independent objects: PSPACE-hardness of the warehousemans problem. *International Journal of Robotics Research* 3(4):76–88.

Silver, D. 2005. Collaborative pathfinding. In *Proceedings of AIIDE*, 23–28.

Sturtevant, N., and Buro, M. 2005. Partial pathfinding using map abstraction and refinement. In *Proceedings of AAAI*, 47–52.