Optimized Algorithms for Multi-Agent Routing (Short Paper)

Akihiro Kishimoto
Department of Media Architecture,
Future University-Hakodate
116-2, Kamedanakano-cho, Hakodate,
Hokkaido, 041-8655, Japan
kishi@fun.ac.jp

Nathan Sturtevant
Department of Computing Science,
University of Alberta
Edmonton, Canada T6G 2E8
nathanst@cs.ualberta.ca

ABSTRACT

Auction methods have been successfully used for coordinating teams of robots in the multi-robot routing problem, a representative domain for multi-agent coordination. Solutions to this problem typically use bids computed using the shortest distance between various locations on a map. But, the cost of this shortest-distance computation has not been considered in previous research. This paper presents a new auction-based algorithm, FastBid, that works to reduce the computational costs associated with bidding in the multi-robot routing problem. We also analyze how a small modification in the bidding algorithm can reduce the computational load of the bidding process. Experiments demonstrate that FastBid not only scales much better than previous approaches, but does so with little or no loss in solution quality.

Categories and Subject Descriptors

I.2 [**Computing Methodologies**]: Artificial Intelligence

General Terms

Algorithms

Keywords

Auction, multi-robot routing, shortest-distance computation, pathfinding, clique abstraction, Dijkstra's algorithm, A*

1. INTRODUCTION

Consider a set of robots assigned to complete a set of tasks. Whatever the exact domain, the completion of tasks requires communication and collaboration. An abstract version of these problems has been studied by AI and robotics researchers as the multi-robot routing problem, where a team of robots must visit a set of targets while also achieving other objectives such as minimizing energy consumption or the total time to complete the plan.

Heuristic approaches are used for "real-time" solutions as exact approaches to this problem are computationally infeasible. The multi-robot routing problem has usually been analyzed from the perspective of maximizing the quality of the plans. We argue, however, that plan quality is not the only metric that should be considered when comparing algorithms. Specifically, the memory and computational costs for existing algorithms quickly become infeasible as map sizes and the number of robots grow. This is important if we are minimizing energy consumption and can also have a significant effect on the time required to complete a plan.

In this paper we present a variety of techniques which allow us to scale the size of multi-robot routing problems which can be solved. The contributions of the paper are:

- A novel combination of a bidding algorithm based on minimum spanning trees [3, 4] with clique abstraction [8] which effectively computes shortest distances on demand during the bidding process.
- Experiments on the multi-robot routing problem with large maps and large numbers of robots and targets. Results demonstrate that computing the shortest distances without abstraction leads to serious performance degradation on large maps. Our implementation only reduces the solution quality a few percent in order to achieve a speedup of two orders of magnitude.

2. MULTI-ROBOT ROUTING PROBLEM

The multi-robot routing problem this paper deals with is defined as follows: n homogeneous robots and m targets are placed on a two-dimensional map. Each target must be visited by one robot. All robots move at the same speed and can move to cardinally and diagonally adjacent points unless blocked by obstacles. The cost of moving to cardinal and diagonal neighbors are 1 and $\sqrt{2}$ respectively.

Let the robots be $R = \{r_1, r_2, \cdots, r_n\}$ and the targets $T = \{t_1, t_2, \cdots t_m\}$. The cost of moving between locations i and j is $c(i, j)$ $(i, j \in R \cup T)$. In our model $c(i, j)$ is not pre-computed. The path cost of a given robot, r_i, is the sum of costs from r_i's initial location to all sequential targets allocated to r_i. The objective of multi-robot routing is to find an allocation of targets to robots for some team objectives. This paper deals with the following two typical team objectives:

- MiniSum: Minimize the path cost sum over all robots.
- MiniMax: Minimize the maximum path cost of any single robot.

Cite as: Optimized Algorithms for Multi-Agent Routing (Short Paper), Akihiro Kishimoto and Nathan Sturtevant, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal, pp. 1585-1588.

2.1 Sequential Single-Item Auctions

The MINISUM and MINIMAX objectives are NP-hard [3]. Approximation methods are important in such domains, particularly when computation cannot take place offline.

Sequential Single-Item (SSI) auctions have shown promise for approximating optimal solutions to the multi-robot routing problem. SSI auctions consist of a series of auctions in which a single item is auctioned off; there are as many rounds as there are items to be auctioned off. In the multi-robot routing problem all robots are eligible to bid on any unallocated targets in each round. The robot that makes the lowest bid on a target wins that target; ties are broken arbitrarily. Only one target is allocated in each round, so each robot bids just on the single target for which the lowest bid can be placed. This process continues until all targets are assigned to the robots. SSI auctions allow for efficient communication and parallelism among robots, as bid costs can be computed in parallel by the robots.

The TREE rules [4] are a popular SSI auction method which guarantees approximation factors of no worse than 2 for MINISUM and $2 \times n$ with n robots for MINIMAX if the triangle inequality holds. Under the TREE rules each robot r_i creates a minimum spanning tree (MST) originating from r_i and passing through each allocated target. There is one node for each allocated target, and one for robot r_i. When building the MST, node locations are known, but edge costs are not; $c(i, j)$ must be computed from the actual travel costs in the map. The MST is then converted to a path. The MST approximates the cost of visiting all allocated targets and can be obtained in polynomial time by Prim's algorithm [6], a greedy algorithm which iteratively adds the node with the smallest edge cost from the current spanning tree.

The main focus of existing research (see [2] for a detailed literature review) has been in improving auction strategies; the cost-effective computation of $c(v, t)$ has been ignored. This is particularly because experiments have used small maps, such as 51×51 [9]. Computing $c(v, t)$ requires finding the shortest path between v and t, for which the computational effort increases with large maps. This computational overhead can be large, especially on large maps. In a naïve approach, the shortest path from all allocated targets to all unallocated targets must be computed before each bid. In the worst case, this could mean that $c(v, t)$ of all possible $O(T^2)$ edges between targets will have to be computed.

2.2 Abstraction and Pathfinding

The exact cost of some edge $c(v, t)$ can be computed exactly by running a pathfinding algorithm like A* in the map. Because the auction methods described in the last section do not guarantee optimality we can use a heuristic estimate of an edge cost in the map as $c(v, t)$. There are a number of heuristics which can be used in the map. The most obvious heuristic is just to use the straight-line estimate (eg. Euclidean or octile distance). These heuristics are fast, but can provide poor estimates of actual cost (see Section 4).

Instead, we leverage an idea which has received much attention in the search literature, but has not been widely applied in other domains. Our approach is to build an abstract version of the map and to use costs in the abstract graph as a heuristic estimate for $c(v, t)$. We could use our heuristic estimate from the abstract graph to find exact costs in the original map more quickly, but the heuristic estimate provides significant speed gains with little or no loss of solution quality (see Section 4). The heuristic is computed online from the abstract graph using standard search algorithms.

Explicit abstractions are built by merging groups of states in the original map into abstract states in the abstract graph. In this paper we use the clique-abstraction from [8]. Clique abstraction is an iterative process which can be recursively applied to provide abstract graphs of many different sizes.

At the lowest level of the map, assuming that a robot can only move to direct and diagonal neighbors on a two-dimensional map, the clique abstraction groups states which form a *clique*. In such maps, the largest possible cliques contain four nodes. Maps are abstracted by successively removing cliques (size 2-4), which become nodes in the abstract graph. Any nodes not abstracted by this process are abstracted as single nodes. This abstraction process continues from lower level to higher levels, creating a hierarchy of abstract graphs. One important feature is that reachability between locations is preserved through this approach. When no path between two locations is found in an abstract graph, there is also no path between them in the original map [1].

3. THE FastBid ALGORITHM

FASTBID works to minimize the cost of the shortest distance computation, based on three ideas: bounding path computations based on known target information, using abstraction to quickly estimate costs, and modifying slightly the bidding scheme to further bound path computations. In order to inherit the merits of auction-based approaches, FASTBID is based on the TREE rules [3].

3.1 Bounding edge cost computations

Each robot maintains a MST of targets which have been successfully bid on. When a robot is considering which target to bid on next, the robot must find the closest unallocated target to its MST. We present two approaches which can do this effectively.

3.1.1 Dijkstra's Algorithm

For each node in the MST, we can run Dijkstra's algorithm from that node to find the cost to all unallocated targets. But, instead of running Dijkstra's algorithm to completion, we run it incrementally. At each node we keep track of the cost of the last path expanded. If a path to a target has not yet been found, we are guaranteed that this will be a lower bound on the cost to any target from that node.

Thus, we can keep all nodes from the MST in a priority queue, sorted by the cost of the last path expanded by Dijkstra's algorithm. At each step we remove the best node from the priority queue, and continue the Dijkstra search from that node until the cost exceeds the cost of the next best node at the top of the priority queue. At this point the current node is put back onto the priority queue, and the process is repeated until a path to an unallocated target is found. The first target found will be the next bid. This process requires one invocation of Dijkstra's algorithm for the robot's location, and one for each target which has already been allocated to that robot.

3.1.2 The A* Algorithm

If multiple targets are nearby, Dijkstra's algorithm is effective, as a single search tree is used. But, Dijkstra's algorithm does not take advantage of a heuristic function.

An alternate approach is to start one A* search from each node in the MST to each unallocated target. This requires many individual searches, but use of a heuristic function will often prevent the searches from expanding any nodes. As in the Dijkstra approach, we focus our efforts on the most promising path between a MST node and an unallocated target. Instead of putting each node from the MST on the priority queue, we put a pair onto the priority queue for each possible node and target. We then sort the pairs by the cost of the best node on A*'s Open list. As with Dijkstra's, the best $(node, target)$ pair is taken off the priority queue at each step, and nodes are expanded in that A* search until the cost exceeds the next best pair in the priority queue or until the target is found. If there are T_u unallocated targets and n nodes in the MST, this will require $O(n \cdot T_u)$ A* searches. The *octile* distance heuristic is used during search.

FastBid computes the path costs locally and does not share any shortest distances computations among robots, which would incur an additional communication overhead.

3.2 Pathfinding on an Abstract Map

The previous analysis assumed that the A* and Dijkstra searches were taking place on the original map. However, we can replace the map used for search with an abstract graph. If we need an admissible heuristic, we must set the cost of all edges in the abstract graph to be the minimum edges cost in the original map. Because we are not computing optimal solutions to begin with, we instead get a heuristic estimate from the abstract graph which is not guaranteed to be admissible. The cost of an abstract edge is defined as the straight-line cost between the abstract node locations. An abstract node is located at the average of the location of all nodes it abstracts. Octile distance will then be an admissible heuristic for search within the abstract graph, but the costs in the abstract graph are not guaranteed to be lower bounds on the true costs in the original map.

Let $c_l(v, w)$ be the distance of the shortest path between v and w on the map with abstraction level l. Although clique abstraction makes $c_l(v, w)$ less accurate than computing $c(v, w)$ in the planning graph, it is not only a reasonable approximation but also preserves reachability. Hence, our approach guarantees that robot r_i can reach target t, if r_i wins t and a map is completely known. For an incomplete map, we believe that techniques described in [1] for repairing abstraction could be incorporated into our algorithm. However, investigating *a priori* unknown maps remains future work.

3.3 Modification to the Bidding Scheme

The server that determines the winner of an auction receives all the bids of robots. If the server broadcasts not only the winner of the previous auction, but also the second best bid in the previous round, each robot can use this value as a threshold for path computation, further reducing search effort. This is only valid if the second-best bid is for a different target than the one that was one in the auction. In FastBid, each robot can bid either the exact bidding cost or a lower bound (i.e., give-up obtaining a target) in each round. Only the second smallest exact biding cost can be used to bound the path computations.

3.4 Other Implementation Details

When converting an MST to a path, our current implementation employs shortcutting as used in TSP [5]. However, since there is a trade-off between solution quality and execution time, the edges selected in the MST for depth-first search are limited to at most two edges. These edges are greedily chosen, based on $c_l(v, w)$ if available or the octile distance otherwise.

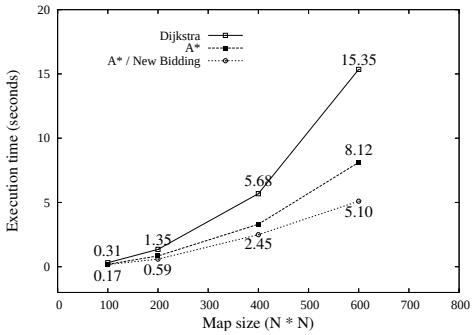


Figure 1: Average planning time on various sized maps (5 robots and 50 targets, MiniSum)

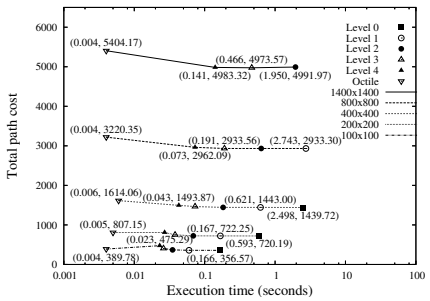
4. EXPERIMENTAL RESULTS

We used a freely available implementation of the clique abstraction, along with a corresponding simulation environment for our experiments [7]. Although FastBid allows for a decentralized implementation, it was simulated sequentially on a single machine. But, none of the shortest distances computations were shared among robots. All experiments were performed on a 1.79 GHz AMD Opteron 265 with 2 GB memory. 117 maps were used, and varied in size from 100×100 to 1400×1400. 5 robots and 50 targets were randomly placed with the condition that each target must be reachable by at least one robot, resulting in 117 problems per map size. We report the average value for each category over all experiments. Each robot has complete knowledge of the map and the locations of the targets, but has no information about the locations of the other robots or the exact costs to travel between targets.

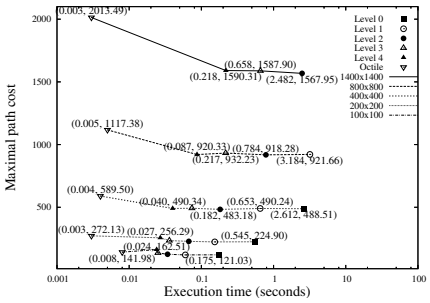
4.1 A* versus Dijkstra's Algorithm

Figures 1 summarizes performance comparison between A* and Dijkstra's algorithm without abstraction. As the map size increases, the planning time considering the shortest distance computation dramatically increases for both MiniMax and MiniSum. Due to memory constraints, maps 800×800 or larger were not solvable without applying abstraction first. For example, finding a task allocation on 600×600 maps requires more than 40 times longer planning time than on 100×100 maps. This indicates the importance of efficiently computing $c(v, w)$ on large maps. The results for both MiniMax and MiniSum are similar so we only show MiniSum results here. A* is twice as fast as Dijkstra's algorithm in each category. Our slight modification to the bidding rule further improves the planning time.

A* usually outperforms Dijkstra's algorithm by a large margin. For example, A* is more than 10 times faster in solving 48 problems for MiniMax and 42 for MiniSum. However, there are a few problems that Dijkstra's algorithm solves much more quickly than A*. For these problems, due to complicated paths between targets, the duplicate search effort of A* offsets the benefit of using a heuristic function.



(a) MiniSum team objective



(b) MiniMax team objective

Figure 2: Performance results on various sized maps (5 robots and 50 targets, A*, new bidding)

Speeding up A* in this scenario remains as future work.

4.2 Performance with Clique Abstraction

Figure 2 summarizes performance on various abstraction levels varied from level 0 (i.e. no abstraction) to level 4 (four iterations of abstraction applied to the low-level map). Solution quality was measured by computing the shortest path for each robot to visit all assigned targets after the bidding phase. The versions with levels 0 and 1 could not find allocations within available memory in some problems on large maps. These cases are excluded from the graphs. Additionally, a version that only uses the the octile distance (heuristic) between v and w as $c(v, w)$ was also compared.

The results show that there is a direct trade-off between planning time and solution quality. Using only the octile distance as a heuristic for $c(v, w)$ is always the fastest approach. But, octile tends to achieve a lower solution quality especially on larger maps for MiniMax. For example, octile returns a 34% larger path cost on the 1400×1400 maps than the version with abstraction level 2.

Octile achieves better solution quality for MiniSum than for MiniMax. Auction-based algorithms usually return much closer solutions to optimal than the theoretical worst case [9]. Moreover, because of the worse approximation factor in MiniMax than in MiniSum ($2 \times 5 = 10$ for MiniMax versus 2 for MiniSum), there is usually much more room for improvement to MiniMax.

Using abstraction improves the planning time, especially on large maps. For example, using abstraction level 4 on the 800×800 maps achieves speedups of two orders of magnitude for both MiniMax and MiniSum. The degradation in solution quality is only a few percent if a proper abstraction level is used. Moreover, when the map is 800×800 or larger, the shortest-path computations sometimes exceeded the available (2GB) memory, even with one level of abstraction. On the other hand, if the map used to compute bids is too abstract, the solution quality becomes worse even than octile, at least on small maps. Informally, this seems to occur when the number of nodes is the abstract map is less than the square root of the number of nodes in the actual map, but more work is needed to understand this.

5. CONCLUSIONS AND FUTURE WORK

This paper has synthesized research on abstraction and the Tree rules for task allocation. Although there is a trade-off between the bidding time and solution quality, our results show that using new bounds for bidding along with map abstraction can dramatically improve the computational costs with only a small degradation in solution quality.

There are many topics for future work. First, we plan to extend the presented algorithm to scenarios where robots do not have *a priori* knowledge of the map, and environments where the maps can change dynamically. We are also interested in whether limited communication can further reduce planning costs. We attempted to use a different search mechanism to speed search further, and plan to continue to look into methods for reducing computational cost.

Acknowledgement

The work was partially funded by a grant from iCore.

6. REFERENCES

[1] V. Bulitko, N. Sturtevant, J. Lu, and T. Yau. Graph Abstraction in Real-time Heuristic Search. *Journal of Artificial Intelligence Research*, 30:51 – 100, 2007.

[2] S. Koenig, C. A. Tovey, M. G. Lagoudakis, V. Markakis, D. Kempe, P. Keskinocak, A. J. Kleywegt, A. Meyerson, and S. Jain. The power of sequential single-item auctions for agent coordination. In *Twenty-First National Conference on Artificial Intelligence*. AAAI Press, 2006.

[3] M. Lagoudakis, V. Markakis, D. Kempe, P. Keskinocak, S. Koenig, A. Kleywegt, C. Tovey, A. Meyerson, and S. Jain. Auction-based multi-robot routing. In *International Conference on Robotics: Science and Systems*, pages 343–350, 2005.

[4] M. G. Lagoudakis, M. Berhault, S. Koenig, P. Keskinocak., and A. J. Kleywegt. Simple auctions with performance guarantees for multi-robot task allocation. In *IEEE International Conference on Intelligent Robots and Systems (IROS 2004)*, volume 1, pages 1957–1962, 2004.

[5] E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley Series in Discrete Mathematics and Optimization, 1985.

[6] R. C. Prim. Shortest connection networks and some generalizations. *Bell Systems Tech. Journals*, pages 1389–1401, 1957.

[7] N. Sturtevant. Hog - hierarchical open graph. http://www.cs.ualberta.ca/∼nathanst/hog.html, 2005.

[8] N. Sturtevant and M. Buro. Partial pathfinding using map abstraction and refinement. In *AAAI*, pages 1392–1397. AAAI Press, 2005.

[9] C. Tovey, M. Lagoudakis, S. Jain, and S. Koenig. The generation of bidding rules for auction-based robot coordination. In *Multi-Robot Sys.: From Swarms to Intel. Automata*, volume 3, pages 3–14. Springer, 2005.