Incremental Replanning for Mapping

Maxim Likhachev
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
maxim+@cs.cmu.edu

Sven Koenig
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
skoenig@cc.gatech.edu

Abstract

Incremental heuristic search methods can often replan paths much faster than incremental or heuristic search methods individually, yet are simple to use. So far, they have only been used in mobile robotics to move a robot to given goal coordinates in unknown terrain. As far as we know, incremental heuristic search methods have not yet been applied to the problem of mapping unknown terrain. In this paper, we therefore describe how to apply our incremental heuristic search method D Lite, that combines ideas from Lifelong Planning A* and Focussed D*, to mapping unknown terrain, which is rather nontrivial. We then compare its runtime against that of incremental search and heuristic search alone, demonstrating the computational benefits of their combination. By demonstrating the versatility and computational benefits of incremental heuristic search, we hope that this underexploited technique will be used more often in mobile robotics.*

1 Introduction

Mobile robots often have to replan quickly as their knowledge of the world changes. In this case, an incremental heuristic search can replan much faster than complete searches from scratch: an incremental search remembers information from previous searches to speed up the search, and a heuristic search uses heuristic knowledge to focus the search and thus speed it up. Researchers from mobile robotics have, so far, exploited their combination only for goal-directed navigation in unknown terrain [3, 15, 12, 5, 6], where Anthony Stentz' Focussed D* [14] algorithm convincingly demonstrates the advantages of incremental heuristic search since it solves this search task with a speedup of one to two orders of magnitude(!) over repeated A* searches, which is important to avoid the robots being idle. However, Focussed D* is very complex and thus hard to understand, analyze, and extend. For example, while Focussed D* has been widely used as a black-box method, it has not been extended by other researchers. We believe that two achievements are needed to make incremental heuristic search methods more popular in mobile robotics. First, one needs to devise simpler incremental heuristic search methods that have well understood theoretical properties. In previous work, we have therefore developed D* Lite [9], a combination of ideas from Lifelong Planning A* [8] and Focussed D* [14]. D* Lite implements the same navigation strategy as Focussed D* but is algorithmically different. It is simpler, easier to understand, easier to analyze and easier to extend than Focussed D*, yet is more efficient. For example, D* Lite has fewer lines of code (without any coding tricks) than Focussed D*, uses only one tie-breaking criterion when comparing priorities, and does not need nested if-statements with complex conditions that occupy up to three lines each. Thus, we use D* Lite in this paper. Second, one needs to demonstrate the advantages of incremental heuristic search methods, such as D* Lite, for additional navigation tasks. In this paper, we therefore demonstrate how D* Lite can be applied to mapping tasks. We believe that this is the first demonstration of how incremental heuristic search methods or, more generally, heuristic search methods can be used for this task. We then compare the computational benefits of the incremental heuristic search performed by D* Lite to the ones of heuristic but nonincremental searches and incremental but nonheuristic searches. By demonstrating the versatility and computational benefits of incremental heuristic search for mapping of unknown terrain, we hope that this underexploited technique will be used more often in mobile robotics.

2 D* Lite Replanning Method

D* Lite [9], shown in Figure 1, is an incremental version of the heuristic search method A* and combines ideas from Lifelong Planning A* and Focussed D*. It implements the same navigation strategy as Focussed D* but is algorithmically simpler. In particular, it moves the robot from its current vertex to a given goal vertex on a given graph as follows: It always moves the robot on a shortest path from its current vertex to the goal vertex, and replans the shortest path when the edge costs change. (In the pseudo code, we have included a comment on line {33} about how the robot can detect that there is no path but do

The pseudocode uses the following functions to manage the priority queue: U.Top() returns a vertex with the smallest priority of all vertices in priority queue U. U.TopKey() returns the smallest priority of all vertices in priority queue U. (If U is empty, then U.TopKey() returns $[\infty; \infty]$.) U.Insert(s, k) inserts vertex s into priority queue U with priority k. U.Update(s, k) changes the priority of vertex s in priority queue U to k. (It does nothing if the current priority of vertex s already equals k.) Finally, U.Remove(s) removes vertex s from priority queue U.

```
procedure CalculateKey(s)
{01} return [min(g(s), rhs(s)) + h(s_start, s) + k_m; min(g(s), rhs(s))];
procedure Initialize()
{02} U = ∅;
{03} k_m = 0;
{04} for all s ∈ S rhs(s) = g(s) = ∞;
{05} rhs(s_goal) = 0;
{06} U.Insert(s_goal, [h(s_start, s_goal); 0]);
procedure UpdateVertex(u)
{07} if (g(u) ≠ rhs(u) AND u ∈ U) U.Update(u, CalculateKey(u));
{08} else if (g(u) ≠ rhs(u) AND u ∉ U) U.Insert(u, CalculateKey(u));
{09} else if (g(u) = rhs(u) AND u ∈ U) U.Remove(u);
procedure ComputeShortestPath()
{10} while (U.TopKey() < CalculateKey(s_start) OR rhs(s_start) > g(s_start))
{11}     u = U.Top();
{12}     k_old = U.TopKey();
{13}     k_new = CalculateKey(u));
{14}     if(k_old < k_new)
{15}         U.Update(u, k_new);
{16}     else if (g(u) > rhs(u))
{17}         g(u) = rhs(u);
{18}         U.Remove(u);
{19}         for all s ∈ Pred(u)
{20}             if (s ≠ s_goal) rhs(s) = min(rhs(s), c(s, u) + g(u));
{21}             UpdateVertex(s);
{22}     else
{23}         g_old = g(u);
{24}         g(u) = ∞;
{25}         for all s ∈ Pred(u) ∪ {u}
{26}             if (rhs(s) = c(s, u) + g_old)
{27}                 if (s ≠ s_goal) rhs(s) = min_{s'∈Succ(s)}(c(s, s') + g(s'));
{28}             UpdateVertex(s);
procedure Main()
{29} s_last = s_start;
{30} Initialize();
{31} ComputeShortestPath();
{32} while (s_start ≠ s_goal)
{33}     /* if (rhs(s_start) = ∞) then there is no known path */
{34}     s_start = arg min_{s'∈Succ(s_start)}(c(s_start, s') + g(s'));
{35}     Move to s_start;
{36}     Scan graph for changed edge costs;
{37}     if any edge costs changed
{38}         k_m = k_m + h(s_last, s_start);
{39}         s_last = s_start;
{40}         for all directed edges (u, v) with changed edge costs
{41}             c_old = c(u, v);
{42}             Update the edge cost c(u, v);
{43}             if (c_old > c(u, v))
{44}                 if (u ≠ s_goal) rhs(u) = min(rhs(u), c(u, v) + g(v));
{45}             else if (rhs(u) = c_old + g(v))
{46}                 if (u ≠ s_goal) rhs(u) = min_{s'∈Succ(u)}(c(u, s') + g(s'));
{47}             UpdateVertex(u);
{48}         ComputeShortestPath();
```

Figure 1: *D* Lite (optimized version)*

not prescribe what it should do in this case.) The pseudo code uses the following notation. S denotes the finite set of vertices of the graph. $s_{start} \in S$ denotes the current vertex of the robot, initially the start vertex, and $s_{goal} \in S$ denotes the goal vertex. $Succ(s) \subseteq S$ denotes the set of successors of $s \in S$. Similarly, $Pred(s) \subseteq S$ denotes the set of predecessors of $s \in S$. $0 < c(s, s') \leq \infty$ denotes the cost of moving from s to $s' \in Succ(s)$. The heuristics $h(s, s')$ estimate the distance between vertex s and s'. D* Lite requires that the heuristics be non-negative and satisfy $h(s, s') \leq c(s, s')$ for all vertices $s \in S, s' \in Succ(s)$ and $h(s, s'') \leq h(s, s') + h(s', s'')$ for all vertices $s, s', s'' \in S$. The heuristics are guaranteed to have these properties if they have been derived by relaxing the graph, which will almost always be the case. We can prove a variety of properties about D* Lite, including its correctness, efficiency, and similarity to A*. Its correctness, for example, follows from the following theorem.

Theorem 1 *ComputeShortestPath() always terminates and one can then follow a shortest path from s_{start} to s_{goal} by always moving from the current vertex s, starting at s_{start}, to any successor s' that minimizes $c(s, s') + g(s')$ until s_{goal} is reached (ties can be broken arbitrarily).*

A detailed description of D* Lite is given in [9] and the proof of Theorem 1 is given in [7].

3 Greedy Mapping with D* Lite

We now show how D* Lite can be used to implement Greedy Mapping [10], a simple but powerful mapping strategy that has repeatedly been used on mobile robots by different research groups [16, 10, 13]. Greedy Mapping discretizes terrain into cells and then always moves the robot from its current cell to a closest cell with unknown blockage status, until the terrain is mapped. It has a number of desirable properties such as:

- **Small Mapping Time:** Greedy Mapping has a solid theoretical foundation that allows one to characterize its behavior analytically. For example, one can prove that its mapping time is reasonably small even though the robot moves greedily [10].
- **Action Recommendations:** Greedy Mapping makes only action recommendations and can thus coexist with other components of a robot architecture that also make action recommendations. For example, it does not need to have control of the robot at all times. If a robot has to re-charge its batteries during mapping, for instance, then it might have to preempt mapping and move to a known power outlet. Once restarted, Greedy Mapping is able to resume mapping from the power outlet, instead of having to return to the cell where mapping was stopped (which could be far away) and resume its operation from there.
- **Prior Knowledge:** Greedy Mapping takes advantage of prior knowledge about parts of the terrain (if available) since it uses all of its knowledge about the terrain when it determines which cell with unknown blockage status is closest to the robot and how to get there quickly. It does not matter whether this knowledge was previously acquired by the robot or provided to it.
- **Distributed Search:** Mapping tasks can be solved with several robots that each run Greedy Mapping and share their maps, thereby decreasing the mapping time. Cooperative mapping is a currently very active research area [4].

Greedy Mapping frequently needs to determine a shortest path from the current cell of the robot to a closest cell with unknown blockage status. In the following, we show how D* Lite can be used to implement Greedy Mapping efficiently. This is the first demonstration of how incremental heuristic search methods can be used to implement Greedy Mapping. The method is rather non-trivial and is, to the best of our knowledge, the first efficient implementation of Greedy Mapping to date.

3.1 Problem Representation

The mapping problem can be formulated as a graph coverage problem, for example, by imposing a regular eight-connected grid over the terrain. The vertices of the resulting directed graph correspond to cells and are either blocked or unblocked. The robot moves along the edges of the graph but cannot move onto blocked vertices. The robot knows the graph but initially does not know exactly which vertices are blocked. It can utilize initial knowledge about the blockage status of vertices in case it is available. For example, Figure 2 (left) shows a terrain with some prior knowledge about the blockage status of cells, and Figure 3 (left) shows the corresponding grid. Black cells (vertices) are known to be blocked and white cells are known to be unblocked. Grey cells have an unknown blockage status that the robot needs to determine. The robot has an on-board sensor that reports the blockage status of cells close to it, including the blockage status of the cells adjacent to the current cell of the robot. Greedy Mapping remembers this information and uses it to always move the robot on a shortest unblocked path to a closest cell with unknown blockage status. In Figure 2 (left), the cells are labeled with their distance to a closest cell with unknown blockage status. The robot follows a shortest unblocked path from its current cell to a closest cell with unknown blockage status by always moving to an adjacent cell so that it greedily decreases the distance of its current cell. The arrow shows such a path. While following this path, the robot is guaranteed to discover the blockage status of at least one cell with unknown blockage status and is thus guaranteed to make progress with mapping. This is so because it will eventually observe the blockage status of the cell it navigates to if it does not gain information about the blockage status of other cells earlier. Figure 2 (right) shows that a robot with a sensor range of two cells observes that cell B5 is blocked after it moves one cell to the east along the planned path. Whenever the robot gains information about the blockage status of cells, the shortest unblocked path from its current cell to a closest cell with unknown blockage status can change, either because the closest cell with unknown blockage status changes or the path to it changes. Similarly, whenever the robot deviates from the planned path, the shortest unblocked path from its current cell to a closest cell with unknown blockage status changes because its current cell is no longer on the planned path. In both

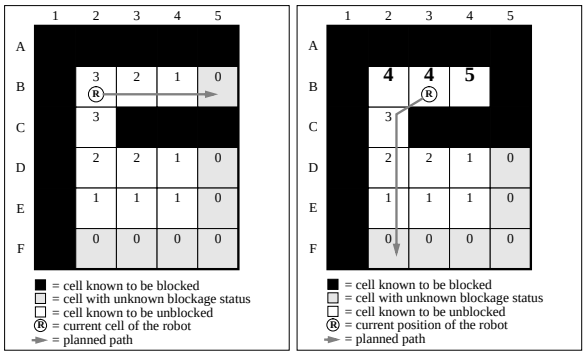

Figure 2: *Example Mapping Task*

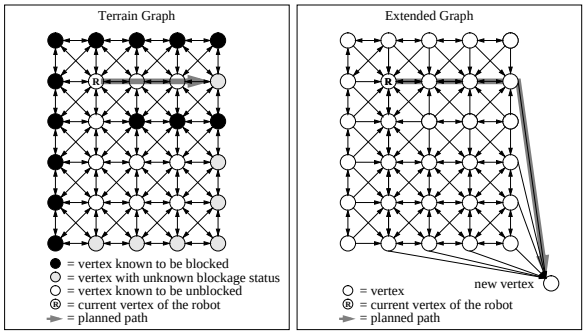

Figure 3: *Graphs for the Example Mapping Task*

cases, Greedy Mapping needs to recalculate a shortest unblocked path to a closest cell with unknown blockage status. Figure 2 (right) shows the new path.

Greedy Mapping could be implemented with any shortest path algorithm. In the following, we explain how we implement it with D* Lite and why this is a good idea. We introduce a new vertex (that becomes the goal vertex of D* Lite) and then construct the so-called extended graph as follows: First, the extended graph contains all edges of the original terrain graph (grid), except those edges that go from vertices that are known to be unblocked or potentially unblocked to vertices that are known to be blocked. This ensures that the planned path is unblocked. (The extended graph contains edges of the terrain graph that go from vertices that are known to be blocked to other vertices. This is important in case the robot has, due to sensor or position uncertainty, mistakenly classified an unblocked vertex as blocked and then, due to actuator uncertainty, deviates from the planned path and reaches this vertex. The robot believes that it can leave this vertex only if the edges of the extended graph that go from it to other vertices have not been deleted.) Some of the edges just described are deleted from the extended graph (by setting their cost to infinity) when the robot discov-

ers additional blocked cells. Second, the extended graph contains also edges that go from any vertex with unknown blockage status that can be reached with one edge traversal from vertices with known blockage status to the new vertex. This ensures that the planned path reaches a vertex with unknown blockage status and, from there, the new vertex. Some of the edges just described are added to or deleted from the extended graph (by setting their cost to one or infinity, respectively) when the robot discovers the blockage status of additional cells. Figure 3 shows the extended graph (right) that corresponds to the graph (left) that models the terrain in Figure 2 (left). A shortest unblocked path on the extended graph from the current vertex of the robot to the new vertex corresponds to a shortest unblocked path on the terrain graph from the current vertex of the robot to a closest vertex with unknown blockage status, and vice versa. Thus, Greedy Mapping can determine a shortest unblocked path from the current vertex of the robot in the terrain graph to a closest vertex with unknown blockage status by finding a shortest path in the extended graph from the current vertex of the robot to the new vertex. Thus, D* Lite can be applied to the extended graph where s_{start} corresponds to the current vertex of the robot and s_{goal} corresponds to the new vertex. This way it finds the path in the extended graph that is shown in Figure 3 (right). This path corresponds to the path in the terrain graph that is shown in Figure 3 (left) and the path in the terrain that is shown in Figure 2 (left).

D* Lite can use heuristic information to speed up its search, in form of estimates of the distances of vertices from the current vertex of the robot, which can be obtained easily for Greedy Mapping. D* Lite can also use information from previous searches to speed up its current search. Figure 2 demonstrates how reusing information can potentially save search effort in the context of Greedy Mapping. The left part of the figure shows the distances of all cells that are known to be unblocked to a closest cell with unknown blockage status. The right part of the figure shows the same distances after the robot moved one cell to the east along the planned path and gained information about the blockage status of cell B5. All but three distances (shown in bold) remain unchanged and therefore do not need to get recomputed even though the path changes completely. This suggests that reusing information from previous searches can potentially reduce the search time of heuristic search methods for Greedy Mapping. For example, when replanning the shortest path in Figure 3 (right), D* Lite only expands the three vertices that correspond to cells whose distances to a closest cell with unknown blockage status have changed (namely B2, B3, and B4), where we say that D* Lite expands a vertex when it executes lines {16-28} and thus calculates the g-value of the vertex, similar to what A* does when it expands a vertex. On the other hand, A* expands five vertices even with the best possible tie-breaking criterion and is thus less efficient than D* Lite.

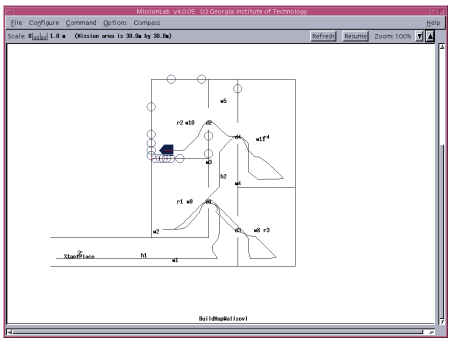

Figure 4: *Greedy Mapping in MissionLab*

3.2 Integration into Robot Architectures

We integrated Greedy Mapping into a multi-task autonomous robot architecture called MissionLab [11], which is a version of the Autonomous Robot Architecture (AuRA) [2]. AuRA is a hybrid system that consists of a schema-based reactive system at the low level and a deliberative system based on finite state automata at the high level. The reactive component consists of primitive behaviors called motor schemata [1] that are grouped into behavioral assemblages. Each behavior of a behavioral assemblage produces its own recommendation for how the robot should move, in form of a vector. The robot then moves in the direction of the weighted average of all vectors.

We utilize that Greedy Mapping makes only action recommendations and can thus coexist with other components of a robot architecture that also make action recommendations. This allows us to implement map building as a behavioral assemblage of three behaviors that takes as parameters the bounds of the area that the robot needs to map: GreedyMapping is a behavior that directs the robot towards a closest cell with unknown blockage status; AvoidObstacles is a behavior that repels the robot from obstacles; and Wander is a behavior that injects some noise. The weight of AvoidObstacle and the distance within which obstacles affect the robot are set depending on the size of the cells. For grids with small (high-resolution) cells, the weight can be set to zero since Greedy Mapping can directly take the obstacles into account. For grids with large cells, the weight can be set to a positive value while the distance within which obstacles affect the robot is made small in comparison to the cell size. This ensures that the robot successfully navigates around small obstacles. The weight of Wander is configured similarly.

We ran Greedy Mapping both on a Nomad 15 with a Sick

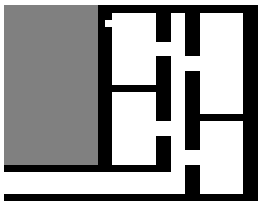

Figure 5: *Acquired Map*

LMS200 laser scanner and in simulation. Figure 4 shows a snapshot of a MissionLab simulation after Greedy Mapping finished mapping the given office environment. The terrain was discretized with a granularity of 0.5 meters, resulting in a maze of size 35 by 27. The robot started at the cell labeled StartPlace. Its obstacle sensor had a range of 8 meters and the circles in Figure 4 show the final sensor readings. The figure also shows the path that the robot took. It is rather close to the shortest path necessary for mapping the terrain, which demonstrates that the mapping time of Greedy Mapping is usually reasonably small. Figure 5 shows the acquired map. Black cells are blocked, white cells are unblocked, and grey cells are unreachable. Thus, their blockage status cannot be determined by the robot. The acquired map corresponds well to the actual floor plan.

3.3 Experiments

In the following, we study to which degree the combination of incremental and heuristic search that D* Lite implements outperforms incremental or heuristic searches individually. We thus compare Greedy Mapping with D* Lite, D* Lite without heuristic search, and D* Lite without incremental search (that is, A*). We decided not to include D* Lite without incremental and heuristic search (that is, breadth-first search) because it performs so poorly that graphing its performance becomes a problem. Since all versions of D* Lite move the robot in the same way, we only need to compare their total planning time. Since the actual planning times are implementation-dependent, we instead use three performance measures that all correspond to common operations performed by D* Lite and thus heavily influence its planning time: the total number of vertex expansions, the total number of heap percolates (exchanges of a parent and child in the heap), and the total number of vertex accesses (for example, to read or change their values). Figure 6 reports not only the means of the three performance measures but also the corresponding 95 percent confidence intervals to demonstrate that our conclusions are statistically significant.

We perform experiments in 25 randomly generated terrains of size 64 by 25 that are represented as eight-connected grids and resemble office environments. Since

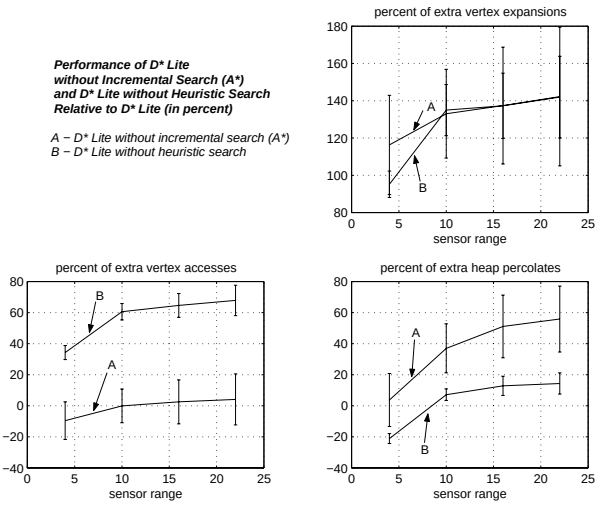

Figure 6: *Experimental Results (1)*

the robot is allowed to move diagonally, we calculate the heuristic approximations of the distances between two vertices as the maximum of the absolute differences of their x and y coordinates. For example, the heuristic of cell B4 is two in Figure 2 (left) and one in Figure 2 (right). We use the MissionLab simulation to be able to average the results over several runs. We varied the sensor range of the robot to simulate both short-range and long-range sensors. For example, if the sensor range is four, then the robot can sense all blocked cells that are up to four cells in any direction away from the robot as long as they are not blocked from view by other blocked cells. Figure 6 shows the three performance measures for Greedy Mapping with D* Lite without heuristic search and D* Lite without incremental search (that is, A*) as percent difference relative to Greedy Mapping with D* Lite. (Thus, D* Lite always scores zero.) As can be seen, the number of vertex expansions of D* Lite is always far less than that of the other two algorithms. This also holds for the number of heap percolates and vertex accesses, with the exception of sensor range four for the heap percolates. The advantage of D* Lite over the other two search methods seems to increase as the sensor range increases, that is, the larger the number of cells is whose blockage status the robot can sense without moving. This implies that the advantage of D* Lite increases if the robot uses sensors with longer ranges (which is important since laser scanners tend to be the sensors of choice for mapping) or discretizes the terrain in a more fine-grained way. Only for the number of vertex accesses is the difference between D* Lite and D* Lite without incremental search statistically not significant although its mean is smaller for D* Lite than D* Lite without incremental search for the larger sensor ranges.

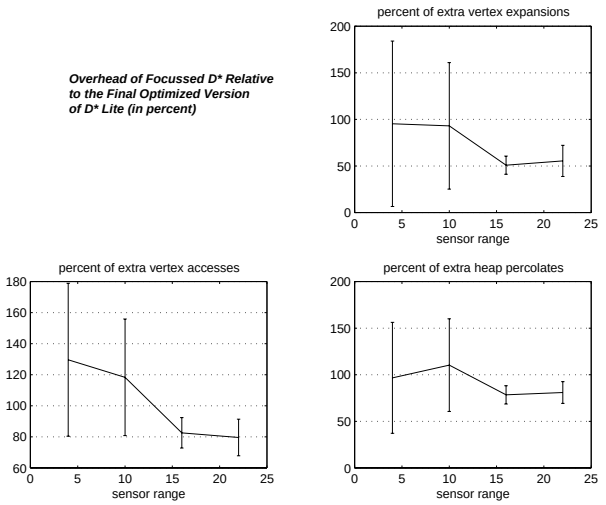

Figure 7: *Experimental Results (2)*

We also use the same setup to compare D* Lite and the original Focussed D*. Figure 7 shows the three performance measures for Greedy Mapping with both search methods and demonstrates that D* Lite performs better than Focussed D* with respect to all three measures. This is consistent with the results we reported when comparing D* Lite and Focussed D* for goal-directed navigation in unknown terrain [9].

4 Conclusions

In this paper, we have explained how incremental heuristic search methods such as our D* Lite apply to mapping of unknown terrain, a new application of incremental heuristic search. Our results show that the combination of incremental and heuristic search that D* Lite implements speeds up the planning time over incremental or heuristic searches individually. These results demonstrate the versatility and computational benefits of incremental heuristic search, an underexploited technique that should be used more often in mobile robotics.

Acknowledgments

The Intelligent Decision-Making Group is partly supported by NSF awards under contracts IIS-9984827, IIS-0098807, and ITR/AP-0113881 as well as an IBM faculty partnership award. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations and agencies or the U.S. government.

References

[1] R. Arkin. Motor-schema based mobile robot navigation. *International Journal of Robotics Research*, 8(4):92–112, 1989.

[2] R. Arkin and T. Balch. AuRA: Principles and practice in review. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2):175–189, 1997.

[3] M. Barbehenn and S. Hutchinson. Efficient search and hierarchical motion planning by dynamically maintaining single-source shortest paths trees. *IEEE Transactions on Robotics and Automation*, 11(2):198–214, 1995.

[4] W. Burgard, D. Fox, M. Moors, R. Simmons, and S. Thrun. Collaborative multi-robot exploration. In *Proceedings of the International Conference on Robotics and Automation*, pages 476–481, 2000.

[5] T. Ersson and X. Hu. Path planning and navigation of mobile robots in unknown environments. In *Proceedings of the International Conference on Intelligent Robots and Systems*, 2001.

[6] Y. Huiming, C. Chia-Jung, S. Tong, and B. Qiang. Hybrid evolutionary motion planning using follow boundary repair for mobile robots. *Journal of Systems Architecture*, 47(7):635–647, 2001.

[7] S. Koenig and M. Likhachev. Improved fast replanning for robot navigation in unknown terrain. Technical Report GIT-COGSCI-2002/3, College of Computing, Georgia Institute of Technology, Atlanta (Georgia), 2001.

[8] S. Koenig and M. Likhachev. Incremental A*. In *Advances in Neural Information Processing Systems 14*, 2001.

[9] S. Koenig and M. Likhachev. Improved fast replanning for robot navigation in unknown terrain. In *Proceedings of the International Conference on Robotics and Automation*, 2002.

[10] S. Koenig, C. Tovey, and W. Halliburton. Greedy mapping of terrain. In *Proceedings of the International Conference on Robotics and Automation*, pages 3594–3599, 2001.

[11] D. Mackenzie, R. Arkin, and J. Cameron. Multiagent mission specification and execution. *Autonomous Robots*, 4(1):29–57, 1997.

[12] L. Podsedkowski, J. Nowakowski, M. Idzikowski, and I. Vizvary. A new solution for path planning in partially known or unknown environment for nonholonomic mobile robots. *Robotics and Autonomous Systems*, 34:145–152, 2001.

[13] L. Romero, E. Morales, and E. Sucar. An exploration and navigation approach for indoor mobile robots considering sensor's perceptual limitations. In *Proceedings of the International Conference on Robotics and Automation*, pages 3092–3097, 2001.

[14] A. Stentz. The focussed D* algorithm for real-time replanning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1652–1659, 1995.

[15] M. Tao, A. Elssamadisy, N. Flann, and B. Abbott. Optimal route re-planning for mobile robots: A massively parallel incremental A* algorithm. In *International Conference on Robotics and Automation*, pages 2727–2732, 1997.

[16] S. Thrun, A. Bücken, W. Burgard, D. Fox, T. Fröhlinghaus, D. Hennig, T. Hofmann, M. Krell, and T. Schmidt. Map learning and high-speed navigation in RHINO. In D. Kortenkamp, R. Bonasso, and R. Murphy, editors, *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*, pages 21–52. MIT Press, 1998.