Does Fast Beat Thorough? Comparing RTAA* and LSS-LRTA*

Shane Kochvi and **Wheeler Ruml**
Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA
sh129 at wildcats.unh.edu and ruml at cs.unh.edu

Abstract

The RTAA* algorithm has been proposed as an alternative to the LSS-LRTA* algorithm for heuristic search under hard time constraints. It uses a cruder but faster learning procedure. Experimental results on pathfinding in mazes in the unknown terrain setting indicated that RTAA* was superior to LSS-LRTA*. In this paper, we attempt to replicate those results and we extend the analysis to additional domains. Our results suggest that RTAA* is superior to LSS-LRTA* only in situations where the heuristic is already relatively accurate, either inherently or because of large lookahead. Overall, neither algorithm seems superior to the other.

Introduction

Applications of heuristic search to on-line planning often require that an agent begin executing a partial plan before a complete path to a goal state can be found. Real-time search algorithms guarantee that an agent's next action is selected within a hard pre-specified time bound. They usually incorporate a learning mechanism that prevents infinite loops and guarantees that, under certain conditions, the agent will reach a goal despite limited planning. These algorithms can also be adapted for planning in a partially-known state space in which additional information becomes available during action execution. Local Search Space Learning Real-Time A* (Koenig and Sun 2009, LSS-LRTA*) is one of the most popular real-time methods. It alternates between an A*-style lookahead phase and a Dijkstra-style learning phase in which the heuristic values of nodes within the local search space are updated. Real-Time Adaptive A* (RTAA*) was proposed by Koenig and Likhachev (2006) as an alternative to LSS-LRTA*. It uses a faster but less thorough strategy for updating heuristic values. The central question is whether this faster strategy, by allowing deeper lookahead within the same time bound, can overcome the less informed heuristic values that are learned. Koenig & Likhachev present empirical results on maze navigation with unknown terrain which suggest that yes, RTAA* performs significantly better for the same time bound. Subsequent authors have built additional algorithms on top of RTAA*, and the algorithm is wide cited. So it is important to know whether the results on unknown mazes generalize to other domains.

In this paper, we replicate the experiments presented by Koenig & Likhachev and evaluate the algorithms' performance in additional domains, including the sliding tile puzzle and a video game with directed state transitions. Unlike the analysis of Arica et al. (2017), in order to capture the potential advantage of RTAA*, we compute trajectory cost subject to a time bound on search iterations. Our empirical results (summarized in Table 1) suggest that LSS-LRTA* is superior to RTAA* in situations where the heuristic is relatively inaccurate or the lookahead is relatively small. RTAA* does well when the heuristic is already quite strong or ample lookahead is available. We did not find support for the notion that RTAA* is generally superior to LSS-LRTA*.

Background

Both RTAA* and LSS-LRTA* interleave planning and action execution. Planning consists of a lookahead phase, during which an expansion-bounded A* generates the local search space around an agent, discovering the optimal path from the initial node to all frontier nodes, and a learning phase, during which the heuristic values of the nodes within the local search space are updated based on information obtained during lookahead. Action execution is simply the transitioning of an agent from node to node along the cost-minimal path discovered during A* lookahead.

The main difference between the two algorithms is in the way they handle the learning phase. LSS-LRTA* uses Dijkstra's algorithm to update the heuristic values of the nodes within the local search space in an outside-in manner, starting with the nodes on the frontier of the local search space. After learning, every node within the local search space has a heuristic value equal to the minimum, over all frontier nodes, of the cost to reach the frontier node plus the heuristic value of that frontier node. In other words, a lower bound on its best path to a goal through the frontier. While this seems like optimal use of the information available to the algorithm, note that Dijsktra's algorithm does require a heap to prioritize the backing up of low h values.

Instead of using a Dijkstra-style propagation, RTAA* updating the heuristics of the nodes in the local search space using:

$$h(n) \leftarrow g(b) + h(b) - g(n)$$

where n is the node to be updated and b is the best node on the frontier (that was about to be expanded during the

A* lookahead). Note that this update can be performed on nodes in any order using simple iteration — in particular, no heap is required. While we would expect LSS-LRTA* to increase heuristic values more, resulting in superior accuracy, we would expect RTAA*'s updates to occur more quickly.

Experimental Set-up

We implemented RTAA* and LSS-LRTA* in C++. We separated the search algorithms and problem domains so that the search algorithms remain general-purpose. We used heap-based open lists and hash tables for closed lists. Ties between nodes with identical f-values were broken in favor of nodes with greater g values. Remaining ties are broken randomly using a random integer assigned to each node. New nodes are allocated from a memory pool. Experiments were run on a machine with a 3.16 GHz Core2 duo E8500 and 8GB RAM.

The A* lookahead phase is limited in the number of nodes it can expand. To infer the performance of the algorithms with a time-based bound, amortized CPU time per search episode was calculated by dividing the total planning time by the number of search episodes. In several of the plots below, we compute the difference in performance between the two algorithms at the same time bound. However, since our time-based data was derived from node bounds, it is rarely the case that we had data for both algorithms at the same time bound. To allow comparison, we used linear interpolation to infer what LSS-LRTA*'s performance would have been for each time per search episode used by RTAA*.

Due to space limitations, we report only the most important results in this paper. For additional plots and analysis, please refer to Kochvi (2017).

Grid Pathfinding

Inspired by video games, grid pathfinding is a popular benchmark for real-time search. Following Koenig & Likhachev, we used four-way movement and the Manhattan distance heuristic, which yields perfect estimates in the absence of obstacles. We used two different versions of each algorithm: one in which the agent knows ahead of time which cells are blocked ("known terrain") and another in which the agent only learns if a cell is blocked when directly adjacent to it ("unknown terrain"). During lookahead, the agent assumes unobserved cells are free of obstacles (the "freespace assumption"). If the agent encounters an obstacle during action execution, it begins a new lookahead phase. The only significant implementation difference is that, in unknown terrain, obstacle states were stored in a separate hash table as they were observed.

Mazes

Following Koenig & Likhachev, we used depth-first search to generate 2,500 mazes of size 151×151. Start and end cells were selected uniformly at random. We note that mazes are a bit atypical as heuristic search benchmarks. During lookahead, most states will have only one non-duplicate successor state. The heuristic is very uninformed. And solutions may involve visiting a relatively large fraction of the state space.

Unknown Terrain The experiments in the original RTAA* paper were performed on mazes with unknown terrain. Our CPU times were about 10 times longer than those reported by Koenig & Likhachev. Because they only report results in one domain, it seems likely that they used a 2D array to store nodes for fast access, while we used a hash table for generality. We also detected some anomalies in their statistical analysis (see (Kochvi 2017) for details). After correcting for this, our overall results matched the original paper well, giving us confidence in our implementation. We did notice one subtle trend when comparing our LSS-LRTA* results: our implementation seems to perform better at lower lookaheads and worse at higher lookaheads than theirs. Our trajectory cost, lookahead node expansions, and search episodes were around 10% lower for a lookahead of 1, rising to a few percent more for a lookahead of 89 (the highest lookahead reported in the original paper). It was not clear what might explain these differences.

Figure 1 presents some of our results. The left panel shows the total number of nodes expanded during all lookahead phases under the goal is reached. At first, as the number of expansions allowed per lookahead increases, better actions can be selected and fewer expansions are required to reach the goal. However, the benefit of lookahead quickly decreases and then becomes a liability. With larger lookaheads in unknown terrain, more of the obstacles that actually lie within the local search space are incorrectly assumed to be traversable due to the freespace assumption and thus a greater proportion of the path given by the A* search will be blocked by obstacles. Only a modest amount of lookahead is actually helpful.

The center panel shows, as expected, that RTAA* has lower CPU time per planning episode (in microseconds) than LSS-LRTA* for a given number of lookahead expansions. This difference can be attributed to the computational overhead of the learning phase.

Finally, the right panel shows the main performance metric: difference in total trajectory cost as a function of CPU time taken per search episode. Darker data points represent time bounds at which RTAA* outperforms LSS-LRTA*. Koenig & Likhachev report that when a search episode is limited to 20.99 microseconds or less, RTAA* delivers lower trajectory costs than LSS-LRTA*. However our data suggest that LSS-LRTA* performs better at the lower and mid-range lookaheads, while RTAA* is better at higher ones, with its advantage increasing as the time per search episode increases. We fit a regression line (-2476.624 + 191233.06/planning time, $r^2 = 0.87$, shown in purple in the figure): its x-intercept is 77.22us per search episode. Above this, RTAA* performed better. Learning is crucial in mazes, where the Manhattan distance heuristic is quite poor. Our working hypothesis is that, as lookahead increases, this compensates for the weaker learning of RTAA*. We believe that the freespace assumption may also play a role here. With larger lookaheads, a larger proportion of the nodes within the local search space will be incorrectly assumed to be traversable. LSS-LRTA*'s fine-grained learning strategy combined with these incorrect estimates may actually result in less accurate updates to the heuristic values than those

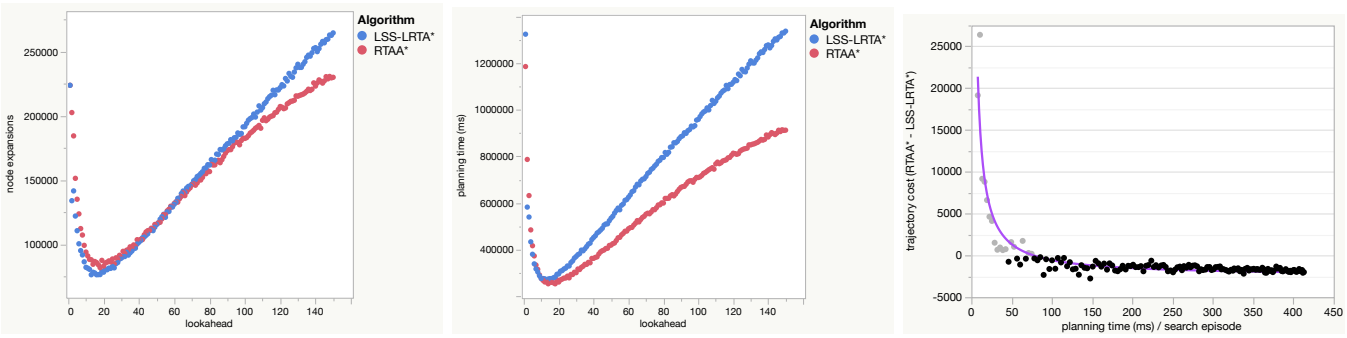

Figure 1: Mazes, unknown terrain: (left) expansions, (center) planning time, (right) trajectory cost difference vs lookahead

given by RTAA*, whose learning strategy depends on only a single node.

Known Terrain Figure 2 shows results on the same mazes when assuming known terrain. The left panel shows very different behavior from the unknown terrain case. Because the terrain is known, increased lookahead results in more viable actions and fewer node expansions overall for both algorithms. LSS-LRTA* requires fewer expansions than RTAA*. The center panel shows performance vs lookahead. For very short time bounds, LSS-LRTA* outperforms RTAA*. For other bounds, neither clearly outperforms the other. This is different from unknown terrain, where RTAA* was clearly superior for mid to large bounds. The thorough learning strategy of LSS-LRTA* is more effective when the heuristic values derived from lookahead are better informed, as they are in known terrain , allowing it to be competitive with RTAA* despite its computational overhead.

Random Maps

In many search problems, the heuristic function is more helpful than it is in mazes. To vary heuristic accuracy, we ran both algorithms over 512×512 grid maps randomly filled with obstacles at 10,20,30, and 40% density. The Manhattan distance heuristic decreases in accuracy as the obstacles density increases. The library (Sturtevant 2012) from which they were sourced orders scenarios into buckets according to optimal solution length. We chose 100 instances from the buckets with the longest optimal solution lengths, because in real-time applications, worst-case performance typically drives design decisions.

Unknown Terrain The right panel of Figure 2 and the three panels of Figure 3 show algorithm performance as the obstacles density increases. With 10% or 20% obstacles, the heuristic is very accurate and LSS-LRTA* has no advantage over RTAA*. With 30%, we start to see the now-familiar advantage at low lookaheads. As with mazes, due to the freespace assumption with unknown terrain, the benefit of lookahead fades as lookahead increases. With 40% obstacles, the problems become difficult and LSS-LRTA* has a clear advantage at all but the largest lookaheads.

Known Terrain Plots for the known terrain case are omitted for space as they are qualitatively similar to those for unknown terrain, although perhaps more extreme: the 10% and 20% cases show significant advantage for RTAA* throughout, the 30% case swings quickly from a large advantage for LSS-LRTA* at low lookaheads to a strong advantage for RTAA* at high lookaheads, and 40% cases illustrates dominance of LSS-LRTA* throughout. As in the unknown terrain case, RTAA* performs better with fewer obstacles and with a longer time-limit. One could say that RTAA* performs well when the heuristic is already accurate or when deep lookahead can provide very informed frontier values.

Game Map

To provide a benchmark more representative of potential applications than random grids, we also ran on the orz100d map from the game Dragon Age: Origins (Sturtevant 2012), shown in Figure 4. The map has a healthy mix of cluttered areas and open space. Overall, 38% of the grid nodes are obstacles. As before, this benchmark comes with start-goal pairs organized by optimal solution length — we used 100 pairs from the buckets with the longest optimal solution lengths.

Results for the unknown terrain setting are shown in the left and center panels Figure 5. They are quite different from the corresponding plots for random maps, and are perhaps most similar to the results obtained from mazes with known terrain. LSS-LRTA* performs strongly for low and mid-range lookaheads, with RTAA* starting to perform competitively for high lookaheads. Very similar results were obtained for the known terrain setting (plots omitted). While we do not have any firm hypotheses why the game map behaves so differently, it is clear that our results do not point to superiority of RTAA* for this problem.

Traffic

While the previous grid pathfinding benchmarks were inspired by video games, they do not model an important characteristic of certain games: directed state transitions. In many state spaces, certain actions are irreversible and several actions can be required in order to revisit the same state (if it is possible at all). We implemented a benchmark used by Kiesel, Burns, and Ruml (2015) that is modeled on the game Frogger. In the version used here, a 100×100 grid is filled 50% with obstacles, with start and goal states in each

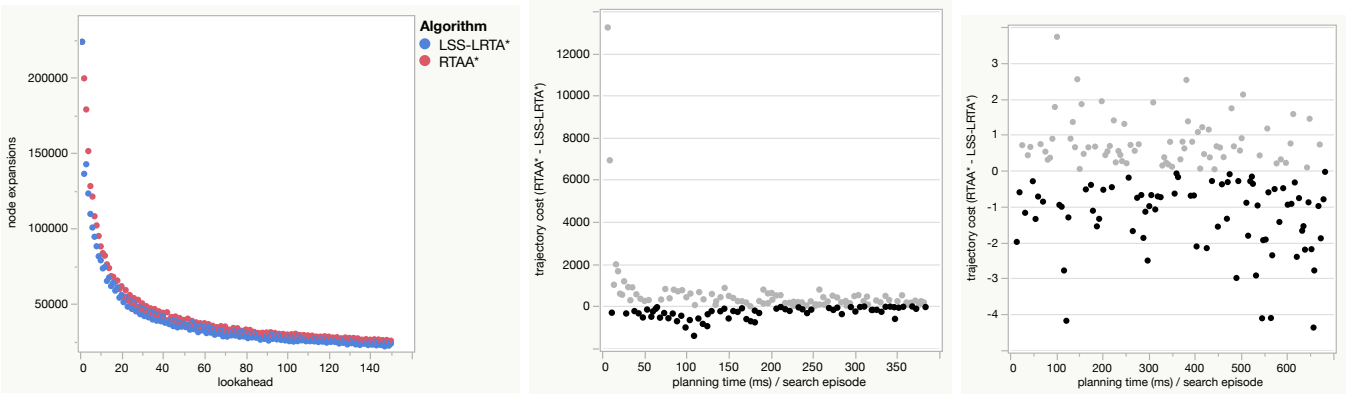

Figure 2: Mazes, known terrain: (left) expansions, (center) performance. (right) Random maps, unknown terrain, performance

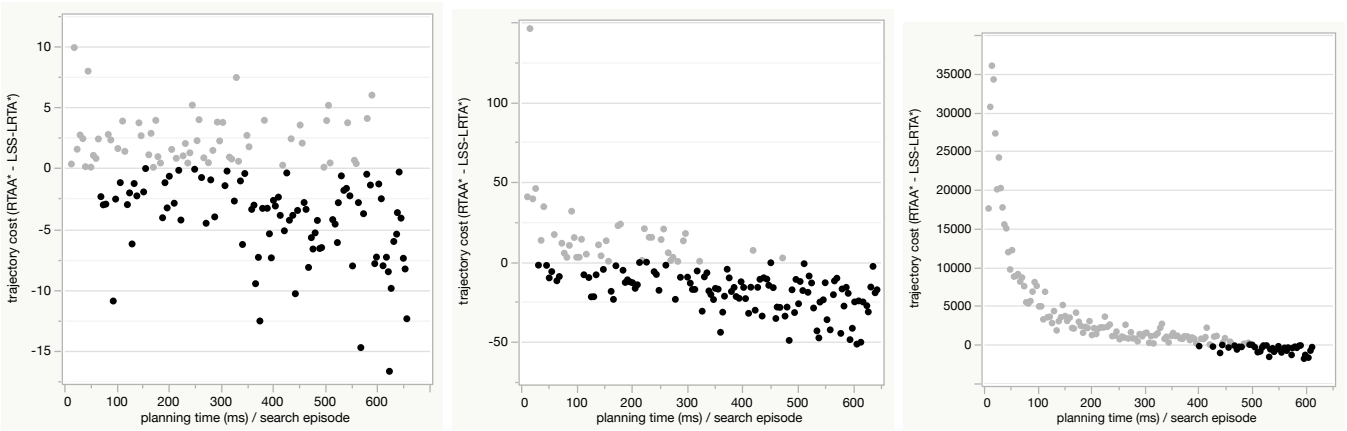

Figure 3: Random Maps, Unknown Terrain: Trajectory Cost vs. Time Bound: (left) 20% obstacles, (center) 30%, (right) 40%

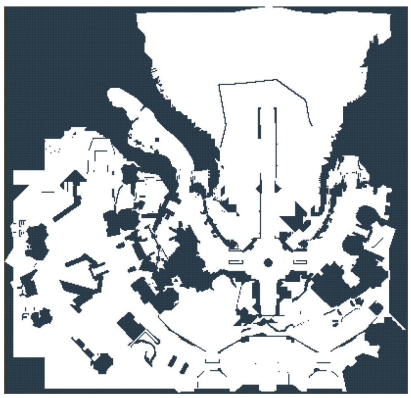

Figure 4: orz100d

corner. Each obstacle has a cardinal direction. At every time step, the obstacles move to adjacent locations according to their direction (bouncing off the edges of the world) and the agent can move in a cardinal direction or stay in the same location. Each state is defined by a particular configuration of obstacles and the location of the agent. If, in the course of events, no successor states are available for the agent, the agent is returned to the start state. Because computing a successor state involves moving all the obstacles, which can be expensive, a hash table was used by the successor generator to cache generated states. The heuristic used is Manhattan distance. The domain is fully observable. We used lookaheads from 1–150 nodes.

The right panel of Figure 5 shows the difference in performance between the two algorithms. There is significant noise, possibly due to the large effect on trajectory cost of restarting back at the start location upon collisions. Certainly, there does not seem to be evidence favoring one algorithm over another.

Sliding Tile Puzzle

We ran both algorithms over 100 instances of the 4 x 4 sliding tile puzzle from the classic problem set from Korf (1985) and averaged their results. We chose as lookaheads multiples of 100 up to 10,000. The heuristic was Manhattan distance. We tested the algorithms with three different cost functions: unit (every action costs 1), heavy (the cost to move a tile is equal to the tile's number), and square root (the cost to move a tile is equal to the square root of the tile's number).

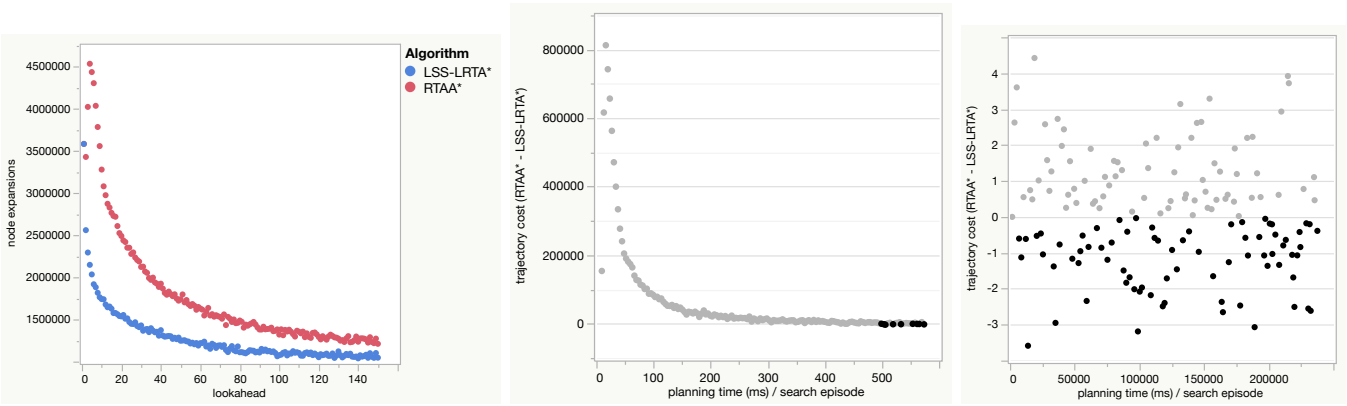

Figure 5: Game map, unknown terrain : (left) expansions, (center) performance. (right) Traffic performance.

	low	high
mazes unk	LSS	RTAA*
mazes known	LSS	same
random unk 10%	same	RTAA*
random unk 20%	LSS	RTAA*
random unk 30%	LSS	RTAA*
random unk 40%	LSS	RTAA*
random known 10%	RTAA*	
random known 20%	RTAA*	
random known 30%	LSS	RTAA*
random known 40%	LSS	
orz100d unk	LSS	same
orz100d known	LSS	same
traffic	same	
tiles	maybe RTAA*	

Table 1: Summary of results by time bound

To summarize the results (omitted for space), the algorithms appeared roughly comparable and there were no discernible trends in performance versus lookahead. Overall, RTAA* outperformed LSS-LRTA* 74%,71%, and 75% of the time for unit, heavy, and square root action costs respectively. Wilt and Ruml (2011) report results suggesting that the Manhattan distance heuristic is most effective for unit, less effective for square root, and least effective for heavy. This accords with the pattern of results that we see: RTAA* is most effective when the heuristic is strong, and LSS-LRTA* becomes more effective as learning becomes more important.

Conclusions

Table 1 summarizes our results. We saw that RTAA* was superior to LSS-LRTA* in cases where the heuristic was relatively strong or the lookahead relatively large. In our benchmarks, there were just as many situations in which LSS-LRTA* was superior as vice versa. Our results do not support a claim that one algorithm is consistently better than the other.

Probably the greatest limitation of our analysis is that we did not compare the implementation using by Koenig & Likhachev to ours. While our comparison is fair in the sense that we maximally shared code between algorithms, implementations with domain-specific optimizations would reduce the time spend on domain operations, magnifying differences in search operations such as the learning overhead advantage of RTAA*.

In addition, although our study uses 16 different benchmarks, any finite set gives only a limited view of algorithm performance. Theoretical work can yield more general results, although often this requires oversimplification.

References

Arica, N.; Mut, A.; Yorukcu, A.; and Demir, K. A. 2017. An empirical comparison of search approaches for moving agents. *Computational Intelligence* 33(3):368 – 400.

Kiesel, S.; Burns, E.; and Ruml, W. 2015. Achieving goals quickly using real-time search: experimental results in video games. *Journal of Artificial Intelligence Research* 54:123–158.

Kochvi, S. 2017. Does fast beat thorough?: Comparing real-time search algorithms LSS-LRTA* and RTAA*. Bachelor's thesis, University of New Hampshire.

Koenig, S., and Likhachev, M. 2006. Real-time adaptive A*. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '06, 281–288.

Koenig, S., and Sun, X. 2009. Comparing real-time and incremental heuristic search for real-time situated agents. *Autonomous Agents and Multi-Agent Systems* 18(3):313–341.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence* 27(1):97–109.

Sturtevant, N. 2012. Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games* 4(2):144 – 148.

Wilt, C., and Ruml, W. 2011. Cost-based heuristic search is sensitive to the ratio of operator costs. In *Proceedings of SoCS*.