Position Paper: Incremental Search Algorithms Considered Poorly Understood

Carlos Hernández[*]
Depto. de Ingeniería Informática
Universidad Católica de la Ssma. Concepción
Caupolican 491, Concepción, Chile
chernan@ucsc.cl

Jorge Baier
Computer Science Department
Pontificia Universidad Católica de Chile
Santiago, Chile
jabaier@ing.puc.cl

Tansel Uras and Sven Koenig
Computer Science Department
University of Southern California
Los Angeles, CA 90089, USA
{turas,skoenig}@usc.edu

Abstract

Incremental search algorithms, such as D* Lite, reuse information from previous searches to speed up the current search and can thus solve sequences of similar search problems faster than Repeated A*, which performs repeated A* searches. In this position paper, we study goal-directed navigation in initially unknown terrain and point out that it is currently not well understood when D* Lite runs faster than Repeated A*. In general, it appears that Repeated A* runs faster than D* Lite for easy navigation problems (where the agent reaches the goal with only a small number of searches), which means that it runs faster than D* Lite quite often in practice. We draw two conclusions, namely that incremental search algorithms need to be evaluated in more diverse testbeds to improve our understanding of their properties and that they can be improved to be more competitive for easy navigation problems.

We study goal-directed navigation with the freespace assumption in initially unknown terrain, as needed in robotics and video games. The terrain is discretized into a grid of known dimensions. The agent does not know initially which cells are blocked but always observes the blockage status of the neighboring cells of its current cell and adds them to its map. It can then move to any unblocked neighboring cell. In this paper, the agent moves to a goal cell with given coordinates using the following navigation strategy: It finds a shortest (unblocked) path from its current cell to the goal cell. If such a path does not exist, it stops unsuccessfully. Otherwise, it follows the path until it either reaches the goal cell, in which case it stops successfully, or observes the path to be blocked, in which case it repeats the process. The agent thus needs to solve a sequence of similar search problems fast. Incremental search algorithms, such as D* Lite (Koenig and Likhachev 2005), reuse information from previous searches to speed up the current search. We claim that it is currently not well understood when D* Lite runs faster than Repeated Forward A*.[1] For example, it appears that Repeated Forward A* runs faster than D* Lite in many cases, typically for easy navigation problems where the agent observes its path to be blocked only a small number of times and thus performs only a small number of searches before it reaches the goal cell (or discovers that this is impossible). Examples are gridworlds where the start and goal cells are close to each other or where the h-values are not misleading (including where only a small number of cells are blocked). The reason appears to be the following: D* Lite has the advantage that it typically expands fewer cells than Repeated Backward A* after the first search since it reuses information from previous searches. However, D* Lite has the disadvantage that Repeated Backward A* typically expands more cells during the first searches than Repeated Forward A* during the first searches. D* Lite also expands cells more slowly than Repeated (Forward or Backward) A*. This means that the first search of D* Lite typically runs more slowly than the first search of Repeated Forward A*, an effect that becomes the more pronounced the further apart the start and goal cells are. If the number of subsequent searches needed for the agent to reach the goal cell is small then typically D* Lite runs more slowly than Repeated Forward A*.

[*]Our research was supported by NSF, ARO and ONR grants to Sven Koenig (while he served at NSF) and a Fondecyt grant to Jorge Baier.

[1]Repeated Forward A* performs repeated A* searches from the current cell of the agent to the goal cell and typically expands fewer cells than Repeated Backward A* during the first searches and thus runs faster, see (Koenig and Likhachev 2005) for an explanation. We use a version of Repeated Forward A* that finds a shortest path whenever the agent observes a blocked cell on its path (as most evaluations do), different from the previous evaluation in (Koenig and Likhachev 2005) where it finds a shortest path whenever the agent observes any blocked cell. D* Lite is a version of Repeated Backward A* that reuses information from previous searches. We use a version of D* Lite that breaks ties among cells with the same f-value in favor of smaller g-values since it typically runs faster than a version of D* Lite that breaks ties in the opposite direction because it a) expands more cells during the first search but fewer cells during subsequent searches and b) expands cells faster since its f-values are pairs rather than triples.

Blocked	Length	Searches	Algorithm	Expansions	Runtime (ms)	Faster
8%	725.8	58.7	Rep. Forw. A*	25,235	6.76	95.4%
			D* Lite	113,368	24.93	4.6%
16%	812.3	128.2	Rep. Forw. A*	52,527	13.47	86.9%
			D* Lite	113,387	26.37	13.1%
24%	996.1	225.8	Rep. Forw. A*	89,792	22.35	77.4%
			D* Lite	114,957	28.81	22.6%
32%	1562.3	429.2	Rep. Forw. A*	170,626	41.52	53.4%
			D* Lite	123,482	36.07	46.6%
40%	13880.3	3671.0	Rep. Forw. A*	21,885,533	4,195.11	3.4%
			D* Lite	299,762	172.95	96.6%
Total	3595.4	902.6	Rep. Forw. A*	4,444,743	855.85	63.3%
			D* Lite	152,991	57.82	36.7%

Table 1: Results for Random Four-Neighbor Grids

Blocked	Length	Searches	Algorithm	Expansions	Runtime (ms)	Faster
10%	569.8	48.0	Rep. Forw. A*	14,725	4.23	91.6%
			D* Lite	45,753	13.41	8.4%
20%	596.2	98.2	Rep. Forw. A*	29,494	8.28	84.5%
			D* Lite	47,334	15.01	15.5%
30%	644.7	155.8	Rep. Forw. A*	46,483	13.05	76.5%
			D* Lite	49,409	16.89	23.5%
40%	740.5	233.1	Rep. Forw. A*	69,382	19.53	67.5%
			D* Lite	51,531	19.93	32.5%
50%	1142.3	421.4	Rep. Forw. A*	126,509	35.75	54.8%
			D* Lite	61,209	30.10	45.2%
Total	738.7	191.3	Rep. Forw. A*	57,319	16.17	75.0%
			D* Lite	51,047	19.07	25.0%

Table 2: Results for Random Eight-Neighbor Grids

Case Study: Random Grids

To support our claims, we generate four-neighbor random grids of size 1024×1024 with 8, 16, 24, 32 and 40 percent blocked cells. For each percentage of blocked cells, we generate 1000 grids with randomly blocked cells and randomly chosen start and goal cells from all unblocked cells, different from many previous evaluations that used fixed start and goal cells (for example, in diagonally opposite corners of the grids). The h-values are the Manhattan distances. We ensure that Repeated Forward A* and D* Lite follow the same trajectory, different from many previous evaluations. We report the average length of the trajectory, the average number of searches performed, the average number of cells expanded and the runtime until the agent reaches the goal cell and the percentage of navigation problems for which the search algorithm runs faster than its competitor. Table 1 shows that the average runtime of Repeated Forward A* over all navigation problems is larger than the one of D* Lite. Yet, perhaps surprisingly, Repeated Forward A* runs faster than D* Lite on the majority of navigation problems with 9, 16, 24 and 32 percent blocked cells and on the majority of all navigation problems. Most research evaluates D* Lite only on four-neighbor grids, with only few exceptions such as (Koenig and Likhachev 2005). However, we now show that it makes sense to evaluate it on eight-neighbor grids as well. We generate eight-neighbor random grids of size 1024×1024 with 10, 20, 30, 40 and 50 percent blocked cells and proceed otherwise as before. Table 2 shows, perhaps surprisingly, that the average runtime of Repeated Forward A* over all navigation problems is now smaller than the one of D* Lite (which is likely due to the fact that navigation problems on eight-neighbor grids are easier than those on four-neighbor grids with the same percentage of blocked cells), which supports our claim that one needs use more diverse testbeds to evaluate incremental search algorithms.

Case Study: Game Maps, Office Maps and Mazes

Navigation problems on random grids get harder as the percentage of blocked cells increases. It is not immediately obvious how to characterize the hardness of navigation problems on other kinds of grids. We use the general solution to classify navigation problems as easy iff the runtime of Repeated Forward A* is small. Consequently, we group navigation problems into buckets according to the runtime of Repeated Forward A* so that each bucket contains the same number of navigation problems. We compare eight-neighbor game maps, office maps and mazes, obtained from Nathan Sturtevant's repository movingai.com. We omit all details due to space constraints but the results are similar to those on random grids. For game and office maps, Repeated Forward A* runs faster than D* Lite on the majority of easy navigation problems (that is, navigation problems in buckets where the runtime of Repeated Forward A* is small) and on the majority of all navigation problems. Only for mazes, Repeated Forward A* runs more slowly than D* Lite on the majority of all navigation problems. Overall, navigation problems on game maps seem to be easier than on office maps, and navigation problems on office maps seem to be easier than on mazes, which supports our claim that Repeated Forward A* is faster than D* Lite for many navigation problems, including those often encountered in practice.

Conclusions

Evaluations of incremental search algorithms are often performed on only one or two kinds of grids. However, we conclude that one needs to evaluate them in more diverse testbeds and with more diverse implementations (for example, of the priority queue) to improve our understanding of their properties. Evaluations of incremental search algorithms also often average over many randomly generated grids of a given kind. We conclude that this can give the wrong impression in cases where D* Lite appears to run faster than Repeated Forward A* because Repeated Forward A* runs faster than D* Lite on many easy navigation problems but much more slowly on a few hard navigation problems. We currently use these insights to develop new incremental search algorithms (for example, by classifying navigation problems and then either running Repeated Forward A* or D* Lite, as appropriate) and to improve existing incremental search algorithms (for example, by speeding up the first search of D* Lite). D* Lite performs its first search essentially on an empty map, which means that the first search can be simulated by initializing the values of all (or a subset of) cells appropriately, namely by setting

$rhs(s) = g(s) = h(s)$. In general, it would be helpful to have computational models that can predict the performance of incremental heuristic search methods as a function of the navigation problem, experimental setup and implementation details. Runtime proxies, such as the number of cells expanded, cannot be used to compare incremental heuristic search methods against Repeated Forward A* since they do not expand cells equally fast. Furthermore, the maps of navigation problems typically fit in memory. For such navigation problems, big O-analyses do not make sense and small implementation details can have a big effect on the runtime.

References

Koenig, S., and Likhachev, M. 2005. Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics and Automation* 21(3):354–363.