Artificial Intelligence 234 (2016) 49–77

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Truncated incremental search

Sandip Aine [a,*], Maxim Likhachev [b]

[a] Indraprastha Institute of Technology Delhi (IIIT Delhi), Delhi, India
[b] Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, United States

ARTICLE INFO

Article history:
Received 6 April 2014
Received in revised form 29 November 2015
Accepted 12 January 2016
Available online 18 January 2016

Keywords:
Planning
Replanning
Anytime planning
Heuristic search
Anytime search
Incremental search

ABSTRACT

Incremental heuristic search algorithms reuse their previous search efforts whenever these are available. As a result, they can often solve a sequence of similar planning problems faster than planning from scratch. State-of-the-art incremental heuristic searches (such as LPA*, D* and D* Lite) work by propagating cost changes to all the states in the search tree whose g values (the costs of computed paths from the start state) are no longer optimal. This work is based on the observation that while a complete propagation of cost changes is essential to ensure optimality, the propagations can be stopped earlier if we are looking close-to-optimal solutions instead of the optimal one. We develop a framework called Truncated Incremental Search that builds on this observation and uses a target suboptimality bound to efficiently restrict cost propagations. We present two truncation based algorithms, Truncated LPA* (TLPA*) and Truncated D* Lite (TD* Lite), for bounded suboptimal planning and navigation in dynamic graphs. We also develop an anytime replanning algorithm, Anytime Truncated D* (ATD*), that combines the inflated heuristic search with truncation, in an anytime manner. We discuss the theoretical properties of these algorithms proving their correctness and efficiency, and present experimental results on 2D and 3D (x, y, heading) path planning domains evaluating their performance. The empirical results show that the truncated incremental searches can provide significant improvement in runtime over existing incremental search algorithms, especially when searching for close-to-optimal solutions in large, dynamic graphs.

1. Introduction

Heuristic search is a fundamental problem solving technique in Artificial Intelligence (AI) [1], that models an optimization problem in terms of planning paths in state-space graphs. Search algorithms explore the state-space using domain specific information (heuristics), which guide the search towards solutions. Over the years, search algorithms have been used in many applications, including classical planning [2], learning [3,4], robotics [5–10], network design [11–13], VLSI [14–16], drug design [17,18], bio-informatics [19], with the list growing every day.

Planning for real-world applications involves two major problems, uncertainty and complexity. Real world is an inherently uncertain and dynamic place, which means accurate models are difficult to obtain and can quickly become out of date. Replanning becomes a necessity when such a change is perceived. The challenge here is to efficiently utilize the information gathered from earlier searches to facilitate the current planning. Incremental search algorithms are meant for such dynamic environments. They reuse previous search efforts to speed up the current search, and thus can often replan faster

* Corresponding author.
E-mail address: sandipaine@gmail.com (S. Aine).

http://dx.doi.org/10.1016/j.artint.2016.01.009
0004-3702/

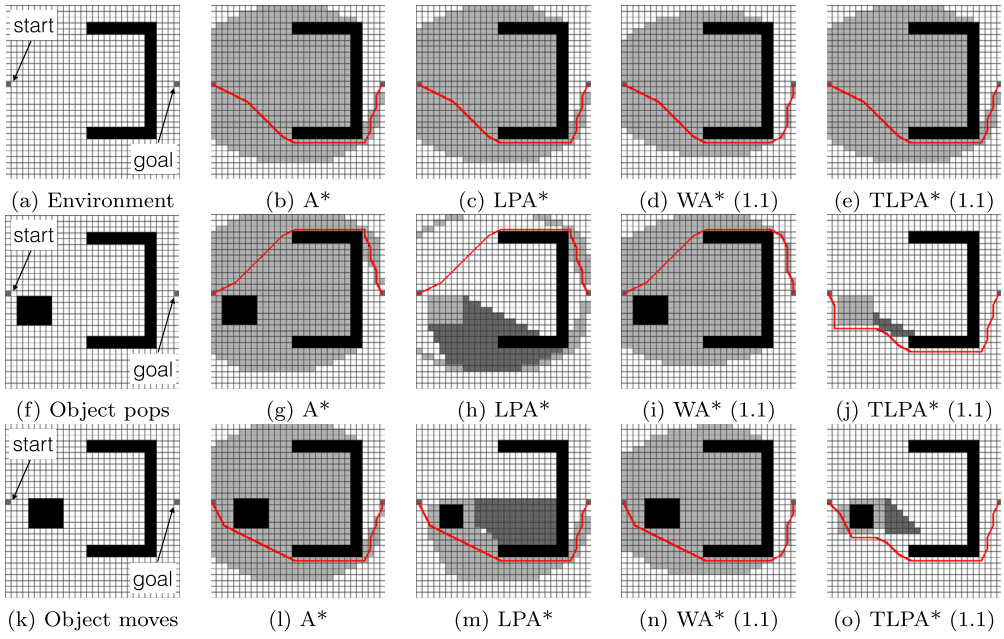

Fig. 1. A 30×30 grid path planning example showing the difference between A*, LPA*, WA* and TLPA* (WA* and TLPA* are run with suboptimality bound 1.1). In each case, we show the expanded states in grey and the path from start to goal in red (grey in print). Note that A* and WA* expand a state once only (per iteration), whereas LPA* and TLPA* can expand a state twice. To distinguish, we show the states expanded twice using dark grey and the states expanded once using light grey. The first search is identical for A*, LPA* and TLPA*, while WA* is a bit more efficient. After the first search, a new obstacle is introduced. A* and WA* recompute a *new* path from scratch. LPA* reuses the previous search tree and only rebuilds where the current tree is different from the previous one, and expands less states than both A* and WA* (1.1). However, it still expands a considerable number of states. TLPA* (1.1) quickly finds a way around the new obstacle and recomputes a bounded path with much fewer expansions. For the next iteration, the obstacle moves, blocking and unblocking some cells from the previous environment. Again, we observe that while A*, WA* (1.1) and LPA* expand a substantial number of states, TLPA* (1.1) terminates much faster, as it only propagates cost changes which are required to satisfy the cost bounds and truncates other expansions. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

than planning from scratch. For complex planning problems, it is often desirable to obtain a trade-off between the solution quality and the runtime, especially when optimal planning becomes infeasible due to resource (time/memory) constraints. Anytime search algorithms target such trade-offs, providing an initial (possibly highly suboptimal) solution quickly, and then iteratively improving the solution quality depending on the deliberation time. In this work, we focus on these two classes of search algorithms, incremental and anytime, and present *novel* algorithms for efficiently planning/replanning bounded suboptimal paths through large, dynamic graphs.

Lifelong planning A* (LPA*) [20] is an incremental version of A* (with a consistent heuristic function) which optimally solves a sequence of search problems in a dynamic environment. Its first iteration is the same as that of A*, but the subsequent searches are potentially faster as they reuse parts of the previous search tree that are identical to the current search tree. The rest of the tree is rebuilt by expanding states for which the g values (path cost from the start state) differ from the previous run (cost propagation). If large parts of the search tree remain unchanged over episodes, i.e., if the environments change only slightly and the changes are close to the goal, LPA* can converge significantly faster than A*. In Fig. 1, we present a simple path planning example in dynamic environment, showing how LPA* can outperform A*. The tree repair approach of LPA* has been used as a backbone for several incremental algorithms, such as D* Lite [21], Field D* [22], Anytime D* [23], that are widely used in practice, especially in robotics.

As LPA* recomputes the optimal solution every time the environment changes, it needs to propagate the cost changes for all the states (in the search tree) whose g values are no longer optimal. This means that even for a small change in the environment, large parts of the search tree may be regenerated, especially if the changes occur close to the root. We start with the observation that while the g values may change for a large number of states, the cost difference between the previous and the current values may not be significant for a majority of such states. LPA* treats all such states equally, as it uses a *binary* notion of change and does not account for the impact of a particular change. In contrast, if we *quantify* the possible impact of the costs changes (find out how much the path cost has changed), we may improve the replanning runtime by only re-expanding the states for which the change is significant and reusing the previous paths for the other states. For example, in Fig. 1, after the first search, an obstacle pops (Fig. 1f). This (potentially) changes the g values for 202 states, and LPA* re-expands all these states (some of them twice) before it recomputes an optimal solution. However, if we only consider states for which the (potential) change in g is more than 5% or 10%, the number of states affected reduces to 78 and 41, respectively. Similarly, when the obstacle moves (Fig. 1k), the total number of states (potentially) affected is