Dynamically Pruned A* for Re-planning in Navigation Meshes

Wouter van Toll and **Roland Geraerts**
Utrecht University, Department of Information and Computing Sciences
Princetonplein 5, 3584 CC Utrecht
E-mail: W.G.vanToll@uu.nl, R.J.Geraerts@uu.nl

Keywords: path planning, dynamic environments, navigation meshes, crowd simulation, intelligent agents.

1 Introduction

In modern simulations and games, crowds of virtual characters (or *agents*) must plan and traverse paths through complex environments in real-time. A *navigation mesh* represents the areas in which the agents can move. Its *dual graph* contains a vertex for each mesh region and an edge for each pair of adjacent regions. Agents use A* search [1] on this graph to find *global* routes, which they traverse while *locally* avoiding other agents.

In *dynamic environments*, obstacles are inserted, deleted, or moved during the simulation. We focus on insertions and deletions; moving obstacles can be represented by sequential deletions and insertions. These dynamic events can have large effects on the walkable space: imagine a bridge collapsing or a vehicle blocking an alley. Local collision avoidance may not be sufficient to guide agents to their goals. Instead, the navigation mesh should be updated and agents should *re-plan* their global paths. Existing re-planning algorithms require too much memory for crowds (because agents need to remember the previous search), or they are difficult to implement for graphs that structurally change [2, 3].

We present *Dynamically Pruned A** (DPA*), an extension of A* that efficiently re-plans an optimal global path when an obstacle has been inserted or deleted. DPA* *prunes* the search based only on the agent's old path and its relation to the event. The algorithm is tailored for applications with limited memory per agent. We show that DPA* outperforms standard A* in large graphs, especially when the dynamic event is visible to the agent. DPA* can be used for the real-time simulation of large crowds in dynamic environments.

The full version of this paper was published in IROS 2015 [4]. We refer the reader to the original publication for more details, including a supplementary video.

2 Difference to A* and Adaptive A*

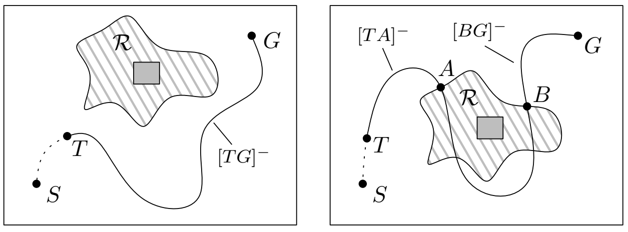
The A* algorithm [1] finds a path through a graph from a vertex S to a vertex G. Starting at S, this algorithm iteratively expands the vertex V for which $g(V) + h(V)$ is lowest. Here, $g(V)$ is the *cost* of the best discovered path from S to V so far, and $h(V)$ is a *heuristic* that estimates the cost of the optimal path from V to G. Costs and heuristics are often distance-based. The vertices to explore are stored in an *open list*, sorted by their values of $g + h$. If h is *admissible* (i.e. it never overestimates the optimal path cost), then A* computes an optimal path.

For re-planning after a dynamic event, *adaptive A** makes h more informed by using the previous query [2]. Under certain conditions, the algorithm can stop when a vertex of the old path is expanded. By contrast, DPA* prunes the standard A* search without changing any costs or heuristics, and without requiring any memory of the previous search besides the old path itself. It uses the 'distance' to the dynamic event to find out if vertices can be skipped. Furthermore, DPA* uses a distinction between four *scenarios* to optimize the search; each scenario has its own pruning rules.

3 Problem Description: Scenarios

A dynamic event affects the navigation mesh: regions can be added, removed, split, or merged. Let the *affected region* $\mathcal{R}$ be the the set of vertices and edges in the dual graph that have (dis)appeared. Intuitively, when an obstacle has been *inserted*, $\mathcal{R}$ has become becomes more costly to traverse; when an obstacle has been *removed*, $\mathcal{R}$ has become easier to traverse. This does not refer to individual edge costs, but to the overall cost of passing through $\mathcal{R}$. Note that $\mathcal{R}$ is already computed during the mesh update, and that it can have an arbitrary shape.

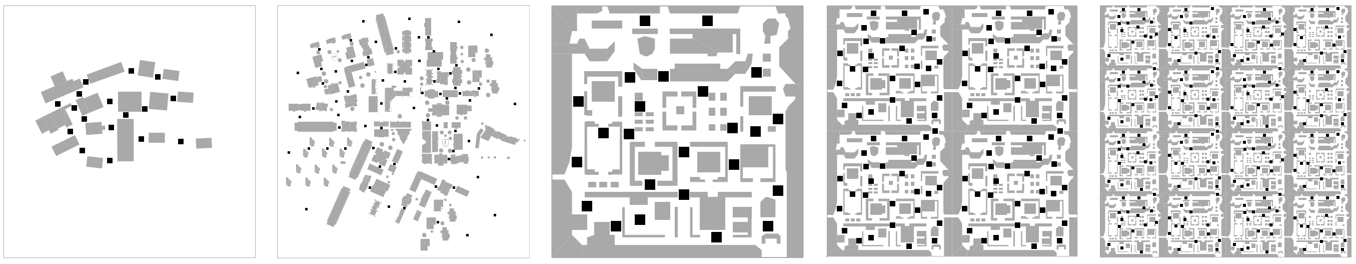
Initially, the agent has used A* to find an optimal path from S to G, which we call $[SG]^-$. Assume that an event occurs later in the simulation, and the agent decides to re-plan when it has traversed the path up to a vertex T, e.g. because it can now see the event. The agent should re-evaluate its path from T to G, i.e. $[TG]^-$. If $[TG]^-$ does *not* run through $\mathcal{R}$ (Figure 1a), the path is still valid, but it may not be optimal anymore. If $[TG]^-$ *does* run through $\mathcal{R}$ (Figure 1b), let A and B be the first and last vertex in $\mathcal{R}$ that occur in $[TG]^-$. We split the path into an *invalid* subsection $[AB]^-$ and two *valid* subsections $[TA]^-$ and $[BG]^-$, which may be empty.



(a) Unaffected path (b) Affected path

Figure 1: Re-planning scenarios after a dynamic event.

Furthermore, the event can be either an *insertion* or a *deletion* of an obstacle. This leads to four possible scenarios. The goal of DPA* is to compute a new optimal path $[TG]^+$ in a given scenario.



(a) *Military* (b) *City* (c) *Zelda* (d) *Zelda2x2* (e) *Zelda4x4*

Figure 2: The environments used in our experiments. Dynamic obstacles are shown in black.

4 Dynamically Pruned A*

A naive way to compute the new path $[TG]^+$ is to perform A* from scratch. DPA* adds *pruning rules* to this search by reusing information from the old path. We will now summarize these rules for each re-planning scenario.

Scenario 1. If an obstacle has been *inserted* and $[TG]^-$ does *not* run through $\mathcal{R}$, then it can be shown that $[TG]^-$ is still optimal. Hence, the agent does not need to re-plan, and DPA* will simply return $[TG]^-$.

Scenario 2. If an obstacle has been *inserted* and $[TG]^-$ *does* run through $\mathcal{R}$, then the agent may need to take a detour around $\mathcal{R}$, but the information in $[TG]^-$ can be reused. If we arrive at a vertex $C \in [BG]^-$, we know that $[CG]^-$ is still optimal. DPA* only adds the successor of C in $[CG]^-$ to the open list. It skips all other neighboring vertices of C because we already know that they cannot yield better paths. Similarly, for each vertex $D \in [TA]^-$, we know that $[TD]^-$ is still optimal. DPA* only adds D to the open list when coming from the predecessor of D in $[TD]^-$.

Scenario 3. If an obstacle has been *deleted* and $[TG]^-$ does *not* run through $\mathcal{R}$, then $[TG]^-$ may contain a detour around an area that has now become more attractive. If there is a better path than $[TG]^-$, then this path *must* pass through $\mathcal{R}$ at least once. Thus, when expanding any vertex V, we estimate the path cost from V to G *via* $\mathcal{R}$, using a second heuristic function that estimates the path cost to $\mathcal{R}$. If it turns out that the cost of $[TG]^-$ cannot be improved, then there are cases in which some or all neighbors of V can be skipped.

Scenario 4. If an obstacle has been *deleted* and $[TG]^-$ *does* run through $\mathcal{R}$, then $[TG]^-$ passes through affected mesh regions. The *geometric* path is still obstacle-free, but it can possibly be improved. As in Scenario 3, the new path should pass through $\mathcal{R}$ and it cannot have a higher cost than $[TG]^-$. The difference is that only the subpath $[BG]^-$ still exists and may be re-used.

5 Experiments and Results

We have implemented DPA* for the Explicit Corridor Map (ECM) navigation mesh [6], which supports real-time dynamic updates [7]. We use Euclidean distance-based costs and heuristics.

We compared the performance of DPA* and A* as follows. In each of the environments shown in Figure 2, we defined a number of dynamic obstacles (squares of 2×2 m). For each such obstacle O, we first planned paths between 500 random start and goal pairs. We then inserted O and re-planned all paths using both DPA* and A*. Finally, we deleted O and re-planned again. The results showed that A* is faster in small graphs, but that DPA* outperforms A* in larger graphs because they allow for more pruning. DPA* was particularly efficient in the 'deletion + path unaffected' scenario.

Next, we altered this experiment such that all start positions lie in the *visibility polygon* of the dynamic obstacle's center, to simulate that agents re-plan when they *see* an event. This greatly improved the results for some scenarios, in particular 'insertion + path affected'. Thus, DPA* is also useful for responding to a dynamic insertion when it is within the agent's visibility range.

Finally, we have integrated DPA* in an ECM-based crowd simulation framework [5]. The software can simulate over 10,000 agents in real-time using 6 CPU cores in parallel. Obstacles can be added and removed interactively; the crowd responds by re-planning. When using visibility as a trigger, re-planning actions are automatically divided over time, allowing real-time performance.

In conclusion, DPA* is an extension of A* for efficient re-planning in dynamic environments, suitable for real-time crowd simulation.

References

[1] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[2] C. Hernández, T. Uras, S. Koenig, J.A. Baier, X. Sun, and P. Meseguer. Reusing cost-minimal paths for goal-directed navigation in partially known terrains. *Autonomous Agents and Multi-Agent Systems*, pages 1–46, 2014.

[3] S. Koenig and M. Likhachev. D* Lite. In *Proc. AAAI Conf. of Artificial Intelligence*, pages 476–483, 2002.

[4] W. van Toll and R. Geraerts. Dynamically Pruned A* for re-planning in navigation meshes. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 2051–2057, 2015.

[5] W. van Toll, N. Jaklin, and R. Geraerts. Towards believable crowds: A generic multi-level framework for agent navigation. In *ASCI.OPEN*, 2015.

[6] W.G. van Toll, A.F. Cook IV, and R. Geraerts. Navigation meshes for realistic multi-layered environments. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 3526–3532, 2011.

[7] W.G. van Toll, A.F. Cook IV, and R. Geraerts. A navigation mesh for dynamic environments. *Computer Animation and Virtual Worlds*, 23(6):535–546, 2012.