Fast Subgoaling for Pathfinding via Real-Time Search

Carlos Hernández
Departamento de Ingeniería Informática
Universidad Católica de la Santísima Concepción
Concepción, Chile

Jorge A. Baier
Departamento de Ciencia de la Computación
Pontificia Universidad Católica de Chile
Santiago, Chile

Abstract

Real-time heuristic search is a standard approach to pathfinding when agents are required to make decisions in a bounded, very short period of time. An assumption usually made in the development and evaluation of real-time algorithms is that the environment is unknown. Nevertheless, in many interesting applications such as pathfinding for automnomous characters in video games, the environment *is* known in advance. Recent real-time search algorithms such as D LRTA* and kNN LRTA* exploit knowledge about the environment while pathfinding under real-time constraints. Key to those algorithms is the computation of subgoals in a preprocessing step. Subgoals are subsequently used in the online planning phase to obtain high-quality solutions. Preprocessing in those algorithms, however, requires significant computation. In this paper we propose a novel preprocessing algorithm that generates subgoals using a series of backward search episodes carried out from potential goals. The result of a single backward search episode is a tree of subgoals that we then use while planning online. We show the advantages of our approach over state-of-the-art algorithms by carrying out experiments on standard real-time search benchmarks.

Introduction

Intelligent agents moving in physical or virtual environments are required to constantly perform *pathfinding*, i.e. search for a sequence of moves that will lead them to reach a destination from their current location. This is an old AI problem for which many approaches exist (Dijkstra 1959; Hart, Nilsson, and Raphael 1968).

In some pathfinding settings the time allowed for computation per move is bounded by a constant independent of the size of the problem. An example is pathfinding in video games, in which companies impose limits in the order of milliseconds on the aggregated per-move time for all their virtual agents. Other settings, like pathfinding for unmanned vehicles in dynamic domains, have similar characteristics.

Real-time heuristic search algorithms, such as LRTA* (Korf 1990) or LSS-LRTA* (Koenig and Sun 2009), are applicable to real-time pathfinding. Nevertheless, most existing approaches produce solutions that look irrational to humans. Indeed, most real-time algorithms get "trapped" in regions of the search space in which the heuristic is imprecise or *depressed* (Ishida 1992). To exit these regions, they learn the correct heuristic values for states in the depression – a process that may require re-visiting the same states several times.

While it is unclear how to (completely) avoid such a problem when the environment is initially unknown to the agent, approaches capable of obtaining much better-quality solutions can be devised if the agent has complete knowledge about the environment *and* is allowed sufficient time for preprocessing. In applications like videogames, preprocessing is a viable option since designers know the environments in advance.

There exist a few approaches that utilize additional preprocessing time effectively. D LRTA* (Bulitko et al. 2007) precomputes optimal paths between pairs of abstracted states and then exploits this information while planning online. On the other hand, kNN LRTA* (Bulitko, Björnsson, and Lawrence 2010) pre-computes several optimal paths between pairs of states extracting a number of subgoals that are later exploited while planning on-line. The preprocessing carried out by kNN LRTA* is expensive since a high number of optimal paths may need to be computed. On the other hand D LRTA* may require a little less preprocessing time but is harder to describe and implement (Bulitko, Björnsson, and Lawrence 2010).

In this paper we propose a novel and simple approach to computing subgoals. There are two key elements that distinguish it from recent previous work. First, subgoals are computed by doing a backward search from a *single* state, generating a *tree* of subgoals. Second, subgoals correspond to states that are *exit points* from heuristic depressions. As a result, while the agent plans online, depressions are exited with a small number of moves. We show that, while our approach does not guarantee optimality, we usually find solutions only about 11% away from the optimal. We also show advantages in terms of preprocessing time over kNN LRTA* and total execution time over LRTA*.

The rest of the paper is organized as follows. We give an overview the basics of real-time search and pathfinding in the next section. We then describe our subgoaling algorithm, and how this information is exploited while planning online. We finish with an experimental evaluation followed by conclusions.

Preliminaries

Pathfinding is the problem of finding a path in a graph. Formally, the problem is described by a tuple (S, A, c, s_0, s_{goal}), where the search space is defined by the digraph (S, A). The finite set S represents the *states* and the arcs in A represent the available actions. A does not contain elements of form (x, x). On the other hand, the cost function $c : A \mapsto \mathbb{R}^+$ associates a positive cost to each of the available actions. Moreover, the search space is *undirected*, i.e. for all $(u, v) \in A$ then $(u, v) \in A$ and furthermore $c(u, v) = c(v, u)$. Finally, $s_0 \in S$ is the start state, and $s_{goal} \in S$ is the goal state. The successors of state u are defined by $Succ(u) = \{v \,|\, (u, v) \in A\}$. If v is a successor of u, we say that u and v are *neighbors*. A heuristic function $h : S \mapsto [0, \infty)$ associates to each state s an approximation $h(s)$ of the cost of a path from s to a goal state. The problem of pathfinding consists of finding a path in (S, A) starting in s_0 and ending in s_{goal}.

Algorithm 1: Pseudo code for LRTA*.

```
1 s ← s_0;
2 while s ≠ s_goal do
3     for each w ∈ Succ(s) do update c(s, w);
4     y ← argmin_{w∈Succ(s)} c(s, w) + h(w) ;
5     h(s) ← max{h(s), c(s, y) + h(y)} ;
6     Move the agent to y and do s ← y ;
```

In real-time pathfinding we additionally assume the an amount of computation between moves is bounded by a constant. Other potential applications are the control of unmanned vehicles acting in highly dynamic domains.

Learning Real-Time A* (LRTA*) (Korf 1990) (Algorithm 1) solves real-time search problems. It iteratively executes an observe-lookahead-update-act cycle until the goal is reached. The procedure has three local variables. Variable s stores the current position of the agent. Variable $c(s, s')$ contains the cost of moving state s to a successor s'. Variable h is such that $h(s)$ contains the heuristic value for the state. All three variables change over time.

In problems in which the domain is initially *unknown*, no obstacles are assumed and $c(s, s')$ is initialized to a value smaller than ∞ for all states $s, s' \in S$. As the agent acts both the cost and heuristic functions are updated (Lines 3 and 5, respectively). On the other hand, when the environment is *known* in advance c is initialized with the correct values, there is no need for updating c while executing.

There are a number of extensions of LRTA*, that improve its performance by doing more extensive lookahead or learning; e.g., LSS-LRTA* (Koenig and Sun 2009), LRTA*(k) (Hernandez and Meseguer 2005). All of these extensions however behave poorly in presence of heuristic depressions; i.e., regions in which the heuristic is imprecise (Ishida 1992). In pathfinding problems, this results in the agent re-visiting the same states multiple times; a behaviour that looks irrational and that has hampered the use of real-time heuristic search for applications like video games.

D LRTA* (Bulitko et al. 2007) and kNN LRTA* (Bulitko, Björnsson, and Lawrence 2010) are algorithms able to perform much better assuming the environment is known in advance that sufficient time is available for preprocessing before the agent acts in the world. In preprocessing time, D LRTA* uses an clique abstraction technique that merges sets of neighbour states into a single abstracted state. Optimal paths are then computed for every pair of abstracted states. In planning time, D LRTA* runs a modified version of LRTA* that uses the pre-computed information. On the other hand kNN LRTA* precomputes optimal paths between a number of pairs of initial-final states. These paths are "compressed" and stored in a database. In planning time, the algorithm searches for the best among the k closest pairs, and uses the pre-computed compressed paths as a sequence of subgoals to pursue.

Computing Subgoals

As mentioned in the previous section, the exploitation of pre-computed subgoals is an effective approach to real-time pathfinding. Recent existing approaches compute subgoals from actual, optimal *paths* between pairs of states. In the rest of the section, we propose a novel approach to computing subgoals which given a (single) goal state, finds a *tree* of subgoals. Subgoals are not extracted from a path; rather, they correspond to states that are regarded as key to escaping from the heuristic depressions in the problem.

Heuristic depressions appear naturally in pathfinding problems. Figure 1 left shows two shaded areas in a grid-world where the heuristic is *depressed* with respect to the goal G; i.e., the heuristic value of shaded cells is lower than the actual cost to reach the goal. To travel from cell $D2$ to $E6$, the agent has to escape the heuristic depression. With no subgoaling, a traditional real-time heuristic search algorithm performs several updates to the heuristic of the states in the depressed region before the region is exited. However, assume the search algorithm were informed that state $B3$ is a good subgoal to pursue. In such a situation, it turns out that the heuristic (manhattan distance) of $D2$ with respect to the subgoal $B3$ ($h(D2, B3)$) is exactly the *actual cost* of reaching $B3$ from $D2$. If we now consider a new search problem in which the goal is $B3$, then $D2$ is no longer in a heuristic depression with respect to this new goal. As such, LRTA* would head straight to the subgoal cell $B3$. Once $B3$ is reached, if the algorithm were informed that $E6$ is the next goal to pursue, we likewise have that $h(B3, E6)$ is the actual distance to the goal, and thus $E6$ will be reached with a little number of moves.

Our subgoaling algorithm (Algorithm 2) discovers states that correspond to *exit points* from a heuristic depression. A state s is an exit point from a heuristic depression with respect to a goal state g if for some successor s' of s, $h(s', g)$ is lower than the actual cost to reach g from s'. These exit states are found by performing a backward search from a given goal state. Our algorithm is a modification of the Dijkstra algorithm which associates to every state in the search space (1) a function $g(s)$, which corresponds to the optimal distance from s to the given goal s_{goal}, (2) a function $Sub(s)$, that returns the goal that will come after s during the real-time search, (3) a function $g_{sub}(s)$ which is the optimal

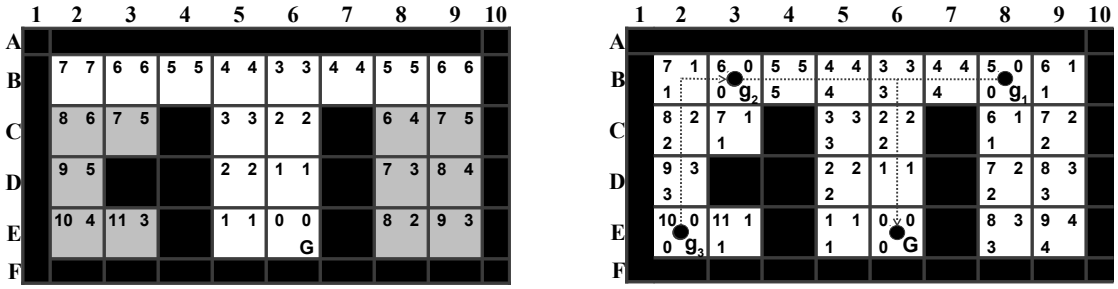

Figure 1: A 4-connected grid, with a goal marked by G. **Left:** The upper-left corner of each cell shows the actual distance to the goal. The upper-right corner shows the value of the heuristic for the cell (manhattan distance). Shaded cells show heuristic depressions. **Right:** The result of running Algorithm 2. g_1, g_2, and g_3 are the computed subgoals. The parent of g_1 and g_2 is G, and the parent of g_3 is g_2. The upper-left corner of each cell shows the actual distance to the goal (g-value), the lower-left shows the actual distance to the corresponding subgoal (g_{sub}-value), and the upper-right corner shows the value of the heuristic (manhattan distance) to the corresponding subgoal ($h(s, Sub(s))$-value) .

Algorithm 2: Subgoaling algorithm

Input: A search problem
Output: A tree of subgoals, stored in variable $Tree$

1 $Tree \leftarrow emptyTree$;
2 **for each** $s \in S$ **do** $g(s) \leftarrow \infty$;
3 $Sub(s_{goal}) \leftarrow s_{goal}$;
4 $g_{sub}(s_{goal}) \leftarrow 0$;
5 $g(s_{goal}) \leftarrow 0$;
6 $Open.push(s_{goal})$;
7 **while** $Open \neq \emptyset$ **do**
8 $s \leftarrow Open.pop()$;
9 **if** $s \neq s_{goal}$ **then**
10 $Sub(s) \leftarrow Sub(parent(s))$;
11 $g_{sub}(s) \leftarrow g_{sub}(parent(s)) + c(parent(s), s)$;
12 **for each** $s' \in Succ(s)$ **do**
13 **if** $g(s') > g(s) + c(s, s')$ **then**
14 $g(s') \leftarrow g(s) + c(s, s')$;
15 $g_{sub}(s') \leftarrow g_{sub}(s) + c(s, s')$;
16 $parent(s') \leftarrow s$;
17 **if** $g_{sub}(s') \neq h(s', Sub(s))$ **then**
18 $g_{sub}(s) \leftarrow 0$;
19 $Tree.push(Sub(s), s)$;
20 $Sub(s) \leftarrow s$;
21 **if** $s' \in Open$ **then** remove s' from $Open$;
22 $Open.push(s')$;

distance from s to $Sub(s)$, (4) a function $h(s', s)$, given as a parameter, which returns an estimate of the cost of an optimal path from s' to s, and (5) a function $parent(s)$ which, as in a standard Dijkstra algorithm, corresponds to the state from which s can be reached optimally from the goal state s_{goal}. In addition, $Open$ is Dijkstra's Algorithm priority queue, which in our variant is ordered by g. Finally, the variable $Tree$ stores the tree of subgoals. A new arc in the tree is added in Line 19 when a state has a child state s' whose optimal distance to the next subgoal is greater than the heuristic value from s' to such a subgoal.

Figure 1 right shows an example execution of the algorithm for the problem shown on the left. The first arc added to the tree is $(B8, E6)$, since the heuristic distance of a children of $B8$'s, $C8$, is such that $h(B8, C8) = 4$ is lower than 6, which is the cost of the optimal distance to $C8$ found by Dijkstra's algorithm when run goal.

Real-Time Pathfinding with Subgoaling

We have shown how to compute a tree of subgoals from a particular goal. We now turn our attention to how we exploit these trees for real-time pathfinding. In the rest of the section we describe a simple way to do it. Later we will show that our approach, although simple, achieves superior performance.

We start off assuming that we have computed a subgoal tree (using Algorithm 2) for every state s of a given pathfinding problem. (Although that is seemingly a very expensive initial step, we later show that this outperforms the state-of-the-art.) Then we use a modified version of LRTA* (shown in Algorithm 3) to plan find a path in real time. The algorithm initializes a variable $nextgoal(s)$ (Lines 1–4) which corresponds to the next goal to be pursued once s is reached, for each state s in the search space. Variable $goal$ contains the goal that is currently being pursued, and is initialized to the state that seems closer given the heuristic (Line 5). Then a slightly modified version of LRTA* ensues, in which subgoals are followed respecting the subgoal tree structure. We assume $h(s, s')$ is initialized to an estimate of the optimal path from s to s' (in 4-connected grids, we use the manhattan distance whereas in 8-connected grids we use the octile distance). In our implementation we do not set the value of $h(s, s')$ for all pairs $(s, s') \in S \times S$, but only for pairs that are required at execution time.

Algorithm 3: LRTA* with subgoaling

1 **for each** $s \in S$ **do** $nextgoal(s) \leftarrow null$;
2 $tree \leftarrow$ subgoal tree for s_{goal};
3 **for each** *state s in tree* **do**
4 $nextgoal(s) \leftarrow$ parent of s in $tree$;
5 $goal \leftarrow \text{argmin}_{\text{s in } tree} h(s_0, s)$;
6 $s \leftarrow s_0$;
7 **while** $s \neq s_{goal}$ **do**
8 **if** $nextgoal(s) \neq null$ **then** $goal \leftarrow nextgoal(s)$;
9 $y \leftarrow \text{argmin}_{w \in Succ(s)} c(s, w) + h(w, goal)$;
10 $h(s, goal) \leftarrow \max\{h(s, goal), c(s, y) + h(y, goal)\}$;
11 Move the agent to y and do $s \leftarrow y$;

	Cost	Time first search	Total time
A*	198	3.101	3.101
LRTA*	39,232	0.001	34.446
LRTA*+sub	220	0.024	0.251

Figure 2: LRTA* with subgoaling compared with A* and LRTA* in on-line search. Time first search and Total time are in milisecond.

Evaluation

We evaluated the performance of our approach over the four videogame maps used by Bulitko and Björnsson (2009) (2 from the game *Baldur's Gate*, and 2 from *World of Warcraft*), and 4 maps from the game *Dragon Age: Origins*. Maps are represented as 8-connected grids where horizontal and vertical movements have cost 1 and diagonal movements have cost $\sqrt{2}$. We use the octile distance as heuristic function. All experiments were run on a Linux PC with a Pentium QuadCore 2.33 GHz CPU and 8 GB RAM.

There are 17,789 unblocked cells on average in the 8 maps used. For each of them we compute a subgoal tree. Trees contain on average 198 states. The largest tree has 255 states whereas the smallest has size 0 and corresponds to a cell surrounded by obstacles. Subgoal trees for a single map are computed in 73.5 sec. on average, with a maximum computation time of 159.78 sec. and a minimum of 31.74 sec.

In the game maps of Bulitko and Björnsson (2009) we obtained an average computation time of 51.6 seconds. kNN LRTA* computes subgoals in a comparable time when its parameter is set to 1000 on the same maps and a very similar hardware. Such a parameter corresponds to the number of pre-computed optimal paths (between pairs of different states) that are precomputed. kNN LRTA*'s preprocessing time increases as the parameter is increased.

Figure 2 shows results obtained by A* (Hart, Nilsson, and Raphael 1968), LRTA* and LRTA* with subgoaling. We average our experimental results over 4000 test cases (500 test cases for each game map). For each test case, we choose the start and goal cells randomly. The average solution cost obtained by LRTA* with subgoaling is only 11% above the optimal. In the maps that were also tested by Bulitko and Björnsson (2009) our solutions were 13% above optimal. The suboptimality of kNN LRTA* (1000) over those maps is 49.91%, whereas the suboptimality of kNN LRTA* (10000) is 19.52%, requiring an average pre-processing time of 6.23 minutes (Bulitko and Björnsson 2009).

LRTA* and LRTA* with subgoaling need a very small time per planning episode. However, LRTA* with subgoaling outperforms LRTA* in 2 orders of magnitude in terms of solution quality. LRTA* with subgoaling requires more time only for the first planning episode (in which it needs to initialize variables and determine the first goal to pursue). The first planning episode takes 0.024 milliseconds on average. The subsequent planning episodes take 0.001 milliseconds on average. LRTA* with subgoaling is 12.4 times better than A* and 137.4 times better than LRTA* in terms of total time.

The quality of the solutions and preprocessing times obtained by our algorithm are comparable to those reported for D LRTA*(3) and D LRTA*(5) by Bulitko and Björnsson (2009). D LRTA*, however, requires more cell expansions during the on-line planning phase; indeed, D LRTA*(5) requires 19.98 more expansions on average than our subgoaling technique.

LRTA* with subgoaling obtains good quality solutions by expanding a single state per move. The state-of-the-art algorithms D LRTA* and kNN LRTA* need several expansions per planning episode yielding higher computation times.

Summary and Conclusions

We presented a simple and fast approach that computes and exploits subgoals for real-time pathfinding when the environment is known in advance. In the pre-computation phase, our subgoaling algorithm performs a backward search which returns a tree of subgoals. Each subgoal intuitively corresponds to an exit point from a heuristic depression. In the online planning phase, we modify LRTA* slightly to sequentially pursue goals in a branch of a the subgoal tree.

The performance exhibited by our approach is better both in terms of preprocessing time and solution quality than those obtained by Bulitko and Björnsson (2009) for kNN LRTA* over the same maps and running on similar hardware. Our approach seems comparable with D LRTA* in terms of preprocessing and quality. However, D LRTA* is more complex and requires more expansions per move. A recent version of kNN LRTA* that improves upon D LRTA* is presented by Bulitko, Björnsson, and Lawrence (2010).

We think our approach has plenty of potential, as there are many ways in which it could be extended. Computing smaller trees and grouping states that have similar subgoal trees are interesting avenues of future research. Our subgoal trees can also be straightforwardly incorporated into other real-time search algorithms such as LSS-LRTA* or LRTA$^*(k)$ (Hernandez and Meseguer 2005).

References

Bulitko, V., and Björnsson, Y. 2009. kNN LRTA*: Simple subgoaling for real-time search. In *AIIDE*, 2–7.

Bulitko, V.; Björnsson, Y.; Lustrek, M.; Schaeffer, J.; and Sigmundarson, S. 2007. Dynamic control in path-planning with real-time heuristic search. In *ICAPS*, 49–56.

Bulitko, V.; Björnsson, Y.; and Lawrence, R. 2010. Case-based subgoaling in real-time heuristic search for video game pathfinding. *Journal of Artificial Intelligence Research* 38:268–300.

Dijkstra, E. W. 1959. A note on two problems in connection with graphs. *Numerische Mathematik* 1:269–271.

Hart, P. E.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimal cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2).

Hernandez, C., and Meseguer, P. 2005. LRTA*(k). In *IJCAI*, 1238–1243.

Ishida, T. 1992. Moving target search with intelligence. In *AAAI*, 525–532.

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. 1990. Real-time heuristic search. *Artificial Intelligence* 42(2-3):189–211.