Time-Bounded Adaptive A*

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

ABSTRACT

In this paper, we investigate real-time path planning in static terrain, as needed in video games. We introduce the game time model, where time is partitioned into uniform time intervals, an agent can execute one movement during each time interval, and search and movements are done in parallel. The objective is to move the agent from its start location to its goal location in as few time intervals as possible. For known terrain, we show experimentally that Time-Bounded A* (TBA*), an existing real-time search algorithm for undirected terrain, needs fewer time intervals than two state-of-the-art real-time search algorithms and about the same number of time intervals as A*. TBA*, however, cannot be used when the terrain is not known initially. For initially partially or completely unknown terrain, we thus propose a new search algorithm. Our Time-Bounded Adaptive A* (TBAA*) extends TBA* to on-line path planning with the freespace assumption by combining it with Adaptive A*. We prove that TBAA* either moves the agent from its start location to its goal location or detects that this is impossible - an important property since many existing real-time search algorithms are not able to detect efficiently that no path exists. Furthermore, TBAA* can eventually move the agent on a cost-minimal path from its start location to its goal location if it resets the agent into its start location whenever it reaches its goal location. We then show experimentally in initially partially or completely unknown terrain that TBAA* needs fewer time intervals than several state-of-the-art complete and real-time search algorithms and about the same number of time intervals as the best compared complete search algorithm, even though it has the advantage over complete search algorithms that the agent starts to move right away.

Categories and Subject Descriptors

I.2.8 [**Problem Solving, Control Methods, and Search**]: Graph and Tree Search Strategies, Heuristic Methods

General Terms

Algorithms, Experimentation

Appears in: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Conitzer, Winikoff, Padgham, and van der Hoek (eds.), June, 4–8, 2012, Valencia, Spain.

Keywords

A*, Incremental Heuristic Search, Learning Real-Time A*, LRTA*, Path Planning with the Freespace Assumption, Real-Time Heuristic Search, TBA*, Time-Bounded A*, Video Games

1. INTRODUCTION

Game characters in video games can execute one movement per game cycle. We therefore introduce the following time model, called game time model. Time is partitioned into uniform time intervals, an agent can execute one movement during each time interval, and search and movements are done in parallel. The objective is to move the agent from its start location to its goal location in as few time intervals as possible. Complete search algorithms, such as A* [4], search first and only then move the agent along the resulting path. They typically need several time intervals to find a complete path from the start location of the agent to its goal location, resulting in a long delay before the agent starts to move and a long time until it reaches its goal location. Real-time search algorithms avoid both issues by executing A* searches and movements in parallel.

Most complete and real-time search algorithms can operate in both known and initially partially or completely unknown terrain. They typically use on-line path planning with the freespace assumption in initially unknown terrain by taking all obstacles into account that the agent has observed so far but assuming that unknown terrain is free of obstacles [11]. For example, Repeated A* is identical to A* except that Repeated A* starts a new A* search from the current location of the agent to its goal location whenever the agent observes obstacles on its current path to its goal location. Incremental search algorithms, such as Adaptive A* [9] and D* Lite [7], behave in the same way but speed up the A* searches by using their experience with prior A* searches to speed up future ones.

In this paper, we study Time-Bounded A* (TBA*) [1]. TBA* is an existing real-time search algorithm for undirected terrain that performs an A* search from the start location of the agent to its goal location. At the end of each time interval, the agent executes a movement towards a location in the OPEN list with the smallest f-value. We show experimentally that TBA* moves the agent in known terrain from its start location to its goal location in fewer time intervals than two state-of-the-art real-time search algorithms and in about the same number of time intervals as A*. However, it cannot be used when the terrain is not known initially. We thus extend it to on-line path planning with the freespace assumption in initially partially or completely unknown (but static) terrain in two steps. In the first step, we extend TBA* to Restarting Time-Bounded A* (RTBA*). RTBA* is identical to TBA* except that, whenever the agent observes obstacles on its current path to a location in the OPEN list with the smallest f-value, RTBA* starts a new A* search from the current location of the agent to its goal location. At the end of each time interval, the agent continues to execute a movement towards a location in the OPEN list with the smallest f-value. In the second step, we extend RTBA* to Time-Bounded Adaptive A* (TBAA*). TBAA* is identical to RTBA* except that TBAA* updates the h-values of the expanded states after each A* search to make them more informed and thus focus future A* searches better, just like Adaptive A* and RTAA* [10]. We prove that RTBA* and TBAA* correctly either move the agent from its start location to its goal location or detect that this is impossible. Many other real-time search algorithms cannot detect efficiently that this is impossible. Furthermore, RTBA* and TBAA* can eventually move the agent on a cost-minimal path from its start location to its goal location if they reset the agent into its start location whenever it reaches its goal location. We show experimentally that TBAA* moves the agent in initially partially or completely unknown terrain from its start location to its goal location in fewer time intervals than several state-of-the-art complete search algorithms (including Adaptive A*) and real-time search algorithms (including RTAA*) and in about the same number of time intervals as the best compared complete search algorithm (namely, D* Lite), even though it has the advantage over complete search algorithms that the agent starts to move right away.

2. GAME TIME MODEL

We now introduce and justify the game time model. Under the game time model, time is partitioned into uniform time intervals, an agent can execute one movement during each time interval, and search and movements are done in parallel. The objective is to move the agent from its start location to its goal location in as few time intervals as possible. The game time model is motivated by video games. Video games often partition time into game cycles, each of which is only a couple of milliseconds long [2]. Each game character executes one movement at the end of each game cycle, which gives the players the illusion of fluid movement. During each game cycle, video games perform all computations necessary to progress the game, which includes the calculation of the next movement of the agent, before redrawing the visuals.

The game time model addresses the fact that the standard way of evaluating search algorithms, namely using their CPU times or path costs, is problematic in real-time situations. A*, for example, needs the smallest CPU time of any search algorithm to find cost-minimal paths (up to tie breaking). Yet, an agent that uses A* might need more time to move from its start location to its goal location than an agent that uses some other search algorithm because A* searches first and only then moves the agent along the resulting path. A* typically needs several time intervals to find a cost-minimal path from the start location of the agent to its goal location, resulting in a long delay before the agent starts to move (which makes the agent unresponsive) and a long time until it reaches its goal location (which makes the agent inefficient) since there is no parallelism of search and movement – the agent does not move until the path is found and does not search afterwards. The advantage of the game time model is that the time needed by the agent to move from its start location to its goal location is proportional to the number of time intervals needed. A search algorithm that computes a path while moving the agent might be able to move the agent to its goal location (along a suboptimal path) in fewer time intervals than A* and thus is more desirable than A*, even though both its CPU time and resulting path cost could be larger than those of A*.

3. NOTATION

A search problem is a tuple $(S, A, c, s_{start}, s_{goal})$, where (S, A) is a finite digraph. S is the set of states, and A is the set of edges. $Succ(s) = \{t \,|\, (s,t) \in A\}$ is the set of successors (or, synonymously, neighbors) of state $s \in S$. $c : A \mapsto \mathbb{R}^+$ is the cost function that associates a cost with each edge. $s_{start} \in S$ is the start state, and $s_{goal} \in S$ is the goal state. We assume that the digraph is undirected, meaning that there is an edge from vertex s to vertex t iff there is one from t to s, and both edges have the same cost. For simplicity, we call the edges undirected. We also assume that the reader is familiar with A* and knows that A*, when searching from s_{root} to s_{goal}, maintains an OPEN list (a priority queue) and, for every state $s \in S$, a g-value $g(s)$ that is the cost of the cost-minimal path from s_{root} to s found so far, an h-value $h(s)$ that is an approximation of the cost of a cost-minimal path from s to s_{goal}, an f-value $f(s) = g(s) + h(s)$, and a parent pointer $parent(s)$ that points to the parent of s in the A* search tree, whose root is s_{root}. $H(s)$ is the user-provided h-value of s. The h-values are consistent iff $h(s_{goal}) = 0$ and $h(s) \leq c(s,t) + h(t)$ for all states $s \in S$ and $t \in Succ(s)$.

4. TBA*

Given a search problem, the objective is to move an agent from s_{start} to s_{goal} in as few time intervals as possible. Real-time search algorithms execute one or more A* searches to determine one movement for the agent per time interval, that is, after a bounded amount of computation. Time-Bounded A* (TBA*) [1] is an existing real-time search algorithm for search problems with known cost functions. We use known four-neighbor grids with blocked and unblocked cells as examples, where the states are the cells and edges connect every unblocked cell to its unblocked neighboring cells to the north, east, south and west with cost one. TBA* performs an A* search from $s_{root} = s_{start}$ to s_{goal}. At the end of each time interval, the agent executes the movement from its current state $s_{current}$ towards a state s_{best} in the OPEN list with the smallest f-value (where s_{goal} should be chosen if possible), as follows: Consider the branch *path* of the A* search tree from its root $s_{root} = s_{start}$ to s_{best}, which is a cost-minimal path from s_{root} to s_{best}. If $s_{current}$ is on *path* (that is, $s_{current}$ is encountered when following the parent pointers from s_{best} to s_{root}), then the agent executes the movement from $s_{current}$ towards s_{best} along *path* (that is, moves to the state after $s_{current}$ on *path*). Otherwise, it executes the movement from $s_{current}$ towards s_{root} along the branch of the A* search tree from s_{root} to $s_{current}$ (that is, follows the parent pointer of $s_{current}$), which is possible since the edges are undirected. The time that it takes to expand a state depends on the size of the OPEN list, and

Algorithm 1 TBA*

```
1: procedure InitializeState($s$)
2: if $search(s) = 0$ then
3:     $h(s) := H(s)$;
4:     $g(s) := \infty$;
5: $search(s) := searchnumber$;

6: procedure InitializeSearch()
7: $searchnumber := searchnumber + 1$;
8: $s_{root} := s_{current}$;
9: InitializeState($s_{root}$);
10: $g(s_{root}) := 0$;
11: $OPEN := \emptyset$;
12: Insert $s_{root}$ into $OPEN$;
13: InitializeState($s_{goal}$);
14: $goalFoundFlag := 0$;

15: function Search()
16: $expansions := 0$;
17: while $OPEN \neq \emptyset$ AND $expansions < k$ AND
18:         $g(s_{goal}) + h(s_{goal}) > \min_{t \in OPEN}(g(t) + h(t))$ do
19:     $s := \arg\min_{t \in OPEN}(g(t) + h(t))$;
20:     Remove $s$ from $OPEN$;
21:     for all $t \in Succ(s)$ do
22:         InitializeState($t$)
23:         if $g(t) > g(s) + c(s,t)$ then
24:             $g(t) := g(s) + c(s,t)$;
25:             $parent(t) := s$;
26:             Insert $t$ into $OPEN$;
27:     $expansions := expansions + 1$;
28: if $OPEN = \emptyset$ then
29:     return false;
30: $s_{best} := \arg\min_{t \in OPEN}(g(t) + h(t))$;
31: if $s_{best} = s_{goal}$ then
32:     $goalFoundFlag := 1$;
33: $path :=$ path from $s_{root}$ to $s_{best}$;
34: return true;

35: function MoveToGoal()
36: $s_{current} := s_{start}$;
37: InitializeSearch();
38: while $s_{current} \neq s_{goal}$ do
39:     print("a new time interval starts now");
40:     if $goalFoundFlag = 0$ then
41:         if Search() = false then
42:             return false;
43:     if $s_{current}$ is on $path$ then
44:         $s_{current} :=$ state after $s_{current}$ on $path$;
45:     else
46:         $s_{current} := parent(s_{current})$;
47:     Execute movement to $s_{current}$;
48: return true;

49: procedure Main()
50: $searchnumber := 0$;
51: for all $s \in S$ do
52:     $search(s) := 0$;
53: if MoveToGoal() = true then
54:     print("the agent is now at the goal state");
55: else
56:     print("the agent cannot reach the goal state");
```

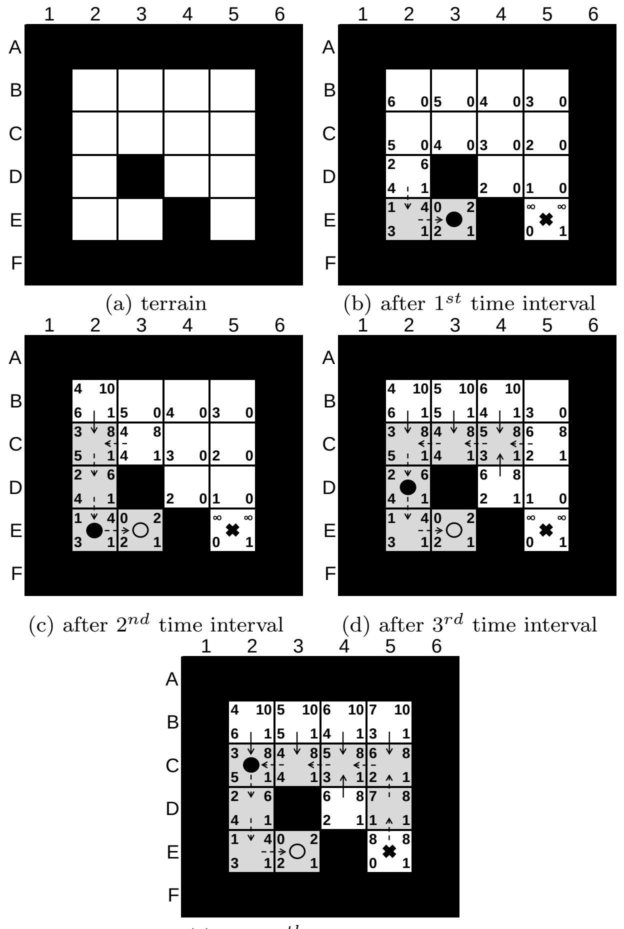

(a) terrain
(b) after 1st time interval
(c) after 2nd time interval
(d) after 3rd time interval
(e) after 4th time interval

Figure 1: Operation of TBA*

the time that it takes to determine the next movement for the agent depends on the length of *path*. In the following, we make the simplifying assumption that the time per state expansion is constant and the time per movement determination is zero, resulting in a constant number k of state expansions during each time interval. The original version of TBA* uses an additional parameter and techniques that make it a true (amortized) real-time search algorithm. We could do the same for RTBA* and TBAA* but will not do so for simplicity.

Algorithm 1 shows TBA*. Procedure InitializeState initializes the h-value of a given state (to the user-provided h-value) and the g-value of the state (to infinity) when they are needed for the first time, to avoid initializing those values that are not needed later. *searchnumber* is the number of the current A* search, and *search*(s) is the number of the A* search during which the g-value of state s was initialized last. It is zero if the g-value has not been initialized yet. Procedure Search performs an A* search from s_{root} to s_{goal} by expanding states until the OPEN list becomes empty (in which case the agent cannot reach s_{goal}), until a cost-minimal path from s_{root} to s_{goal} has been found (that is, $s_{best} = s_{goal}$) or until k states have been expanded. *expansions* is the number of expanded states in the current time interval, and *goalFoundFlag* is true iff the A* search has found a cost-minimal path from s_{root} to s_{goal}. Procedure Search sets *goalFoundFlag* and *path* before it terminates. Procedure MoveToGoal sets the agent into s_{start} and calls procedure InitializeSearch to initialize the A* search from $s_{root} = s_{current}$ to s_{goal}. It then repeatedly calls procedure Search (if *goalFoundFlag* is false) and then executes a movement on Lines 43-47, until the agent reaches s_{goal} or the A* search indicates that the agent cannot reach s_{goal}. Procedure Main calls procedure MoveToGoal for a given search problem and reports the results.

Figure 1 shows the operation of TBA* with $k = 2$ in the known four-neighbor grid with blocked (black) and unblocked (white) cells shown in Figure 1(a). $s_{start} = E3$, and $s_{goal} = E5$. The user-provided h-values are the Manhattan distances, that is, the costs of cost-minimal paths from the cells to the goal cell on a four-neighbor grid without blocked cells. The OPEN list contains the non-expanded cells that have a neighboring expanded (grey) cell. The bottom-left and bottom-right corners of each cell show its h- and search-values, respectively. The top-left and top-right corners of each cell show its g- and f-values, respectively, iff its g-value has been initialized. The arrow shows its parent pointer iff its parent pointer has been initialized. Dotted arrows indicate the cost-minimal path from $s_{root} = s_{start}$ to s_{best}. Figure 1(b) shows the situation at the end of the first time interval (but before the first movement of the agent). The OPEN list contains only $D2$ with f-value 6. $s_{current} = E3$ is on the branch *path* of the A* search tree from $s_{root} = E3$ to $s_{best} = D2$. The agent thus executes the movement along this branch to $E2$. Figure 1(c) shows the situation at the end of the second time interval (but before the second movement of the agent). The OPEN list contains $C3$ with f-value 8 and $B2$ with f-value 10. $s_{current} = E2$ is on the branch *path* of the A* search tree from $s_{root} = E3$ to $s_{best} = C3$. The agent thus executes the movement along this branch to $D2$, and so on. Figure 1(e) shows that the A* search finds a cost-minimal path from $s_{root} = E3$ to $s_{goal} = E5$ at the end of the fourth time interval (but before the fourth movement of the agent). The agent then executes movements along this branch without any further cell expansions until it reaches $s_{goal} = E5$.

Algorithm 2 RTBA*

```
1: procedure InitializeState($s$)
2: if $search(s) = 0$ then
3:     $h(s) := H(s)$;
4:     $g(s) := \infty$;
5: else if $search(s) \neq searchnumber$ then
6:     $g(s) := \infty$;
7: $search(s) := searchnumber$;

8: procedure StartNewSearch?()
9: if edge costs on $path$ have increased
10:            since the last call to InitializeSearch then
11:     InitializeSearch();
12:     $path := empty$;

13: function MoveToGoal()
14: $s_{current} := s_{start}$;
15: InitializeSearch();
16: Observe edge cost increases (if any);
17: while $s_{current} \neq s_{goal}$ do
18:     print("a new time interval starts now");
19:     if $goalFoundFlag = 0$ then
20:         if Search() = false then
21:             return false;
22:         StartNewSearch?();
23:     if $path \neq empty$ then
24:         if $s_{current}$ is on $path$ then
25:             $s_{current}$ := state after $s_{current}$ on $path$;
26:         else
27:             $s_{current} := parent(s_{current})$;
28:         Execute movement to $s_{current}$;
29:         Observe edge cost increases (if any);
30:         StartNewSearch?();
31: return true;
```

5. RTBA*

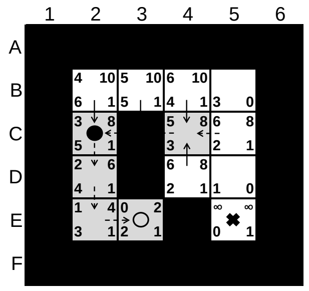
TBA* does not apply to search problems with unknown cost functions. We require that the agent observes the cost of an edge before it traverses it. In cases where a lower bound on the cost function is known, search algorithms can use the lower bound when an edge cost is unknown. This way, the observation of an actual edge cost can change the assumed edge cost only once, namely from the lower bound to the actual edge cost, which cannot decrease the assumed edge cost - properties that we exploit in the following.[1] We use initially unknown four-neighbor grids with blocked and unblocked cells as examples. The agent knows its start and goal cells. It does not know the blockage status of the cells initially but always observes the blockage status of its four neighboring cells to the north, east, south and west. It can use path planning with the freespace assumption by assuming that all cells are unblocked, that is, edges connect every cell to its neighboring cells to the north, east, south and west with cost one. If, after executing a movement, it observes that a neighboring cell is blocked, it increases the costs of all incoming and outgoing edges of that cell to infinity, which is equivalent to removing the edges from the set of edges A. The agent could run TBA* again whenever edge costs have increased during the current A* search but would then often abandon the current A* search unnecessarily. Instead, it runs TBA* again only when edge costs on the path from $s_{current}$ to s_{best} have increased during the current A* search. We refer to this search algorithm as Restarting Time-Bounded A* (RTBA*).

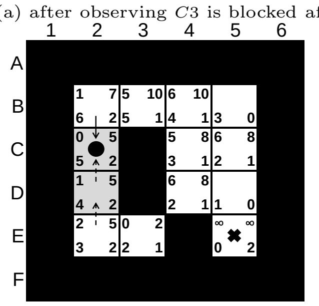
Algorithm 2 shows RTBA* without repeating the InitializeSearch, Search and Main procedures from Algorithm 1. Procedure InitializeState now initializes the h-value of a given state (to the user-provided h-value) when it is needed for the first time. It initializes the g-value of the state (to infinity) when it is needed by the current A* search for the first time. Procedure StartNewSearch? checks whether edge costs on the path from $s_{current}$ to s_{best} have increased during the current A* search and, if so, starts a new A* search from $s_{root} = s_{current}$ to s_{goal}.[2] Procedure MoveToGoal now calls procedure StartNewSearch? on Line 30 after the agent executes a movement since the agent might observe additional edge costs which result in increases of edge costs on the path from $s_{current}$ to s_{best}. Procedure MoveToGoal also calls procedure StartNewSearch? on Line 22 after the call to procedure Search since it might result in a new path from $s_{current}$ to s_{best} that contains edge costs increases that have been ignored earlier.

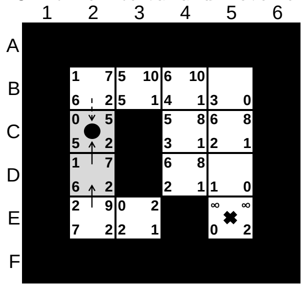
Figure 2(a,b,d) shows the operation of RTBA* with $k = 2$ in an initially unknown four-neighbor grid with blocked and unblocked cells. $s_{start} = E3$, and $s_{goal} = E5$. The user-provided h-values are the Manhattan distances. The first three time intervals are as in Figure 1 except that the agent observes blocked cell $C3$ after the third time interval and movement of the agent, which increases edge costs on the

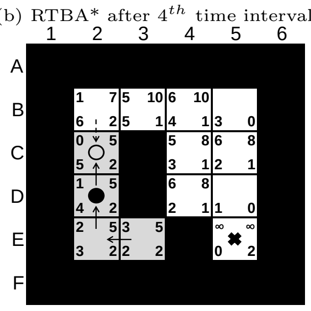
[1]We just refer to edge costs and the context determines whether we mean the actual or assumed edge costs.

[2]Actually, procedure StartNewSearch? checks the branch *path* of the A* search tree from s_{root} to s_{best} rather than the path from $s_{current}$ to s_{best}. The agent traverses only edges that belong to branches of the A* search tree. Consider the branches of the A* search tree from s_{root} to $s_{current}$ and from s_{root} to s_{best}. There cannot be any edge cost increases on the branch from s_{root} to $s_{current}$ since the edges are undirected and the agent has traversed the edges of the branch earlier already. The path from $s_{current}$ to s_{best} is made up of both branches without their common prefix. Thus, edge costs increase on the path iff they increase on the branch from s_{root} to s_{best}.


(a) after observing $C3$ is blocked after 3rd time interval and movement


(b) RTBA* after 4th time interval


(c) TBAA* after 4th time interval


(d) RTBA* after 5th time interval

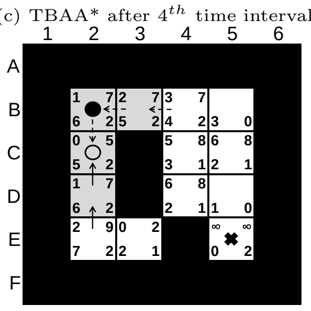

(e) TBAA* after 5th time interval

Figure 2: Operation of RTBA* and TBAA*

path from $s_{current} = C2$ to $s_{goal} = E5$. Thus, RTBA* starts a new A* search from $s_{root} = s_{current} = C2$ to $s_{goal} = E6$. Figure 2(b) shows the situation at the end of the fourth time interval (but before the fourth movement of the agent). The OPEN list contains $E2$ with f-value 5 and $B2$ with f-value 7. $s_{current} = C2$ is on the branch *path* of the A* search tree from $s_{root} = C2$ to $s_{best} = E2$. The agent thus executes the movement along this branch to $D2$. Figure 2(d) shows the situation at the end of the fifth time interval (but before the fifth movement of the agent). The OPEN list contains only $B2$ with f-value 7. $s_{current} = D2$ is not on the branch *path* of the A* search tree from $s_{root} = C2$ to $s_{best} = B2$. The agent thus executes the movement that follows the parent pointer of $s_{current} = D2$ to $C2$.

6. TBAA*

Each time RTBA* starts a new A* search, all information from the previous A* search is lost. However, real-time search algorithms often update the h-values to make them more informed. We therefore propose Time-Bounded Adaptive A* (TBAA*). TBAA* works like RTBA* but updates h-values of cells in the way (Lazy) Adaptive A* [9] and (Lazy) RTAA* do. Each time TBAA* starts a new A* search, it updates the h-values of all generated states s by the previous A* search by assigning $h(s) := f(s_{best}) - g(s)$ if this increases $h(s)$, which maintains the consistency of the

Algorithm 3 TBAA*

```
1:  procedure InitializeState($s$)
2:  if $search(s) = 0$ then
3:      $h(s) := H(s)$;
4:      $g(s) := \infty$;
5:  else if $search(s) \neq searchnumber$ then
6:      if $h(s) < pathcost(search(s)) - g(s)$ then
7:          $h(s) := pathcost(search(s)) - g(s)$;
8:      $g(s) := \infty$;
9:  $search(s) := searchnumber$;

10: procedure StartNewSearch?()
11: if edge costs on $path$ have increased
12:         since the last call to InitializeSearch then
13:     $pathcost(searchnumber) := \min_{s \in OPEN}(g(s) + h(s))$;
14:     InitializeSearch();
15:     $path := empty$;
```

h-values. TBAA* performs this h-value update only once the h-value of a state is needed by the current A* search for the first time, to avoid computing those h-values that are not needed later.

Algorithm 3 shows TBAA* without repeating the InitializeSearch, Search and Main procedures from Algorithm 1 and the MoveToGoal procedure from Algorithm 2. Procedure StartNewSearch? now remembers $f(s_{best})$ on Line 13 by assigning $pathcost(searchnumber) := f(s_{best})$, and Procedure InitializeState now updates the h-value of the given state s on Line 7 when it is needed by the current A* search for the first time by assigning $h(s) := pathcost(search(s)) - g(s)$ if this increases $h(s)$.

Figure 2(a,c,e) show the operation of TBAA* with $k = 2$ in the same scenario as described for RTBA*. TBAA* sets $pathcost(1) = f(C5) = 8$ and starts a new A* search from $s_{root} = s_{current} = C2$ to $s_{goal} = E6$. Figure 2(c) shows the situation at the end of the fourth time interval (but before the fourth movement of the agent). The OPEN list contains $E2$ with f-value 9 (four higher than for RTBA*) and $B2$ with f-value 7. $s_{current} = C2$ is on the branch *path* of the A* search tree from $s_{root} = C2$ to $s_{best} = B2$. The agent thus executes the movement along this branch to $B2$ and needs fewer movements to reach $s_{goal} = E5$ than RTBA*, demonstrating the advantage of updating the h-values.

7. ANALYSIS

We now prove that RTBA* and TBAA* correctly either move the agent from s_{start} to s_{goal} or detect that this is impossible. Furthermore, RTBA* and TBAA* can eventually move the agent on a cost-minimal path from s_{start} to s_{goal} if they reset the agent into s_{start} whenever it reaches s_{goal}. We make use of the following assumptions and properties: The user-provided h-values are consistent. The h-value updates of Adaptive A* (and thus also the ones of TBAA*) maintain the consistency of the h-values [9]. An A* search with consistent h-values correctly finds a path or detects that none exists [12], and the found path is cost-minimal [12].

THEOREM 1. *RTBA* and TBAA* correctly either move the agent from s_{start} to s_{goal} or detect that this is impossible.*

PROOF. Each time the search algorithm starts a new A* search, it has observed at least one additional edge cost. Thus, the number of times it starts a new A* search is bounded. Consider the last A* search, and let s_{root} be the root of the A* search tree. This A* search correctly finds a

Algorithm 4 Repeated Runs of RTBA* and TBAA*

```
1: procedure Main()
2: $searchnumber := 0$;
3: for all $s \in S$ do
4:     $search(s) := 0$;
5: repeat
6:     if MoveToGoal() = false then
7:         print("the agent cannot reach the goal state");
8: until $s_{root} = s_{start}$;
9: print("cost-minimal path from start state to goal state: ");
10: print($path$);
```

path from s_{root} to s_{goal} or detects that none exists. There exists a path from s_{root} to s_{goal} iff there exists a path from s_{start} to s_{goal} since the agent moves along undirected edges. If the A* search finds a path from s_{root} to s_{goal}, then the agent repeatedly follows the parent pointers of its current states until it is on the path and then traverses the path to s_{goal}. The agent is able to follow the parent pointers since edges are undirected and the agent has traversed the edges earlier already during the A* search. The agent reaches the path this way since the agent moves in the A* search tree and thus reaches s_{root} by repeatedly following the parent pointers if it does not reach the path from s_{root} to s_{goal} earlier already. The agent is able to traverse the path to s_{goal} since the A* search correctly found a path from s_{root} to s_{goal} and the search algorithm would have started a new A* search if edge costs on the path had increased during the A* search. $\square$

The following theorem states that RTBA* and TBAA* eventually find a cost-minimal path from s_{start} to s_{goal} if they reset the agent into s_{start} whenever it reaches s_{goal}. Algorithm 4 defines this process precisely.

Theorem 2. *Algorithm 4 prints a cost-minimal path for RTBA* and TBAA* if a path from s_{start} to s_{goal} exists.*

Proof. Each time after the search algorithm resets the agent into s_{start} and starts a new A* search, it moves the agent from s_{start} to s_{goal} according to Theorem 1. Each time the search algorithm starts a new A* search while the agent moves to s_{goal}, it has observed at least one additional edge cost. Thus, the number of times it starts a new A* search while the agent moves to s_{goal} is bounded, and it eventually moves the agent from s_{start} to s_{goal} without starting a new A* search while the agent moves to s_{goal} (implying that $s_{root} = s_{start}$). The last A* search finds a cost-minimal path from s_{start} to s_{goal}, and there cannot be any edge cost increases on the path since edges are undirected and the agent has traversed the edges earlier already during the A* search. The agent thus traverses a cost-minimal path if it moves along this path. $\square$

8. EXPERIMENTAL EVALUATION

We compare RTBA* and TBAA* against several state-of-the-art complete and real-time search algorithms, which we implemented in a similar way, for example, using a standard binary heap for the OPEN list and breaking ties among states with the same f-value in favor of larger g-values. All complete search algorithms first find a complete path for the agent from the start state to the goal state (over several time intervals) and then move the agent along it (again over several time intervals). All real-time search algorithms perform only state expansions during the first time interval. In each subsequent time interval, they compute a path for the agent, execute one movement for the agent, and then perform repeatedly state expansions until the end of the time interval is reached, implying that all time intervals have approximately the same length but not necessarily the same number of state expansions. All real-time search algorithms perform an h-value update only once the h-value of a state is needed by the current A* search for the first time. The scaling behavior of the search algorithms is less important than the hardware and implementation details since the search problems are small. It is difficult to compare the search algorithms with proxies, such as the number of state expansions, instead of the runtime itself since they perform different basic operations. We thus do not know of any better method for evaluating them than to implement them as best as possible and let other researchers validate the results with their own and thus potentially slightly different implementations.

We use known and initially partially or completely unknown eight-neighbor grids with blocked and unblocked cells in the experiments. Four-neighbor grids make for good illustrations since they result in integer-valued g-, h-, and f-values but we believe that eight-neighbor grids are more realistic for video games [3]. Also, some incremental search algorithms speed up A* searches less on eight-neighbor grids. The user-provided h-values are the octile distances, that is, the costs of cost-minimal paths from the cells to the goal cell on an eight-neighbor grid without blocked cells. The agent knows the dimensions of the grid and its start and goal cells. It can always move from its current unblocked cell to one of the eight unblocked neighboring cells with cost one for horizontal or vertical movements and cost $\sqrt{2}$ for diagonal movements. We ran the search algorithms with time intervals whose lengths ranged from 0.3 to 1.5 milliseconds and report results for the average number of time intervals and number of movements until the agent reaches the goal cell for the first time.

We use six game maps and generated 300 search problems with randomly chosen start and goal cells for each game map, for a total of 1,800 search problems.[3] In known grids, it knows the blockage status of all cells initially. In initially partially or completely unknown grids, it does not know the blockage status of some or all, respectively, cells initially but always observes the blockage status of its eight neighboring cells. Partially unknown grids are motivated by video games where the layout of the terrain is known to the agent but other players can build structures which are initially unknown to the agent [13]. We randomly blocked 15 percent of the unblocked cells in the game maps. The agent knows the blockage status of all blocked cells in the game maps but does not know the blockage status of the additional blocked cells.

8.1 Known Terrain

In known terrain, we compare TBA* against the complete search algorithm (forward) A* and the real-time search algorithms RTAA* and daRTAA*. Table 1 shows the following

[3]We use three Starcraft maps, namely Enigma of size 768×768, Inferno of size 768×768, and WheelofWar of size 768×768. We use three Dragon Age: Origins maps, namely orz103d of size 456×463, orz702d of size 939×718, and orz703d of size 502×652.

Table 1: Known Terrain

	RTAA*		daRTAA*		TBA*		A*	
Length of Time Intervals (ms)	# Time Intervals	# Movements	# Time Intervals	# Movements	# Time Intervals	# Movements	# Time Intervals	# Movements
0.3	2,193	2,192	1,729	1,728	**568**	567	584	545
0.6	1,541	1,540	1,303	1,302	**556**	555	565	545
0.9	1,362	1,361	1,151	1,150	**552**	551	558	545
1.2	1,196	1,195	1,057	1,056	**550**	549	555	545
1.5	1,087	1,086	970	969	**549**	548	553	545

Table 2: Initially Completely Unknown Terrain

	RTAA*		daRTAA*		RTBA*		TBAA*		Repeated A*		Adaptive A*		D* Lite	
Length of Time Intervals (ms)	# Time Intervals	# Movements	# Time Intervals	# Movements	# Time Intervals	# Movements	# Time Intervals	# Movements	# Time Intervals	# Movements	# Time Intervals	# Movements	# Time Intervals	# Movements
0.3	3,245	3,244	2,879	2,878	4,613	4,604	2,290	2,286	7,155	2,004	3,230	2,010	**2,203**	2,027
0.6	2,598	2,597	2,472	2,471	3,368	3,360	2,147	2,144	4,487	2,004	2,572	2,010	**2,090**	2,027
0.9	2,451	2,450	2,418	2,417	2,918	2,910	2,101	2,099	3,611	2,004	2,361	2,010	**2,062**	2,027
1.2	2,310	2,309	2,305	2,304	2,695	2,688	2,086	2,083	3,178	2,004	2,260	2,010	**2,051**	2,027
1.5	2,281	2,280	2,272	2,271	2,560	2,553	2,070	2,068	2,920	2,004	2,202	2,010	**2,045**	2,027

Table 3: Initially Partially Unknown Terrain

	RTAA*		daRTAA*		RTBA*		TBAA*		Repeated A*		Adaptive A*		D* Lite	
Length of Time Intervals (ms)	# Time Intervals	# Movements	# Time Intervals	# Movements	# Time Intervals	# Movements	# Time Intervals	# Movements	# Time Intervals	# Movements	# Time Intervals	# Movements	# Time Intervals	# Movements
0.3	2,694	2,693	2,460	2,459	2,734	2,730	**1,505**	1,504	6,324	1,409	2,430	1,399	1,659	1,418
0.6	2,039	2,038	1,863	1,862	2,037	2,034	**1,442**	1,441	3,812	1,409	1,875	1,399	1,532	1,418
0.9	1,840	1,839	1,779	1,778	1,860	1,857	**1,431**	1,430	2,979	1,409	1,695	1,399	1,490	1,418
1.2	1,707	1,706	1,643	1,642	1,726	1,724	**1,421**	1,420	2,564	1,409	1,608	1,399	1,470	1,418
1.5	1,620	1,619	1,642	1,641	1,668	1,666	**1,415**	1,414	2,316	1,409	1,556	1,399	1,458	1,418

relationships for known terrain.

- The average number of movements until the agent reaches the goal cell does not depend on the length of the time intervals for A* since complete search algorithms search first and only then move the agent along the resulting path.
- The average number of movements until the agent reaches the goal cell decrease as the length of the time intervals increases for TBA*, RTAA*, and daRTAA*.
- The average number of time intervals until the agent reaches the goal cell decreases as the length of the time intervals increases for all search algorithms.

All real-time search algorithms move the agent in known terrain from its start cell to its goal cell in about the same or more time intervals than A*. However, TBA* moves the agent in known terrain from its start cell to its goal cell in fewer time intervals than the two real-time search algorithms RTAA* and daRTAA* and in about the same number of time intervals as A*. TBA* has the advantage over A* that the agent moves right away for TBA* (namely, during time interval 2 in our implementation) rather than only in time intervals 8 to 40 on average for A*.

8.2 Initially Unknown Terrain

In initially unknown terrain, we compare RTBA* and TBAA* against the complete search algorithms (forward) Repeated A*, Adaptive A*, and D* Lite and the real-time search algorithms RTAA* and daRTAA*. Our implementation of Repeated A* starts a new A* search only when edge costs on the path from $s_{current}$ to s_{goal} have increased, rather than whenever edge costs have increased, different from [8], which explains the difference in experimental results compared to [8]. Tables 2 and 3 show the following relationships for initially completely and partially unknown terrain, respectively.

- The average number of movements until the agent reaches the goal cell does not depend on the length of the time intervals for Repeated A*, Adaptive A* and D* Lite. Furthermore, they are similar for all search algorithms, since they execute the same movements (modulo tie breaking).
- The average number of movements until the agent reaches the goal cell decreases as the length of the time intervals increases for RTBA*, TBAA*, RTAA*, and daRTAA*.
- The average number of time intervals until the agent reaches the goal cell decreases as the length of the time intervals increases for all search algorithms.

All real-time search algorithms move the agent in initially partially or completely unknown terrain from its start cell to its goal cell in fewer time intervals than Repeated A*. The game time model is thus able to explain the importance of real-time search in this case. TBAA* moves the agent in initially partially or completely unknown terrain from its start cell to its goal cell in fewer time intervals than the two complete search algorithms Repeated A* and Adaptive A*, and the two real-time search algorithms RTAA* and daRTAA* and in about the same number of time intervals as the best compared complete search algorithm D* Lite. TBAA* seems to have a slight advantage over D* Lite in initially partially unknown terrain and vice versa in initially completely unknown terrain (although this difference might not be statistically significant). The reason appears to be that the h-value surface does not have local minima on grids without blocked cells, which allows D* Lite to speed up A* searches significantly during the first searches in initially completely unknown terrain but not in initially partially unknown terrain. For example, D* Lite is often able to find the very first path in initially completely unknown terrain in one time interval.

9. SUMMARY

In this paper, we introduced the game time model, where time is partitioned into uniform time intervals, an agent can execute one action during each time interval, and search and action execution run in parallel. We then extended Time-Bounded A* (TBA*) to on-line path planning with the freespace assumption in initially partially or completely unknown (but static) terrain, resulting in Time-Bounded Adaptive A* (TBAA*). Similar to TBA*, TBAA* performs an A* search from the start location of the agent to its goal location. At the end of each time interval, the agent executes a movement towards a location in the OPEN list with the smallest f-value. TBAA* starts a new A* search whenever the agent observes obstacles on its path to this location. Similar to Adaptive A*, TBAA* updates the h-values of the expanded states after each A* search to make them more informed and thus focus future A* searches better. We proved that TBAA* correctly either moves the agent from its start location to its goal location or detects that this is impossible. Many other real-time search algorithms cannot detect efficiently that no path exists. Furthermore, TBAA* can eventually move the agent on a cost-minimal path from its start location to its goal location if it resets the agent into its start location whenever it reaches its goal location. We then showed experimentally that TBAA* moves the agent in initially partially or completely unknown terrain from its start location to its goal location in fewer time intervals than several complete and real-time search algorithms and in about the same number of time intervals as the best compared complete search algorithm, even though it has the advantage over complete search algorithms that the agent starts to move right away. In future work, we intend to let TBAA* use techniques that speed up daRTAA* over RTAA* [6, 5] to avoid local minima in the h-value surface. Since TBAA* uses less than 30 percent of the available time on average, we also intend to let it use more sophisticated h-value update techniques to make its h-values even more informed.

Acknowledgments

We thank Nathan Sturtevant for making his game maps publicly available at www.movingai.com. We also thank the anonymous reviewers, whose comments resulted in substantial revisions of our original manuscript. This material is based upon research supported by NSF (while Sven Koenig was serving at NSF). It is also based upon research supported by ARL/ARO under contract/grant number W911NF-08-1-0468 and ONR in form of a MURI under contract/grant number N00014-09-1-1031. The research was performed while Carlos Hernández visited the University of Southern California. Jorge Baier was partly funded by Fondecyt grant number 11110321. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies or the U.S. government.

10. REFERENCES

[1] Yngvi Björnsson, Vadim Bulitko, and Nathan Sturtevant. TBA*: Time-bounded A*. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, pages 431–436, 2009.

[2] Vadim Bulitko, Yngvi Björnsson, Nathan Sturtevant, and Ramon Lawrence. *Real-time Heuristic Search for Pathfinding in Video Games*. available from: https://sites.google.com/a/ualberta.ca/ircl/projects/rths.

[3] Vadim Bulitko and Greg Lee. Learning in real time search: a unifying framework. *Journal of Artificial Intelligence Research*, 25:119–157, 2006.

[4] Peter Hart, Nils Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimal cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[5] Carlos Hernández and Jorge Baier. Real-Time Adaptive A* with depression avoidance. In *Proceedings of the 7th International Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, pages 146–151, 2011.

[6] Carlos Hernández and Jorge Baier. Real-time heuristic search with depression avoidance. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 578–583, 2011.

[7] Sven Koenig and Maxim Likhachev. D* Lite. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI)*, pages 476–483, 2002.

[8] Sven Koenig and Maxim Likhachev. Fast replanning for navigation in unknown terrain. *Transactions on Robotics*, 21(3):354–363, 2005.

[9] Sven Koenig and Maxim Likhachev. A new principle for incremental heuristic search: Theoretical results. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 402–405, 2006.

[10] Sven Koenig and Maxim Likhachev. Real-Time Adaptive A*. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 281–288, 2006.

[11] Sven Koenig, Craig Tovey, and Yury Smirnov. Performance bounds for planning in unknown terrain. *Artificial Intelligence*, 147(1-2):253–279, 2003.

[12] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, Boston (Massachusetts), 1984.

[13] Peter Yap, Neil Burch, Robert Holte, and Jonathan Schaeffer. Any-angle path planning for computer games. In *Proceedings of the 7th International Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2011.