Real-Time Pathfinding in Unknown Terrain via Reconnection with an Ideal Tree

Nicolás Rivera[1], León Illanes[2], and Jorge A. Baier[2]

[1] Department of Informatics
King's College London
London, UK
[2] Departmento de Ciencia de la Computación
Pontificia Universidad Católica de Chile
Santiago, Chile

Abstract. In real-time pathfinding in unknown terrain an agent is required to solve a pathfinding problem by alternating a time-bounded deliberation phase with an action execution phase. Real-time heuristic search algorithms are designed for general search applications with time constraints but unfortunately in pathfinding they are known to produce poor-quality solutions. In this paper we propose p-FRIT$_{\text{RT}}$, a real-time version of FRIT, a recently proposed algorithm able to produce very good-quality solutions in pathfinding under strict, but not fully real-time constraints. The idea underlying p-FRIT$_{\text{RT}}$ draws inspiration from bug algorithms, a family of pathfinding algorithms. Yet, as we show, p-FRIT$_{\text{RT}}$ is able to outperform a well-known bug algorithm and is able to solve graph search problems that are more general than pathfinding. p-FRIT$_{\text{RT}}$ also outperforms significantly—generating solutions six times shorter when time constraints are tight—a previously proposed real-time version of FRIT and the real-time heuristic search algorithm that is considered to have state-of-the-art performance in real-time pathfinding.

1 Introduction

Pathfinding in an a priori unknown terrain is an important problem, with applications ranging from videogames to robotics. In many of those applications, time is a very limited resource. One example is videogames, where characters are required to move fluently but game developers are not willing to design pathfinding algorithms which would spend more than one millisecond per game cycle, for all simultaneously moving characters [1]. Under such time constraints it is often not possible to compute complete solutions offline before all agents have to be moved.

Real-Time heuristic search algorithms, as conceived by Korf [2], solve general search problems—including pathfinding—and are designed to produce movements given a constant time bound for planning. However, it is known that in pathfinding applications they generate poor-quality solutions, because they rely on a heuristic function that needs to be updated for several states in the search space [3]. When time constraints are tight, the agent is required to re-visit many states before completing search, generating *scrubbing*-like behavior [1].

Recently, Rivera *et al.* [4] proposed FRIT (Follow and Reconnect with the Ideal Tree), a general search algorithm that performs very well at pathfinding in unknown terrain, requiring very little time resources. Unfortunately FRIT cannot produce an action given a constant time bound. In a follow-up paper [5], however, they proposed FRIT$_{\mathrm{RT}}$, a fully real-time version of FRIT. In their evaluation they showed that in pathfinding, the resulting algorithm outperformed a state-of-the-art real-time heuristic search algorithm. However, the quality of the solutions produced could be up to one order of magnitude *worse* than those obtained by its predecessor, unless significant time was given per move.

In this paper we propose a real-time version of FRIT$_{\mathrm{RT}}$, p-FRIT$_{\mathrm{RT}}$, that unlike FRIT$_{\mathrm{RT}}$, is able to produce solutions comparable to those of FRIT in pathfinding. The key idea underlying this algorithm draws inspiration from a family of algorithms known as *bug algorithms* [6], which are pathfinding-specific algorithms that imitate the behavior of bugs by "going around" obstacles as they move. Our version of FRIT$_{\mathrm{RT}}$ is designed to restrict to borders too; specifically, it restricts reconnection search—a key phase of FRIT—to only expand nodes that are in the border of obstacles.

We prove that our algorithm always finds a solution in a general class of problems which subsumes pathfinding in 8-connected grids. In an experimental evaluation on game map benchmarks, we show that p-FRIT$_{\mathrm{RT}}$ improves upon FRIT$_{\mathrm{RT}}$ significantly and that it is able to outperform a bug algorithm.

The outline of the paper is as follows. In the next section we present background knowledge, including FRIT and FRIT$_{\mathrm{RT}}$. Then we present our algorithm, p-FRIT$_{\mathrm{RT}}$, and prove it always terminates. Next, we present our experimental evaluation. The paper finishes with an analysis of related work and conclusions.

2 Preliminaries

A search problem is a tuple $P = (G, c, s_{start}, g)$, where $G = (S, A)$ is a directed graph that represents the search space. The set S represents the *states* and the arcs in A represent all available actions. We define the successors of s as $Succ(s) = \{s' \mid (s, s') \in A\}$. State $s_{start} \in S$ is the *initial state* and state $g \in S$ is the *goal state*. A standard assumption in real-time search is that S is finite, that A does not contain elements of the form (s, s), that G is such that g is reachable from all states reachable from s_{start}. In addition, we have a non-negative cost function $c : A \to \mathbb{R}$ which associates a cost with each of the available actions.

Given a subset T of S we define the frontier of T as $\partial T = \{s \in S \setminus T : \exists t \in T \text{ such that } (t, s) \in A\}$. Intuitively, ∂T corresponds to the states that surround the region of states T, i.e., it contains the neighbors of states in T that are not in T. A subset T of S is said connected if for every pair of vertices s, t in T there exists a path that only uses states in T that connects s and t and vice versa.

The objective in *offline* search is to compute a path from s_{start} to g. Heuristic search algorithms solve search problems using a heuristic function to guide search. A heuristic for a search graph G is a non-negative function $h : S \to \mathbb{R}$ such that $h(s)$ estimates the distance between state s and state g, $d_G(s, g)$. We say that h is *admissible* iff $h(s) \leq d_G(s, g)$, for every $s \in S$. Furthermore, h is *consistent* if for every $(s, t) \in A$ it holds that $h(s) \leq c(s,t) + h(t)$, and furthermore that $h(g) = 0$. It is simple to prove that consistency implies admissibility.

Below we assume familiarity with the heuristic-search algorithm A* [7], which ranks states in its search frontier with a function $f(s) = g(s) + h(s)$, where $g(s)$ is the cost of a path from s_{start} to s, and h is the heuristic. We may refer to $h(s)$ and $g(s)$ as, respectively the h-value and g-value of s.

A pathfinding problem in an $n \times m$ grid can represented as a search problem by representing each cell as a state. Specifically, the set of states is defined by $\{0, 1, \ldots, n, n{+}1\} \times \{0, 1, \ldots, m, m{+}1\}$, where cells of the form $(0, x)$, $(n{+}1, x)$, $(x, 0)$, or $(x, m{+}1)$, for some x, are *border cells* and are regarded as *obstacles*. In this paper we focus on 8-connected grids, which are such that each of the cells have eight possible neighbors (two horizontal, two vertical, and four diagonal neighbors). Formally, we denote by $dist(s,t)$ the euclidean distance between s and t, and we say that $(s,t) \in A$ if and only if $dist(s,t) \leq \sqrt{2}$. For any pair such that $(s,t) \in A$, we define $c(s,t) = dist(s,t)$ if neither s nor t are obstacles. If s is an obstacle, then $c(s,t) = \infty$ and $c(t,s) = \infty$, for every suitable t. Finally, we assume $Obs \subseteq S$ contains all obstacle cells.

Our pathfinding algorithm will restrict search to states that are adjacent to an obstacle. To that end, we define an order for successor states. Specifically, $Succ(s,i)$ denotes the i-th successor of s, such that two successive successors are adjacent to each other. Formally, $Succ(s,i) = s + \delta_i$, for $i \in \{1, \ldots, 8\}$, where δ_i is the i-th element of the following vector,

$$\delta = \big((1,0)\ (1,1)\ (0,1)\ (-1,1)\ (-1,0)\ (-1,-1)\ (0,-1)\ (1,-1)\big),$$

defines the 8 successors of a cell in clockwise order.[3] In addition, for simplicity, we define $Succ(s,0) \overset{\text{def}}{=} Succ(s,8)$ and $Succ(s,9) \overset{\text{def}}{=} Succ(s,1)$. Finally, if $Succ(s,i) \notin S$, we say $Succ(s,i)$ is undefined, which may only happen for an s that lies in the border of the grid, which we defined above as obstacles.

An admissible and consistent heuristic often used in 8-connected grid navigation is the octile distance, which is an analogue of the Manhattan distance in 4-connected grids.

2.1 Real-Time Heuristic Search

Real-time search algorithms move an agent from the initial state to the goal. They are given a bounded amount of time for deliberating, independent of the size of the problem, after which the agent is expected to move. After such a move, more time is given for deliberation and the loop repeats.

Real-time *heuristic* search algorithms are akin to heuristic search algorithms and thus guide search with a heuristic. An example is Real-Time Adaptive A* (RTAA*) [8], which in pathfinding problems in unknown terrain can be described using the following algorithm. (1) Set s to s_{start}. (2) Observe the environment, updating Obs and c. (3) Carry out a bounded A* search that will not extract the goal from $Open$ or expand more

[3] We chose this particular δ for convenience but any vector that allows defining the successors in a clockwise order will work, as well as any vector that reflects a counter-clockwise order.

than k states. (4) Set $next$ to $\arg\min_{t \in Open} g(t)+h(t)$. (5) Set $h(s) \leftarrow f(next)-g(s)$, for every s in $Closed$. (6) Follow the path identified by A* towards $next$; update Obs while moving; stop if an obstacle is blocking the path or if $next$ is reached. Step 5 is called the *learning step*, in which it makes h more informed. It can be shown that Step 5 preserves heuristic consistency [8], which implies RTAA* always terminates.

RTAA* and many other generalizations of LRTA* (e.g., [9–11]) perform poorly in the presence of *heuristic depressions* [3, 12]. A heuristic depression is an area of the search space in which the heuristic function returns values that are much lower than the actual cost required to reach a goal state.

2.2 FRIT and FRIT$_{\mathbf{RT}}$

Follow and Reconnect with the Ideal Tree (FRIT) [4], is a family of algorithms for solving search problems in unknown search graphs. FRIT is a framework for general search problems but since the focus of this paper is pathfinding we describe it using pathfinding notions, differing slightly from [4].

In an unknown terrain the search graph G is not known to the agent at the outset; instead, the agent knows the dimensions of the grid and the start and goal cells, and furthermore believes that the search graph is given by G_M, which intuitively defines an idealistic pathfinding problem in which the set of obstacles is a subset of the obstacles in G. Specifically, G_M is a search graph that materializes the free-space assumption [13], in which unobserved cells whose blockage status is unknown are assumed obstacle-free.

In its initialization, FRIT defines a so-called *ideal tree*, $\mathcal{T}$, which contains each non-border cell and is defined via *parent pointers*, which point from children nodes to parent nodes. Specifically, each cell s in $\mathcal{T}$, except for the goal cell, has a parent pointer $\mathrm{p}(s)$ such that $\mathrm{p}(s)$ is a state not in Obs. In addition, given any cell s in $\mathcal{T}$, there exists a natural n such that $\mathrm{p}^n(s) = g$. In other words, by "following" the parent pointers from any cell s in $\mathcal{T}$, one eventually reaches the goal in the idealistic world defined by G_M.

For pathfinding in unknown terrain we can generate an initial ideal tree using a consistent heuristic h, setting $\mathrm{p}(s) = \arg\min_{t \in Succ(t)} c(s,t) + h(t)$, for every non-border cell which is not the goal cell. Moreover, the parent pointers do not need to be set explicitly for every cell but rather computed when needed. Fig. 1(a) illustrates the ideal tree defined for an 8-connected grid.

To solve a pathfinding problem, FRIT follows the parent pointers of the ideal tree, observing the environment as it moves, until it discovers that this is not possible because a newly discovered obstacle is blocking the path the pointers define. When this happens it invokes a search procedure whose objective is to find a path connecting the current state to a state that is connected to the ideal tree. We call this search *reconnection search*.

Reconnection search can be carried out with any graph search algorithm. The goal condition is the only aspect that is rather different: instead of looking for a specific goal state, reconnection search needs to verify whether or not a state is connected to the search graph, which can be done verifying that there exists a path to the goal state via p pointers. After reconnection, the p pointers define a forest, and the current state is in the ideal tree $\mathcal{T}$.

FRIT(BFS) [4] is the simplest instance of FRIT. It uses breadth-first search (BFS) for reconnection and was shown to have very good performance in pathfinding, with very little time requirements. Fig. 1(b)-(d) shows a few iterations of FRIT(BFS) on a pathfinding problem.

Unfortunately, FRIT(BFS), unlike standard real-time search algorithms, is not able to produce an action given a time bound. This is because BFS takes time bounded by the *size of the search graph* to return a solution. To address this pitfall, Rivera *et al.* [5] proposed to use a real-time search algorithm for reconnection search. In particular, when RTAA* is used for reconnection, one produces FRIT$_{\text{RT}}$ (RTAA*).

Algorithm 1: FRIT$_{\text{RT}}$ (RTAA*): FRIT with RTAA* reconnection.

```
   Input: A search graph $G_M$, an initial state $s_{start}$, and a goal state $g$
1  Initialization: Let $\mathcal{T}$ be an ideal tree for $G_M$.
2  Set $s$ to $s_{start}$.
3  Set $c$ to 0 and the color of each state in $G_M$ to 0.
4  while $s \neq g$ do
5      Observe the environment around $s$.
6      for each newly discovered inaccesible state $o$ do
7          Prune from $\mathcal{T}$ any arcs that lead to $o$, and add $o$ to $Obs$.
8      if $\mathrm{p}(s) = null$ then
9          $c \leftarrow c + 1$
10         Call RTAA*, using InTree$[c]$ as termination condition.
11     Movement: Move the agent from $s$ to $\mathrm{p}(s)$ and set $s$ to the new position of the agent.
```

Algorithm 1 shows the pseudo-code for FRIT$_{\text{RT}}$ (RTAA*). Line 10 invokes a slightly modified version of RTAA* (Algorithm 3), which differs from its original in (1) that it sets the parent pointers following the path traversed by the agent, and (2) that it uses function InTree$[c]$ to determine whether or not a state is connected to the ideal tree. A simplified version of the pseudo-code of InTree$[c]$ is shown in Algorithm 2. It is simplified because it still may need a number of iterations bounded by the size of the graph. We can modify InTree$[c]$ to make it real-time, making it return *false* if there is no more time for a new iteration in the main loop. If time constraints are extremely tight InTree$[c]$ will return *false* almost always. This is no problem since RTAA* is still guaranteed to lead the agent to the goal state. For more details on this discussion we refer the reader to [5]. In the next section we will show that this implementation of InTree$[c]$ will not guarantee termination; this will motivate a new version of InTree$[c]$.

Algorithm 2: InTree$[c]$ function

```
   Input: a vertex $s$
1  while $s \neq g$ do
2      Paint $s$ with color $c$.
3      if $\mathrm{p}(s) = null$ or $\mathrm{p}(s)$ has color $c$ then
4          return false
5      $s \leftarrow \mathrm{p}(s)$
6  return true
```

An important aspect of using a real-time search algorithm for reconnection is what heuristic to use. In contrast to the traditional use of real-time search algorithms, which is to guide the agent to the goal, here RTAA* is used to find a reconnecting path. Rivera *et al.* [5] defined the concept of *admissible reconnecting heuristic*. By using them termination is guaranteed. In addition, they showed that both the null heuristic ($h(s) = 0$, for every $s \in S$), and the octile distance are admissible reconnecting heuristics and hence can be used along with FRIT$_{\text{RT}}$. Experimentally, they showed that best results in pathfinding are obtained with the null heuristic. The intuition for this is that, upon reconnection, search must be guided away from heuristic depressions rather than towards the goal. The null heuristic seems to better serve that purpose.

Algorithm 3: Real-Time Adaptive A* for FRIT

```
Input: A search problem $P$, and a heuristic function $h$.
Effect: The agent is moved from the initial state to a goal state if a trajectory exists
1  $h_0 \leftarrow h$
2  $s_{current} \leftarrow s_0$
3  while $s_{current} \notin G$ do
4      $\mathtt{A*}\ (k)$
5      if $Open = \emptyset$ then return no-solution
6      $s_{next} \leftarrow \arg\min_{s \in Open} f(s)$
7      for each $s \in Closed$ do
8          $h(s) \leftarrow f(s_{next}) - g(s)$
9      Follow the path connecting $s_{current}$ and $s_{next}$ that was identified by $\mathtt{A*}$ and that can be
       extracted by following the back pointers from $s_{next}$. Update $Obs$ while moving. Stop if an
       obstacle is detected in the path or if $s_{next}$ is reached.
10     Assuming $\sigma = s_0 s_1, \ldots s_n$ is the path traversed in this iteration, make $\mathrm{p}(s_i) = s_{i+1}$ for every
       $i \in \{0, \ldots, n-1\}$.
11     Set $s_{current}$ to the current position of the agent.
```

2.3 Bug Algorithms

Bug algorithms [6] are pathfinding algorithms for continuous 2D terrain. Unlike real-time heuristic search algorithms, they do not work in general search problems and do not exploit heuristics.

An algorithm relevant to our evaluation is Bug2 [14], which behaves as follows. First it defines a straight line connecting the initial position with the final position, which henceforth we call *m-line*. During execution Bug2 follows the m-line until encountering an obstacle or reaching the goal. If an obstacle is encountered, it saves the position at which the obstacle was hit in a variable called *hit point* and then starts following the boundary of the obstacle (either clockwise or counterclockwise) until the m-line is encountered again. Then, if the current position is closer to the goal than the hit point, the agent starts following the m-line again and the process repeats.

The trajectories generated by Bug2 "go around" obstacles. In the pathfinding problem defined by Fig. 1, there are two trajectories that can be returned by Bug2, depending on the side that is chosen when the obstacle in F6 is observed.

3 A Pathfinding-Specific Version of FRIT$_{\mathrm{RT}}$

FRIT$_{\mathrm{RT}}$ was shown in [5] to have to have good performance in pathfinding relative to other real-time heuristic search algorithms. Indeed, its performance was shown to be comparable to that of daRTAA* [12], which is considered state-of-the-art in real-time pathfinding. Nevertheless, there are situations in which paths returned by FRIT$_{\mathrm{RT}}$ have a large number of steps; much larger than those returned by pathfinding-specific algorithms like Bug2 or even FRIT(BFS). There are two reasons for this. First, reconnection is carried out using a standard real-time search algorithm (in this case, RTAA*), the agent may be required to revisit some states many times while reconnecting.

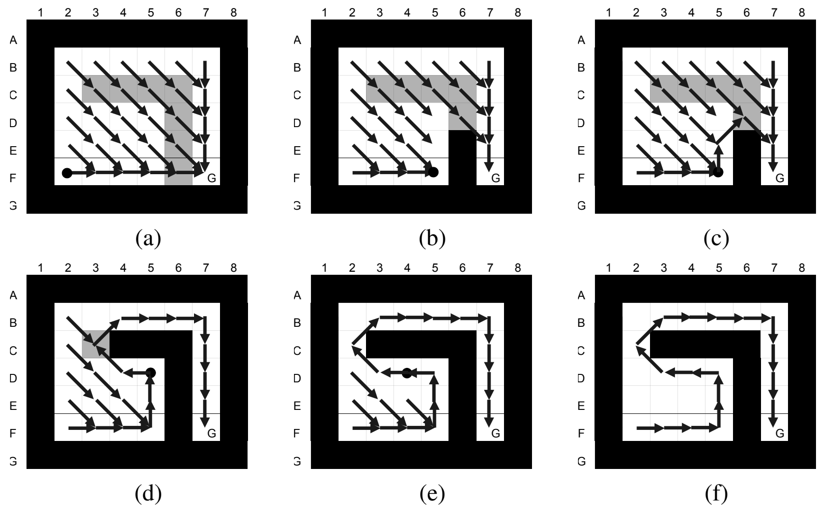

Fig. 1. An execution of FRIT (BFS) when solving the pathfinding problem of moving an agent from cell F2 to cell F7. The gray areas represent obstacles that have not been detected yet by the agent. The position of the agent is shown by the dot. The arrows are a representation of the ideal tree as maintained by FRIT. The first few moves, (a)-(b), trivially follow the ideal tree moving the agent from F2 to F5. In (b), the ideal tree has been updated by removing edges that go through the observed obstacles. (c) shows the agent in the same location after the reconnection through BFS has been completed connecting F5 to D6 through F6. (d)-(e) show the next few stages of the search after reconnection is performed. After (e), the rest of the search is straightforward and the search effort is minimal. The path followed by the agent throughout the whole execution is shown in (f).

The second reason explaining poor behavior of FRIT_{RT} is that the search space considered by FRIT_{RT} during reconnection is large. An illustration is given in Fig. 2. In that situation the path followed FRIT_{RT} (RTAA*) covers a complete *area* underneath the obstacle, while a bug algorithm would only move on the *perimeter* of that obstacle. As a general conclusion, FRIT_{RT} (RTAA*) may return solution paths whose size is quadratic on the size of paths returned by pathfinding-specific bug algorithms.

Our approach to reducing the search space is strongly inspired by the design principle of bug algorithms. We propose to restrict the reconnection search space only to states that are in the border of an obstacle. This can be ensured by replacing the A* search in RTAA* by Algorithm 4, which in Line 10 guarantees that a state is added to $Open$ only if it has a neighbor which is an obstacle too. We call the resulting algorithm p-FRIT_{RT}. It is not hard to verify that p-FRIT_{RT} (RTAA*) solves the problem of Fig. 2 in 19 steps.

A rather important detail to notice is that since Algorithm 4 only considers states next to obstacles, RTAA*'s learning rule applies only to those states. Although Algorithm 4 imposes a clockwise order to look for new open states, any order can be implemented. On the other hand, in order to ensure that FRIT_{RT} always reaches the goal, the $\text{I}\text{N}\text{T}\text{REE}[c]$ function needs modification. As mentioned above, if time constraints are tight InTree$[c]$ will return **false** most of the times putting reconnection at risk. Since Algorithm 4 only inserts into *Open* states that are next to an obstacle, RTAA* moves the agent only through those states. If the goal state is not adjacent to a wall and the agent does not have enough time to check whether states are connected to the ideal tree then it may end up looping forever around an obstacle. To address this, we propose a modification of the InTree$[c]$ function, shown in Algorithm 5.

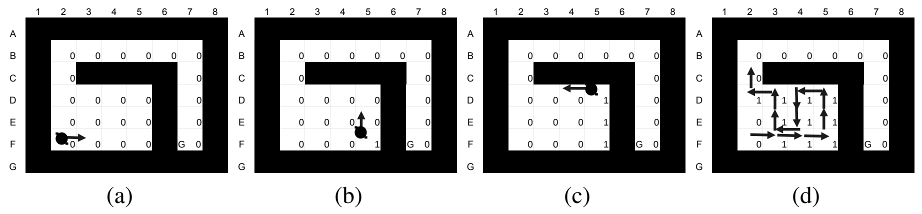

Fig. 2. An illustration of the execution of FRIT$_{\mathrm{RT}}$ (RTAA*), run with lookahead parameter equal to 1, at solving a pathfinding problem whose start state is cell F2, and whose goal cell is F7. In (a)-(d), the dot shows the position of the agent, the arrow denotes the next action to be carried out, and the number in the cell denotes the value of the reconnecting heuristic that we use in this example ($h = 0$). For the first three moves (a)-(b) the agent follows the ideal tree. Once it reaches F5, the obstacle cell at F6 becomes disconnected from the tree and reconnection is initiated via running RTAA*. Now we assume the cells are added to A* priority queue in clockwise order, starting from the top cell, and that, moreover ties are broken in the priority queue in the same order. After expanding F5, E5 is the best successor, and thus the agent moves upward, updating the heuristic value of F5 to 1. Analogously, the agent moves upwards until reaching cell D5 (c). D5's best successor is D4, and due to tie-breaking rules the agent moves downward, then left, and then upwards again, following the path shown in (d).

Algorithm 4: Bounded A* lookahead restricted to follow walls

```
procedure A* ($k$)
    for each $s \in S$ do $g(s) \leftarrow \infty$
    $g(s_{current}) \leftarrow 0$; $Open \leftarrow \emptyset$
    Insert $s_{current}$ into $Open$
    $expansions \leftarrow 0$
    while each $s' \in Open$ with minimum $f$-value is such that InTree$[c](s')$ is not true and $expansions < k$ do
        Remove state $s$ with smallest $f$-value from $Open$
        Insert $s$ into $Closed$
        for $i$ in $1, \ldots, 8$ do
            if $Succ(s, i-1) \in Obs$ or $Succ(s, i+1) \in Obs$ then
                if $g(s') > g(s) + c(s, s')$ then
                    $g(s') \leftarrow g(s) + c(s, s')$
                    $s'$.back $= s$
                    if $s' \in Open$ then remove $s'$ from $Open$
                    Insert $s'$ in $Open$
        $expansions \leftarrow expansions + 1$
```

Algorithm 5: Modified IN<span>TREE</span>$[c]$ function

```
Input: a vertex $s$
1  $s' \leftarrow s$
2  if $\mathrm{sp}(s') = null$ then
3      return false
4  else
5      $s' \leftarrow \mathrm{sp}(s')$
6  while $s' \neq g$ do
7      Paint $s'$ with color $c$.
8      if $\mathrm{p}(s') = null$ or $\mathrm{p}(s')$ has color $c$ then
9          return false
10     $s' \leftarrow \mathrm{p}(s')$
11     $\mathrm{sp}(s) \leftarrow \mathrm{p}(s')$
12 return true
```

Algorithm 5 introduces a new attribute for states called the *super parent* sp. At the outset of each reconnection (i.e., each call of RTAA* in Algorithm 1) we set $\mathrm{sp}(s) = s$ for every s. The idea of Algorithm 5 is now that if the agent does not have enough time to complete the INTREE$[c]$ process from a state s and it had to move, if we perform the INTREE$[c]$ function again from s, we can recover the work done starting the search from $\mathrm{sp}(s)$ instead of s. It is easy yet important to observe that if we repeat Algorithm 5 from the same state a sufficient number of times the algorithm will eventually return true if the state is connected to the goal state.

3.1 Theoretical Analysis

In this section we prove that p-FRIT$_{\mathrm{RT}}$ is *correct*; i.e., that it guides an agent to the goal when a solution exists. Though p-FRIT$_{\mathrm{RT}}$ was designed for pathfinding, we extend it now to a more general class of graphs. Below, $G = (S, A)$ is an undirected graph, i.e., one in which $(s, s') \in A$ if and only if $(s', s) \in A$.

We say that $B \subset S$ is a *fence* with respect to state s and goal state g if $g \notin B$, B is a connected graph and whenever there is a path from s to g then $\partial B \cap L(s, B)$ is connected, where $L(s, B)$ is the set of all vertices reachable by a path from s without using states of B. The idea of the fence is that the agent cannot cross it but can "skirt around it". Note that, for technical reasons, B is a fence in the trivial cases when $s \in B$ or B cuts all paths from s to g.

We say G is *nice* with respect to state s and goal state g, if every $B \subset V$ is a fence with respect to s and g. A graph is nice (with respect to g) if it is nice with respect to s and g for all $s \in S$. As a way of example an $n \times m$ grid that has a solution is nice.

Remark 1. This simple observation about the definition of fence is the key to prove that the algorithm always finds a solution. Let G be a connected, nice undirected graph with g the goal state. Let s be a state such that $\pi = ss_1 \ldots s_n g$ is a path from s to g. Let B be a fence w.r.t. s and g, and $s \notin B$ but $s_i \in B$ for some i (the reader should think B as a set of obstacles that cut the path π). We can always reconstruct a new path P' with the following idea, let N be the greater N such that $s_N \in B$, then $s_{N+1} \in \partial B$ (note that s_{N+1} might be g), then we can construct a path starting from s, then move along the path until we hit B, then move through states in ∂B until we reach s_{N+1} and continue using π again.

Theorem 1. *Consider a search problem P, and assume that at every moment, the current search graph G_M is nice. Then the algorithm finds a solution.*

To prove Theorem 1 we use the following intermediate result.

Lemma 1. *Suppose that from the current state p-FRIT$_{RT}$ follows the path defined by the* p *pointers and that when the agent reaches state s the path becomes blocked by an obstacle. Then reconnection search can find a reconnecting path to the ideal tree.*

Proof. Recall that when the ideal tree is disconnected then the agent runs RTAA* using the modified version of A* in Algorithm 4 which restricts to states next to an obstacle. Let G_M be the current graph, let x be the obstacle in the path that was being followed and let B be the set of all obstacles connected with x. Since G_M is nice, and B is a fence the agent can move on the wall of B. By Remark 1, there exists a set of states on the wall of B, say U, such that the vertices of U are connected with g and thus if the agent can identify them, it can reconnect to the ideal tree. Finally, even though it may take several runs of Algorithm 5 from the same vertex, a connected vertex is eventually recognized since the graph is finite and connected. RTAA* will make the agent move on the wall of B until it recognizes a state of U connected to the ideal tree.

Proof (Of theorem 1). Let $\mathcal{T}$ be the ideal tree defined by the p pointers after a reconnection or at the beginning of the algorithm. Moreover, let s be the position of the agent after such reconnection. Let π be the path in $\mathcal{T}$ that goes from s to g. Recall the agent will follow π until it reaches g or finds an obstacle. Now assume an obstacle is found, and let G_M be the known graph at the point the agent found the obstacle. The above lemma says that we can reconnect the tree on G_M, recovering a new tree that connects the position of the agent with the goal state. Since the initial tree $\mathcal{T}$ connects the initial position with g (in the initial known graph) and the set of connected components of obstacles is finite, we repeat the argument inductively, finishing the proof.

4 Experimental Evaluation

We implemented our algorithm over the same codebase of FRIT. The objective of our experimental evaluation was to compare p-FRIT$_{\mathrm{RT}}$ (RTAA*) with the state of the art in real-time pathfinding, which is represented by daRTAA* and FRIT (RTAA*). The objective of our evaluation was not to establish a relationship between p-FRIT$_{\mathrm{RT}}$ and the state of the art in bug algorithms. However, for reference we also include a comparison with Bug2, which we is easy to implement.

We evaluated over 12 game maps from N. Sturtevant's pathfinding repository [15]. The maps we considered come from the games *Dragon Age*, and *StarCraft*.[4] We generated 500 problems for each of them. We ran the real-time algorithms in 9 lookahead configurations $(1, 2, 4, 8, 16, 32, 64, 128, 512)$. We assume the agent can observe the blockage status of its neighbor cells. All experiments were run on a Linux 2.00GHz QuadCore Intel Xeon machine with 128MB of RAM.

[4] **Map details.** Dragon Age: brc202d, orz702d, orz900d, ost000a, ost000t and ost100d; sizes: 481×530, 939×718, 656×1491, 969×487, 971×487, and 1025×1024 resp. StarCraft: ArcticStation, Enigma, Inferno, JungleSiege, Ramparts and WheelofWar; sizes: 768×768, 768×768, 768×768, 768×768, 512×512 and 768×768 resp.

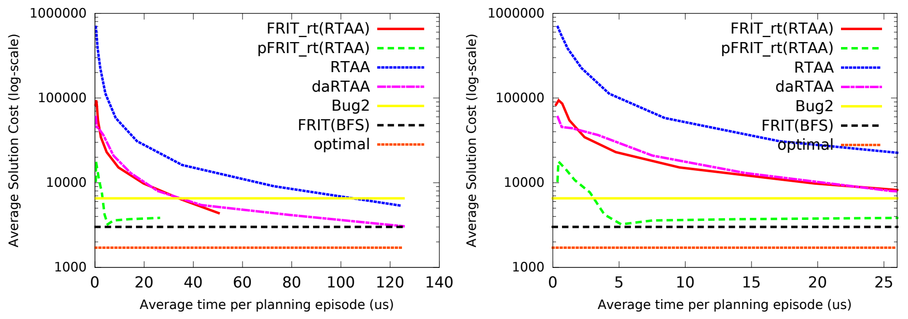

Fig. 3. A comparison of p-FRIT$_{\mathrm{RT}}$, FRIT$_{\mathrm{RT}}$ (RTAA), and the state-of-the-art daRTAA* over 12 standard game maps. The plot on the right-hand side is a zoomed version of the one on the left.

Figure 3 compares solution quality obtained by the various algorithms. FRIT(BFS) is not a real-time algorithm and its average solution cost is included in the plots for reference only. On the other hand, Bug2 returns a single solution in very little time; it is shown in the figure as a horizontal line, rather than a simple dot, for clarity. As well, "optimal" is the average optimal solution (obtained using RTAA* using an infinite lookahead parameter) and it is included for reference as a horizontal line.

p-FRIT$_{\mathrm{RT}}$ significantly outperforms the real-time search algorithms daRTAA* and FRIT$_{\mathrm{RT}}$ (RTAA*) with respect to average time spent per search episode. In fact, for lookahead equal to 1 p-FRIT$_{\mathrm{RT}}$ (RTAA*) generates a solution that is 6.9 times cheaper than that produced by daRTAA* and 8.3 times better than the one returned by FRIT$_{\mathrm{RT}}$ (RTAA*). For other values of the lookahead parameter we observe a similar behavior. For example, for lookahead 16, the solution returned by p-FRIT$_{\mathrm{RT}}$ (RTAA*) is 2.7 times cheaper than that produced by daRTAA* and 4.5 times better than the one returned by FRIT$_{\mathrm{RT}}$ (RTAA*). For lack of space we omit total runtime plots. However, p-FRIT$_{\mathrm{RT}}$ (RTAA*) clearly outperforms the other real-time search algorithms on that metric. For example, it is 5.52, 23.62, and 2.81 times faster than daRTAA* for lookaheads 1, 16, and 512, respectively, and on average 9.81 times faster than daRTAA*, 22.8 times faster than RTAA*, and 7.55 times faster than FRIT$_{\mathrm{RT}}$ (RTAA*).

The average cost of the solutions returned by Bug2 is 6,546, with an average runtime of 2,317 μs. p-FRIT$_{\mathrm{RT}}$ (RTAA*), on the other hand, obtains solutions of cost 7,629 on average for lookahead parameter 16, with an average runtime of 4,777 μs. For lookahead parameter 32, p-FRIT$_{\mathrm{RT}}$ (RTAA*) obtains an average solution of cost 4,210 with an average total runtime of 4,008 μs. The best average solution cost obtained by p-FRIT$_{\mathrm{RT}}$ (RTAA*) is for a lookahead equal to 64, which yields an average solution of cost 3,225—about half of the cost obtained by Bug2—, with a runtime of 4,838 μs.

We conclude that p-FRIT$_{\mathrm{RT}}$ significantly outperforms the state of the art in real-time heuristic search for pathfinding tasks. Given sufficient time, p-FRIT$_{\mathrm{RT}}$ may also return solutions substantially better than those obtained by Bug2 and thus should be preferred in situations in which time constraints allow for at least 64 node expansions.

5 Summary & Final Remarks

We presented p-FRIT$_{RT}$, a real-time algorithm tailored to pathfinding which we showed outperforms by a large margin the state of the art in real-time heuristic search algorithms for pathfinding. p-FRIT$_{RT}$ is a modification of FRIT$_{RT}$ that searches for reconnection only on states that are in the border of obstacles. We proved p-FRIT$_{RT}$ always terminates leading the agent to a goal in a class of problems that subsumes 8-connected grids.

The main idea underlying FRIT$_{RT}$ draws inspiration from bug algorithms, but is also related to other recent trends in offline pathfinding in grids which restrict the search space exploiting the fact that optimal paths must touch the borders of obstacles (i.e., [16, 17]).

References

1. Bulitko, V., Björnsson, Y., Sturtevant, N., Lawrence, R. Applied Research in Artificial Intelligence for Computer Games. In: Real-time Heuristic Search for Game Pathfinding. Springer Verlag (2011) 1–30
2. Korf, R.E.: Real-time heuristic search. Artificial Intelligence **42**(2-3) (1990) 189–211
3. Ishida, T.: Moving target search with intelligence. In: Proc. of the 10th National Conf. on Artificial Intelligence (AAAI). (1992) 525–532
4. Rivera, N., Illanes, L., Baier, J.A., Hernández, C.: Reconnecting with the ideal tree: An alternative to heuristic learning in real-time search. In: Proc. of the 6th Symposium on Combinatorial Search (SoCS). (2013)
5. Rivera, N., Illanes, L., Baier, J.A., Hernández, C.: Reconnection with the ideal tree: A new approach to real-time search. Journal of Artificial Intelligence Research **50** (2014) 235–264
6. LaValle, S.M.: Planning algorithms. Cambridge University Press (2006)
7. Hart, P.E., Nilsson, N., Raphael, B.: A formal basis for the heuristic determination of minimal cost paths. IEEE Transactions on Systems Science and Cybernetics **4**(2) (1968) 100–107
8. Koenig, S., Likhachev, M.: Real-time Adaptive A*. In: Proc. of the 5th Int'l Joint Conf. on Autonomous Agents and Multi Agent Systems (AAMAS). (2006) 281–288
9. Hernández, C., Meseguer, P.: LRTA*(k). In: Proc. of the 19th Int'l Joint Conf. on Artificial Intelligence (IJCAI). (2005) 1238–1243
10. Hernández, C., Meseguer, P.: Improving LRTA*(k). In: Proc. of the 20th Int'l Joint Conf. on Artificial Intelligence (IJCAI). (2007) 2312–2317
11. Koenig, S., Sun, X.: Comparing real-time and incremental heuristic search for real-time situated agents. Autonomous Agents and Muti-Agent Systems **18**(3) (2009) 313–341
12. Hernández, C., Baier, J.A.: Avoiding and escaping depressions in real-time heuristic search. Journal of Artificial Intelligence Research **43** (2012) 523–570
13. Zelinsky, A.: A mobile robot exploration algorithm. IEEE Transactions on Robotics and Automation **8**(6) (1992) 707–717
14. Lumelsky, V.J., Stepanov, A.A.: Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. Algorithmica **2** (1987) 403–430
15. Sturtevant, N.R.: Benchmarks for grid-based pathfinding. IEEE Transactions Computational Intelligence and AI in Games **4**(2) (2012) 144–148
16. Harabor, D.D., Grastien, A.: Online graph pruning for pathfinding on grid maps. In: Proc. of the 26th AAAI Conf. on Artificial Intelligence (AAAI). (2011)
17. Uras, T., Koenig, S., Hernández, C.: Subgoal graphs for optimal pathfinding in eight-neighbor grids. In: Proc. of the 23rd Int'l Conf. on Automated Planning and Scheduling (ICAPS). (2013)