Envelope-based Approaches to Real-Time Heuristic Search

Kevin C. Gall and **Bence Cserna** and **Wheeler Ruml**
Department of Computer Science
University of New Hampshire, USA
kcg245 at gmail.com, bence at cs.unh.edu, ruml at cs.unh.edu

Abstract

In real-time heuristic search, the planner must return the next action for the agent within a pre-specified time bound. Many algorithms for this setting are 'agent-centered' in that, at every iteration, they only expand states near the agent's current state, discarding the search frontier afterwards. In this paper, we investigate the alternative paradigm in which the search expands a single ever-growing envelope of states. Previous work on envelope-based methods restricts the agent to move along the generated search tree. We propose a more flexible approach in which an auxiliary search is performed within the envelope to guide the agent toward a promising frontier node. Experimental results indicate that intra-envelope search is beneficial in state spaces that are highly interconnected, such as those for grid pathfinding.

Introduction

Interactive systems, such as robots or user interfaces, often must be controlled in real time. In this paper, we address real-time planning, where the planner must return the next action for the system to take within a specified wall-clock time bound (Korf 1990). The traditional approach to real-time planning is agent-centered, meaning a portion of the state space immediately surrounding the agent — the Local Search Space (LSS) — is expanded to some bounded lookahead (Koenig 2001). For example, in the popular LSS-LRTA* algorithm (Koenig and Sun 2009), the agent generates the LSS using A* and then moves toward the frontier node of lowest f. This paradigm has a significant drawback: the agent can easily fall into local minima, also known as heuristic depressions, in which the h values are misleadingly low, distracting the agent from paths leading toward a goal. To guarantee completeness in domains with local minima larger than the lookahead, the heuristic values of nodes within the LSS are raised (Koenig and Likhachev 2006). However, multiple visits to the same state may be required before the values are high enough, a process known as 'scrubbing' (Sturtevant and Bulitko 2016). Many attempts have been made at improving the ability of LSS-style algorithms to escape heuristic depressions including analysis of heuristic error and dynamically changing lookahead (Hernández and Baier 2012; Kiesel, Burns, and Ruml 2015). However for any lookahead size achieved with finite resources, one can always imagine a domain with local minima larger than the achieved lookahead.

Time-Bounded A* (TBA*) (Björnsson, Bulitko, and Sturtevant 2009) represents an alternative to agent-centered methods and is the first example of what we call Envelope-based Search (ES) for real-time search problems. TBA* iteratively expands a single A* search tree throughout the life of the algorithm and periodically commits the agent toward the best node on the A* frontier by tracing and following the pointers of the tree. Because A* with a consistent heuristic will expand a state at most once, scrubbing is eliminated. TBA* is able to escape heuristic depressions by directing the agent toward a more promising branch of the tree without spending computation time updating heuristic values. However, the algorithm also has drawbacks stemming from the fact that its search is rooted at the agent's start state, an increasingly irrelevant state with respect to the agent as it transitions away. In the worst case, when a new target node is selected the agent will backtrack all the way to the start state before reaching the new branch.

Variants of TBA* that use Weighted A* and Greedy Best First Search to expand the search graph (Hernández, Baier, and Asín 2016) alleviate this problem by biasing toward nodes with low h, discouraging the search from finding new paths that deviate from the tree near the root (reminiscent of the 'low h bias' described by Richter, Thayer, and Ruml (2010)). However, even with this enhancement, the agent's path through the constructed search graph is still inflexible since the graph is not typically rewired to exploit shortcuts between the agent's current branch and a new target branch.

We propose a new ES algorithm called Intra-Envelope Search (I-ES) that seeks to address the flaws in TBA* by splitting time between expanding nodes on the envelope frontier and searching through the expanded envelope to connect the agent with a target node. This target is chosen periodically as the current best node on the envelope frontier. We show that I-ES is complete in undirected and directed domains under reasonable assumptions. Experimental results suggest that the new method has superior performance in domains in which it is easy to find alternative paths between states, such as grid pathfinding.

Preliminaries

A heuristic search problem $\langle S, E, Goal, h, s_{root} \rangle$ consists of a set of states S, a set of edges E labeled with positive costs between states, a predicate $Goal(s)$ that returns true if s is a goal, a heuristic function $h(s, s')$ that estimates the cost of an optimal path $c^*(s, s')$ between s and s', where $h(s)$ denotes $\min_{\{s' | Goal(s')\}} h(s, s')$, and the agent's start state $s_{root} \in S$. All heuristics used in this paper are *admissible*, meaning that they do not overestimate c^*. We denote by $g(s)$ the cost of the cheapest known path to reach s from s_{root}, $f(s) = g(s) + h(s)$, OPEN holds all nodes that have been generated but not expanded, and f_{min} is the minimum f value on OPEN. The successors of state s, denoted s_{succ}, are those s' such that $\exists\ edge \in E\ :\ edge = (s, s'),\ c(s, s') < \infty$. Similarly, s_{pred} denotes the predecessors.

Envelope-Based Search

Envelope Search (ES) offers an alternative to the LSS paradigm of limited lookahead followed by heuristic updates. An ES algorithm maintains a search envelope containing every node expanded throughout the life of the algorithm. ES algorithms direct the agent toward the best node discovered during *the entire search so far*, not simply the best node in the agent's immediate vicinity. Heuristic depressions are much more easily avoided by ES algorithms because minima can be fully explored over multiple iterations and thereafter never considered for the agent's target. In this way, they can reason about the entire portion of the state space generated so far, avoiding the obvious blunders that plague agent-centered algorithms.

The problem of discovering and extracting a path from the agent to some arbitrary target node is non-trivial. Agent-centered search in the absence of dead ends typically guarantees that a path will be available and easily extracted for the agent within the time bound by virtue of the fact that no nodes are considered which are farther away from the agent than the bound. In Envelope Search, however, the agent may be arbitrarily far away from the node set as its target. ES algorithms must be able to incrementally construct paths connecting the agent to the target so that the time constraint is not violated. TBA* and I-ES represent alternative approaches to this problem.

Time-Bounded A*

Time-Bounded A* (TBA*) (Björnsson, Bulitko, and Sturtevant 2009) is an Envelope Search algorithm that maintains a single A* search tree rooted at s_{root} throughout the life of the algorithm. The tree is constructed incrementally exactly as in offline A* by expanding nodes until the constant time bound is reached. TBA* periodically commits the agent to the path leading to the current-best node on the frontier with respect to f_{min}. This is achieved by tracing back paths from target nodes via the parent pointers in the A* search tree. TBA* inherits from A* completeness in solvable infinite undirected domains. Though the algorithm can make no guarantees about overall solution quality since it is real-time, in practice the underlying optimality of the A* search tree tends to create high-quality solutions.

Algorithm 1: TBA*

```
input : $bound, tracebackLimit$
1  OPEN $\leftarrow \{root\}$
2  $loc \leftarrow root$
3  while $loc \neq goalState$ do
4      A* from OPEN until $bound$ is exhausted
5      if $goal\ node\ not\ traced$ then
6          $newTarget \leftarrow$ TracePath($tracebackLimit$)
           if $newTarget\ is\ not\ null$ then
7              $agentTarget \leftarrow newTarget$
8      if $loc\ is\ on\ path\ to\ agentTarget$ then
9          $loc \leftarrow loc.next$
10     else
11         $loc \leftarrow loc.parent$
```

Pseudocode for TBA* appears in Algorithm 1. TBA* begins precisely as A*: seeding the open list with the root state [line 1] and expanding nodes sorted on f [line 4]. The A* search is paused when *bound* is reached. Then, the algorithm performs its "Traceback" phase [line 6].The *traceback* function takes *tracebackLimit* as a parameter which places a constant bound on the amount of time the algorithm can spend tracing a path. If no trace-in-progress is cached, TBA* selects the best node from the open list and begins extracting a path by following parent pointers from the search tree. The trace terminates on any of three criteria: the trace reaches the root state, the trace reaches the agent's location, or the *tracebackLimit* is reached. In the last scenario, the trace-in-progress is cached so that it can be resumed during the next iteration, and *null* is returned. When a trace terminates at the root or the agent's location, the node at the end of the path is returned as *newTarget* and is set as the agent's target [line 6]. The agent moves based on a simple decision: if it is on the path to its current target, continue on that path. If not, move to the previous state on its current path.

Both the number of nodes expanded by A* and the number of nodes traced in the Traceback phase are bounded by a constant, so an iteration guarantees constant time complexity, i.e. $O(bound + tracebackLimit)$.

Björnsson, Bulitko, and Sturtevant (2009) describe some simple optimizations to the basic algorithm. The 'threshold' optimization prevents the agent from choosing a new path unless the g value of *newTarget* is at least as high as the g value of *agentTarget*. This prevents the agent from switching rapidly between paths that are roughly as good. This optimization was shown to produce 2% better paths in certain circumstances, and so we include it in our implementation. A second technique, 'shortcutting,' will be discussed below.

Variants: Weighted, Greedy, and Restarting

The original TBA* algorithm, while effective compared to agent-centered competitors, is flawed due to its reliance on g as a function of s_{root}. Hernández, Asín, and Baier (2014) introduce two variants: Time-Bounded Weighted A* (TB(WA*)) and Time-Bounded Best-First

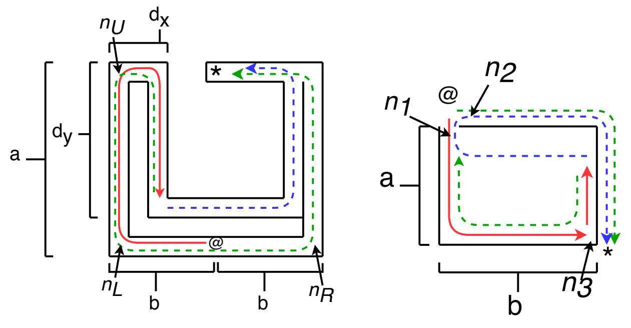


Figure 1: Challenges for TB(WA*) (left), TB(BFS) (right). Red solid lines approximate the trajectory of TBA* to a certain time, green dotted lines represent the backtracking path TBA* takes when a new branch leads to a goal, and blue dotted lines represent a shortcut from the red path to the goal.

Search (TB(BFS)). The modification is simple but powerful: instead of an A* search, the underlying search tree is constructed using Weighted A* or Greedy Best-First Search respectively. After all, there is no reason that the agent must necessarily find an optimal path from the root to a goal, as its actual trajectory is likely to be suboptimal.

TB(WA*) reduces the agent's dependence on s_{root} by using the weighted A* equation $f(s) = g(s) + w * h(s)$ where $w > 1.0$. This means that the algorithm is not restricted to finding the optimal path, but rather any path within the suboptimality bound w. As a result, increasing the weight can often lead to lower-cost total solutions since the agent is not forced to backtrack toward an optimal branch of the tree when the goal is closer to the agent's current location.

TB(BFS) divorces the search from the concept of g at all and instead conducts a Greedy Best-First Search ordered on h. This allows the agent to follow the heuristic greedily. The search tree is still rooted at s_{root}, but when the tree is long and skinny, as it would be with a heuristic accurate enough to be consistently decreasing, the agent rarely has to backtrack.

Restarting Time-Bounded Weighted A* (TB_R(WA*)) (Hernández, Baier, and Asín 2016) is a variant that is complete on directed search graphs (i.e. graphs with non-reversible edges). In this variant, when the agent attempts to backtrack but no edge to the previous state exists, the search graph is discarded. The algorithm then begins expanding a new search graph rooted at the agent's current state. In order to achieve completeness, the heuristic values of all nodes in the discarded search tree are updated using a Dijkstra backup in exactly the same manner as LSS-LRTA*.

Shortcuts

Even with greedy envelope expansion, the problem remains that the agent's behavior is tied to the search tree anchored at s_{root}, a state which is irrelevant after the first iteration. (In fact, the low h bias of greedy expansion can even worsen the problem.) Figure 1 presents examples illustrating this poor behavior. Each case is an instance of grid pathfinding with four-way movement and the Manhattan distance heuristic. The start location is marked by @ and the goal by *. As we will explain, the red solid lines approximate the trajectory of TBA* up to a certain time. The green dotted lines represent the backtracking path that TBA* follows when it discovers another path to the goal. The blue dotted lines represent the shortcut path through the state space that an agent could have taken from the end of the red path to reach the goal more quickly.

The left panel shows a troublesome instance for TB(WA*). Because h is Manhattan distance and increases as we move away from the start, the search will begin by alternately expanding the paths to the left and right until either the bottom left (n_L) or bottom right (n_R) corner node is expanded. Observe that $h(n_L) = h(n_R) = a + b$. Call this value h_b, so $f(n_L) = f(n_R) = b + wh_b$. Assume without loss of generality that TB(WA*) breaks the tie in favor of n_L. For the ith successor of n_L such that $i < a$, observe that $h(n_i) = (a - i) + b$. Therefore, $f(n_i) = b + i + wh(n_i) = b + i + w(a+b) - wi = b + wh_b + i(1-w)$, so $f(n_i) < f(n_R)$ if $w > 1$. Therefore, all nodes in the left corridor will then be expanded before expanding n_R. Assuming that a and b are large relative to the lookahead size, the agent will move along the red path toward n_U. Then, let $d_i = d_{x_i} + d_{y_i}$ be the distance of a node l_i past n_U, so $h(l_i) = b - d_{x_i} + d_{y_i}$ and $f(l_i) = b + a + d_i + wh(l_i)$. TB(WA*) will continue to expand along the red path as long as:

$$\begin{aligned}
f(l_i) &< f(n_R) \\
b + a + d_i + wh(l_i) &< b + wh_b \\
w(h(l_i) - h_b) &< -a - d_i \\
w(h_b - h(l_i)) &> a + d_i \\
w &> \frac{a + d_i}{h_b - h(l_i)} \\
w &> \frac{a + d_{x_i} + d_{y_i}}{a + d_{x_i} - d_{y_i}}
\end{aligned}$$

which can be made true for any $w > 1$ by increasing a. But as d_{y_i} increases relative to a, there will come a point where w is no longer large enough and n_R is expanded. In the figure, this is where the red path ends. Since n_R then becomes the best node on OPEN, it or its successors will become the agent's target. Because TB(WA*) moves the agent only along the search tree, the agent will be forced to backtrack to s_{root}, instead of discovering the shortcut represented by the blue dotted arrow.

The right panel shows a troublesome instance for TB(BFS). Let l be the lookahead. Observe that $h(n_1) = h(n_2) = a + b$. Without loss of generality, assume the search breaks ties in favor of moving south, so n_1 is expanded first and the agent falls into the local minimum where all nodes besides n_1 have $h < a + b$. Because the search order is greedy, nodes along the red path will be expanded until n_3 is reached, at which point the search will fill the room by diagonals. Once the room is filled, TB(BFS) will expand n_2 and backtrack the agent along the path it came in by. This will be longer than a shortcut (such as the blue path) if the agent has already passed n_3 by the time the room has filled. This is true when the number of actions the agent has taken, $\frac{ab}{l}$, is greater than $a + b$, which happens when $l = 1$ and

$a = b = 3$ or when $l > 1$ and $a = b = l^2$.

The original TBA* paper proposes a ‘shortcut’ optimization. When a goal is traced but the agent is not on the goal branch, a new search is conducted every iteration from the entire goal path toward the agent. If the agent is found, the path is extracted and followed (Björnsson, Bulitko, and Sturtevant 2009). We implemented this technique, but do not include it in our experiments because, for many of the domains we tested on, the algorithm could not complete within the allotted time. The technique requires seeding a new open list from s_{root} to the discovered goal every iteration after a goal is found, which is worst-case linear in the size of the domain, making it impractical in domains of even moderate size.

Iterative Updates in TB$_\mathrm{R}$(WA*) Hernández, Baier, and Asín (2016) make reference to the fact that the heuristic update operation of TB$_\mathrm{R}$(WA*) is not bounded by a constant and suggest it can be done iteratively. In consultation with the original authors, the following implementation was used in our experiments: When the agent attempts to backtrack but cannot, the existing OPEN is cached and reordered on h, and a new open list is allocated. We will call this new open list OPEN$_{current}$ and the cached one OPEN$_{cached}$. We choose the best successor s' of the agent's current state s with respect to the priority function and seed it into OPEN$_{current}$. The action transitioning s to s' is returned for execution. For every iteration while OPEN$_{cached}$ is not empty, a number of nodes are expanded from OPEN$_{cached}$ equal to the number of nodes expanded in the forward search, thus bounding the number of updates performed by a constant. Note that expansions from OPEN$_{cached}$ are executed as in LSS-LRTA* (Koenig and Sun 2009). When OPEN$_{cached}$ is empty it is discarded. If the algorithm attempts to restart when there is already a non-empty OPEN$_{cached}$, then the existing OPEN$_{current}$ is not cached but simply discarded. This ensures that there is no way any state can be “discarded” from OPEN$_{cached}$ repeatedly and thus never updated.

Observe that the above maintains completeness by not disrupting any invariants of the original TB$_\mathrm{R}$(WA*). Any state that is continually visited will eventually be moved to OPEN$_{cached}$ and updated accordingly.

If OPEN is ordered on weighted f, reordering OPEN$_{cached}$ on h is a linear operation in the size of the frontier and thus still not bounded by a constant. However, it is likely to be significantly less than the operation of updating the entire closed list when restarting is triggered, which is linear in CLOSED, not OPEN, and so represents an improvement over the scheme presented by Hernández, Baier, and Asín (2016).

Intra-Envelope Search

We now turn to a new approach to Envelope Search we call Intra-Envelope Search (I-ES). Like TBA* and variants, a single search frontier is maintained throughout the life of the algorithm. Periodically, the best node on the frontier is selected as the agent's target, and a search is performed within the envelope to connect the agent to that target node. These connection searches allow the agent to “shortcut” through the state space toward a target node. Pseudocode is presented in Algorithm 2.

I-ES maintains an open list OPEN$_{fr}$ representing the Envelope frontier. An additional open list OPEN$_{bk}$ is maintained for the intra-envelope search and is described in detail below. The algorithm is agnostic to the ordering of the nodes on these open lists except that the priority function must increase as h increases in order to provide completeness guarantees (Theorem 2).

The algorithm takes a parameter $bound$ which is a resource limit on the iteration execution time. This limit is split into 2 parts by multiplying by a factor $0 < r_1 < 1$. In our experimentation we used $r_1 = 0.8$, implying more frontier expansion and less search within the envelope. The “cache” of the agent's path is set to an empty list, and OPEN$_{fr}$ is seeded with the root node [line 7].

While the agent has not reached the goal, the algorithm proceeds in two parts: Exploration and Path Extraction. The EXPLORE function expands nodes from and adds successors to OPEN$_{fr}$ until $bound_{fr}$ is exhausted [line 9]. Every state s expanded by EXPLORE has its heuristic value updated to $s.h = \min_{s' \in s_{succ}} s'.h + c(s, s')$. This update is only relevant in directed graphs, where it provides completeness guarantees (Theorem 2).

In Path Extraction, I-ES searches through the envelope to connect the agent to a target node. There are many possible techniques for searching within the envelope, but for clarity, we focus on a simple backward search from a target to the agent. First, we seed the intra-envelope search open list if a search is not in progress [line 11]. s_{target} is set as the root of the backward search and $s_{forward}$ is set as its goal. Note that if our path cache is non-empty we seed the forward search from the end of the cached path [line 21].

Next the ENVELOPESEARCH function is executed. Using the backward strategy (pseudocode in Algorithm 3), nodes are expanded by examining their predecessors. Note that only predecessors that have already been generated by the Exploration stage are examined, ensuring the search remains within the envelope. The goal of the search is to connect s_{target} with $s_{forward}$, so a state s is ordered within OPEN$_{bk}$ using $h(s_{forward}, s)$. EXPAND sets pointers on predecessors so that the path to s_{target} can be extracted.

If $s_{forward}$ is expanded from OPEN$_{bk}$, the search is terminated and a path is extracted. The EXTRACTPATH function constructs a path by walking forward from $s_{forward}$ to s_{target} following pointers set in the envelope search. The resulting path is appended to $pathCache$ [line 28 - 29].

If no path is available after ENVELOPESEARCH, then a partial path is constructed by simply taking best successor of $s_{forward}$ with respect to h. Note that if $s_{forward}$ has not been expanded yet it is expanded as by EXPLORE during EXTRACTPARTIALPATH [line 14]. Finally, the agent transitions to the next state in the cached path.

Directed Domains I-ES works in directed search graphs with one modification. If OPEN$_{bk} = \emptyset$ and no connection path has been extracted, the entire envelope is reset: CLOSED$_{fr} \leftarrow \{\}$, OPEN$_{fr} \leftarrow \{s_{agent}\}$. Crucially, heuristic updates for states explored are maintained. If the backward search is empty, this means that the agent cannot reach

Algorithm 2: Intra-Envelope Search (I-ES)

```
input : $bound$
$s_{agent} \leftarrow root$
$s_{target} \leftarrow null$
$s_{forward} \leftarrow null$
$bound_{fr} \leftarrow r_1 * bound$
$bound_i \leftarrow (1.0 - r_1) * bound$
$pathCache \leftarrow [\,]$
$\text{OPEN}_{fr} \leftarrow \{root\}$
while $\neg Goal(s_{agent})$ do
    // Exploration
    Explore($bound_{fr}$)
    // Path Extraction
    if Envelope Search not in progress then
        ReseedEnvelopeSearch
    EnvelopeSearch($bound_i$)
    if $pathCache$ is empty then
        $pathCache \leftarrow$ ExtractPartialPath
    $s_{agent} \leftarrow pathCache.first$
    $pathCache.removeFirst()$
define ReseedEnvelopeSearch:
    $s_{target} \leftarrow \text{OPEN}_{fr}.top()$
    $\text{OPEN}_{bk} \leftarrow \{s_{target}\}$
    if $pathCache$ is not empty then
        $s_{forward} \leftarrow pathCache.last$
    else
        $s_{forward} \leftarrow s_{agent}$
```

Algorithm 3: EnvelopeSearch - Backward Strategy

```
while $bound_i$ not exhausted do
    $s \leftarrow \text{OPEN}_{bk}.pop()$
    Expand($s$)
    add $s_{pred} \cap (\text{CLOSED}_{fr} \cup \text{OPEN}_{fr})$ to $\text{OPEN}_{bk}$
    if $s = s_{forward}$ then
        $pathCache \leftarrow pathCache +$ ExtractPath($s$)
        break
```

s_{target} from within the expanded envelope, and so we begin re-exploring the area of the state space we know the agent will be able to reach.

Properties of I-ES

We now turn to characterizing conditions under which I-ES terminates and is complete.

Definition 1 *In an undirected domain, $\forall s \in S$, $\forall s' \in s_{succ}$, $s \in s'_{succ}$. Otherwise, the domain is directed.*

Definition 2 *In a solvable domain, $\forall s \in S$, $\exists$ path p : $p.first = s \wedge Goal(p.last)$ is true. Any s for which this does not hold is a dead end.*

Theorem 1 *I-ES is complete in finite undirected solvable domains.*

Proof: The EXPLORE function will continue to expand nodes until all descendants of the start state have been expanded, which by definition of a finite solvable state space must include a goal node. Once a goal is generated, the backward search will be seeded with it during the next call to RESEEDENVELOPESEARCH. The agent is always either following a path extracted from the intra-envelope search or transitioning to its best successor as discovered by expanding its current state. Therefore the agent is always within the envelope. The backward search will eventually expand s_{agent} since the domain is finite and the agent is always within the envelope. Thus a path will be extracted to the goal. □

Furthermore, I-ES is also complete in a broad class of directed domains.

Theorem 2 *I-ES is complete in finite directed solvable domains with no dead ends, positive finite edge costs, and positive finite heuristics.*

Proof: We reuse the classic RTA* proof (Korf 1990, Theorem 1): To be incomplete in a finite domain, the agent must forever traverse a finite set of states S that does not include a goal. (We note that this implies resetting the envelope frontier an infinite number of times, as at least two nodes will need to be chosen as targets an infinite number of times.) The h value of every node expanded by EXPLORE is set to a value higher than the minimum of its successors, due to the positive action costs. Therefore, a minimum h value in S will iteratively be raised until a successor outside S appears more attractive than any node in S, causing the agent to leave S and implying a contradiction. □

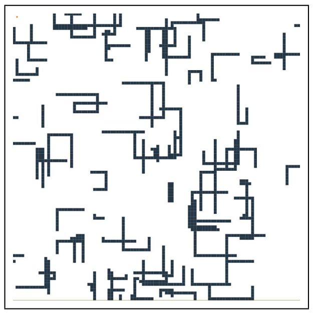
Theorem 3 *I-ES terminates in finite unsolvable undirected domains.*

Proof: If OPEN_{fr} empties without finding a goal, EXPLORE will cause the algorithm to terminate. □

Ordering the Open Lists

I-ES and its theorems are agnostic to the ordering of the open lists. We explored several priority functions including Greedy Best First on h for all open lists, weighted f for OPEN_{bk}, and a weighted f-like function for OPEN_{fr} calculated as $f(s) = w * h(s) + h(s, s_{agent})$ where s_{agent} is the agent's current state. This function allows us to approximate g without having to maintain a rigid graph. Since each open list could theoretically be ordered using a different priority function, the number of configurations possible is combinatorial in the number of priority functions considered. For experimental comparison with TBA*, we used a Greedy Best-First priority function for OPEN_{fr} and either a greedy or a weighted f function for OPEN_{bk}. The weight parameter reported in experiments applies only to the backward open list. In the backward search, $g(s) = c(s, s_{target})$.

These configurations were identified as the strongest performers in the domains considered; we omit plots and detailed analysis of the other configurations for brevity.



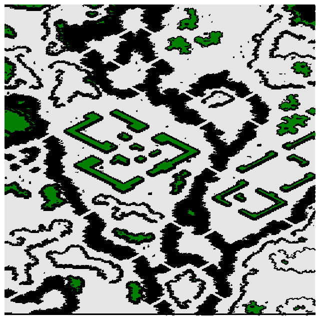


Figure 2: left: 100 $\times$ 100 Minima map; right: Starcraft Cauldron map

Time Complexity

As a real-time algorithm, we analyze time complexity with respect to per-iteration guarantees. All operations are either O(1) or otherwise bounded by the constant $bound$ except where noted below.

The open lists are implemented for our experimental analysis using binary heaps. Insertion time into a binary heap takes log time in the size of the heap, which is not bounded by a constant. Data structures exist for constant time insert, such as the Fibonacci Heap, but were not used for ease of implementation.

The EXTRACTPATH function is worst-case linear in the number of states generated so far since it is possible that all states generated are on the extracted path. Therefore, EXTRACTPATH is not bounded by a constant. It is possible to iteratively construct the path over mutliple search iterations in a manner similar to TBA*, however, we decided not to focus on those details since following pointers is a relatively cheap operation. Profiling of our implementation revealed that our traceback operations take approximately 200ns as opposed to our expansion operations which take approximately 4000ns, and furthermore our tracebacks account for only 2% of total algorithm runtime. For fair comparison with TBA* and its variants, we set the TBA* parameter $tracebackLimit = \infty$ allowing it to also extract paths unbounded by constants.

$pathCache$ was implemented as a linked list which allows constant time insert, delete, and append for both the start and end of the list.

Experimental Results

We tested variants of TBA* and I-ES on: 1) the 100 15-puzzles of Korf (1985) using Manhattan distance (MD); 2) a deterministic version of the racetrack game (Barto, Bradtke, and Singh 1995) in which a collision brings the velocity to zero, using MD divided by the maximum achievable speed and a version of the original Barto racetrack that is scaled and extended to be 414 x 66; 3) grid pathfinding using four-way movement and MD in a) the Starcraft cauldron map (Sturtevant 2012) and b) 1500 x 1500 grids with large randomly-generated minima, referred to as the Minima domain (Figure 2 shows example maps).

Each experiment was given 7GB RAM and 5 minutes. Real-time bounds were specified using limits on the number of expanded nodes and the CPU time of every iteration before the goal was found was recorded. To compensate for not using a real-time OS, we used the 99th percentile of the per-iteration CPU times, which was much more stable than the maximum.

We present the most successful variants of TBA* and I-ES, with LSS-LRTA* reported as a baseline. For this analysis, the "Greedy" I-ES variant uses GBFS for both frontier and intra-envelope expansion, whereas the "Weighted" variant uses GBFS for frontier expansion and Weighted A* for intra-envelope expansions. A weight of 3.0 was selected as a high-performing weight in these domains and compared against greedy variants.

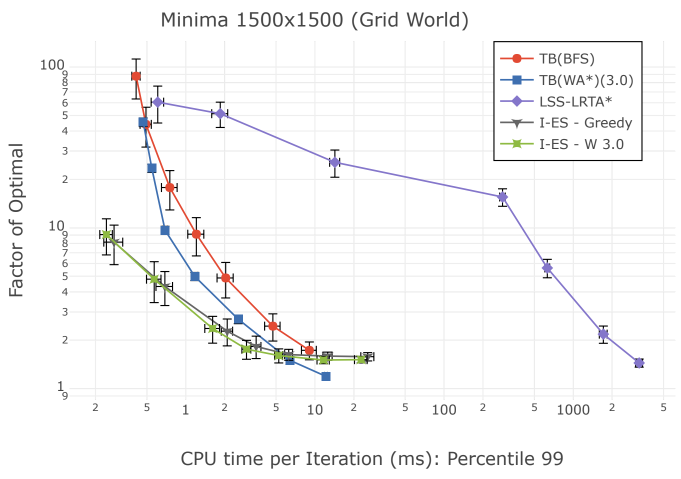
Figure 3 presents the results. The x-axis represents the mean over the test instances of the 99th percentile of the per-iteration CPU time, with error bars representing 95% confidence intervals. The y-axis represents the mean agent trajectory cost, expressed as a factor of A*'s off-line optimal solution, again with confidence intervals. Figures 3a and 3b show that, in grid pathfinding, I-ES is significantly superior to the other tested algorithms for lower time bounds. Above 15 ms/iteration, TB(WA*) become slightly superior. Note that even though both TB(BFS) and I-ES expand their frontiers on a greedy best-first strategy, I-ES has much better performance, suggesting the shortcutting inherent to I-ES makes a crucial contribution in these highly-connected domains.

The Sliding Tile experiments (Figure 3c) offer an interesting contrast. I-ES is somewhat competitive with the TBA* variants at low iteration durations, but TB(WA*) becomes much better at higher lookaheads. This suggests that in this domain the ability of TB(BFS) to find a suitable path quickly is more important than the ability to switch between paths opportunistically. This is evidenced by the trajectories of the I-ES and TB(BFS) plots, which are very similar. Recall that both I-ES and TB(BFS) expand their frontiers greedily, but that I-ES spends computation time searching within the envelope. The time spent on intra-envelope search reduces the effectiveness of the greedy expansion strategy, though it is still able to outperform TB(WA*) at low durations even without a rigid internal graph structure.

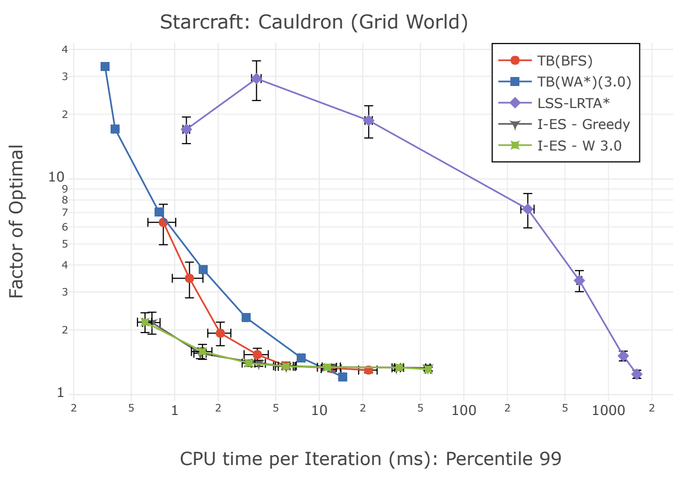
Racetrack (Figure 3d) represents a domain where I-ES is not as competitive, though it does ultimately converge with TBA* in solution quality. We speculate that this is because connections between branches in the search tree are difficult to discover. The distance with respect to node transitions between two states which share a common location but vary in speed may be large, and so discovering these connections takes more computation time than in other domains.

Discussion

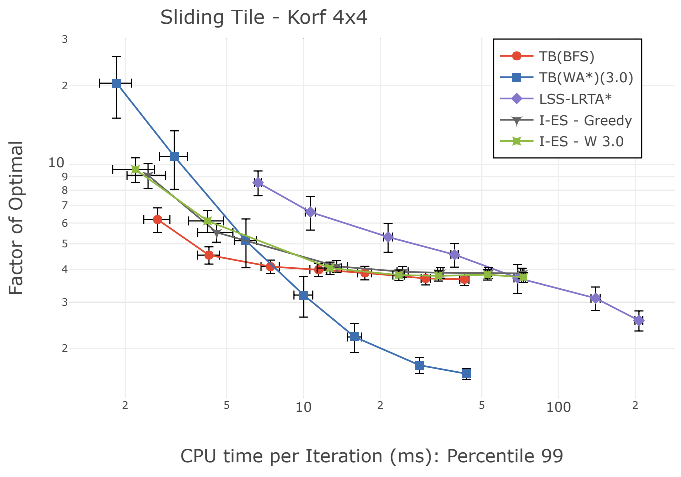
Intra-Envelope Search techniques have clear advantages in certain domains. They exhibit the crucial behavior of being able to escape heuristic depressions without scrubbing as evidenced by the high performance of I-ES in the Minima domain. When compared with the leading real-time



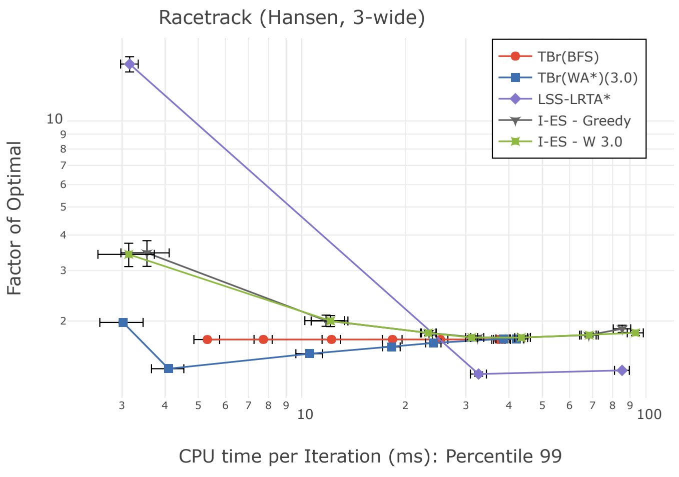
(a)



(b)



(c)



(d)

Figure 3: Solution cost as a function of the real-time bound for the best-performing variants of TBA* and I-ES.

search algorithm for these domains, TBA* and variants, I-ES achieves state-of-the-art performance. By anchoring neither the search nor the path extraction in the increasingly less relevant root node, I-ES is very effective at navigating dense graphs via shortcuts when new target nodes are selected.

The performance of I-ES in grid pathfinding does not translate as easily to the Sliding Tile and Racetrack domains. This characteristic is a result of the fact that I-ES spends computation on its extra intra-envelope search instead of continuing to expand the frontier with all available time. In grids, this split search pays dividends in better path quality because shortcuts between target nodes are discovered. This is in contrast to TBA* and its variants which restrict themselves to their pre-constructed graphs. In Racetrack and Sliding Tile, the state space is not as interconnected and so the extra time spent trying to find shortcuts ultimately proves less effective than simply following pointers of a rigidly defined search tree.

We experimented with seeding OPEN_{bk} with the entire search frontier. This resulted in the agent being drawn toward frontier nodes which happened to be near it rather than the goal. We also experimented with replacing the backward envelope search with a naive bidirectional search, however, this did not improve performance. Incorporating more recent work on bidirectional search (Holte et al. 2015; Shperberg et al. 2019) could be a promising future direction.

I-ES could be extended into dynamic domains where edge costs change over time. For example, the forward frontier search could be replaced with a version of Lifelong Planning A* (Koenig and Likhachev 2001), or the backward search could be replaced with a version of D* Lite (Koenig and Likhachev 2002).

Although the LSS and ES paradigms are the most popular in real-time search, there are others. For example, the FRIT algorithm (Rivera et al. 2014) does not rely on LSS expansion as the only means to guide the agent. Instead, a relaxed search graph (i.e. one with no obstacles) is constructed and is then updated as the agent discovers that assumed edges do not exist in the true search graph. This approach shows improvements when compared against other LSS algorithms, but still restricts its learning to areas local to the agent and is not limited to expanding states at most once.

Real-Time D* (Bond et al. 2010) uses both a forward search for action selection and a backward search to connect the agent with the goal. However, the backward search limits the algorithm to domains featuring a single goal state. I-ES is able to connect the agent to the frontier in the absence of a discovered goal and can guide the agent toward the closest goal if multiple have been discovered.

Conclusion

We advanced the paradigm of envelope-based real-time heuristic search as a contrast to the more traditional agent-centered approach. We examined TBA*, the previous state-of-the-art envelope-based approach, and presented examples of troublesome problems for its way of guiding the agent towards the promising area of the search frontier. We proposed a new approach, Intra-Envelope Search (I-ES), that conducts an auxiliary search within the main envelope to connect the agent to the frontier. We showed that I-ES is complete in both undirected and directed domains under reasonable assumptions and compared its best configurations with TBA*'s in several search benchmarks. I-ES exhibits superior performance in highly-connected domains like grid pathfinding where shortcuts were easily extracted, although it did not provide any advantage in more sparsely-connected state spaces such as racetrack. Given that the main advantage of envelope-based methods is their avoidance of scrubbing, which appears so irrational to an observer, we hope I-ES's sensible approach to guiding the agent to the frontier will raise the profile of envelope-based search as an effective approach for real-time planning.

References

Barto, A. G.; Bradtke, S. J.; and Singh, S. P. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence* 72(1):81–138.

Björnsson, Y.; Bulitko, V.; and Sturtevant, N. 2009. TBA*: Time-Bounded A*. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09).*

Bond, D. M.; Widger, N. A.; Ruml, W.; and Sun, X. 2010. Real-time search in dynamic worlds. In *Proceedings of the Symposium on Combinatorial Search (SoCS-10).*

Hernández, C., and Baier, J. A. 2012. Avoiding and escaping depressions in real-time heuristic search. *Journal of Artificial Intelligence Research* 43:523–570.

Hernández, C.; Asín, R.; and Baier, J. A. 2014. Time-bounded best-first search. In *Proceedings of the Seventh Annual Symposium on Combinatorial Search (SoCS 2014).*

Hernández, C.; Baier, J. A.; and Asín, R. 2016. Time-bounded best-first search for reversible and non-reversible search graphs. *Journal of Artificial Intelligence Research* 56:547–571.

Holte, R. C.; Felner, A.; Sharon, G.; and Sturtevant, N. R. 2015. Bidirectional search that is guaranteed to meet in the middle. In *AAAI.*

Kiesel, S.; Burns, E.; and Ruml, W. 2015. Achieving goals quickly using real-time search: experimental results in video games. *Journal of Artificial Intelligence Research* 54:123–158.

Koenig, S., and Likhachev, M. 2001. Incremental A*. In *Proceedings of the Neural Information Processing Systems.*

Koenig, S., and Likhachev, M. 2002. D* lite. In *AAAI*, 476–483. American Association for Artificial Intelligence.

Koenig, S., and Likhachev, M. 2006. Real-time adaptive A*. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-06)*, 281–288.

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.

Koenig, S. 2001. Agent-centered search. *AI Magazine* 22(4).

Korf, R. E. 1985. Iterative-deepening-A*: An optimal admissible tree search. In *Proceedings of IJCAI-85*, 1034–1036.

Korf, R. E. 1990. Real-time heuristic search. *Artificial Intelligence* 42:189–211.

Richter, S.; Thayer, J. T.; and Ruml, W. 2010. The joy of forgetting: Faster anytime search via restarting. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS-10).*

Rivera, N.; Illanes, L.; Baier, J. A.; and Hernández, C. 2014. Reconnection with the ideal tree: A new approach to real-time search. *Journal of Artificial Intelligence Research* 50:235–264.

Shperberg, S. S.; Felner, A.; Shimony, S. E.; Sturtevant, N. R.; and Hayoun, A. 2019. Improving bidirectional heuristic search by bounds propagation. In *Twelfth Annual Symposium on Combinatorial Search.*

Sturtevant, N. R., and Bulitko, V. 2016. Scrubbing during learning in real-time heuristic search. *Journal of Artificial Intelligence Research* 57:307–343.

Sturtevant, N. 2012. Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games* 4(2):144–148.