Proceedings of the Sixth International Symposium on Combinatorial Search

Online Detection of Dead States in Real-Time Agent-Centered Search

Guni Sharon
ISE Department
Ben-Gurion University
Israel
gunisharon@gmail.com

Nathan R. Sturtevant
Department of Computer Science
University of Denver
USA
sturtevant@cs.du.edu

Ariel Felner
ISE Department
Ben-Gurion University
Israel
felner@bgu.ac.il

Abstract

In this paper we introduce techniques for state pruning at runtime in *a priori* unknown domains. We describe how to identify states that can be deleted from the state-space when looking for both optimal and suboptimal solutions. We discuss general graphs and special cases like 8-connected grids. Experimental results show a speed up of up to an order of magnitude when applying our techniques on *real-time agent-centered search* problems.

Introduction

Many recent techniques aim to reduce the effective size of a search space. Techniques like the dead-end heuristic (Björnsson and Halldórsson 2006) and Hierarchical Swamps (Pochter *et al.* 2010) mark areas of the state space which can be avoided when searching for optimal paths. Reach (Goldberg *et al.* 2006) builds information about the length of optimal paths through a state and uses it for pruning parts of the state space. All these techniques share two important assumptions: (1) The state space is explicitly given a priori and is stored in memory (2) An off-line preprocessing phase is activated to gather the information that allows the pruning for the main search. The time for the preprocessing is usually omitted as it is done only once and then used for solving a large number of problems.

In this paper we explore cases where these assumptions do not hold. In many cases the search space is not given explicitly, as in an agent moving through unexplored terrain or when the graph is exponentially large. Similarly, preprocessing might not be effective, e.g., if we are only interested in solving a single problem instance. For these scenarios, we introduce the idea of pruning states *on-line*, during the actual search. We are particularly interested in online pruning using the local structure around a state which can be used to prove that a state can safely be deleted from the state-space.

Online pruning incurs an overhead and it should only be applied in scenarios where despite this overhead the pruning is likely to reduce the overall effort. If an algorithm is guaranteed to expand each state at most once (e.g., A* with a consistent heuristic) then online pruning as we describe here will likely provide little or no benefit. But, there are many scenarios where agents may revisit states many times and, for these cases, our pruning methods can be very effective.

Our focus here is the scenario of *Real-time agent-centered search*. In this scenario an agent has limited power of computation and only local perception of the environment. As such, *learning* heuristic values from local neighbors must be used to reach the goal. In areas where large heuristic errors exist, agents must revisit states many times. In such cases, the number of times a given state is revisited is potentially polynomial in the state space (Koenig 1992), and in practice there is significant room for improvement[1]. We introduce two types of states that may be pruned from the state-space:
(1) Expendable: States that are not necessary in order to reach the goal. These are effective when the agent is seeking to reach the goal, not necessarily via an optimal path.
(2) Swamps: States that are not part of the optimal path. These are important when the optimal path should be found.

We present methods for detecting and pruning swamps and expendable states in general graphs and in 8-connected grids. Experimental results show a speed up of up to an order of magnitude.

Real-Time Agent-Centered Search

Real-Time Agent-Centered Search (RTACS) is defined by an undirected graph G and *start* and *goal* states denoted as s and g. The agent is located in s and its task is to arrive at g. We assume that we have an admissible heuristic $h(n) \leq cost(n, g)$. As a *real-time* problem, the agent can only perform a constant-bounded amount of computation before it must act by following an edge from its current state. Then, a new *search-act* cycle begins from its new position. As an *agent-centered* problem, the agent is constrained to only sense and reason about the states which are local to the agent; these are usually assumed to be contiguous around the agent. We also assume that we are only allowed to write a limited (constant) amount of information into each state (i.e., an estimate of the distance to the goal). In this way RTACS is an example of an 'ant' algorithm, with limited

[1]Other examples where online pruning may help are: *Search with inconsistent heuristics*, which can require polynomial number of re-expansions (Mero 1984; Felner *et al.* 2011), and *Search in unknown environments*, where algorithms like D*-lite (Koenig and Likhachev 2002) re-plan as obstacles are discovered.

Algorithm 1: LRTA*(1)

```
Input: Vertex $start$, Vertex $goal$
1  $v_{current} = start$
2  while $v_{current} \neq goal$ do
3      $v_{current}.h = \infty$
4      foreach (Vertex $v_n$ in neighbors($v_{current}$)) do
5          if ($v_{current}.h > v_n.h + cost(v_{current}, v_n)$)
           then
6              $bestNeighbor = s_n$
7              $v_{current}.h = v_n.h + cost(v_{current}, v_n)$
8      // optional - online pruning
9      if (ShouldKill($v_{current}$)) then
10         Kill($v_{current}$)
11     $v_{current} = bestNeighbor$   //physical move
```

computation and memory (Shiloni *et al.* 2009).

Work on RTACS is quite diverse. Published work comes from a range of applications with diverse evaluation metrics. The original research in this area was used to suboptimally solve sliding-tile puzzle (Korf 1990) instances. Other work has modeled this as a robotics problem (Koenig 2001), or as a path planning problem in video games (Bulitko *et al.* 2008). Common metrics for evaluation are:

1: Minimize travel distance: This is relevant when the time of physical movement of the agent (between states) dominates the computational time done by the CPU.

2: Minimize computational time: This is relevant when the CPU time dominates the time of the physical movement of the agent. CPU time can be measured exactly or approximated by counting the number of visited states.

Koenig (2004), for example, uses these and other variants.

Previous Work on RTACS

The dominant approach to RTACS is based on *learning heuristic distances from neighbors* – learning an accurate estimate of the distance to the goal using a form of the Bellman update rule between neighbors. The baseline algorithm for this is the LRTA* algorithm (Korf 1990) illustrated in Algorithm 1. Lines (8-10) are optional; we add them to support our new ideas. In each loop, the agent updates the heuristic of the current state using the distance to a neighboring state along with the h-cost of that neighbor (lines 4-7). It then moves to the best neighbor (line 11).

We illustrate LRTA* in Figure 1. The numbers in parenthesis denote the initial heuristic value for that vertex. The agent starts at S and its task is to reach G. LRTA* starts by updating the heuristic of S to 1 before moving to vertex B, its most promising neighbor. In B the heuristic is updated to 2, and then the agent moves back to S. The agent then moves again to B, updating the heuristic of S to be 3. This illustrates one drawback of this approach – an agent may visit a state many times during learning. In the worst case, an agent may perform $O(n^2)$ moves before finding the goal in a state space with n states (Koenig 1992).

In a simple implementation of LRTA* the agent only performs a *1-step lookahead* into the environment. Variants of LRTA* follow the same framework of acting and then learning a heuristic, but vary the size of the lookahead, the shape of the lookahead, the movement rule, and other parameters (Bulitko and Lee 2006). Representative examples include LSS-LRTA* (Koenig and Sun 2009), which performs an A*-shaped lookahead, and daLSS-LRTA* (Hernández and Baier 2012), which prioritize moving to states where least learning occurred in. By doing so it can avoid revisiting states belonging to a local heuristic depression many times. A new approach is f-LRTA* (Sturtevant and Bulitko 2011) which *learns* both the heuristic to the goal (h) and the distance from the start state (g). f-LRTA* uses g values both alone and in combination with h values to prune states from the state space.

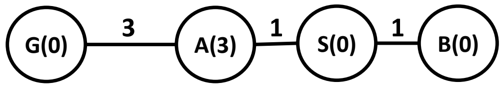


Figure 1: Example graph

It is important to note that due to the real-time nature of interleaving of planning and acting, the path followed by the agent is not necessarily optimal and an optimal path is usually not known even when the agent reaches the goal. We split these algorithms into two phases. The *first phase* is getting the agent to the goal as fast as possible (as defined by the evaluation metrics). Some work focuses only on this (Hernández and Baier 2011). However, as already noted by (Korf 1990), when solving the same problem *repeatedly* from the start to the goal, the paths traversed by the agent in the remaining runs converge to the optimal solution. We call this the *convergence phase*. Other work focuses on this (Sturtevant and Bulitko 2011) as well.

We would like to stress that our online pruning approaches are based only on the local structure of the state space and are thus applicable for all the algorithms and evaluating metrics of RTACS. Additionally, we distinguish special pruning rules for the first phase in the form of *expendable states* and for the convergence phase in the form of *swamps states* to preserve the optimal solution.

Previous online pruning techniques.

(Pochter *et al.* 2011) explored on-line pruning using symmetries that arise from the logical state-space descriptions with application to implicit exponential domains. (Sturtevant *et al.* 2010) introduced the notion of *redundant states* and of *dead ends* for g-cost learning algorithms. If a state can be reached from several different states by the same g-cost, only one of these is necessary for finding the optimal path. If a state is *redundant* with respect to all of its successors, it can be pruned without changing the length of the optimal path between the start and the goal. If all neighbors of a given state s are closer to the start location than s, then s is a *dead end*. *Dead ends* cannot be on any of the optimal paths to the goal and thus can also be pruned. f-LRTA* uses these and other concepts for pruning. Online redundant states pruning is also applied in the *jump-point search*

(JPS) algorithm (Harabor and Grastien 2011). JPS is an enhancement of the A* algorithm with application limited to grid worlds. JPS is not for real-time search nor is it agent-centered, and adapting it to meet these restrictions is non-trivial. JPS uses special rules to find states that are redundant with respect to the current start and goal, so the work here is in the same vein as JPS. The pruning techniques in this paper can be applied in addition to the existing pruning techniques in all RTACS problems and on general undirected graphs.

Expendable States

We now define the online pruning techniques, starting with expendable states.

Definition 1 Expendable state: *State e is* expendable *if excluding e and marking it as dead will not affect the* reachability *of any pair of non-expendable states. More formally, a state e is expendable, if for every pair of states x and y, there exists a path from x to y that does not pass through e.*

An equivalent definition is to focus only on the states that are neighbors of e (labeled $N(e)$) as follows:

Definition 2 Expendable state: *State e is expendable if for every pair of states $x \in N(e)$ and $y \in N(e)$, there exists a path from x to y that does not pass through e.*

Lemma 1 *The two definitions are equivalent.*

Proof: Direction 1 Definition 1 implies Definition 2: Neighbors are a special case of two general states.

Direction 2 Definition 2 implies Definition 1: Let $P = \{p_1, p_2, ..., p_l\}$ be path of length l between two given states p_1 and p_l that passes through state e where $e \neq p_1$ and $e \neq p_l$. Let i be the position of e in P. Since p passes through e it must also pass through two neighbors of e, p_{i-1} and p_{i+1}. Since all neighbors of e are reachable from one another not through e as per definition 2, there exists a path between p_{i-1} and p_{i+1} that does not pass through e. Thus, there is a valid path between p_1 and p_l that does not pass through e. □

Removing expendable states from the graph will likely reduce the search effort for reaching the goal, especially when such states would have been revisited many times without the pruning process.

The expandable attribute of a state, as defined above, depends only on the graph and does not depend on a specific start and goal state. Given a specific pair of start and goal states s, g, one can remove every expandable state and still find a valid solution. There is only one exception for this: s and g cannot be removed even if they are expandable.

Online detection of expendable states

Pruning can be added to LRTA* in Algorithm 1 as follows: Before moving out of state v, (line 11 of Algorithm 1) we introduce a *should-kill(v)* function (line 9). If *true* was returned, we remove the state and its connecting edges from the graph so that v will never be revisited. The straightforward implementation for *should-kill()* would be to run a breadth-first search from one of the neighbors and check whether all other neighbors are reachable. This has two

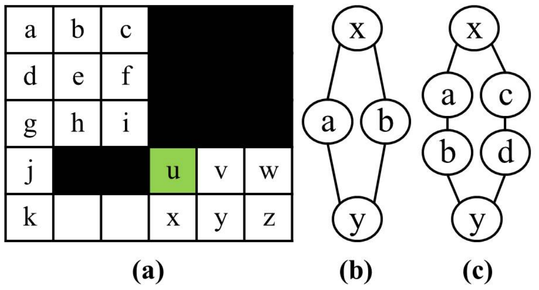


Figure 2: (a) Example 8-connected graph. (b,c) Cases where simple rules will not detect swamps.

shortcomings. First, in the worst case, this check will involve visiting the entire graph. Second, in RTACS we assume that we can only sense the local neighborhood. Thus, a complete breadth-first search might not be possible. We thus give a more restrictive definition of expendable states.

Definition 3 Locally Expendable State: *State e is* locally expendable*, if for every pair of states $x \in N(e)$ and $y \in N(e)$ there exists a path from x to y that passes only through other neighbors of e.*

This can be computed more efficiently within the sensing radius; a breadth-first search in $N(e)$ is performed to see that all states in $N(e)$ are reachable without going through e. Definition 3 is restrictive and only uses 1-step lookahead for pruning. Therefore, a variety of expendable states according to Definitions 1 and 2 will not be declared as locally expendable by Definition 3. This will occur if an alternative path (that does not go through e) exists, but it includes states which are not neighbors of e (i.e., outside $N(e)$).

Our experimental results show that our restrictive definition of *locally expendable* is simple to implement and quite effective in practice. We leave the treatment of the larger lookaheads for future work and hereafter, we only refer to *locally expendable* states.

Expendable States in 8-Connected Grids

Our primary example domain is an 8-connected grid, which is common in real world environments such as robotics and games (Thrun 2003; Sturtevant 2007), and is similar to other models for terrain and discretization of free space. While we present general methods for pruning we will also provide special cases techniques for 8-connected grids.

An 8-connected grid is shown in Figure 2(a). States in the grid can either be blocked (black) or unblocked (white). An agent can move one square in any of the four cardinal directions given that the destination location is unblocked. In our variant of the 8-connected grid, an agent can move diagonally to an adjacent square only if *both* of the cardinal directions are free. For example, an agent in location j or i cannot move SE, and an agent at h cannot move SW.

For the special case of an 8-connected grid, the detection for expendable states is simple and inexpensive: a state is *expendable* if all its reachable neighbors are contiguous. We test a state by going through its neighbors in a clock-wise manner and verify that its reachable neighbors are contiguous. For example for state e in Figure 2(a) we iterate through (a, b, c, f, i, h, g, d). Since they are all reachable then e is expendable. State i only has three reachable neighbors (e, f, and h) but since they form a contiguous block i is also expendable. State j has g and k as reachable neighbors. Since they are not contiguous j is not expendable.

The order by which checks for expendable states are performed significantly influences the results. If in Figure 2 (a) we start by checking state a, we see that it is expendable and remove it from the state space. Continuing in alphabetical order from b up to state f followed by i, h, and g, we detect that all of them are expendable. Now, assume that state e is checked first. Since it is expendable, it will be removed. Then, no other neighbor of e is expendable because every state is a neighbor of e and is also adjacent to a blocked cell (i.e., it has other, not connected, blocked neighbors). Thus not all unblocked neighbors are connected to each other. In the first ordering 9 states are pruned; in the latter one, only one state is pruned.

As our aim is to prune as many states as possible, we added the rule that if all 8 neighbors of a state are unblocked it is not checked nor marked as expendable. Thus, practically, our search proceeds until it hits a wall and only then starts to prune expendable nodes. For general graphs, pruning is considered only when learning updates the heuristic estimation of the current node; this corresponds to hitting an obstacle in grids.

Online Swamps

If we are interested in preserving shortest paths within the environment, we need to define a much more restrictive pruning rule in order to not prune states that lie on the optimal paths (even if they are expendable). We do this by using the concept of *swamps* introduced by Pochter et al. (2009; 2010). This is useful for the *convergence* phase (described above) of algorithms from the LRTA* family.

A *swamp* S is a set of states so that if S is removed from the original graph, the length of the shortest path between every two states in the remaining graph is equal to the shortest path in the original graph. In other words excluding the states of a swamp will not affect the length of any of the shortest paths that start and end outside the swamp. In prior work, swamps were detected offline in a preprocessing phase.

The formal definition, taken from (Pochter *et al.* 2010) is:

Definition 4 Swamp: *A swamp S in a graph $G = (V, E)$ is a set of states $S \subseteq V$ such that for each $v_1, v_2 \in V \backslash S$, there exists a shortest path $P_{1,2}$ that connects v_1 and v_2 and traverses only through nodes in $V \backslash S$.*

Online Swamp Detection

Online detection of swamps in general graphs can be done as follows. **General swamp detection:** For a given state s, compute the shortest path between all the neighbors of the state with and without s in the graph. If the shortest path distances do not change when s is removed from the graph and s is not the start or the goal states then s is a *swamp* and can be removed. Again, we face the same two problems described above when discussing expendable states. (1) This may be an expensive online computation and future gains from removing the state do not necessarily outweigh the costs of testing whether the state can be removed. (2) we only assume local sensing capabilities.

Define the *local neighborhood* of s as $LN(s) = N(s) \cup s$. A more efficient rule, but one that is less general, is to again restrict the discussion to shortest-paths in $LN(s)$. We thus define a *local swamp* as follows.

Definition 5 Local Swamp: *s is a local swamp s if for each $x, y \in N(s)$, there exists a shortest path in $LN(s)$ that connects x and y but does not pass through s.*

From this definition we get **local swamp detection:** We remove state s only if the lengths of shortest paths in $LN(s)$ between any two states in $N(s)$ are not changed when removing s. The *local swamp* definition is more restrictive than the general swamp definition (when the shortest path is defined over the entire graph) and it fails to catch cases like state a in Figure 2(b). Using local swamp detection, a is on the shortest path between x and y when we are restricted to only pass through $LN(a) = \{a, x, y\}$. Thus, a cannot be pruned. However, according the general swamp detection rule (when the shortest path between x and y can also pass through b), a can be marked as a swamp.

Pochter et al 2010 showed that swamps may include a large number of states. For example, in Figure 2(c) states a and b can be jointly marked as swamps, but not individually. We do not yet have efficient rules for detecting these cases. Our online swamp detection only detects swamps in the form of a single state at a time.[2]

Online swamp detection in 8-connected grids

In grids it is easy to implement the *local swamp detection* rule. As with the check for expendable states, we scan the eight neighbors of a state in clockwise order. If the state is a swamp according to Lemma 2 then prune the state.

Lemma 2 *A state s in an 8-connected grid is a swamp if* **(1)** *s is expendable (its unblocked neighbors are consecutive) and* **(2)** *s has no more than four unblocked neighbors.*

Proof: If state s is not expendable, i.e. the unblocked neighbors of s are nonconsecutive, than the shortest path between at least two neighbors must go through s. This can be seen in state g of Figure 2 (a), where the shortest path between j and d, e, h goes through state g. Thus g cannot be a swamp. If a state s has five or more consecutive unblocked neighbors in a grid, then two of them must be on opposite

[2]Swamps are a special case of Contraction Hierarchies (CH) (Geisberger *et al.* 2008). In CHs any node can be removed from the graph, but the optimal path remains by adding edges to preserve shortest paths that went through any nodes that were removed. Swamps use the CH rule, but only prune states which require no adding of edges for preserving remaining shortest paths.

sides. For example, state f in Figure 2 (a) has 5 neighbors, where c and i are on opposite sides of f. In this case, the shortest path between c and i must go through f, and f cannot be a swamp. It is easy to verify this for all combinations of five or more consecutive neighbors.

Because of our diagonal movement rule, a state can only have four neighbors if it originally had five neighbors and one of them was marked as a swamp, which is illustrated by state v in Figure 2 (a) (if u was an obstacle, x would not be a neighbor of v). When a state s only has four consecutive neighbors (or less), there will always be an optimal path between all pairs of neighbors which does not go through s. For state v the path w, y, x is an alternate to going through v. This can be verified manually by trying all possible combinations of four or fewer neighbors. □

As just suggested, additional pruning of states may be possible after their neighbors are marked as swamps. State b in Figure 2 (a) can only be marked as a swamp only after a or c have been marked as swamps.

Expendable vs Swamps

Every swamp is also an expendable state but not every expendable state is a swamp. Therefore, pruning expendable states will usually result in a smaller state space then pruning only swamps. However, if the optimal solution is sought one cannot prune expendable states as they may lay on the optimal path. In these cases one should only mark swamps as dead states. This is demonstrated in the left side of Figure 3 which presents runs of the first phase of LRTA* on the grid of Figure 3(I) where white cells are unblocked, black are blocked and S and G are the start and goal. In Figure 3(II) swamp states were marked as dead (gray) and in Figure 3(III) expendable states were marked as dead. Clearly, more expendable states were detected compared to swamps. However, many of the expendable states block the optimal path to the goal. For example, in the optimal solution the agent would traverse diagonally from S to the left bottom corner. Hence, killing expendable states will prevent the algorithm from converging to the optimal solution. For the single run presented in these figures, plain LRTA* without our state pruning traversed a total distance of 1,055. With swamps pruning it was reduced to 1,000 while with expendable pruning a total distance of 709 was traversed.

Experimental Results

The aim of our experiments is to show that online pruning helps a wide range of algorithms. We illustrate the difference between the domain-specific pruning (with the domain-specific ordering) rules for 8-connected grid and generic pruning (with generic ordering) between the first and convergence phases, and between different map types.

To demonstrate the effect of using online pruning we implemented five RTACS algorithms: LRTA*, daLRTA*, f-LRTA*, LSS-LRTA* and daLSS-LRTA*. We experimented with benchmark maps and problems from Sturtevant's repository (Sturtevant 2012). The maps were set to be either 8 connected or 16 connected. The legal movement in a 16-connected grid is illustrated in Figure 3 right. We first provide results for the *first phase* (a single run in a previously unexplored map) and then for the *convergence phase* (multiple runs solving the same problem). We measured both the *distance traveled* and the *time elapsed*. In some cases we also provide the number of states expanded (the states that were examined during the lookahead phase). If the lookahead is set to one, the amount of expanded states, naturally, highly correlates to the distance traveled. We observed that even when the lookahead is greater than one (we label the lookahead in parenthesis) there is still a high correlation between expanded states and distance traveled, so in most cases we omit this data and present only the distance.

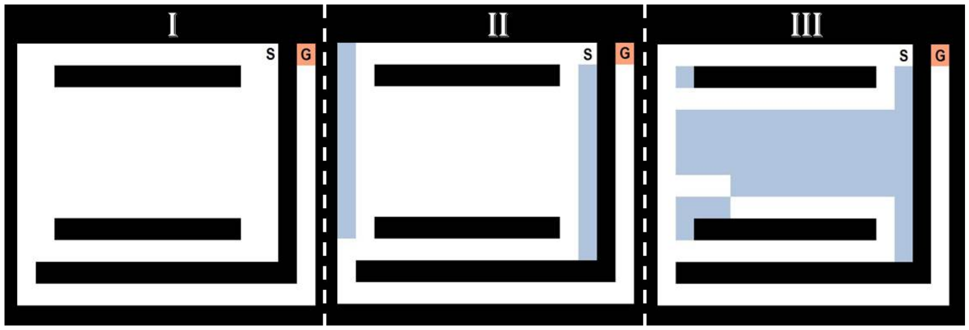
First phase

We experimented with the first phase of the different algorithms on the 1,302 instances of the *Dragon Age Origins* maps (DAO) from buckets 64 and 128 of (Sturtevant 2012). Unless stated otherwise, in all our plots the x-axis is ordered in increasing order of difficulty of the problems. The y-axis (in logarithmic scale) corresponds to the evaluation method.

Figure 4 presents *distance* results for LRTA*, daLRTA*, FLRTA*, LSS-LRTA*(10) and daLSS-LRTA*(10) in an 8-connected environment. For each of the algorithms three curves are shown: blue (solid line) - the original algorithm, red (dotted line) - using online swamp pruning, green (dashed line) - using online expendable states pruning. As can be seen, swamp detection reduced the travel distance and expendable state pruning reduces the travel distance even more dramatically. Gains are up to an order of magnitude.

There was one exception. In daLSS-LRTA*(10) no gains were obtained when online pruning was applied. The explanation is as follows. daLRTA* was designed to escape local heuristic depressions and dead ends. It does so by giving preference for exploration of the graph over exploitation of potentially shorter paths. In other words, daLRTA* directs the agent mainly towards unexplored areas of the graph. While in most cases daLRTA* significantly outperforms other algorithms, we observed that it is less effective in maps with many bottlenecks because daLRTA* may cause the agent to leave the bottleneck on the side away from the goal and to fully explore that area before returning to the bottleneck and passing through it towards the goal. Our pruning techniques narrow the map, causing more bottlenecks and worsening the performance of daLRTA*. Thus, with larger lookahead, the pruning techniques with LSS-LRTA* provide better performance than daLRTA*.

As the lookahead of any RTACS algorithm grows, more knowledge about the graph is found in the lookahead phase and the effectiveness of online pruning diminishes. In the extreme case, the lookahead performs a full A* search and no gains can be obtained from online pruning. This tendency is shown in Figure 5. The figure presents average distance traveled for the LSS-LRTA* algorithm over all the DAO scenarios. The x-axis is the lookahead depth in log scale. The y-axis is the average distance. With smaller lookaheads there is a clear advantage for using swamps pruning over no pruning at all. Using expandable pruning outperforms both swamps and no pruning. As the lookahead is increased, the benefit of using online pruning diminishes, although we



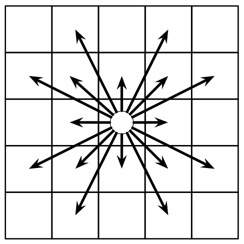


Figure 3: Left: (I) the initial grid (II) Detection of swamps only (III) Detection of Expendable states. Right: 16-connected grid

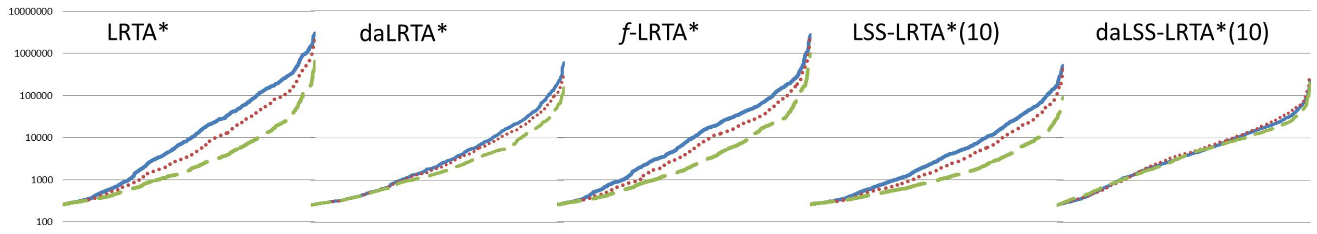


Figure 4: Distance for the first phase. Blue: no pruning. Red: swamp pruning. Green: expendable pruning

presume this is also relative to the size of the problems and maps being solved.

Table 1 presents average values over the same 1,302 instances. It shows distance traveled (Distance) and number of expanded states (Expansions) for an 8-connected grid. CPU runtime (in milliseconds) has two columns for the 8-connected gird. The first (Time) is for the *general* pruning method designed for arbitrary graphs but applied on the 8-connected grid. The second (Time*) is for the special methods designed for the 8-connected grids. Average expansions per millisecond are provided for both the general (E/T) and specific pruning (E/T*) methods. Then, results are provided for the same problems as 16-connected grids. The $+S$ line refers to using the same algorithm but with swamp pruning; $+E$ is similar for expendable pruning.

In general, online pruning incurs overhead. However, we observed that for some cases the constant time per node is reduced when pruning methods were applied. When using online pruning, many open spaces in the map are reduced to smaller more narrow spaces. This in turn reduces the effective branching factor (i.e. average number of neighbors per state) and the cost of learning in each move.

Distance gains were achieved with our pruning methods (with one exception - swamps pruning for daLRTA*); a factor of 2 for swamps and up to an order of magnitude for expendable states was obtained in many cases. As for CPU time, the specific 8-connected expendable pruning method (Time*) achieved a speedup of up to 10x, while swamps pruning achieved little or no gains due to its larger overhead for the *should-kill()* check. Naturally, general pruning methods have much larger overhead per check. Thus the general swamp pruning (Time) always runs slower while expendable pruning is faster only for the basic LRTA* and algorithms with lookahead > 1. This is shown in the algorithms with lookahead of 10 (other lookahead > 1 performed similarly). This is mainly due to the harder problems in the set; Figure 6 demonstrate this trend for LSS-LRTA*(10) with generic pruning method. Runtime is presented for all problems as in table 1 sorted according to difficult.

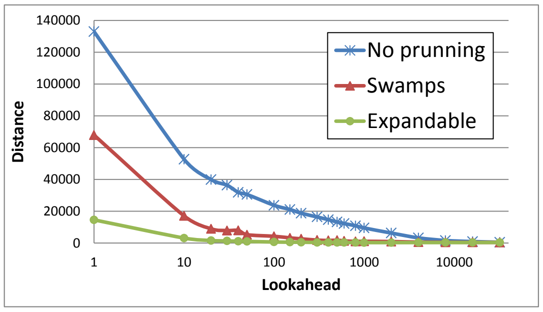


Figure 5: Distance traveled for LSS-LRTA*

For the 16-connected grid there are distances reductions of up to an order of magnitude. Only expendable pruning achieved time speedup (for f-LRTA* and LSS-LRTA*(10)) for general pruning (Time). We also implemented a specific rule for 16-connected grids (Time*) which was very similar to the specific 8-connected grids rules. Similar speedups were obtained (not shown).

Influence of Domain Properties We have also experimented with mazes (with corridors of width 1) and with random maps with 40% obstacles.[3] In mazes, all expendable

[3]We experimented with all the relevant instances from Sturtevant's repository using buckets 50,60,70,80,90,100,110,120,

	8-connected						16-connected	
Algorithm	Distance	Expansions	Time	E / T	Time*	E / T*	Distance	Time
LRTA*	132,982	129,732	198	655	198	655	125,386	307
+S	67,919	65,924	1,296	51	109	605	74,061	4,398
+E	14,588	13,073	97	135	15	872	8,627	310
daLRTA*	22,291	20,921	34	615	34	615	11,792	25
+S	14,788	13,885	288	48	24	579	10,465	646
+E	7,179	6,557	109	60	10	656	9,178	444
f-LRTA*	98,809	87,690	98	895	98	895	212,142	374
+S	71,050	63,042	1,129	56	77	819	139,592	5,634
+E	24,979	22,118	106	209	27	822	11,687	292
LSS(10)	22,696	93,765	281	334	281	334	22,829	589
+S	13,559	54,577	947	58	164	333	14,511	4,015
+E	3,505	6,852	74	93	20	343	2,914	352
daLSS(10)	11,130	81,103	420	193	420	193	5,354	463
+S	13,498	99,151	1,890	52	475	209	5,859	3,068
+E	9,168	28,511	265	108	68	419	7,916	1,243

Table 1: +S: swamp pruning, +E: expendable pruning

	After 10 runs			After 100 runs			Until convergence			
Algorithm	Distance	Expansions	Time	Distance	Expansions	Time	Runs	Distance	Expansions	Time
LRTA*	176,530	172,006	52	299,021	289,201	96	3,270	2,415,471	2,227,165	906
LRTA*+S	92,418	89,472	57	196,432	188,064	126	2,016	1,301,281	1,183,579	856
Ratio	1.91	1.92	0.92	1.52	1.54	0.76	1.62	1.86	1.88	1.06
f-LRTA*	137,866	122,842	140	278,341	248,183	297	413	485,008	430,628	522
f-LRTA*+S	99,697	88,957	135	171,602	153,426	248	226	262,407	234,102	385
Ratio	1.38	1.38	1.04	1.62	1.62	1.20	1.83	1.85	1.84	1.36
LSS(10)	41,709	165,650	524	120,287	418,296	1,341	484	375,768	998,675	3,323
LSS(10)+S	27,832	105,200	366	90,485	282,398	988	312	211,983	506,201	1,816
Ratio	1.50	1.57	1.43	1.33	1.48	1.36	1.55	1.77	1.97	1.83

Table 2: 8-connected DAO maps. Convergence phase.

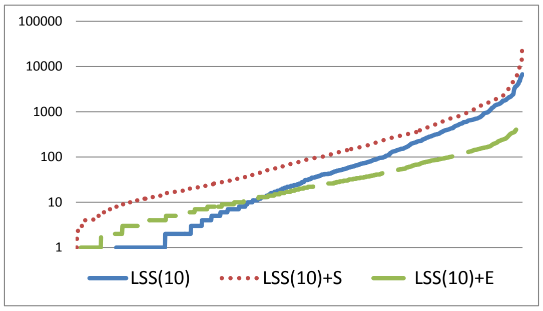


Figure 6: Runtime in ms for LSS-LRTA*(10)

states are also swamps. Pruning for swamps and expendable states performs equally well; for both cases up to an order of magnitude gains were obtained over not using any pruning. For the random maps the gains achieved by the pruning (up to x2) were smaller than mazes and the DAO maps, as open spaces are very rare and very few states can be pruned.

Convergence phase

Results are presented only for swamps for the convergence phase, as expendable states prevent convergence to optimal solutions. daLSS-LRTA*, which is only designed for the first phase has similar results to LSS-LRTA*, and is omitted. The same 8-connected DAO maps were tested and only the specific 8-connected swamp detection was used. Table 2 presents the average distance (Distance), states expanded (Expansions) and runtime (Time) for each algorithm after 10 runs, 100 runs and all runs until convergence. The *runs* column presents the number of runs performed until convergence to optimality was proven. The *ratio* rows present the improvement factor from using swamps. Gains of up to a factor of two were achieved for both metrics. Considerably better results were achieved on mazes, which have a larger proportion of swamps. Random maps, which have very few swamps, had worse results.

130,140,150,160,170,180,190,200 - a total of 1600 instances.

Conclusions

We presented methods for online pruning of *expendable* states and of *swamps* and showed how to implement them in general and in the special case of 8-connected grids. Experimental results showed gains of up to an order of magnitude for the distance traveled across all algorithms we tried. Significant time speedups were mostly seen for the domain specific pruning rules (for 8- and 16- connected grids). This calls for future development of other (non-grid) specific rules. In this paper we only used the restricted *local neighborhood* variants which only include the immediate neighbors of nodes. Future work will explore rules for detecting larger regions of swamps or expendable states, particularly when using algorithms with larger lookaheads.

Acknowledgements

This research was supported by the Israeli Science Foundation (ISF) under grant #305/09 to Ariel Felner.

References

Yngvi Björnsson and Kári Halldórsson. Improved heuristics for optimal path-finding on game maps. In *AIIDE*, pages 9–14, 2006.

Vadim Bulitko and Greg Lee. Learning in real-time search: A unifying framework. *J. Artif. Intell. Res. (JAIR)*, 25:119–157, 2006.

Vadim Bulitko, Mitja Luštrek, Jonathan Schaeffer, Yngvi Björnsson, and Sverrir Sigmundarson. Dynamic Control in Real-Time Heuristic Search. *JAIR*, 32:419 – 452, 2008.

Ariel Felner, Uzi Zahavi, Robert Holte, Jonathan Schaeffer, Nathan R. Sturtevant, and Zhifu Zhang. Inconsistent heuristics in theory and practice. *Artif. Intell.*, 175(9-10):1570–1603, 2011.

Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In Catherine C. McGeoch, editor, *WEA*, volume 5038 of *Lecture Notes in Computer Science*, pages 319–333. Springer, 2008.

Andrew V. Goldberg, Haim Kaplan, and Renato F. Werneck. Reach for A*: Efficient point-to-point shortest path algorithms. In *Workshop on algorithm engineering and experiments*, pages 129–143, 2006.

Daniel Damir Harabor and Alban Grastien. Online graph pruning for pathfinding on grid maps. In Wolfram Burgard and Dan Roth, editors, *AAAI*. AAAI Press, 2011.

Carlos Hernández and Jorge A. Baier. Real-time heuristic search with depression avoidance. In *IJCAI*, pages 578–583, 2011.

Carlos Hernández and Jorge A. Baier. Avoiding and escaping depressions in real-time heuristic search. *J. Artif. Intell. Res. (JAIR)*, 43:523–570, 2012.

S. Koenig and M. Likhachev. D* Lite. In *Proceedings of the National Conference on Artificial Intelligence*, pages 476–483, 2002.

Sven Koenig and Xiaoxun Sun. Comparing real-time and incremental heuristic search for real-time situated agents. *Autonomous Agents and Multi-Agent Systems*, 18(3):313–341, 2009.

S. Koenig. The complexity of real-time search. Technical Report CMU–CS–92–145, School of Computer Science, Carnegie Mellon University, Pittsburgh, 1992.

Sven Koenig. Agent-centered search. *AI Magazine*, 22(4):109–132, 2001.

Sven Koenig. A comparison of fast search methods for real-time situated agents. In *In Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume*, pages 864–871, 2004.

Richard E. Korf. Real-time heuristic search. *Artif. Intell.*, 42(2-3):189–211, 1990.

L. Mero. A heuristic search algorithm with modifiable estimate. *Artificial Intelligence*, 23:13–27, 1984.

Nir Pochter, Aviv Zohar, and Jeffrey S. Rosenschein. Using swamps to improve optimal pathfinding. In Carles Sierra, Cristiano Castelfranchi, Keith S. Decker, and Jaime Simão Sichman, editors, *AAMAS (2)*, pages 1163–1164. IFAAMAS, 2009.

Nir Pochter, Aviv Zohar, Jeffrey S. Rosenschein, and Ariel Felner. Search space reduction using swamp hierarchies. In *AAAI*, 2010.

Nir Pochter, Aviv Zohar, and Jeffrey S. Rosenschein. Exploiting problem symmetries in state-based planners. In *AAAI*, 2011.

Asaf Shiloni, Noa Agmon, and Gal A. Kaminka. Of robot ants and elephants. In *AAMAS '09: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, pages 81–88, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems.

N.R. Sturtevant and V. Bulitko. Learning where you are going and from whence you came: h-and g-cost learning in real-time heuristic search. *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 365–370, 2011.

Nathan R. Sturtevant, Vadim Bulitko, and Yngvi Börnsson. On learning in agent-centered search. In *AAMAS*, pages 333 – 340, 2010.

Nathan R. Sturtevant. Memory-efficient abstractions for pathfinding. In *AIIDE*, pages 31–36, 2007.

N. Sturtevant. Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games (to appear)*, 2012.

Sebastian Thrun. Learning occupancy grid maps with forward sensor models. *Auton. Robots*, 15(2):111–127, 2003.