Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence

Exponential Deepening A* for Real-Time Agent-Centered Search

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

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

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

Abstract

In the *Real-Time Agent-Centered Search* (RTACS) problem, an agent has to arrive at a goal location while acting and reasoning in the physical world. Traditionally, RTACS problems are solved by propagating and updating heuristic values of states visited by the agent. In existing RTACS algorithms the agent may revisit each state many times causing the entire procedure to be quadratic in the state space. We study the Iterative Deepening (ID) approach for solving RTACS and introduce *Exponential Deepening A** (EDA*), an RTACS algorithm where the threshold between successive *Depth-First* calls is increased exponentially. EDA* is proven to hold a worst case bound that is linear in the state space. Experimental results supporting this bound are presented and demonstrate up to 10x reduction over existing RTACS solvers wrt distance traveled, states expanded and CPU runtime.

Introduction

The focus of this paper is *real-time agent-centered search* (RTACS). In this scenario an agent has limited computational power and only local perception of the environment. The task of the agent is to reach a given goal state while acting and reasoning in the physical environment. Algorithms for solving RTACS work in repeated *plan-act* cycles. Each planning phase is restricted to the agent's local sensing radius and is time/memory bounded.

Most existing RTACS solvers belong to the LRTA* family (Korf 1990; Koenig and Sun 2009; Hernández and Baier 2012). The core principle in algorithms of this family is that when a state is visited by the agent, its heuristic value is updated through its neighbors. In areas where large heuristic errors exist, agents must *revisit* states many times, potentially linear in the state space per state (Koenig 1992). Consequently, the total number of states visited in the entire procedure is quadratic in the state space.

We define an RTACS algorithm to be *efficient* if the number of *revisits* is of the same order as the number of *first visits*. *Iterative Deepening* (ID) can be used to solve RTACS. In common RTACS domains (e.g., grids) IDA* (Korf 1985), similar to LRTA*, may revisit states many times and may have complexity quadratic in the state space. IDA* is thus *inefficient* according to our definitions.

To tackle the problem of extensive state revisiting we introduce *Exponential Deepening A** (EDA*), a variant of IDA*. Unlike IDA*, where the threshold for the next iteration grows linearly, in EDA* the threshold for the next iteration is multiplied by a constant factor and thus grows exponentially. We prove that in common RTACS domains EDA* is efficient according to our definition, and thus the complexity of EDA* is linear in the size of the state space.

Finally we provide experimental results that validate this theoretical claim. EDA* outperforms existing RTACS solvers, especially in worst-case scenarios, where its linear bound is most important.

RTACS: Background and Definitions

Given a graph G, *start* and *goal* states, and an admissible heuristic to the goal we differentiate between two types of problems and related algorithms: *classic* search problems and RTACS problems. In a classic search problem the task is to search and *return* a full start-to-goal path (in a single planning phase). Later, this path may be followed by a real-world agent (acting phase), but this is beyond the scope of the classic search problem. Traditional search algorithms in their common use (e.g., A*, IDA*) belong to this class.

By contrast, in RTACS, a moving agent is located in *start* and its task is to *physically arrive* at the *goal*. RTACS algorithms perform cycles that include a *planning phase* where search occurs and an *acting phase* where the agent physically moves. Several *plan-act* cycles are performed due to the following restrictive assumptions:

Assumption 1: As a *real-time* problem, the agent can only perform a constant-bounded number of computations before it must *act* by following an edge from its current state. Then, a new *plan-act* cycle begins from its new position.

Assumption 2: The internal memory of the agent is limited. But, agents are allowed to write a small (constant) amount of information into each state (e.g., g- and h- values). In this way RTACS solvers are an example of 'ant' algorithms, with limited computation and memory (Shiloni *et al.* 2009).

Assumption 3: As an *agent-centered* problem, the agent is constrained to only manipulate (i.e., read and write information) states which are in close proximity to it; these are usually assumed to be contiguous around the agent.

An RTACS agent has two types of state visits:

First visit - the current state was never visited previously by the agent. We denote the number of first visits by F.

Revisit - the current state was visited previously by the agent. We denote the number of revisits by R.

Evaluation Metrics

Common metrics for evaluating RTACS algorithms are:
(1) Travel distance: the total distance (sum of edge costs) that the agent moved during the acting phase. This is relevant when the time of physical movement of the agent (between states) dominates the computational CPU time.
(2) Computational time: the sum of CPU runtime over all planning phases; relevant when the CPU time dominates the time of a physical movement. CPU time can be measured precisely or approximated by counting the number of states expanded in the planning phase.[1]

In this paper we also introduce a new metric:
(3) First Visits Ratio (FVR): The ratio between the number of first visits and total visits, defined as $FVR = \frac{F}{F+R}$.

Optimality

We distinguish between two types of optimality:
Optimal-0: Returning the optimal solution. Defined only for *classic* search algorithms. If the search algorithm is guaranteed to return the shortest path (after its single planning phase) we say that it is *Optimal-0*. For example, A*, IDA* and DFBnB are all Optimal-0. However, Optimal-0 is not defined for RTACS algorithms because they do not return paths; their task is to *arrive* at the goal.
Optimal-1: Converging to the optimal solution. Defined for RTACS. In some cases it is assumed that the agent must repeatedly solve the same problem, sharing knowledge across all trials. In these cases, the heuristics can be updated by successive trials until they converge to the true value. Once complete, this will allow the agent to follow the optimal path in all future trials. Algorithms of the LRTA* family are Optimal-1 (Korf 1990).

Learning Real-Time A* (LRTA*)

A basic building-block for RTACS solvers is propagation (*learning*) of h-values and/or g-values between neighboring states. Since the agent is assumed to have no or limited memory, any needed data regarding a state must be stored in the actual state (according to assumption 3 above). The *propagation* procedure (known as the *Bellman equation* (Bellman 1957)) is defined as: $x(v) = min_u(x(u) + cost(v,u))$, where $x(v)$ is the g- or h-value of vertex v, u is a vertex within the lookahead radius of vertex v and $cost(v,u)$ is the cost along the path from v to u.

Many RTACS algorithms belong to the LRTA* family (Korf 1990). Basic LRTA* with *lookahead* of radius 1 is presented in Algorithm 1. In its planning phase the agent performs a *lookahead* search and propagates the heuristic of the current state from its neighbors (lines 4-7). Then, the agent moves (act phase) to the most promising neighbor v_{next} (line 8). LRTA* was proved to be Optimal-1.

A closely related algorithm is Real-time A* (RTA*) (Korf 1990). In RTA*, $h(v)$ is propagated not from v's best neighbor, like in LRTA*, but from its second-best neighbor. This allows escaping from local heuristic depressions faster than LRTA*. But, over multiple trials RTA* may end up with inadmissible heuristic values, so RTA* is not Optimal-1.

Variants of LRTA*/RTA* follow the same plan-act framework but vary the size or shape of the lookahead, the movement rule, or other parameters (Bulitko and Lee 2006). LSS-LRTA* (Koenig and Sun 2009) performs an A*-shaped lookahead in the *local search space* (LSS) near the agent. Larger LSSs yield shorter distance traveled, but require a longer planning phase for each step. daLRTA*/daRTA* (Hernández and Baier 2012), prefers moving to states where the least learning has occurred. By doing so, it may avoid revisiting states belonging to the same local heuristic depression many times. SLA* and f-LRTA* (Sturtevant and Bulitko 2011) are designed to converge to optimal (Optimal-1) in as few trials as possible rather than solving the problem only once (one trial).

In areas where large heuristic errors exist, all LRTA* algorithms may revisit states many times, potentially linear in the state space per state (Koenig 1992). Thus, denoting the size of the state-space by N, while $F = O(N)$, R is $O(N^2)$ in the worst case. Consequently, the total number of states visits ($F + R$) is $O(N^2)$ - quadratic in N.

Algorithm 1: LRTA* with lookahead radius 1

```
   Input: Vertex start, Vertex goal
1  v_current = start
2  while v_current ≠ goal do
3      v_current.h = ∞
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              v_next = v_n
7              v_current.h = v_n.h + cost(v_current, v_n)
8      v_current = v_next   //physical move
```

Algorithm 2: IDA*/RIBS/EDA*

```
   Input: Vertex start, Vertex goal, Int C
1  T = start.h
2  while BDFS(start, goal, T) = FALSE do
3      Case IDA* : T = T + C
4      Case EDA* : T = T × C
```

Iterative-Deepening Algorithms

We now discuss the Iterative-Deepening (ID) framework and its attributes which are relevant for RTACS. Then, we discuss RIBS - the IDA* variant for RTCAS. Finally, we introduce our new algorithm, Exponential-Deepening A*.

ID acts according to the high-level procedure presented in Algorithm 2. T denotes the threshold for a given Bounded DFS (BDFS) iteration where all states with $f \leq T$ will be visited. In IDA*, T is initialized to $h(start)$ (line 1). For the next iteration, T is incremented to the lowest f-value seen in the current iteration that is larger than T. For simplicity,

[1] Koenig (2004), discussed these and other metric variants. Recently, Burns *et al.* (2013) suggested combining the two metrics to one utility function that takes both into account. Working with this metric is left for future work.

we assume that T is incremented by a constant C (line 3). A lower bound for C is the minimal edge cost.

In many domains, multiple paths to the same node exist; these are called *transpositions* or *duplicates*. Transpositions may blow up the search tree to be exponentially larger than the state space. In a 2D, 8-connected grid of radius r there are $O(r^2)$ unique states. But, due to transpositions, the number of states examined is $O(7^r)$. In such cases a mechanism called *duplicate detection and pruning* (DD) may be optionally employed. Since in RTACS the agent is allowed to leave small pieces of information in each state, marking and detecting previously visited states is allowed.

RIBS (Sturtevant *et al.* 2010) is a variant of IDA* for RTACS problems. At the high level RIBS is identical to IDA* and executes Algorithm 2. However, the BDFS iterations are physically performed by a moving agent. As an RTACS algorithm, RIBS stores the index of the current BDFS iteration in each state it visits. Consequently, RIBS can detect and prune duplicates (DD) within any given iteration. In addition, since the threshold T corresponds to the f-value($= g + h$) of states, RIBS stores both g-values and h-values in each state. RIBS is Optimal-1.

Sturtevant et al. (2010) described pruning techniques that can be employed on top of RIBS. These techniques find and mark *dead states* that can be removed from the state space while still maintaining the optimal solution.

Theoretical analysis of ID

For simplicity of discussion, hereafter we make the following assumptions: **(1)** All ID-based RTACS algorithms use DD within a given iteration. **(2)** The heuristic h is admissible and consistent. **(3)** Unit edge costs. **(4)** Fixed branching factor b and fixed dimensionality k for polynomial domains. **(5)** For worst-case analysis we assume $h = 0$ for all states.

For an ID iteration, i, we denote the number of states visited for the first time by F_i and the number of states that are revisited (were visited in previous iterations) by R_i. Note that $\sum F_i = F$ and $\sum R_i = R$ where F and R are the total *first visits* and *revisits* across the entire run as define above.

Theoretical studies of ID usually focus on other quantities: N_{prev}, the total number of states visited in all iterations prior to the last one, and N_{last}, the number of states visited in the last iteration. We now show that $N_{last} = F$ and that $N_{prev} = R$. We denote the last iteration index by n.

Lemma 1 $F = N_{last}$

Proof: States revisited in iteration i are exactly the states visited during iteration $i-1$, i.e., $R_i = F_{i-1} + R_{i-1}$. Note that there are no revisits at the first iteration, i.e., $R_1 = 0$. Now, $N_{last} = F_n + R_n = F_n + F_{(n-1)} + R_{(n-1)} = F_n + F_{(n-1)} + F_{(n-2)} + R_{(n-2)} = ... = \sum_{i=1}^{n} F_i = F$ □

Lemma 2 $R = N_{prev}$

Proof: We again use the fact that $R_i = F_{i-1} + R_{i-1}$. Thus, $N_{prev} = \sum_{i=1}^{n-1} (F_i + R_i) = \sum_{i=1}^{n} R_i = R$ □

Definition: We say that any RTACS algorithm and ID in particular is *efficient* if $F = \theta(R)$ (or equivalently, if $N_{last} = \theta(N_{prev})$ in ID). We break this into two conditions:
Condition 1 - $R = O(F)$, meaning that the order of F is *greater than or equal to* R.
Condition 2 - $F = O(R)$, meaning that the order of R is *greater than or equal to* F.

If $R >> F$ (condition 1 is violated) the agent spends most of the time revisiting previously seen (non-goal) states. Many existing RTACS algorithms (e.g., the LRTA* family) do not satisfy condition 1. In the worst-case, they are quadratic in the state space due to extensive revisits.

If $F >> R$ (condition 2 is violated) the agent might spend too much time in exploring new but irrelevant states.

Lemma 3 *An algorithm has a worst case complexity linear in the size of the state space N iff it satisfies condition 1.*

Proof: Since in the worst case the entire state space will be visited, and each state can be visited for the first time only once, $F = O(N)$. If condition 1 is satisfied then $R = O(F)$. Now, since $F = O(N)$ then $R = O(N)$ too. Thus the complexity of the algorithm $(F + R)$ is also $O(N)$. On the other hand, if $F + R = O(N)$ and since $F = O(N)$, R must also be $O(N)$, so $R = O(F)$ and condition 1 is satisfied. □

Efficiency of IDA*/RIBS

We now discuss the circumstances under which IDA* (or RIBS which performs DD when possible) satisfies both conditions 1 and 2 (and is considered *efficient*). We differentiate between two types of domains:

1. Exponential domains: In exponential domains, usually given implicitly, the number of states at depth d is b^d (exponential in d) where b is the branching factor. At each level there are a factor of b more states. Therefore, in IDA* the total number of states visited in the last iteration is $N_{last} = F = O(b^d)$. The sum of states visited over all prior iterations, $N_{prev} = R$, is also $O(b^d)$ (See (Russell and Norvig 2010), pp 90 for more details). Therefore, $F = \theta(R)$, both conditions 1 and 2 are met, and according to our definition, IDA* is *efficient* in exponential domains.

2. Polynomial domains: In polynomial domains the number of states at radius r from the start state is r^k where k is the *dimension* of the domain. The last iteration visits $N_{last} = F = O(r^k)$ states. However, the total number of states visited prior to the last iteration is (we denote $q = r - 1$): $N_{prev} = R = \sum_{i=1}^{q} i^k > \sum_{i=q/2}^{q} i^k > \sum_{i=q/2}^{q} (\frac{q}{2})^k = \frac{q}{2} \cdot (\frac{q}{2})^k = \frac{q^{k+1}}{2^{k+1}}$

Since 2^{k+1} is a constant and $q = r - 1$, $R = \Omega(r^{k+1})$. Condition 2 is satisfied ($F = O(R)$). But, since $R = \Omega(r^{k+1})$ and $F = O(r^k)$ then $R >> F$ and condition 1 is violated. For example, in a straight line of length d, IDA*/RIBS will visit $O(d)$ first states, but it will revisit $O(d^2)$ states.[2]

[2]to Satisfy Conditions 1 and 2 for varying branching factor or for non-uniform edge costs, the threshold of IDA* should be dynamically changed. Some prediction methods that may determine the next effective threshold exist (Korf *et al.* 2001; Lelis *et al.* 2013; Burns and Ruml 2013; Sarkar *et al.* 1991; Wah and Shang 1994) but they all make specific assumptions, need significant computation and are not directly applicable to RTACS.

Since condition 1 is violated in polynomial domains, the complexity of IDA*/RIBS is larger than the size of the state space (according to Lemma 3). EDA* remedies this and satisfies both conditions 1 and 2 for RTACS problems on polynomial domains such as grids.

Exponentially Deepening A* (EDA*)

EDA* is similar to IDA* in that it performs multiple *cost-bounded depth-first searches* until the solution is found. EDA* only requires one change to the IDA* pseudo code presented in Algorithm 2 (line 4). That is, instead of adding a small constant to the threshold, we multiply the previous threshold by a constant. For example, assuming unit edge costs and $h \equiv 0$, IDA* will perform iterations with thresholds $1, 2, 3..., i$. By contrast, if $C = 2$, the EDA* thresholds will be $2, 4, 8, 16 \ldots, 2^i$. Similar to RIBS, EDA* in RTACS environments performs full DD within a given iteration.

Optimality of EDA*

Lemma 4 *EDA*, as a classic* single planning phase *search, without the RTACS restrictions, is Optimal-0.*

Proof: Let d be the cost of the optimal solution. Let i be the value for which $C^{i-1} < d \leq C^i$. The goal will not be found in iteration $i-1$ as $d > C^{i-1}$. On the other hand, EDA* must find the optimal solution in the i^{th} iteration if it completes that iteration (i.e., the iteration does not halt when the goal is first reached). In iteration i, *all* states with $f \leq C^i$ are visited. In particular, we will see the goal via its optimal solution as $d \leq C^i$.[3]□

Lemma 5 *EDA* when used as a RTACS algorithm is Optimal-1: Let d be the cost of the optimal solution. EDA* will converge to the optimal solution after d trials if it completes the final iteration in each trial.*[4]

Proof: By induction on d: For $d = 1$, the optimal path of length 1 will be discovered in the first trial via the start state. Assume the induction holds for all paths of length $z-1$. Now, consider the optimal path of length z composed of states $s_1, s_2 ... s_z$. The optimal path to state s_{z-1} was found after $z-1$ trials, according to the assumption. State s_{z-1} must be expanded during the last iteration. When s_{z-1} is expanded the optimal path to state s_z will be found, that is, the known shortest path to s_{z-1} plus the connecting edge $E(s_{z-1}, s_z)$. Tracing back from the goal state will reveal the optimal path. □

Efficiency of EDA*

To deal with the efficiency conditions for EDA*, we again distinguish the two types of domains.

1. Exponential domains: Assume that the depth of the goal is $d = C^i + 1$. In this case, the goal will not be found in iteration i, and will instead be found in iteration $i+1$. All states with $f \leq C^{i+1}$ will be visited during the last iteration. There are $N_{last} = F = O(b^{(C^{(i+1)})})$ such states in total. In all previous iterations $N_{prev} = R = \sum_{j=0}^{i} b^{(C^j)} = O(b^{(C^i)})$ states will be visited. In exponential domains EDA* satisfies Condition 1, $R = O(F)$.

EDA*, however, violates Condition 2. Let d be the optimal solution. We say that states with $f > d$ are *surplus* (Felner *et al.* 2012). Since the EDA* threshold may be increased beyond $d = C^i + 1$ up to C^{i+1}, the number of surplus nodes that EDA* will visit is $O(b^{C^{i+1}})$. This is exponentially more than the $b^{(C^i)+1}$ necessary nodes to verify the optimal solutions, i.e., those with $f \leq d$ (which are expanded by A*). Since $R = O(b^{(C^i)+1})$, $R << F$ and condition 2 is violated. Thus, EDA* is not efficient for exponential domains.

2. Polynomial domains:

We assume that in a polynomial domain of dimension k the number of unique states visited by EDA* within a threshold T is $\theta(T^k)$.[5] If the goal is found in iteration i, EDA* will visit $F = (C^i)^k = (C^k)^i = (\hat{C})^i$ states, where $\hat{C} = C^k$ is a constant. In all previous iterations the agent will visit $R = \sum_{j=0}^{i-1}(C^j)^k = \sum_{j=0}^{i-1}(\hat{C}^j) = \theta(\hat{C}^i)$. Consequently, $F = \theta(R)$. EDA* satisfies both conditions 1 and 2. Since EDA* satisfies condition 1, its worst case complexity is linear in the state space, as proven in Lemma 3. Since it satisfies condition 2, the number of surplus nodes visited will not hurt the complexity. As a result, EDA* is considered fully efficient on polynomial domains.

Solver	Exponential ($N = b^d$)				Polynomial ($N = d^k$)			
	Nodes	C1	C2	ef.	Nodes	C1	C2	ef.
IDA*	b^d	+	+	✓	d^{k+1}	-	+	×
EDA*	$b^{C(d-1)}$	+	-	×	$\hat{C}d^k$	+	+	✓

Table 1: IDA* vs EDA*. Ci = condition i. ef = efficent

Table 1 summarizes the total complexities of IDA*/RIBS and EDA* on polynomial and exponential domains and also points which of the conditions they satisfy. Note that for both exponential and polynomial domains EDA* satisfies condition 1 and has a linear worst case complexity.

BDFS in EDA*

The low-level BDFS procedure (when EDA* is used as an RTACS algorithm) is presented in Algorithm 3 and is described next. EDA* writes the following pieces of information in each visited state s:

1: Iteration index (labeled $s.i$): The iteration index of the last BDFS iteration in which this state was visited. If for iteration I and state s, $s.i = I$ then state s is treated as a duplicate state. If, however, $s.i \neq I$ it means that s was never visited during iteration I and EDA* sets $s.i = I$ (Line 6).

2: Best g-value (labeled $s.g$): The shortest known distance from the start state, the g-value, of state s. When a state s_n is examined (among the other neighbors, Line 7) via s, $s_n.g$ is

[3]Strictly speaking EDA* is Optimal-0 only if DD is not applied (e.g., in implicitly given exponential domains), or, when DD is applied (e.g., in explicitly given polynomial domains) if we allow the re-expansion of nodes if they are seen again with smaller g-values.

[4]Note that a *trial* is the entire procedure of solving a given problem while an *iteration* is a single BDFS call within a trial.

[5]This is true in state spaces with no transpositions. When transpositions exist there are pathological cases (with irregular movements of the agent) that require a different treatment and a different proof (omitted here), so this is a simplifying assumption.

Algorithm 3: BDFS for EDA* with DD

Input: State $start$, State $goal$, Threshold T, Iteration I

```
1  start.g = 0
2  s = start
3  while s ≠ goal do
4      if (s = NULL) then
5          return FALSE //Backtracked from root
6      s.i = I
7      s_next = best unvisited neighbor, null if non exist.
8      if (s.g + h(s, goal) > T) OR s_next = null then
9          s = s.prev //Physical move - Backtrack
10         continue //Next while loop
11     s_next.prev = s
12     s = s_next //Physical move
13 return TRUE
```

updated according to $s_n.g = min(s_n.g, s.g + cost(s, s_n))$. The best g-value is stored across visits and across iterations.
3: Parent pointer (labeled $s.prev$): As all depth-first searches, EDA* must backtrack once it reaches a leaf. Thus, a backpointer is stored. When the agent moves away from state s to a neighboring state s_{next} which was previously unvisited during the current iteration, the parent pointer $s_{next}.prev$ is set to s (Line 11).

When EDA* moves to a new state $s \neq goal$ in iteration I (Line 12), a new while loop begins (Line 3) and the iteration index $s.i$ is updated to I (Line 6). All neighbors of state s are examined; during this process their g-value is updated if needed. The neighbor with the lowest $f = g + h$, that was not yet visited during the current iteration, is chosen as s_{next}. If no unvisited neighbor exist, s_{next} is set to null. Next, two cases exist:
(1) (Lines 8-10) $f(s) = g(s) + h(s) > T$ or $s_{next} = null$. In this case, the agent backtracks to state $s.prev$.
(2) (Line 11-12) $f(s) = g(s)+h(s) \leq T$ and $s_{next} \neq null$. In this case, $s_{next}.prev$ is updated to be s. Then, the agent physically moves to s_{next}.

Experimental Results

In all experiments we add results for A* which is a classic search algorithm and not a RTACS algorithm. Nevertheless, A* never revisits states ($FVR = 1$). Thus, it can be used as a lower bound. Moreover, when the different RTACS algorithms use a Local Search Space (LSS) they perform an A* search over the LSS. In the limit where the LSS includes the entire state space these algorithms will act identically to A*.

Supporting the Theoretical Analysis

Our first experiment is intended to support the claim that EDA* is efficient in polynomial domains. For this we used a large grid (2000×2000) with no obstacles so as to clearly show the trends of the algorithms. As we aim to prove the worst case performance bounds for the different algorithms, the heuristic values of all states were set to zero ($h \equiv 0$).

The distance d between *start* and *goal* varied from 1 to 500. Figure 1(left) presents the number of expanded states (y-axis) as a function of the distance from *start* to the *goal* (d) (x-axis) for IDA*, RIBS (+dead-state pruning), and EDA* with $C = 2$. Clearly, EDA* outperforms RIBS which in turn outperforms IDA*. The quadratic growth of IDA*/RIBS vs. the linear growth of EDA* is clearly shown.

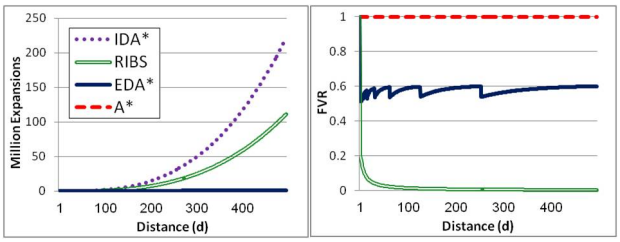

Figure 1: Open map experiments: A*, IDA*, RIBS, EDA*.

Alg.	Expanded	Distance	Time	FVR
Classic search algorithms				
A*	**11,345**	**380**	**78**	**1.00**
IDA*	6,142,549	14,617,700	9,975	0.19
Real-Time Agent-Centered algorithms				
RIBS	330,397	742,138	1,971	0.29
f-LRTA*	82,111	92,149	340	0.49
LRTA*	237,233	243,075	284	0.42
daLRTA*	33,486	35,645	105	0.66
RTA*	60,744	70,481	**78**	0.60
daRTA*	**26,664**	**30,978**	82	**0.74**
EDA*(1.1)	48,797	109,146	135	0.40
EDA*(1.5)	18,984	38,518	51	0.59
EDA*(2)	15,243	29,764	40	0.65
EDA*(4)	12,970	24,248	34	0.70
EDA*(8)	**12,714**	**23,553**	**33**	**0.71**
EDA*(16)	12,785	23,689	**33**	**0.71**

Table 2: Average measurements over all DAO problems.

Figure 1(right) presents the FVR (y-axis). Here, A*, RIBS and EDA* are reported. A* never checks the same state more than once, therefore, its FVR is always 1. Recall that for *efficient* algorithms (where $F = \theta(R)$) we expect that FVR is neither decreasing nor increasing. The FVR for RIBS decreases quickly, converging to 0. This supports the claim that for polynomial domains IDA* performs a factor of d more revisits compared to first visits ($F = d^k$ and $R = d^{(k+1)}$). By contrast, EDA* has an FVR which is always very close to 0.5 and is not significantly decreasing nor increasing. This supports the claim that EDA* is efficient ($F = \theta(R)$). The "steps" in EDA* correspond to the jumps of T and correspond to the powers of $C = 2$. For any i, once the goal is further than C^i, the $i+1$ iteration must be performed and all states at radius C^{i+1} will be visited. Note that A* has a larger FVR than EDA*. However, since EDA* has a lower constant time per expansion it is faster than A*. The average runtime of EDA* was 223ms vs 328ms for A* (39,998ms for RIBS). Note that other RTACS solvers cannot be compared when $h \equiv 0$, as they will move randomly around the state space.

RTACS Experiments on Video Game Maps

We experimented with the entire set of *Dragon-Age: Origins* (DAO) problems (all buckets, all instances) from (Sturtevant 2012). h was set to *octile distance*. The follow-

Alg.	Expanded	Distance	Time	FVR
A*	**123,392**	**2,893**	**1,146**	**1.00**
IDA*	160,128,683	366,936,525	260,343	0.00
Real-Time Agent-Centered algorithms				
RIBS	6,165,745	14,010,238	32,093	0.01
f-LRTA*	37,817,668	44,415,331	134,964	0.00
LRTA*	7,274,692	7,449,572	8,411	0.01
daLRTA*	1,371,132	1,456,906	4,534	0.03
RTA*	3,070,169	3,619,821	4,337	0.02
daRTA*	2,283,872	2,748,504	6,587	0.02
EDA*(16)	**141,455**	**312,137**	**581**	**0.31**

Table 3: Worst case measurements over all DAO problems.

ing algorithms were used for this experiment: A*, LRTA*, RTA*, daLRTA* and daRTA* (Hernández and Baier 2011), f-LRTA* (Sturtevant and Bulitko 2011), RIBS and EDA*[6]. For all algorithms, lookahead (sensing radius) was set to 1. For EDA*, the number in parenthesis denotes the size of the constant factor C. C was chosen from $\{1.1, 1.5, 2, 4, 8, 16\}$.

Table 2 reports the averages over all instances of four measures aspects: **(1)** The number of node expansions (expanded). **(2)** The total distance traveled during the solving process (for RTACS algorithms) (Distance). **(3)** CPU runtime in *ms*. The CPU time spent in the planning phases (Time). **(4)** First Visit Ratio (FVR).

The best algorithm in each category is in bold. The total distance traveled highly correlates to the number of expanded states. Distance is slightly higher than the nodes expanded due to the agent's ability to move diagonally; diagonal moves cost $\sqrt{2}$ while only expanding one state. In the DFS algorithms (IDA*, RIBS, EDA*), the distance traveled correlates to twice the number of expanded states because the agent travels in each edge twice (generating successors and then backtracking).

Different C values for EDA* influence the performance. The value of $C = 8$ was best for all 4 measures. EDA* outperformed *all* other algorithms in all measures. If we disregard RTA* and daRTA* which are not Optimal-1 then the advantage of EDA* is even more dramatic.

Worst case experiment EDA* has the best worst-case complexity (linear in the state space) among all other RTACS solvers (most are quadratic in the state space). Consequently, the advantage of EDA* is much clearer when we look at the worst instance for each algorithm. Table 3 has the same format as Table 2 but on a single instance. For each algorithm it is the instance where the algorithm performed worst, thus, the instances may vary among the algorithms. Our aim here is to show that EDA*, unlike other algorithms, is very robust among the instances and that it performs well even in its worst-case behavior. The results show that when comparing the worst cases, EDA* significantly outperforms all other RTACS solvers.

Varying the Lookahead Some of the algorithms, namely, LRTA*, RTA*, daLRTA* and f-LRTA* use a Local Search Space (LSS) for lookahead. If allowed, these algorithms search a local space around the agent in order to perform more learning and find a better path. The quality of the plan returned by the planning phase is better and so the total distance traveled by the agent is reduced at the cost of a longer planning phase and more CPU time. We implemented lookahead in EDA* by simulating the algorithm without actually moving the agent. Once the planning phase is done, the agent then moves to the final simulated location.

[6]The *dead states pruning* of RIBS are also applicable to EDA*. Unlike RIBS, the complexity of EDA* is not dominated by state revisits. Consequently EDA* does not benefit greatly from dead state pruning; we found the overhead to be not worthwhile.

Alg.	LSS	Exp.	Dist.	Time	#Cycles	MS/P
daLRTA*	5	51,472	**14,664**	745	10,295	0.072
daRTAA*	5	40,616	14,697	212	8,124	0.026
EDA*(8)	5	**12,714**	23,090	**30**	**4,063**	**0.007**
daLRTA*	10	69,266	11,657	882	6,927	0.127
daRTAA*	10	46,492	**9,719**	226	4,650	0.049
EDA*(8)	10	**12,714**	22,891	**28**	**2,032**	**0.014**
daLRTA*	100	62,575	**3,809**	647	626	1.033
daRTAA*	100	76,373	4,657	406	764	0.531
EDA*(8)	100	**13,114**	22,529	**31**	**210**	**0.148**

Table 4: Average results for varying lookaheads.

Table 4 compares the strongest existing RTACS solvers, daRTAA* and daLRTA* to EDA* with $C = 8$, with varying limits on the size of the LSS. Here we also present the number of planning/acting cycles (#Cycles) and the time (in millisecond) per planning phase (MS/P). Again, all values are averaged over all problem instances.

Increasing the LSS reduces the distances for daRTAA*/daLRTA*. The reduction for EDA* is less dramatic and only occurs if the agent backtracks during the lookahead phase. EDA*, however, outperformed daRTAA*/daLRTA* in all other metrics. If large LSS is allowed, CPU time is abundant and the agent moves slowly, daRTAA* with a large LSS will yield the best travel distance. But, if the LSS is limited to radius 1, or CPU time is restricted, or the agent moves extremely fast, EDA* should be chosen.

Conclusions and Future Work

We introduced new terminology with regards to efficiency of RTACS solvers. We then presented EDA*. EDA* is intuitive and very simple to implement. To the best of our knowledge EDA* is the only RTACS algorithm that is, in the worst case, linear in the state space. Experimental results on grids support our theoretical claims; EDA* outperforms other algorithms in all the measurements if standard lookahead of radius 1 is assumed. If deeper lookahead is allowed, EDA* is best in all measurements except Distance. In addition, EDA* is shown to be very robust across different instances.

Many of the claims and proofs along the paper made specific assumptions but the trends were generally supported by our experiments. Future papers will **(1)** remove these assumptions and provide a more thorough theoretical treatment of other cases, such as variable edge costs, variable dimensionality etc. **(2)** cover the relations between DD and reexpansion of nodes when better g-values are found **(3)** treat various (and irregular) movement ordering of the agent, and **(4)** devise advanced methods for performing LSS for EDA*.

Acknowledgements

The research was supported by the Israeli Science Foundation (ISF) under grant #417/13 to Ariel Felner.

References

Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

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

Ethan Burns and Wheeler Ruml. Iterative-deepening search with on-line tree size prediction. *Annals of Mathematics and AI*, 2013.

Ethan Burns, Wheeler Ruml, and Minh Binh Do. Heuristic search when time matters. *J. Artif. Intell. Res. (JAIR)*, 47:697–740, 2013.

Ariel Felner, Meir Goldenberg, Guni Sharon, Roni Stern, Tal Beja, Nathan R. Sturtevant, Jonathan Schaeffer, and Robert Holte. Partial-expansion A* with selective node generation. In *AAAI*, 2012.

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.

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.

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

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, Michael Reid, and Stefan Edelkamp. Time complexity of iterative-deepening-A^*. *Artif. Intell.*, 129(1-2):199–218, 2001.

R. E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *AIJ*, 27(1):97–109, 1985.

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

Levi H. S. Lelis, Sandra Zilles, and Robert C. Holte. Predicting the size of ida*'s search tree. *Artif. Intell.*, 196:53–76, 2013.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.

Uttam K. Sarkar, Partha P. Chakrabarti, Sujoy Ghose, and S. C. De Sarkar. Reducing reexpansions in iterative-deepening search by controlling cutoff bounds. *Artif. Intell.*, 50(2):207–221, 1991.

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.

Nathan R. Sturtevant and Vadim 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. Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games*, 2012.

Benjamin W. Wah and Yi Shang. A comparative study of ida*-style searches. In *ICTAI*, pages 290–296, 1994.