



## Controlling the learning process of real-time heuristic search<sup>☆</sup>

Masashi Shimbo<sup>a,\*</sup>, Toru Ishida<sup>b</sup>

<sup>a</sup> Graduate School of Information Science, Nara Institute of Science and Technology, Nara 630-0192, Japan

<sup>b</sup> Department of Social Informatics, Kyoto University, Kyoto 606-8501, Japan

Received 2 June 2001; received in revised form 18 August 2002

---

### Abstract

Real-time search provides an attractive framework for intelligent autonomous agents, as it allows us to model an agent's ability to improve its performance through experience. However, the behavior of real-time search agents is far from rational during the learning (convergence) process, in that they fail to balance the efforts to achieve a short-term goal (i.e., to safely arrive at a goal state in the present problem solving trial) and a long-term goal (to find better solutions through repeated trials). As a remedy, we introduce two techniques for controlling the amount of exploration, both overall and per trial. The *weighted real-time search* reduces the overall amount of exploration and accelerates convergence. It sacrifices admissibility but provides a nontrivial bound on the converged solution cost. The *real-time search with upper bounds* insures solution quality in each trial when the state space is undirected. These techniques result in a convergence process more stable compared with that of the Learning Real-Time A\* algorithm.

© 2003 Elsevier Science B.V. All rights reserved.

*Keywords:* Real-time heuristic search; Adaptive learning; Convergence process; Rational agent; Resource-boundedness

---

### 1. Introduction

Being an overall problem solving architecture rather than a planner in the sense of traditional off-line search, real-time search [22] provides an attractive model for intelligent autonomous agents [15,31]. It interleaves look-ahead search (partial planning) and action

---

<sup>☆</sup> This paper is an extended version of [18], augmented with proofs and additional experimental results.

\* Corresponding author.

*E-mail address:* shimbo@is.aist-nara.ac.jp (M. Shimbo).

execution in an online manner, by limiting look-ahead search to be performed within real time. This architectural design endows real-time search with great flexibility to cope with unknown or changing environments. It does not require a complete model of the state space to be known a priori, and it can adapt itself to changing goals [17] and to non-determinism in the state space [1,3,20]. All these are tasks not easily carried out with off-line search, but which are commonly faced by autonomous agents in practical applications.

As the look-ahead search is non-exhaustive, the solution obtained with real-time search is generally sub-optimal. Yet, the Learning Real-Time A\* (LRTA\*) [22] and its family of real-time search algorithms have the property called *convergence* to compensate for sub-optimality. If these algorithms are successively applied to the same or similar problems, they eventually converge to optimal solutions. Stated differently, the algorithms are capable of *learning* better ways of solving problems through trial and error. This property is again favorable to autonomous agents, as it allows them to improve their performance while at their tasks. The learning process of LRTA\*, however, presents two problematic issues from the standpoint of rational agents.

First, LRTA\* is designed to boost performance to optimality, but such a task often requires abundant computational resources. It therefore violates the fundamental design policy of rational agents, namely, resource-boundedness [31], and consequently, application domains are severely constrained. As argued by Simon [33] and advocated by Korf [22], it is relatively rare that optimal solutions are required in real-world problems, and near-optimal solutions are often acceptable.

The other issue is the instability of the learning process. Performance during convergence often worsens considerably before an optimal solution is reached. This instability is incurred by the ‘greedy’ nature of the algorithm, which eagerly explores unvisited regions of the state space.

If finding optimal solutions is the only objective of the algorithm, eager exploration makes sense, as it increases the chance of finding optimal solutions in a shorter time. But from the viewpoint of agent architecture, the instability resulting from this strategy is not always acceptable; if the agent performed reasonably well in a run, the user, knowing that the agent has the ability to self-improve performance, would not expect it to perform orders of magnitude worse in the next run. The real-time search agent should hence pursue two objectives simultaneously: the short-term objective of arriving at a goal state safely in the present run, in addition to the long-term objective of finding better solutions through repeated runs. These objectives are often in a trade-off relation, as the first biases the agent in favor of attempted actions, while the second requires aggressive trial of non-attempted actions at the risk of degrading the present solution quality. How this trade-off should be resolved depends on the application domain, but LRTA\* does not provide ways of controlling the amount of exploration it performs, and always pursues optimality.

Based on this observation, we present two techniques<sup>1</sup> to overcome the problematic behavior of LRTA\* during convergence. The first of the two methods, *weighted LRTA\**, abandons the convergence to optimal solutions to reduce the overall amount of exploration, and to avoid inherent intractability present in most AI problems. The algorithm is sub-

---

<sup>1</sup> In [18], weighted LRTA\* and upper-bounded LRTA\* were called  $\epsilon$ - and  $\delta$ -search, respectively.

optimal, but it is still accompanied by a useful bound on converged solution costs. The other method, *upper-bounded LRTA\**, allows us to control the amount of exploration in each problem solving trial (a run of the algorithm) when the state space is undirected. This is achieved with the help of an extra heuristic function, which supplies the problem solver with *upper bounds* for the exact costs. Such upper bounds are either known for some problems, or obtained from the result of the first problem solving trial. The use of additional estimates requires extra memory, and may appear to contradict resource-boundedness. However, as we will demonstrate in Section 5, such additional estimates bring about an overall saving in memory required to solve large problems, compared to algorithms using only lower bounds.

The rest of the paper is organized as follows. Section 2 reviews the LRTA\* algorithm and presents an experimental result that illustrates the issues in its learning process. Sections 3 and 4 introduce weighted LRTA\* and upper-bounded LRTA\*, respectively. Section 5 empirically compares the performance of the proposed methods with other real-time search algorithms, including LRTA\* and the more recent FALCONS [9]. In Section 6, we discuss the relation between our proposed methods and other search algorithms. We conclude in Section 7.

## 2. Learning Real-Time A\*

Learning Real-Time A\* (LRTA\*), due to Korf [22], is probably the most fundamental and popular of all the real-time search algorithms that are capable of learning. In this section, we first review the algorithm and relevant notions briefly. We then run LRTA\* in a typical AI benchmark problem in order to illustrate the aforementioned issues concerning its learning performance.

### 2.1. State space and heuristic function

The state space is defined as a quintuple  $(X, A, k, s, G)$ , which is a finite simple graph  $(X, A)$  where  $X$  is a nonempty finite set of states (nodes), and  $A \subset X \times X - \{(x, x) \mid x \in X\}$  is a set of actions (edges) with each edge labeled with a cost assigned by an action cost function  $k: A \mapsto [0, \infty)$ , together with a distinct state  $s \in X$ , called initial state, and a set  $G \subset X$  of goal states. A state space is *undirected* if  $A$  defines a symmetric relation and  $k(x, y) = k(y, x)$  for any  $(x, y) \in A$ ; otherwise, it is *directed*.

We denote the set of (immediate) successors of  $x$  by  $\text{Succ}(x)$ , or,  $\text{Succ}(x) = \{y \mid (x, y) \in A\}$ . A *path* is a nonempty sequence  $(x_0, x_1, \dots)$  of states, with every pair  $(x_j, x_{j+1})$  of successive states in the sequence belonging to  $A$ . We assume that for every non-goal state  $x \in X - G$  in the state space, there exists at least one path from  $x$  to a goal state. The *length* of a path is the cumulative number of actions involved in the path. The *cost* of a path is the sum of the action costs associated with the successive state pairs in the path. The (exact) cost  $h^*(x)$  of a state  $x$  is the minimum cost incurred to go from  $x$  to a goal state. In particular,  $h^*(g) = 0$  for every goal state  $g \in G$ . A path that yields the exact cost of state  $x$  is called an *optimal path* from  $x$ . A *heuristic function*  $h$  is a mapping  $h: X \mapsto [0, \infty]$ . Intuitively, a heuristic function assigns an estimate of the exact cost to each state; hence

the value  $h(x)$  is called the *heuristic estimate* for state  $x$ . The heuristic estimate  $h(x)$  is *correct* in  $x$  iff  $h(x) = h^*(x)$ . Such a state  $x$  is called a correct state (wrt  $h(\cdot)$ ). The heuristic estimate  $h(x)$  of a state  $x$  is said to be *admissible* in  $x$  iff  $h(x) \leq h^*(x)$  holds. A heuristic function that assigns admissible values to all states is called an admissible heuristic function.

The following is a summary of the assumptions on the state space.

**Assumption 2.1.**  $|X|$  and  $|A|$  are finite.

**Assumption 2.2.** Every action (edge) cost is positive.

**Assumption 2.3.** Every state has at least one path to a goal state.

**Assumption 2.4.** The underlying graph  $(X, A)$  is simple.

**Assumption 2.5.** The state space does not contain a loop (a cycle of length one); i.e.,  $(x, x) \notin A$  for any  $x \in X$ .

The first three assumptions are standard in real-time search literature (e.g., [9,22]). Assumptions 2.4 and 2.5 are made only for the sake of brevity. With minor modifications in the arguments, the results established in this paper apply to state spaces violating these two assumptions.

In Section 4 and Appendix A.2, we make the additional assumption that the state space is undirected (Assumption 4.1). These sections discuss the properties of upper-bounded LRTA\*, which is only applicable to undirected state spaces. In contrast, LRTA\* and weighted LRTA\* equally apply to directed state spaces.

## 2.2. The LRTA\* algorithm

In AI search framework, the task of the problem solver, which is initially placed in the initial state of a given state space, is to arrive at a goal through a path that costs as low as possible. Real-time search further demands that individual action decisions be made in real time, based only on information gathered from a vicinity of the problem solver's current state. In this paper, this *local-search* constraint is modeled by limiting the look-ahead search to depth one. Thus, the problem solver is only allowed to look ahead at the immediate successors of the current state.

The LRTA\* algorithm, which conforms to the above search model, is shown in Algorithm 1. The problem solver maintains its current state in variable  $x_{\text{cur}}$ . It departs from the initial state  $s$ , and iterates steps (3a) and (3b) until it reaches a goal state. The algorithm uses an external heuristic function  $h_0(\cdot)$  which returns the initial (admissible) estimate associated with each state. The actual estimates are maintained in the table  $h(\cdot)$  and are updated during the course of problem solving to account for the information collected with look-ahead search.

**Algorithm 1** (*LRTA\**).

- (1) For each state  $x \in X$ , set  $h(x)$  to its initial value  $h_0(x)$ .
- (2) Reset the problem solver to its initial state  $s$ . Set  $x_{\text{cur}} \leftarrow s$ .
- (3) Repeat the following sub-steps until  $x_{\text{cur}}$  is a goal state:
  - (a) Look ahead and value update: Update the heuristic estimate  $h(x_{\text{cur}})$  of the current state  $x_{\text{cur}}$  by

$$h(x_{\text{cur}}) \leftarrow \max\{h(x_{\text{cur}}), \min_{y \in \text{Succ}(x_{\text{cur}})} [k(x_{\text{cur}}, y) + h(y)]\}. \quad (1)$$

- (b) Action execution: Move to a successor  $y \in \text{Succ}(x_{\text{cur}})$  of  $x_{\text{cur}}$  such that

$$y \in \underset{z \in \text{Succ}(x_{\text{cur}})}{\text{argmin}} [k(x_{\text{cur}}, z) + h(z)]. \quad (2)$$

Break ties arbitrarily. Set  $x_{\text{cur}} \leftarrow y$ .

In actual implementations, it is common to omit the heuristic initialization step (1) and cache the estimates in an on-demand manner; the estimates are stored in the table only after they have changed from the initial values. As the stored estimates persist in the table, the number of states cached, or, in search terminology, *expanded*, serves as a measure of the amount of memory required to run the algorithm.

Under Assumptions 2.1–2.5,<sup>2</sup> when the initial heuristic function  $h_0(\cdot)$  is admissible, the *LRTA\** algorithm enjoys the following properties [22]. It is *complete* in the sense that it never fails to reach a goal. In other words, a *problem solving trial* (a run of Algorithm 1) always terminates. We call the path traversed by the problem solver in a trial the *solution (path)* obtained in the trial, and its cost the *solution cost*. Furthermore, we can show that the admissibility of estimates  $h(\cdot)$  is preserved throughout and after a trial of *LRTA\**. Thanks to this property and the completeness of the algorithm, it can be repeatedly applied to the same problem by resetting the problem solver to the initial state after it arrives at a goal, and reusing the updated heuristic estimates as the initial heuristic function for the next trial. In such repeated trials (called an *episode*), *LRTA\** is *convergent* to optimal solutions, in the sense that after a certain trial, only optimal paths are traversed, and in addition, all the states along these paths have correct  $h(\cdot)$  estimates.

### 2.3. The learning process of *LRTA\**

To examine the convergence process of *LRTA\**, we run it in the gridworld of size  $100 \times 100$  depicted in Fig. 1. Obstacles were placed on 35% of the states (grid cells) chosen at random. At each iteration, the problem solver is allowed to move to an adjacent state horizontally or vertically, unless it is occupied by an obstacle. All moves have a uniform unit cost. There is a unique goal state, placed 100-units apart from the initial state as measured by the Manhattan distance (i.e., the sum of the horizontal and vertical distances in the grid). Only one optimal path exists between them, whose exact cost is 122 units.

---

<sup>2</sup> As mentioned earlier, Assumptions 2.4 and 2.5 can be removed.

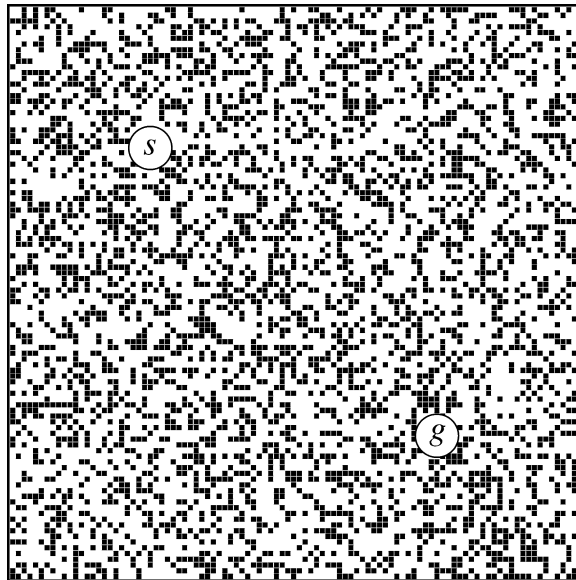


Fig. 1. Gridworld with 35% obstacle ratio.  $s$  and  $g$  respectively indicate the approximate locations of the initial and the goal states.

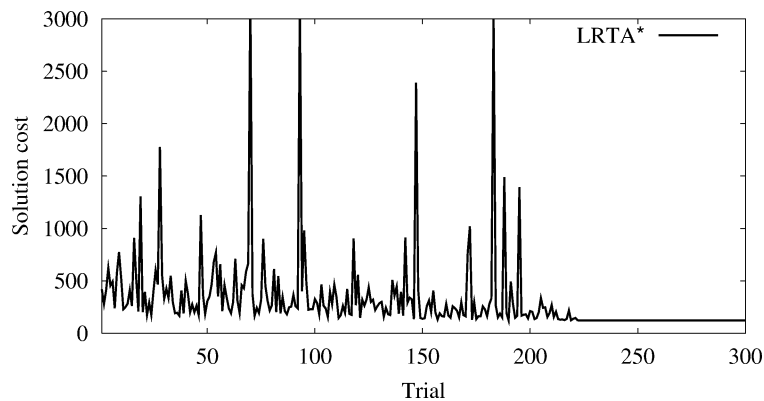


Fig. 2. Typical learning process of LRTA\*: number of moves per trial.

Fig. 2 shows a typical learning episode of LRTA\* in this environment, with the Manhattan distance as the initial heuristic function and a random tie-breaking strategy. The horizontal axis of the graph represents the number of problem solving trials, and the vertical axis measures the solution cost (or, in this domain, the number of moves made by the problem solver) in each trial. As the graph shows, convergence is not monotonic. The solutions in some trials are considerably worse than in earlier trials, and such performance degradations can happen immediately after a near-optimal solution is found, or even in the last stages of the convergence process.

This phenomenon is not specific to this problem (see, for instance, [25]), but is inherent to the LRTA\* algorithm. The characteristics and explanation of this undesirable behavior can be summarized as follows.

- (1) Even after it finds a near-optimal solution, the algorithm continues to explore the rest of the state space seeking for an optimal solution. Although it is feasible to find optimal solutions in this small grid, it is inherently intractable to attain optimality in most of the AI tasks.
- (2) LRTA\* does not guarantee a stable improvement of solution quality. It uses an admissible initial heuristic function, which does not overestimate the cost for any state. This often makes the estimated values considerably smaller than the exact costs. Consequently, as learning progresses, estimates are increased for visited states, while those for unvisited states remain small. Since the problem solver's move is biased towards successor states whose estimates are smaller, it tends towards unexplored regions in the state space.<sup>3</sup> As a result of such explorations, it frequently traverses paths that are far costlier than the one already found.

### 3. Weighted real-time search

The use of an admissible initial heuristic function allows LRTA\* to find optimal solutions, but it has some limitations and undesirable side-effects. As argued in Section 2.3, admissible initial heuristic values open a large room for the discrepancy of heuristic values between explored and unexplored states, and causes the instability in the convergence process. Moreover, there are problem domains for which it is difficult to construct an admissible heuristic function which works effectively [12,27]. Last but not least, it may be computationally intractable to find optimal solutions for some problems. These limitations raise the following questions. Does repeated application of LRTA\* have any use in such domains? What would happen if the initial estimates violated admissibility? Can we expect any performance improvement in such cases? We answer these questions through the analysis of the *weighted LRTA\** method, which adds an extra *weight* to the initial heuristic functions thus making them non-admissible.

#### 3.1. Adding weights to heuristics

Even if we remove the admissibility constraint from the initial heuristic function, we can still find an upper bound for  $h_0(\cdot)$  because the state space is finite. In other words, there is a constant  $\varepsilon \geq 0$  such that  $h_0(x) \leq (1 + \varepsilon)h^*(x)$  for every state  $x \in X$ .

**Definition 3.1.** Let  $h(\cdot)$  be a heuristic function. For a constant  $\varepsilon \geq 0$ , if the heuristic value for state  $x$  does not exceed the exact cost by more than a factor of  $(1 + \varepsilon)$ , or,

$$h(x) \leq (1 + \varepsilon)h^*(x), \quad (3)$$

<sup>3</sup> Similar 'optimism in the face of uncertainty' has been observed in reinforcement learning as well [26].

then the heuristic value  $h(x)$  is said to be  $\varepsilon$ -admissible in  $x$ . A state  $x$  satisfying condition (3) is called an  $\varepsilon$ -admissible state wrt  $h(\cdot)$ . A heuristic function that assigns an  $\varepsilon$ -admissible value to every state in the state space is called an  $\varepsilon$ -admissible heuristic function. When such a value  $\varepsilon$  is known for a heuristic function  $h(\cdot)$ , we call it a *weight* of  $h(\cdot)$ .

For a parameter  $\varepsilon \geq 0$ , the *weighted LRTA\** is a modification of LRTA\* given in Algorithm 1 in which the initial heuristic function is relaxed to be only  $\varepsilon$ -admissible. All other constraints given in Algorithm 1 remain the same. Viewed in another way, since these methods are algorithmically identical and differ only in the class of the initial heuristic functions used, this section discusses the properties of LRTA\* when the initial heuristic function overestimates. In particular, since 0-admissibility is equivalent to (plain) admissibility, weighted LRTA\* subsumes LRTA\* as a special case when  $\varepsilon = 0$ .

### 3.2. Completeness of weighted LRTA\*

In this subsection, we prove the completeness of weighted LRTA\*. By (*intra-trial*) *time instant*  $t$ , we refer to the moment of time immediately after the  $t$ th iteration of step (1) in Algorithm 1. Let  $x_t$  be the state occupied by the problem solver at time  $t$ , and for each state  $y \in X$ , let  $h_t(y)$  denote the heuristic estimate  $h(y)$  at this instant. In addition, we use  $t = 0$  to denote the time just before the first iteration of step (1) starts; hence,  $x_0 = s$ , and the initial heuristic function is  $h_0(\cdot)$  (as already defined). Note that  $t$  denotes *intra-trial* time, and is therefore reset to 0 every time the problem solver is reset to the initial state in the repeated problem solving scenario. We use the time notation exclusively in the discussion of the intra-trial properties. Repeated applications of weighted LRTA\* will be addressed in Section 3.3, but without using this notation.

Our first lemma states that the  $\varepsilon$ -admissibility of heuristic estimates is never violated by the problem solver's value update operations.

**Lemma 3.2.** *Weighted LRTA\* preserves the  $\varepsilon$ -admissibility of heuristic function. In other words, if  $h_t(x) \leq (1 + \varepsilon)h^*(x)$  holds for every state  $x$  at time instant  $t = 0$ , then it holds for any  $t \geq 0$ .*

**Proof.** The proof is by induction on time  $t$ . The base case  $t = 0$  is given by the assumption of the lemma. Suppose  $h_t(x) \leq (1 + \varepsilon)h^*(x)$  for every  $x \in X$ . On the  $(t + 1)$ -st move, the problem solver makes a transition from  $x_t$  to  $x_{t+1}$ . It only updates the heuristic estimate for  $x_t$ , and leaves the estimates for the other states unaffected. So it suffices to prove that  $h_{t+1}(x_t)$  is  $\varepsilon$ -admissible. If  $h_t(x_t) \geq k(x_t, x_{t+1}) + h_t(x_{t+1})$ , then update rule (1) does not alter the estimate for  $x_t$  either. Hence assume  $h_t(x_t) < k(x_t, x_{t+1}) + h_t(x_{t+1})$ , and let  $y \in \text{Succ}(x_t)$  be an optimal successor of  $x_t$ , or, the one that satisfies  $h^*(x_t) = k(x_t, y) + h^*(y)$ . Such a successor  $y$  always exists for any non-goal state  $x_t$ , due to Bellman's optimality equation [2] and Assumption 2.3. Then, because  $h(x_t)$  was updated by rule (1), and  $x_{t+1}$  was chosen according to rule (2), we have

$$h_{t+1}(x_t) = k(x_t, x_{t+1}) + h_t(x_{t+1}) \leq k(x_t, y) + h_t(y). \quad (4)$$

Since  $y$  is  $\varepsilon$ -admissible, we have  $h_t(y) \leq (1 + \varepsilon)h^*(y)$ . Substituting this inequality in formula (4) yields

$$h_{t+1}(x_t) \leq k(x_t, y) + (1 + \varepsilon)h^*(y) \leq (1 + \varepsilon)[k(x_t, y) + h^*(y)].$$

Since  $y$  is an optimal successor of  $x$ , the RHS is equal to  $(1 + \varepsilon)h^*(x_t)$ . This completes the induction.  $\square$

Next, we show that with a minor modification, a lemma originally developed for *moving-target search* [16,17], a real-time search algorithm for changing goals, is applicable to state spaces with stationary goals and non-uniform action costs.<sup>4</sup> The lemma holds regardless of the admissibility of the initial heuristic function.

**Lemma 3.3.** *For each time instant  $\tau = 0, 1, 2, \dots$ , the following relation holds.*

$$\sum_{t=1}^{\tau} k(x_{t-1}, x_t) \leq \sum_{x \in X} h_{\tau}(x) - h_{\tau}(x_{\tau}) - \sum_{x \in X} h_0(x) + h_0(s).$$

**Proof.** See Appendix A.1.  $\square$

Using the above lemmas, we prove the completeness of weighted LRTA\*.

**Theorem 3.4** (Completeness of weighted LRTA\*). *In a state space as defined in Section 2.1, weighted LRTA\* never fails to reach a goal.*

**Proof.** Because each non-goal state has at least one successor (Assumption 2.3), the algorithm terminates only when it reaches a goal. We prove its termination by deriving a bound on the algorithm's running time  $\tau$ . Let  $k_{\min} = \min_{(x,y) \in A} k(x, y)$  be the minimum action cost in the state space. Then, for time  $\tau = 0, 1, 2, \dots$ , we have

$$\begin{aligned} \tau k_{\min} &\leq \sum_{t=1}^{\tau} k(x_{t-1}, x_t) \\ &\leq \sum_{x \in X} h_{\tau}(x) - h_{\tau}(x_{\tau}) - \sum_{x \in X} h_0(x) + h_0(s) \\ &\leq \sum_{x \in X} (1 + \varepsilon)h^*(x) - \sum_{x \in X} h_0(x) + h_0(s). \end{aligned}$$

The last two inequalities follow from Lemmas 3.3 and 3.2, respectively. Since  $|X|$  and  $h^*(\cdot)$  are both upper-bounded, the RHS and consequently the LHS ( $\tau k_{\min}$ ) are also upper-bounded. As the positive cost assumption implies  $k_{\min} > 0$ , the algorithm's running time  $\tau$  is upper-bounded.  $\square$

<sup>4</sup> For state spaces with uniform action costs, see also [19,32].

### 3.3. The learning property of weighted LRTA\*

We now turn our attention to the repeated application of weighted LRTA\*. According to Theorem 3.4, a problem solving trial of weighted LRTA\* always terminates, and Lemma 3.2 guarantees that the  $\varepsilon$ -admissibility of heuristic estimates is preserved after each trial. It is hence possible to run weighted LRTA\* successively in the same manner as described in Section 2.2 for LRTA\*; namely, by reusing updated heuristic estimates as the initial heuristic function for the next trial.

The goal of this subsection is to derive a bound on the solution costs eventually obtained. We first need the notions of  $\varepsilon$ -correct states and  $\varepsilon$ -optimal paths.

**Definition 3.5.** Given a constant  $\varepsilon \geq 0$ , if a state  $x$  meets the condition  $h^*(x) \leq h(x) \leq (1 + \varepsilon)h^*(x)$ , then,  $x$  is said to be an  $\varepsilon$ -correct state (wrt  $h(\cdot)$ ).

**Definition 3.6.** Let  $\pi$  be a path from a state  $x \in X$  to a goal  $g \in G$ . If the cost of  $\pi$  does not exceed the optimal cost  $h^*(x)$  by more than a factor of  $(1 + \varepsilon)$ , then,  $\pi$  is said to be an  $\varepsilon$ -optimal path from  $x$ . More precisely, if  $\pi = (x_0, x_1, \dots, x_n)$  where  $x = x_0$  and  $x_n \in G$ , and

$$\sum_{i=0}^{n-1} k(x_i, x_{i+1}) \leq (1 + \varepsilon)h^*(x)$$

is met, then  $\pi$  is an  $\varepsilon$ -optimal path from  $x$ .

The following lemma states that when the problem solver makes a move to an  $\varepsilon$ -correct state, update rule (1) propagates  $\varepsilon$ -correctness from the state to its predecessor.

**Lemma 3.7.** Assume in a state space as defined in Section 2.1, all states have  $\varepsilon$ -admissible estimates  $h(\cdot)$ . Then, if the problem solver makes a transition from a state  $x$  to an  $\varepsilon$ -correct successor state,  $x$  is also  $\varepsilon$ -correct after the transition.

**Proof.** Let  $x_t$  and  $x_{t+1}$  be the states occupied by the problem solver before and after the  $(t + 1)$ -st move, respectively. Suppose that at time  $t$ , every state has an  $\varepsilon$ -admissible estimate, and  $x_{t+1}$  is  $\varepsilon$ -correct at time  $t$ . We show that  $x_t$  is  $\varepsilon$ -correct at time  $(t + 1)$ . According to Lemma 3.2, we have  $h_{t+1}(x_t) \leq (1 + \varepsilon)h^*(x_t)$ . Hence it suffices to show that  $h^*(x_t) \leq h_{t+1}(x_t)$ .

Let  $y \in \text{Succ}(x_t)$  be an optimal successor of  $x_t$ , i.e., the one which immediately follows  $x_t$  in an optimal path from  $x_t$ . Then,

$$h^*(x_t) = k(x_t, y) + h^*(y) \leq k(x_t, x_{t+1}) + h^*(x_{t+1}). \quad (5)$$

Since state  $x_{t+1}$  is assumed to be  $\varepsilon$ -correct at time  $t$ ,  $h^*(x_{t+1}) \leq h_t(x_{t+1})$ . Substituting this inequality in formula (5) yields  $h^*(x_t) \leq k(x_t, x_{t+1}) + h_t(x_{t+1})$ . On the other hand, from update rule (1), we have  $k(x_t, x_{t+1}) + h_t(x_{t+1}) \leq h_{t+1}(x_t)$ . Chaining these inequalities yields  $h^*(x_t) \leq h_{t+1}(x_t)$ .  $\square$

Our first theorem in this subsection is on the convergence of heuristic estimates (not necessarily to their exact values) along the paths traversed by the problem solver after a certain trial.

**Theorem 3.8.** *If we apply weighted LRTA\* repeatedly to a state space as defined in Section 2.1, there is a trial after which all the states visited by the problem solver are  $\varepsilon$ -correct.*

**Proof.** Since the goal states are initially  $\varepsilon$ -correct (and remains so throughout the problem solving trials because their heuristic values are never updated), the completeness of weighted LRTA\* and Lemma 3.7 insure that a trial of weighted LRTA\* changes at least one state to  $\varepsilon$ -correct unless all the states visited by the problem solver in the trial are already  $\varepsilon$ -correct. As the number of states in the state space is finite (by Assumption 2.1), the statement of the theorem follows.  $\square$

Theorem 3.8 also implies that there will be a trial after which no modification is made to the values of the heuristic estimates. The next theorem gives a way to detect  $\varepsilon$ -optimality of the solution obtained in a trial.

**Theorem 3.9.** *If the problem solver arrives at a goal without modifying any of  $h(\cdot)$  estimates in a trial, then the path traversed in the trial is  $\varepsilon$ -optimal.*

**Proof.** Suppose in a trial, no modification was made to the values of  $h(\cdot)$ . Let  $\bar{h}(x)$  be the (stationary) heuristic estimate for each  $x \in X$  during the trial, and let  $\pi = (x_0, x_1, \dots, x_\tau)$  be the path traversed by the problem solver in the trial where  $x_0 = s$  and  $x_\tau \in G$ . We show that  $\pi$  is indeed  $\varepsilon$ -optimal. It is trivial when  $\tau = 0$ , so let  $\tau > 0$ . Since the problem solver has moved from  $x_{t-1}$  to  $x_t$ ,

$$k(x_{t-1}, x_t) + \bar{h}(x_t) = \min_{y \in \text{Succ}(x_{t-1})} [k(x_{t-1}, y) + \bar{h}(y)]$$

holds for all  $1 \leq t \leq \tau$ . From update rule (1), we have

$$\min_{y \in \text{Succ}(x_{t-1})} [k(x_{t-1}, y) + \bar{h}(y)] \leq \bar{h}(x_{t-1})$$

for all  $1 \leq t \leq \tau$ . Combining the above two formulas yields  $k(x_{t-1}, x_t) \leq \bar{h}(x_{t-1}) - \bar{h}(x_t)$ , and further summing this inequality over  $t = 1, 2, \dots, \tau$  yields

$$\sum_{t=1}^{\tau} k(x_{t-1}, x_t) \leq \bar{h}(x_0) - \bar{h}(x_\tau) = \bar{h}(x_0),$$

where the last equality follows from  $\bar{h}(x_\tau) = 0$  given that  $x_\tau \in G$ . By Lemma 3.2, the RHS is upper-bounded by  $(1 + \varepsilon)h^*(x_0)$ . This means  $\pi$  is  $\varepsilon$ -optimal because the LHS equals the cost of  $\pi$ .  $\square$

As a corollary to Theorem 3.9, we have an upper bound on the solution costs obtained after an infinite number of trials.

**Theorem 3.10.** *If we apply weighted LRTA\* repeatedly, there will be a trial after which the problem solver traverses only  $\varepsilon$ -optimal paths.*

**Proof.** For each state  $x$ , since  $h(x)$  is nondecreasing with time, and since it is upper-bounded by  $(1 + \varepsilon)h^*(x)$ , there is a trial  $m$  after which the value of  $h(x)$  will no longer be modified for any  $x \in X$ . Theorem 3.9 thus holds for each trial after  $m$ , and the statement of the theorem follows.  $\square$

### 3.4. Variation: measuring the error additively

Weighted LRTA\*, as defined above, measures the amount of overestimation multiplicatively relative to the exact cost  $h^*(\cdot)$ . It is also possible to evaluate the overestimation additively. This reformulation establishes a different bound on the costs of the solution paths eventually obtained, which may be more suitable for assessing the quality of solutions in some applications. We begin with a definition.

**Definition 3.11.** Given a constant  $e \geq 0$ , the heuristic estimate  $h(x)$  of a state  $x$  is said to be *e-additively admissible* in  $x$  iff  $h(x) \leq h^*(x) + e$  when  $x$  is a non-goal state, and  $h(x) = 0$  otherwise. The estimate  $h(x)$  is *e-additively correct* in  $x$  iff  $h(x)$  is *e-additively admissible* and  $h(x) \geq h^*(x)$ . A heuristic function  $h(\cdot)$  is *e-additively admissible* iff it assigns *e-additively admissible* estimates to all states.

For  $e \geq 0$ , weighted LRTA\* with *e-additive weight* is a variation of LRTA\* that uses an *e-additively admissible* initial heuristic function instead of an admissible one. This variation again subsumes LRTA\* as a special case when  $e = 0$ . The completeness of this additive version follows immediately from that of the multiplicative version because all *e-additively admissible* heuristic functions can be made  $\varepsilon$ -admissible by choosing an adequate  $\varepsilon$ . Hence we focus on its learning property. Concerning *e-additive admissibility*, the following lemma holds.

**Lemma 3.12.** *If the heuristic estimates are initially e-additively admissible for all states, this property is maintained throughout and after a trial of weighted LRTA\*.*

**Proof.** Assume that at time  $t$ , the heuristic estimates are *e-additively admissible* over all states. By an argument similar to the one used in the proof of Lemma 3.2, it suffices to prove that  $h_{t+1}(x_t)$  is *e-additively admissible* after the heuristic value for  $x_t$  is updated to  $h_{t+1}(x_t) = k(x_t, x_{t+1}) + h_t(x_{t+1})$ . We have

$$\begin{aligned} h_{t+1}(x_t) &= \min_{y \in \text{Succ}(x_t)} [k(x_t, y) + h_t(y)] \\ &\leq \min_{y \in \text{Succ}(x_t)} [k(x_t, y) + h^*(y) + e] \\ &= \min_{y \in \text{Succ}(x_t)} [k(x_t, y) + h^*(y)] + e. \end{aligned}$$

Here, the first term on the RHS equals to  $h^*(x)$  by Bellman's optimality equation. It follows that  $h_{t+1}(x_t) \leq h^*(x_t) + e$ .  $\square$

In parallel to Theorems 3.8 and 3.10 (using Lemma 3.12 in place of Lemma 3.2), we have the following theorem.

**Theorem 3.13.** *Consider a state space as defined in Section 2.1, in which the initial heuristic function  $h_0(\cdot)$  is  $\epsilon$ -additively admissible. If we repeatedly run weighted LRTA\* in this state space, there is a trial after which each state in the solution paths has an  $\epsilon$ -additively correct  $h(\cdot)$  estimate, and in addition, the costs of these solution paths do not exceed the optimal solution cost by more than  $\epsilon$ .*

### 3.5. Summary and discussion

We have shown that learning is still possible with weighted LRTA\*, which use non-admissible initial heuristic functions. After a certain trial, the heuristic estimate at each state the problem solver visits has an  $\epsilon$ -correct (or  $\epsilon$ -additively correct) value, and the cost of the path traversed in each trial falls within a multiplicative factor of  $(1 + \epsilon)$  (or an additive factor of  $\epsilon$ ) of the optimal solution cost.

This, however, does not necessarily mean the paths traversed by weighted LRTA\* will converge to a unique path. When ties are broken in non-deterministic ways, the traversed paths may be oscillating among different paths with non-equal costs (yet they are all guaranteed to be  $\epsilon$ -optimal), even after all heuristic estimates become stationary. The oscillation of traversed paths may also happen with LRTA\* when there is more than one optimal path in the state space, but this is less of a problem as their costs are equal. To make weighted LRTA\* converge to a unique  $\epsilon$ -optimal path under non-deterministic tie-breaking, the problem solver has to either (1) cache the last action performed in each state, or (2) keep the path traversed in the present trial temporarily in memory. These schemes allow the problem solver to follow the same path as soon as an  $\epsilon$ -optimal path is found.

The remaining question is how weighted LRTA\* contributes to the rationality of real-time search agents. Although we do not have a theoretical justification, it is expected that these methods alleviate the first problematic behavior of LRTA\* pointed out in Section 1, namely, the violation of resource boundedness, as they do not seek for optimal solutions any more. The empirical evaluation of Section 5 will demonstrate the effect of weighted LRTA\*, especially in speeding-up convergence and saving memory.

## 4. Real-time search with upper bounds

In this section, we propose a method to overcome the instability of solution quality, the second problematic behavior of LRTA\* pointed out in Section 1. The *upper-bounded LRTA\** algorithm proposed in this section provides an explicit (non-trivial) upper bound on the worst-case performance in each trial. However, such an upper bound cannot be given in state spaces containing irreversible actions. Since the depth of look-ahead search is limited, a path must be traversed in order to determine whether it is optimal or not, but there may be no way back when the problem solver recognizes that the path is not optimal. It must therefore go through the whole path until a goal is reached, even though its cost may be

extremely large. Hence, throughout Section 4, we make the additional assumption that the state space  $(X, A, k, s, G)$  as defined in Section 2.1 is *undirected*, in the sense formally given below.

**Assumption 4.1** (*Undirected state space*). The set  $A$  of actions is a symmetric relation and  $k(x, y) = k(y, x)$  for every  $(x, y) \in A$ .

Not all problem domains satisfy this assumption, but the planning framework (*agent search problems*) as formulated by Dasgupta et al. [5], as well as typical AI benchmark problems such as the gridworld and the sliding-tile puzzle fall into this category. To emphasize that we are specifically dealing with undirected state spaces in this section, we use the term *neighbor* instead of *successor* or *predecessor*. The notation  $\text{Ngh}(x)$  is used to denote the set of neighbors of a state  $x \in X$ .

#### 4.1. Introducing upper bound estimates

Upper-bounded LRTA\* uses another heuristic function  $u(\cdot)$  along with  $h(\cdot)$  used by LRTA\*. While  $h(x)$  gives a lower bound for the exact cost  $h^*(x)$  of each state  $x$ , the purpose of  $u(x)$  is to provide the problem solver with an *upper bound* for  $h^*(x)$ .

For some problems such as sliding-tile puzzles or Rubik's Cube, upper bounds on the solution costs are known either empirically or analytically. It is possible to use these upper bounds as the initial values  $u_0(\cdot)$  for  $u(\cdot)$ , as long as they meet a few constraints described shortly (Proposition 4.3). When no such prior knowledge on upper bounds is available, we can use the following 'non-informative' initial heuristic function.

$$u_0(x) = \begin{cases} 0, & \text{if } x \text{ is a goal state,} \\ \infty, & \text{otherwise.} \end{cases} \quad (6)$$

We use the following rule to update the estimates. Let  $x$  be a state and  $y$  be one of its neighbors.

$$u(x) \leftarrow \min[u(x), k(x, y) + u(y)]. \quad (7)$$

Compare rule (7) with (1), which is used by LRTA\* (and by upper-bounded LRTA\* as well) for updating the lower bounds given by  $h(\cdot)$ . The rule for  $u(\cdot)$  is applicable to any two neighboring states  $x$  and  $y$ . It does not require examining all the neighbors to update a value, as is the case with  $h(\cdot)$ . Thus, we can safely update the upper bound estimate as soon as an estimate for a neighboring state is obtained. This is beneficial if the state space has a large branching factor which makes it too costly to look ahead at all neighbors. One practical application with such a state space is the 'multiple sequence alignment problem' [14,24,34]. Such a finer granularity of the update rule also makes it easier to perform additional update operations, for instance, in the problem solver's spare time.

The remainder of the subsection discusses the invariants over application of update rule (7), and the properties of  $u(\cdot)$  implied thereby. We first need the notion of an anti-consistent heuristic function.

**Definition 4.2.** A heuristic function  $u(\cdot)$  is said to be *anti-consistent* iff for every non-goal state  $x \in X - G$ , there exists a neighbor  $y \in \text{Ngh}(x)$  of  $x$  such that  $u(x) \geq k(x, y) + u(y)$ , and  $u(x) = 0$  for every goal state  $x \in G$ .

Anti-consistency differs from *consistency* (monotonicity) [27] and its inverse notion, *inconsistency*, of a heuristic function. The exact cost function  $h^*(\cdot)$  is both consistent and anti-consistent. An anti-consistent heuristic is either inconsistent or exact, but not all inconsistent heuristics are anti-consistent. Inconsistency only requires a single pair of adjacent states  $(x, y) \in A$  to exist such that  $u(x) > k(x, y) + u(y)$ . Also note the strict inequality in this case.

When  $u(\cdot)$  is updated exclusively according to rule (7), we can prove Proposition 4.3 below by induction on the number of updates.

**Proposition 4.3.** *Suppose we perform a series of updates of  $u(\cdot)$  according to rule (7), with each update using an arbitrary pair of neighboring states  $(x, y)$ . If the following statements initially hold, then they continue to hold at any subsequent moment.*

- (1) For every state  $x \in X$ ,  $u(x)$  is an upper bound for  $h^*(x)$ , or,  $u(x) \geq h^*(x)$ .
- (2)  $u(\cdot)$  is anti-consistent.

**Proof.** See Appendix A.2.  $\square$

It is easy to verify that the non-informative initial heuristic function given by (6) satisfies the conditions of Proposition 4.3. Because they are the only requirements that are imposed by upper-bounded LRTA\* on the initial upper bound heuristic function  $u_0(\cdot)$ , any heuristic function can be used as  $u_0(\cdot)$  as long as it satisfies these conditions.

Now recall that the objective of upper-bounded LRTA\* is to insure the problem solver's arriving at a goal within a predetermined cost in each trial. The estimate  $u(\cdot)$  facilitates this purpose by maintaining information about the paths already traversed in previous trials, in an efficient way. This is achieved through the notion of the *u-path* defined below. It is a path obtained by tracing a neighbor state that gives the lowest of the quantity  $k(x, y) + u(y)$  among all the neighbors  $y$  at each state  $x$ . By traversing a *u-path* from its current state  $x$ , the problem solver can arrive at a goal before its travel cost exceeds  $u(x)$ , as shown in Proposition 4.5.

**Definition 4.4.** Let  $x_0 \in X$  be a state in which  $u(x_0)$  is finite. A *u-path* from  $x_0$  is a path  $(x_0, x_1, \dots)$  satisfying

$$x_t \in \operatorname{argmin}_{y \in \text{Ngh}(x_{t-1})} \{k(x_{t-1}, y) + u(y)\}$$

for every  $t \geq 1$  and terminating when and only when  $x_t \in G$ .

**Proposition 4.5.** *If  $u(\cdot)$  initially satisfies the conditions of Proposition 4.3 and is updated exclusively by rule (7), then at any moment and for any state  $x \in X$  such that  $u(x)$  is finite, the cost of any *u-path* is at most  $u(x)$ .*

**Proof.** See Appendix A.2.  $\square$

Proposition 4.5 also allows us to discriminate the states which can be safely explored from the rest, through the following definition.

**Definition 4.6.** The set of *safe successors* of a state  $x$  with respect to parameter  $\theta$ , denoted by  $\text{Safe}(x, \theta)$ , is defined as follows.

$$\text{Safe}(x, \theta) = \{y \in \text{Ngh}(x) \mid k(x, y) + u(y) \leq \theta\}.$$

Assume that the problem solver is in a state  $x$  and wants to arrive at a goal before the travel cost exceeds  $\theta$ . According to Proposition 4.5, it is safe to move to a member of  $\text{Safe}(x, \theta)$  as it has a  $u$ -path that meets the upper bound imposed by  $\theta$ .

#### 4.2. The upper-bounded LRTA\* algorithm

The upper-bounded LRTA\* algorithm is shown in Algorithm 2. It enjoys completeness, and in addition, it never allows the solution cost in a trial to exceed  $u_0(s)$  by more than a factor of  $(1 + \delta)$ , where  $u_0(s)$  is the initial upper bound heuristic estimate for the initial state  $s$ , and  $\delta \geq 0$  is a user-configurable parameter controlling the amount of exploration performed in a trial.

**Algorithm 2** (Upper-bounded LRTA\*).

- (1) For each state  $x \in X$ , initialize  $h(x)$  with some admissible heuristic function  $h_0(x)$ , and  $u(x)$  with an anti-consistent upper bound heuristic function  $u_0(x)$ .
- (2) Set the problem solver to its initial state  $s$ . Set  $x_{\text{cur}} \leftarrow s$ .
- (3) Compute the limit  $\theta \leftarrow (1 + \delta)u(s)$ , where  $\delta \geq 0$  is a parameter.
- (4) Repeat the following steps until  $x_{\text{cur}} \in G$ .

- (a) Propagation of upper bounds towards neighbors: For each neighbor  $y$  of the problem solver's current state  $x_{\text{cur}}$ , update  $u(y)$  by

$$u(y) \leftarrow \min[u(y), k(y, x_{\text{cur}}) + u(x_{\text{cur}})]. \quad (8)$$

- (b) Look-ahead and value update: Update the estimates  $h(x_{\text{cur}})$  and  $u(x_{\text{cur}})$  with

$$h(x_{\text{cur}}) \leftarrow \max\{h(x_{\text{cur}}), \min_{y \in \text{Ngh}(x_{\text{cur}})} [k(x_{\text{cur}}, y) + h(y)]\}, \quad (9)$$

$$u(x_{\text{cur}}) \leftarrow \min\{u(x_{\text{cur}}), \min_{y \in \text{Ngh}(x_{\text{cur}})} [k(x_{\text{cur}}, y) + u(y)]\}. \quad (10)$$

- (c) Action selection: Find a neighbor  $y$  which satisfies

$$y \in \underset{y \in \text{Safe}(x_{\text{cur}}, \theta)}{\text{argmin}} [k(x_{\text{cur}}, y) + h(y)]. \quad (11)$$

Break ties arbitrarily.

- (d) Recomputation of the limit: Set

$$\theta \leftarrow \theta - k(x_{\text{cur}}, y). \quad (12)$$

(e) Action execution: Move to state  $y$  chosen in step (4c). Set  $x_{\text{cur}} \leftarrow y$ .

The algorithm uses variable  $x_{\text{cur}}$  to hold the current state of the problem solver. It also maintains variable  $\theta$  to keep track of the remaining distance that it is allowed to move in the present trial. It is initialized in step (3) and updated in step (4d) to account for the move about to be made in step (4e). This limit is used, in formula (11), to filter out the neighbor states which are not safe to explore. The value-update rule (9) for  $h(\cdot)$  is identical to the one used in LRTA\*. Rule (10) for  $u(\cdot)$  is the aggregation of the atomic update rule (7) over all neighbors. The other update operation for  $u(\cdot)$ , rule (8) in step (4a), is also an instance of (7). It updates the upper bound estimates for the neighbors in reference to the one for the current state  $x_{\text{cur}}$ , in the opposite direction of (10). It is not a valid operation if the state space is directed, and this requires Assumption 4.1. This step is essential for upper-bounded LRTA\* to achieve convergence to optimal solutions (but see Remark 4.7 below); without it, once the limit  $\theta$  becomes finite, the problem solver would no longer move to unvisited states (which have upper bound estimates of  $\infty$ ).

**Remark 4.7.** What is essential in step (4a) for convergence is not the update operation performed, but the look-ahead computation that it involves. In fact, it is not necessary to update *all* the estimate  $u(y)$  for neighbors  $y$  in this step. Instead, we can compute  $\min[u(y), k(y, x) + u(x)]$  for each  $y$  without updating  $u(y)$ , and temporarily storing it in  $u'(y)$  which is then used in place of  $u(y)$  in steps (4b) and (4c).<sup>5</sup> One update operation  $u(y) \leftarrow u'(y)$  is performed at the end of the iteration for the neighbor  $y$  that the problem solver chose to move to in step (4c). For ease of exposition, we described step (4a) as performing real updates for all neighbors.

In the extreme case where  $\delta = \infty$ , the limit  $\theta$  remains  $\infty$  and  $\text{Safe}(x, \theta) = \text{Ngh}(x)$  holds for every state  $x$  throughout the running time of the algorithm, and consequently, upper-bounded LRTA\* reduces to ordinary LRTA\*. The same is true of the case where  $u_0(s) = \infty$ .

#### 4.3. Completeness of upper-bounded LRTA\*

In the following, we prove the two properties of upper-bounded LRTA\* in a single problem solving trial, namely, completeness and the insurance of the solution cost. The property of upper-bounded LRTA\* in repeated problem solving trials will be discussed in the next subsection. By time  $t$  we mean the (intra-trial) time instant immediately after the  $t$ th iteration of step (4) in Algorithm 2. Let  $x_t$  be the state occupied by the problem solver at time  $t$ , and for any state  $x \in X$ , let  $h_t(x)$  and  $u_t(x)$  be the values of  $h(x)$  and  $u(x)$  at time  $t$ , respectively. Again, we use time  $t = 0$  to denote the instant just before the first iteration of step (4). Further define  $c_t$  to be the cost of the path the problem solver has traversed by

<sup>5</sup> Note that  $\text{Safe}(\cdot, \cdot)$  involves  $u(\cdot)$  implicitly.

time  $t$  in the present trial. Using this notation together with rule (12), the value of limit  $\theta$  in the algorithm at time  $t$ , denoted by  $\theta_t$  hereafter, can be rewritten as

$$\theta_t = \theta_0 - c_t = (1 + \delta)u_0(s) - c_t.$$

Furthermore, we use  $\text{Safe}_t(x, \theta)$  to denote the set  $\text{Safe}(x, \theta)$  at time  $t$ . Thus we have

$$\text{Safe}_t(x, \theta) = \{y \in \text{Ngh}(x) \mid k(x, y) + u_t(y) \leq \theta\}. \quad (13)$$

This notation is necessary because even for a fixed state  $x$  and a fixed  $\theta$ ,  $\text{Safe}(x, \theta)$  may vary with  $t$  as it involves  $u(\cdot)$ . Therefore, on making the  $t$ th move in step (4c), the problem solver chooses its destination from the members of  $\text{Safe}_t(x_{t-1}, \theta_{t-1})$ , or the set of neighbors  $y$  of  $x_{t-1}$  such that

$$k(x_{t-1}, y) + u_t(y) \leq (1 + \delta)u_0(s) - c_{t-1}.$$

The subscript of  $u(y)$  is  $t$  and not  $t - 1$  here, because it is updated from  $u_{t-1}(y)$  to  $u_t(y)$  in step (4a) prior to action selection.

The following lemma states that because of the way upper bound estimates are maintained, there exists at least one neighbor  $y$  of  $x_{t-1}$  satisfying  $y \in \text{Safe}_t(x_{t-1}, \theta_{t-1})$  when the problem solver is deciding on the  $t$ th move; otherwise, upper-bounded LRTA\* would not have chosen  $x_{t-1}$  as the destination on the previous move. This lemma eliminates the possibility of the problem solver getting stuck in the state space without being able to find a state to move to.

**Lemma 4.8.** *For any time  $t \geq 1$ , the set  $\text{Safe}_t(x_{t-1}, \theta_{t-1})$ , from which the problem solver chooses a destination on the  $t$ th move, is nonempty.*

**Proof.** If  $\theta_0 = (1 + \delta)u_0(s) = \infty$ , then  $\theta_t$  remains  $\infty$  for all  $t$ . This means  $\text{Safe}_t(x_t, \theta_t) = \text{Ngh}(x_t) \neq \emptyset$ .

When  $\theta_0 < \infty$ , the proof is by induction on time  $t$ . The nonemptiness of  $\text{Safe}_1(x_0, \theta_0)$  is obvious from the anti-consistency of  $u_0(\cdot)$  and the fact that  $\theta_0 = (1 + \delta)u_0(x_0) \geq u_0(x_0)$ . Now suppose  $\text{Safe}_t(x_{t-1}, \theta_{t-1})$  is nonempty and the problem solver has made a transition from  $x_{t-1}$  to  $x_t$  on the  $t$ th move. Hence,  $x_t \in \text{Safe}_t(x_{t-1}, \theta_{t-1})$ , or,

$$u_t(x_t) \leq \theta_0 - c_t. \quad (14)$$

Let  $y$  be a neighbor of  $x_t$  which immediately follows  $x_t$  in a  $u$ -path from  $x_t$ . Then, by the definition of a  $u$ -path (Definition 4.4),

$$u_{t+1}(y) + k(x_t, y) = \min_{z \in \text{Ngh}(x_t)} u_{t+1}(z) + k(x_t, z).$$

Note that by the anti-consistency of  $u_{t+1}(\cdot)$ , we have

$$\min_{z \in \text{Ngh}(x_t)} u_{t+1}(z) + k(x_t, z) \leq u_{t+1}(x_t).$$

Combining these formulas yields

$$u_{t+1}(y) + k(x_t, y) \leq u_{t+1}(x_t). \quad (15)$$

On the other hand, since  $u(\cdot)$  is nonincreasing, it holds that  $u_{t+1}(x_t) \leq u_t(x_t)$ . Chaining this inequality with (14) and (15), we obtain

$$u_{t+1}(y) + k(x_t, y) \leq \theta_0 - c_t,$$

or,  $y \in \text{Safe}_{t+1}(x_t, \theta_t)$  and the induction is complete.  $\square$

The last lemma leads to the completeness of upper-bounded LRTA\*.

**Theorem 4.9.** *Upper-bounded LRTA\* is complete in an undirected state space given by Assumptions 2.1–2.5 and 4.1, if the initial heuristic function  $u_0(s)$  satisfies the two conditions of Proposition 4.3.*

**Proof.** When  $u_0(s) = \infty$ , upper-bounded LRTA\* reduces to LRTA\*; thus, we concentrate on the case where  $u_0(s)$  is finite. In view of Lemma 4.8, we can rule out incompleteness resulting from the problem solver being unable to find any place to move. So the problem solver either arrives at a goal eventually or strays in the state space forever. Suppose the latter. According to rule (12), the limit  $\theta$  tends towards  $-\infty$  with the increase in  $t$ , because the cost  $k(\cdot, \cdot)$  is positive and the number  $|X|$  of states is finite. But this would eventually make  $\text{Safe}(x, \theta)$  empty, contradicting Lemma 4.8. It follows that the problem solver never fails to arrive at a goal.  $\square$

The next theorem insures that if  $u_0(s)$  is finite in the beginning of a trial, the solution cost in the trial is bounded by a function of  $u_0(s)$ . The bounds are obtained by suppressing the amount of exploration performed in each trial, and is controllable through parameter  $\delta$ . This property will be used to achieve a stable convergence process under the repeated application scenario, which will be discussed in the next subsection.

**Theorem 4.10.** *Consider an undirected state space as given by Assumptions 2.1–2.5 and 4.1. If the initial upper bound heuristic  $u_0(\cdot)$  satisfies the conditions of Proposition 4.3 and  $u_0(s)$  of the initial state  $s$  is finite, then, the solution cost obtained in a single trial of upper-bounded LRTA\* is at most  $(1 + \delta)u_0(s)$ .*

**Proof.** We first prove that the problem solver is in state  $x_t$  at time  $t$  only if the following inequality is satisfied.

$$c_t + u_t(x_t) \leq (1 + \delta)u_0(s). \quad (16)$$

It is obvious when  $t = 0$ , because  $\delta \geq 0$ ,  $c_0 = 0$  and  $x_0 = s$ . So let  $t \geq 1$ . Since  $x_t$  is chosen as the destination of the  $t$ th move from  $x_{t-1}$ , we have  $x_t \in \text{Safe}_t(x_{t-1}, \theta_{t-1})$ , or,

$$k(x_{t-1}, x_t) + u_t(x_t) \leq (1 + \delta)u_0(s) - c_{t-1}$$

for each  $t \geq 1$ . Because  $c_t = c_{t-1} + k(x_{t-1}, x_t)$  by the definition of  $c_t$ , this inequality is equivalent to (16).

Now suppose the problem solver has arrived at a goal after making  $\tau$  moves. Substituting  $t = \tau$  in (16) and using the fact that  $u_\tau(x_\tau) = 0$  yields  $c_\tau \leq (1 + \delta)u_0(s)$ .

Because  $c_\tau$  is the cost of the path traversed by the problem solver in the trial, this establishes the theorem.  $\square$

Note that to apply Theorem 4.10,  $u(\cdot)$  needs to be maintained so that a  $u$ -path from the current state  $x$  with cost less than or equal to  $u(x)$  always exists; that is, the conditions of Propositions 4.3 and 4.5 should never be violated. As mentioned earlier, they are met as long as  $u(\cdot)$  is initialized with rule (6) and updated exclusively with rule (7), but it would require caution if different initial upper bound estimates or update schemes were to be used.

#### 4.4. Convergence of upper-bounded LRTA\*

Since upper-bounded LRTA\* is complete and  $u(\cdot)$  continues to meet the conditions in Proposition 4.3 after its termination, it is possible to run upper-bounded LRTA\* repeatedly, in the same manner as done with LRTA\* and weighted LRTA\*. In this repeated problem solving scenario, Theorem 4.10 insures that once the initial state  $s$  has a finite  $u_0(s)$  in the beginning of a trial, the solution costs in the present as well as the subsequent trials will be at most a factor of  $(1 + \delta)$  larger than  $u_0(s)$ .

The question remains whether upper-bounded LRTA\* converges to optimal solutions. As the algorithm incorporates extra machinery to discourage exploration, the optimality of converged solutions may sound unlikely at first glance. However, we can prove their optimality if  $\delta \geq 2$ .

The proof is based on the fact that when  $\delta \geq 2$ , upper-bounded LRTA\* behaves just like LRTA\* until its travel cost exceeds  $u_0(s)$ . We begin with a lemma.

**Lemma 4.11.** *Let  $\pi = (x_0, x_1, \dots, x_\tau)$  be a path traversed by upper-bounded LRTA\*, where  $x_0 = s$  and  $x_\tau \in G$ . Then, the following relation holds for any time  $0 \leq t \leq \tau$ .*

$$u_t(x_t) \leq c_t + u_0(s). \quad (17)$$

**Proof.** The proof is by induction on time  $t$ . The base case  $t = 0$  is straightforward. Assuming inequality (17) holds at some time  $t \geq 0$ , and given update rule (8), we have

$$u_{t+1}(x_{t+1}) \leq k(x_{t+1}, x_t) + u_t(x_t) \leq k(x_{t+1}, x_t) + c_t + u_0(s).$$

By the definition of  $c_t$  and the symmetry of the cost function, in particular,  $k(x_t, x_{t+1}) = k(x_{t+1}, x_t)$ , the RHS is equal to  $c_{t+1} + u_0(s)$ . This completes the induction.  $\square$

**Proposition 4.12.** *Suppose  $\delta \geq 2$ . We run two problem solvers independently, one using upper-bounded LRTA\* and the other using (original) LRTA\*, in the same undirected state space with the same initial (lower-bound) heuristic function  $h_0(\cdot)$ . Then, the sequences of actions executed by the two are identical up to the outcomes of tie-breaking, until the cost of the traversed path exceeds  $u_0(s)$ .*

**Proof.** Assume that on the  $t$ th move, upper-bounded LRTA\* has made a transition from state  $x_{t-1}$  to  $x_t$ , whereas LRTA\* problem solver has chosen state  $y$  as the destination.

Further assume that the transition to  $y$  does not make the cost of the traversed path exceed  $u_0(s)$ , or,

$$c_{t-1} + k(x_{t-1}, y) \leq u_0(s). \quad (18)$$

Now suppose that on deciding the  $t$ th move, upper-bounded LRTA\* excluded  $y$  from the candidates for transition, or equivalently,  $y \notin \text{Safe}_t(x_{t-1}, (1 + \delta)u_0(s) - c_{t-1})$ . From Eq. (13), we have

$$(1 + \delta)u_0(s) - u_t(y) < c_{t-1} + k(x_{t-1}, y).$$

Chaining this inequality with formula (18) yields  $\delta u_0(s) < u_t(y)$ . Since  $\delta \geq 2$ , we have

$$2u_0(s) < u_t(y). \quad (19)$$

On the other hand, by rule (8) for forward propagation of upper bounds,  $u_t(y) \leq k(y, x_{t-1}) + u_{t-1}(x_{t-1})$ , which, when combined with Lemma 4.11, yields

$$u_t(y) \leq k(y, x_{t-1}) + c_{t-1} + u_0(s).$$

However, plugging  $k(y, x_{t-1}) = k(x_{t-1}, y)$  and (18) in this inequality yields  $u_t(y) \leq 2u_0(s)$ , which contradicts formula (19). Hence  $y \in \text{Safe}_t(x_{t-1}, (1 + \delta)u_0(s) - c_{t-1})$ , meaning that  $y$ , as well as  $x_t$ , were the candidates considered by upper-bounded LRTA\*.

Since upper-bounded LRTA\* chose  $x_t$ , we have  $k(x_{t-1}, x_t) + h(x_t) \leq k(x_{t-1}, y) + h(y)$ . Similarly, LRTA\*'s choice of  $y$  yields  $k(x_{t-1}, y) + h(y) \leq k(x_{t-1}, x_t) + h(x_t)$ . Therefore,  $k(x_{t-1}, y) + h(y) = k(x_{t-1}, x_t) + h(x_t)$ . It follows that either  $x_t = y$ , or upper-bounded LRTA\* could have moved to  $y$  as well if the tie-breaking strategy had not preferred  $x_t$  over  $y$ .  $\square$

To take advantage of Proposition 4.12, we need the following proposition that states the convergence of the (original) LRTA\* algorithm under an unconventional setting, in which a trial may be cut off before a goal is reached.

**Proposition 4.13** (Convergence of LRTA\* with cutoff). *LRTA\* converges to optimal solutions in the following scenario. Each trial may be terminated as soon as the cost of the path traversed so far in the trial exceeds a certain cutoff, regardless of whether a goal state has been reached. After a trial is terminated this way, the problem solver is reset to the initial state and a new trial is started as usual. The cutoff may vary between trials, as long as either (i) it is strictly greater than  $h^*(s)$ , or (ii) it is greater than or equal to  $h^*(s)$  and an additional update operation of rule (1) is performed in the non-goal state at which the trial is cut off.*

**Proof.** See Appendix A.3.  $\square$

We show that Proposition 4.13 applies to upper-bounded LRTA\*, as the upper bound  $u_0(s)$  meets the requirement (ii) for the cutoff given in the proposition. First, being an upper bound for  $h^*(s)$ , the distance  $u_0(s)$  satisfies  $u_0(s) \geq h^*(s)$ . Moreover, according to Proposition 4.12, the upper-bounded LRTA\* acts like LRTA\* until the cost of the traversed path exceeds  $u_0(s)$ . Suppose the two problem solvers are in the same state  $x$ . If the next

move by  $LRTA^*$  makes the cost of the traversed path exceeds  $h^*(s)$  for the first time in the trial, upper-bounded  $LRTA^*$  may choose a different move. Even in this case, the update operations that accompany the moves yield the same value of  $h(x)$  for both algorithms, as they use the same update rule for  $h(x)$ ; compare rules (1) and (9). The upper bound  $u_0(s)$  thus meet the condition (ii) for the cutoff in Proposition 4.13.

It remains to show that extra updates of  $h(\cdot)$ , which are performed by upper-bounded  $LRTA^*$  after the intra-trial travel cost exceeds  $h^*(s)$ , do not interfere with the convergence process. To see this, note that even if  $h(\cdot)$  were updated and increased in arbitrary states, its value would never exceed the exact cost in each state, according to Lemma 3.2. Therefore, it cannot be modified infinitely many times. This means that there is a trial after which  $h(\cdot)$  is no longer modified after the intra-trial travel costs exceed  $h^*(s)$ . It follows that after such a trial, Proposition 4.13 applies to upper-bounded  $LRTA^*$  as well, and we have the following theorem.

**Theorem 4.14** (Convergence of upper-bounded  $LRTA^*$ ). *Consider an undirected state space as given by Assumptions 2.1–2.5 and 4.1. If we run upper-bounded  $LRTA^*$  with  $\delta \geq 2$  repeatedly in this state space, the paths traversed by the problem solver will eventually converge to optimal solutions.*

#### 4.5. Obtaining upper bound estimates from the first solution

As shown in the previous subsections, upper-bounded  $LRTA^*$  bounds the worst-case solution cost in each trial. But to make such a bound effective, the initial upper bound estimate  $u_0(s)$  of the initial state  $s$  must be finite at the beginning of a trial; otherwise, upper-bounded  $LRTA^*$  behaves exactly like  $LRTA^*$ . Unfortunately, when no prior knowledge of an upper bound is at hand, it often takes a number of trials before  $u(s)$  becomes finite. This is because the non-informative initial heuristic function of (6) assigns finite  $u(\cdot)$  values only to goal states, and upper-bounded  $LRTA^*$  updates  $u(\cdot)$  only in the neighborhood of the problem solver's current state.

Even if such is the case, the cost of the first-trial solution is usable as  $u_0(s)$  for subsequent trials, given that it cannot be less than  $h^*(s)$ . It does not, however, suffice to assign the obtained cost just to  $u(s)$ , as it would violate the anti-consistency of  $u(\cdot)$  hence rendering the algorithm incomplete.

There are two ways to make  $u(s)$  finite while preserving the conditions of Proposition 4.3. The first scheme backtracks the traversed path after the first trial, performing a series of updates of  $u(\cdot)$  along the path using rule (7) off-line. This method was used in our preliminary version [18]<sup>6</sup> of the paper, and a similar operation can be found in the  $CRTA^*$  algorithm [8] proposed by Edelkamp and Eckerle. This off-line backtracking operation, however, violates the local-search characteristic of the algorithm on which real-time search essentially relies, in the sense that look-ahead and update operations are permitted only within a vicinity of the problem solver's current state.

There is an alternative 'online' scheme for making  $u(s)$  finite after the first trial, which conforms to the local-search characteristic but still maintains the conditions of

---

<sup>6</sup> In [18], this backtracking operation was performed after *every* trial.

**Proposition 4.3.** In the first trial, each visited state  $x$  is marked with the distance  $g(x)$  traveled by the problem solver so far in the trial. If  $x$  is later revisited in the first trial, the value is overwritten. States are marked only during the first trial. In later trials, when the problem solver accesses  $u(x)$  of a state  $x$  which has been marked with  $g(x)$ ,  $u(x)$  is first updated with  $(C - g(x))$ , where  $C$  is the cost of the first-trial solution. After  $u(x)$  is updated this way,  $g(x)$  can be discarded. Hence, the memory overhead incurred by this scheme is minimal, given that the storage for  $u(x)$  and  $g(x)$  can be shared if an extra (1-bit) flag is reserved for each state  $x$  to indicate whether  $u(x)$  or  $g(x)$  is stored.

This scheme provides a domain-independent way of obtaining anti-consistent upper bound heuristic estimates in an online manner. It even allows running more efficient real-time search methods such as  $RTA^*$  [22] exclusively for the first trial to find better upper bounds. After the first trial with  $RTA^*$  in which  $g(\cdot)$  is computed in the same manner as above, all other heuristic estimates besides  $g(\cdot)$  are discarded, and upper-bounded  $LRTA^*$  is used for subsequent trials. Another possibility is to first run weighted  $LRTA^*$  repeatedly until an  $\varepsilon$ -optimal path is found, and then switch to upper-bounded  $LRTA^*$  by using this path as a  $u$ -path from  $s$ .

## 5. Evaluation

We tested the performance of weighted  $LRTA^*$  and upper-bounded  $LRTA^*$  in comparison with three other real-time search algorithms:  $LRTA^*$ ,  $RTA^*$  both due to Korf [22], and FALCONS [9,10], an algorithm recently proposed by Furcy and Koenig. It is known that  $RTA^*$  often outperforms  $LRTA^*$  in a single problem solving trial, but it does not converge under the repeated problem solving model, not even in the sense of weighted  $LRTA^*$ . In contrast, FALCONS achieves a significant acceleration of convergence over  $LRTA^*$  without losing optimality of converged solutions, but is known to perform poorly in the first trial [9].

### 5.1. Problem domains

The problem domains we use are randomly-generated instances of the Gridworld and the sliding-tile puzzles [30]. As these domains satisfy the undirectedness assumption (Assumption 4.1), upper-bounded  $LRTA^*$  is applicable as well. The setting of each domain is as follows.

*Gridworld* 100 instances of the Gridworlds with 35% obstacle ratio were generated, which are similar to the one in Section 2.3 except that here they consist of  $1000 \times 1000$  cells. Accordingly, the distance between the initial state and the unique goal state is increased to 1000 units as measured by the Manhattan distance. The exact cost between these states is 1067.36 on average over the 100 instances. The Manhattan distance (or the one multiplied by  $1 + \varepsilon$  in the case of weighted  $LRTA^*$ ) is used as the initial heuristic function.

*Sliding-tile puzzles* We used the 100 instances of the Fifteen Puzzle listed in [21]. The mean of the optimal solution costs is 53.05. Because of the large state space,

not all of the Fifteen Puzzle instances could be solved until convergence in our experimental environment.<sup>7</sup> In such cases, we used the 100 Eight Puzzles (whose average optimal solution cost is 22) instead, which were randomly sampled from  $9!/2$  solvable instances. In both cases, the initial heuristic function is the sum of the Manhattan distance between the initial position and the goal for each of the 15 or 8 tiles (multiplied by  $(1 + \varepsilon)$  when used in conjunction with weighted LRTA\*).

These problems can be solved optimally with off-line search methods such as A\* [13] or IDA\* [21], but the task of finding optimal or near-optimal solutions in these domains still provides a good benchmark for real-time search algorithms with one-step look-ahead. Some of these problems are much larger in scale than the problems conventionally used for evaluating the convergence property of such algorithms. For example, [9,10] use the (eight-connected) Gridworlds of size  $20 \times 20$  and the Eight Puzzle. These larger problems are used here to examine the scalability of the proposed methods.

## 5.2. Details of the algorithms

All the algorithms used in the experiments break ties at random in action selection. This choice reflects the fact that the LRTA\* and RTA\* algorithms, which we use as baselines for comparison, may perform worse when deterministic<sup>8</sup> tie-breaking is used [22]. In contrast, the original definition of FALCONS recommends deterministic (systematic) tie-breaking, but even with FALCONS, random tie-breaking produced better results in most of our experiments.

Convergence is detected when no heuristic estimates are updated during a trial, as described in Section 3.5 for weighted LRTA\* (which subsumes LRTA\* when  $\varepsilon = 0$ ). It can be readily seen that the path traversed in such a trial is optimal for upper-bounded LRTA\*, as it behaves like LRTA\* until the travel cost in a trial exceeds  $h^*(s)$  (Theorem 4.12). A parallel proof for FALCONS can be found in [10].

Throughout the experiments, the initial upper bound heuristic used by upper-bounded LRTA\* is the non-informative one given by formula (6). The implementation of upper-bounded LRTA\* incorporates the modification described in Remark 4.7; i.e., it only simulates the updates of  $u(y)$  for each neighbor  $y$  in step (4a) of Algorithm 2. The algorithm is also augmented with the online update scheme described in Section 4.5, which is used to compute the  $u(\cdot)$  estimates from the solution of the first trial; in the first trial only, the distance  $g(x)$  from the initial state is stored in each visited state  $x$ , and the values are used to compute  $u(x) = C - g(x)$  for each  $x$  in later trials on demand. Here,  $C$  is the solution cost of the first trial. The effect of this scheme will be discussed in Section 5.4.3.

<sup>7</sup> The experiments were conducted on two machines, one with an AthlonXP 1800+ processor and 1.5 GB RAM, and the other with dual Xeon 2.2 GHz processors and 4 GB RAM.

<sup>8</sup> By deterministic tie-breaking, we mean that given the same set of tying actions in the same state, the problem solver always chooses the same action.

### 5.3. Performance of weighted LRTA\*

Two experiments were conducted with weighted LRTA\*, to evaluate the effects of weighting on (1) single-trial performance, and (2) convergence process.

#### 5.3.1. First-trial performance

Table 1 shows the first-trial performance of weighted LRTA\* in the Gridworld and Fifteen Puzzle domains with various weight parameter  $\varepsilon$ . It shows the means of the first-trial solution costs over 100 independent trials performed for each value of  $\varepsilon$ . As baselines for comparison, the performances of LRTA\* ( $\varepsilon = 0$ ), RTA\* and FALCONS are included as well.

Table 1  
Means of the solution costs obtained in the first trial, over 100 independent trials. % shows the percentage relative to LRTA\*, and cost/opt shows the factor of the obtained costs relative to the optimal solution costs

(a) Gridworld			
	Cost	%	cost/opt.
$\varepsilon = 0.2$	18119.9	127.6%	17.0
$\varepsilon = 0.4$	18711.1	131.8%	17.5
$\varepsilon = 0.6$	19355.8	136.3%	18.1
$\varepsilon = 0.8$	20712.2	145.9%	19.4
$\varepsilon = 1$	21855.2	153.9%	20.5
$\varepsilon = 1.2$	22788.5	160.5%	21.4
$\varepsilon = 1.4$	23760.1	167.4%	22.3
$\varepsilon = 1.6$	24566.2	173.0%	23.0
$\varepsilon = 1.8$	25268.1	178.0%	23.7
$\varepsilon = 2$	33014.2	232.5%	30.9
LRTA* ( $\varepsilon = 0$ )	14197.7	100%	13.3
RTA*	<b>6574.3</b>	<b>46.3%</b>	<b>6.2</b>
FALCONS	914899.3	6444.0%	857.2
(b) Fifteen Puzzle			
	Cost	%	cost/opt.
$\varepsilon = 0.2$	5551.3	19.6%	104.7
$\varepsilon = 0.4$	5551.3	19.6%	104.7
$\varepsilon = 0.6$	5551.3	19.6%	104.7
$\varepsilon = 0.8$	<b>5392.6</b>	<b>19.0%</b>	<b>101.7</b>
$\varepsilon = 1$	5725.8	20.2%	108.0
$\varepsilon = 1.2$	5804.2	20.4%	109.4
$\varepsilon = 1.4$	5829.8	20.5%	109.9
$\varepsilon = 1.6$	5827.4	20.5%	109.9
$\varepsilon = 1.8$	5827.4	20.5%	109.9
$\varepsilon = 2$	7244.3	25.5%	136.6
LRTA* ( $\varepsilon = 0$ )	28400.2	100%	535.5
RTA*	10677.6	37.6%	201.3
FALCONS	491933.3	1732.1%	9274.8

In the Gridworld,  $RTA^*$  performed best, reducing the solution cost mean to 46% of that of  $LRTA^*$ . The performance of weighted  $LRTA^*$  was inferior to these two algorithms regardless of the value of  $\varepsilon$ , and the solution quality deteriorated as  $\varepsilon$  increased.

In contrast, weighted  $LRTA^*$  outperformed  $RTA^*$  and  $LRTA^*$  in the Fifteen Puzzle domain. It reduced the solution cost mean to about 20% of that of  $LRTA^*$  for a range of  $0.2 \leq \varepsilon \leq 2$ , while  $RTA^*$  reduced the cost to 38%.

In both domains, the performance of FALCONS was an order of magnitude worse than  $LRTA^*$ .

### 5.3.2. Learning process

The next experiment examined the learning process of weighted  $LRTA^*$ , in comparison with  $LRTA^*$  and FALCONS.  $RTA^*$  was excluded from this experiment because the solutions diverge. As the test beds, we used the Eight Puzzle in addition to the Gridworld and the Fifteen Puzzle, because not all the instances of the Fifteen Puzzle could be solved until convergence by the algorithms involved.

The results are reported in Fig. 3, and Tables 2 and 3. They contain the results for  $\varepsilon = 0.2$  through 2, as well as  $LRTA^*$  and FALCONS.

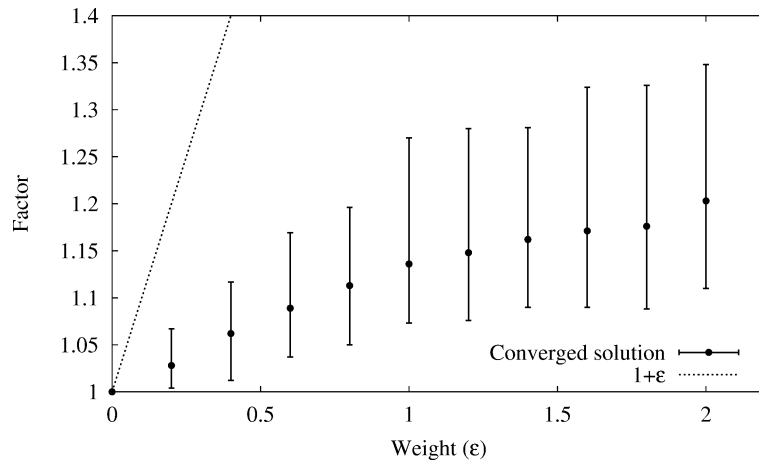
Fig. 3 presents the mean, minimum and maximum of converged solution costs over all solved instances in the Gridworld and the Fifteen Puzzle domains, shown as a factor relative to the optimal solution costs. As argued in Section 3.3, the costs of converged solution paths can be worse than the cost of optimal solutions by a factor of  $(1 + \varepsilon)$ . The figure backs up this claim, as the solution costs were considerably lower than the given upper bounds  $(1 + \varepsilon)$  plotted in the figure as dotted lines.

Table 2 lists the total solution cost, the number of trials and the number of expanded states needed for convergence in the Gridworld and the Eight Puzzle domains. In both domains, these numbers tend to decrease as  $\varepsilon$  increase (but recall that these reductions are achieved at the sacrifice of the quality of converged solutions; cf. Fig. 3). Even with  $\varepsilon = 0.2$ , both the travel cost and the number of expanded states reduce to a tenth that of  $LRTA^*$  in the Gridworld. The reduction in the number of trials by FALCONS over  $LRTA^*$  is due not only to its efficiency, but also to its tendency to make the solution cost in each trial larger until its convergence.

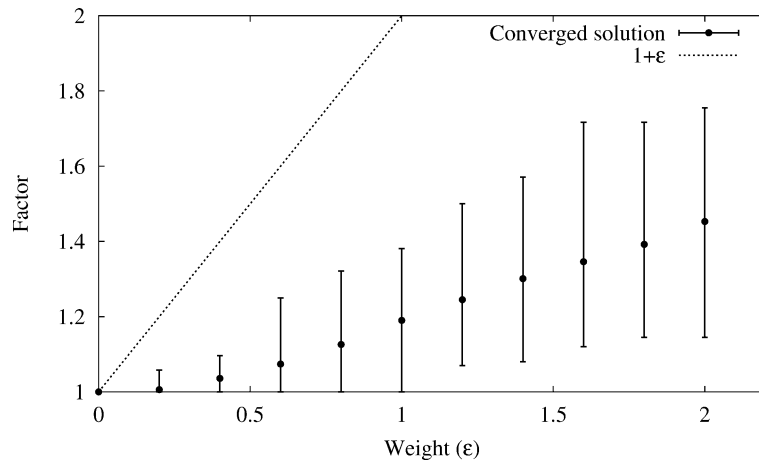
In the Fifteen Puzzle domain, neither  $LRTA^*$  nor FALCONS could solve any of the 100 instances after expanding 40 million states. Weighted  $LRTA^*$  solved 17, 51, 87, and 98 instances when  $\varepsilon = 0.2, 0.4, 0.6,$  and  $0.8$ , respectively, and all 100 instances when  $1 \leq \varepsilon \leq 2$  within the same number of expanded states. The result is summarized in Table 3.

### 5.4. Performance of upper-bounded $LRTA^*$

Since we use the non-informative initial upper bounds, the behavior of upper-bounded  $LRTA^*$  is identical to that of  $LRTA^*$  in the first trial. Hence, we concentrate on its learning process under  $\delta \geq 2$ , and compared it with  $LRTA^*$  and FALCONS, all of which enjoy convergence to optimal solutions.



(a)



(b)

Fig. 3. Quality of converged solutions obtained with weighted LRTA\*. The vertical axis shows the degradation factor relative to the optimal solutions. The dotted lines plot the predicted upper bound ( $1 + \epsilon$ ). (a) Gridworld. (b) Fifteen Puzzle.

#### 5.4.1. Efficiency in convergence

The results for the Gridworld and the Eight Puzzle are shown in Table 4. Upper-bounded LRTA\* reduced the overall travel cost compared to LRTA\* (but not to the level achieved by FALCONS) in these domains, although the properties of upper-bounded LRTA\* proven in Section 4 do not anticipate such a reduction.

Upper-bounded LRTA\* sometimes made the number of trials slightly larger than LRTA\*. This is a side-effect of limiting the amount of exploration performed in each trial.

Meanwhile, the number of expanded states decreased as  $\delta$  decreased, which shows that the search space was effectively reduced by the use of upper bounds. However, because

Table 2

Performance of weighted LRTA\* in repeated trials: the sum of the solution costs, and the numbers of trials and expanded states needed for convergence (mean over 100 instances). Percentages are relative to LRTA\*

(a) Gridworld						
	Cost	%	Trials	%	States	%
$\varepsilon = 0.2$	1678305.8	10.3%	1249.8	22.5%	22508.3	10.6%
$\varepsilon = 0.4$	726585.4	4.5%	510.1	9.2%	14752.3	7.0%
$\varepsilon = 0.6$	453374.6	2.8%	299.7	5.4%	11446.0	5.4%
$\varepsilon = 0.8$	371428.4	2.3%	233.8	4.2%	9970.8	4.7%
$\varepsilon = 1$	307136.8	1.9%	181.3	3.3%	8907.5	4.2%
$\varepsilon = 1.2$	300991.7	1.9%	176.2	3.2%	8384.5	4.0%
$\varepsilon = 1.4$	288821.9	1.8%	166.3	3.0%	7781.6	3.7%
$\varepsilon = 1.6$	263416.2	1.6%	146.0	2.6%	7410.1	3.5%
$\varepsilon = 1.8$	258987.0	1.6%	142.2	2.6%	7153.3	3.4%
$\varepsilon = 2$	148794.2	0.9%	49.9	0.9%	6851.9	3.2%
LRTA* ( $\varepsilon = 0$ )	16256719.3	100%	5557.4	100%	212222.5	100%
FALCONS	9584184.7	59.0%	2680.6	48.2%	183707.2	86.6%
(b) Eight Puzzle						
	Cost	%	Trials	%	States	%
$\varepsilon = 0.2$	53513.1	59.7%	384.3	116.9%	13813.7	43.6%
$\varepsilon = 0.4$	41251.5	46.0%	275.5	83.8%	11053.2	34.9%
$\varepsilon = 0.6$	33309.2	37.2%	217.1	66.1%	9029.2	28.5%
$\varepsilon = 0.8$	27970.1	31.2%	181.6	55.3%	7598.7	24.0%
$\varepsilon = 1$	24982.8	27.9%	156.9	47.7%	6705.9	21.2%
$\varepsilon = 1.2$	22342.0	24.9%	146.9	44.7%	5866.2	18.5%
$\varepsilon = 1.4$	20637.1	23.0%	136.6	41.6%	5351.4	16.9%
$\varepsilon = 1.6$	19769.7	22.1%	129.6	39.4%	5130.1	16.2%
$\varepsilon = 1.8$	19677.6	22.0%	129.0	39.3%	5085.4	16.0%
$\varepsilon = 2$	16011.7	17.9%	71.2	21.7%	3584.2	11.3%
LRTA* ( $\varepsilon = 0$ )	89635.4	100%	328.6	100%	31693.5	100%
FALCONS	37564.0	41.9%	82.3	25.0%	16811.7	53.0%

Table 3

Number of Fifteen Puzzle instances ( $\varepsilon$ -)optimally solved within the limit of 40 million state expansions (out of 100 problem instances)

	FALCONS	LRTA*	Weighted LRTA*				
			$\varepsilon = 0.2$	$\varepsilon = 0.4$	$\varepsilon = 0.6$	$\varepsilon = 0.8$	$1 \leq \varepsilon \leq 2$
# solved	0	0	17	51	87	98	100

upper-bounded LRTA\* maintains two estimates for each state, it must lower the number of expanded states below half that of LRTA\* to be more memory efficient. Consequently, upper-bounded LRTA\* in the Gridworld domain is inferior to LRTA\* in this respect, as the reduction rate was only 15% at maximum. The same applies to FALCONS which also uses two heuristic functions. In contrast, the use of upper bound estimates paid off in the Eight Puzzle, because upper-bounded LRTA\* with  $\delta = 2$  expanded only 46% of states expanded by LRTA\*.

Table 4

Performance of upper-bounded LRTA\* in repeated trials: the sum of the solution costs, and the numbers of trials and expanded states needed for convergence (mean over 100 instances). Percentages are relative to LRTA\*

(a) Gridworld						
	Cost	%	Trials	%	States	%
$\delta = 2$	13570647.7	83.5%	5567.3	100.2%	<b>182978.9</b>	<b>86.2%</b>
$\delta = 4$	14582489.5	89.7%	5548.8	99.8%	192827.2	90.9%
$\delta = 10$	15673326.9	96.4%	5554.6	99.9%	206115.1	97.1%
LRTA*	16256719.3	100%	5557.4	100%	212222.5	100%
FALCONS	<b>9584184.7</b>	<b>59.0%</b>	<b>2680.6</b>	<b>48.2%</b>	183707.2	86.6%
(b) Eight Puzzle						
	Cost	%	Trials	%	States	%
$\delta = 2$	48048.8	53.6%	317.6	96.6%	<b>14502.7</b>	<b>45.8%</b>
$\delta = 4$	64978.6	72.5%	344.0	104.7%	21453.6	67.7%
$\delta = 10$	82701.6	92.3%	334.0	101.6%	28926.5	91.3%
LRTA*	89635.4	100%	328.6	100%	31693.5	100%
FALCONS	<b>37564.0</b>	<b>41.9%</b>	<b>82.3</b>	<b>25.0%</b>	16811.7	53.0%

Table 5

Number of Fifteen Puzzle instances optimally solved (out of 100 problem instances)

	LRTA*	FALCONS	Upper-bounded LRTA* ( $\delta = 2$ )
# expanded states	$8 \times 10^7$	$4 \times 10^7$	$4 \times 10^7$
# instances solved	0	0	32

The effect of upper bound heuristics was more evident in the Fifteen Puzzle (Table 5). While neither LRTA\* nor FALCONS could optimally solve any of the 100 instances after expanding 40 million states, upper-bounded LRTA\* with  $\delta = 2$  found optimal solutions for 32 instances within the same limit on the number of expanded states. Since upper-bounded LRTA\* (and FALCONS) require twice the memory consumed by LRTA\* when the numbers of expanded states are equal, we further ran LRTA\* until it expanded 80 million states for fair comparison. The result was that none of the instances were solved optimally either.

#### 5.4.2. Stability of the convergence process

The stability of the convergence process consists of factors such as frequency of overshoots, their amplitude, and time required for convergence. Because the relative importance of these factors depends on the application domain, we use several different performance indices to measure stability.

The first four performance indices are adopted from optimal control theory [7]: the integral (or sum, in discrete time case) of absolute error (IAE), the integral (or sum) of square error (ISE), and their time-weighted variants (ITAE and ITSE). These indices are used for tuning feedback control systems, in a way that the optimal settings are obtained by minimizing them. The following are the definitions of these indices adapted to our experimental setting. Here,  $c(i)$  denotes the solution cost for the  $i$ th trial.

$$\begin{aligned} \text{IAE} &= \sum_{i=1}^{\infty} |c(i) - h^*(s)|, \\ \text{ISE} &= \sum_{i=1}^{\infty} (c(i) - h^*(s))^2, \\ \text{ITAE} &= \sum_{i=1}^{\infty} i |c(i) - h^*(s)|, \\ \text{ITSE} &= \sum_{i=1}^{\infty} i (c(i) - h^*(s))^2. \end{aligned}$$

As shown above, integrals are changed to sums because we interpret each (discrete) trial as a time point. IAE provides essentially the similar performance measure as the sum of the solution costs (see Table 4), as seen from the above definition. In contrast, ISE penalizes large overshoots more severely and the two time-weighted versions impose large penalties on sustained errors.

In addition, we compute a new index which we denote SOD (sum of one-sided difference) given below, which sums up the difference in solution costs between two consecutive trials but only when the solution worsens.

$$\text{SOD} = \sum_{i=1}^{\infty} \max[0, c(i+1) - c(i)].$$

SOD takes its minimum value 0 if the convergence process is monotonic (i.e., the solution costs are non-increasing over trials), no matter how fast or slow convergence is. Thus, unlike the first four performance indices from control theory, SOD does not take the number of trials into account, but it provides an insight into the degree of ‘smoothness’ of the process which is not captured by the first four indices.

Table 6 lists the computed performance indices for each algorithm in the Gridworld and the Eight Puzzle domains. Upper-bounded LRTA\* with  $\delta = 2$  outperformed LRTA\* on all indices, and FALCONS on ISE, ITSE, and SOD. The advantage of FALCONS with respect to ITAE owes to its performing a great deal of moves in earlier trials when weight (trial number)  $i$  is small, which also has the merit of keeping the total number of trials small. However, the number of moves made by FALCONS in earlier trials is so large (see Table 1) that when errors are squared and overshoots are penalized more severely as in ISE or ITSE, it becomes inferior to upper-bounded LRTA\*.

#### 5.4.3. Effect of upper bounds obtained from the first-trial solutions

How crucial is the quality of the initial upper bound estimates to the learning performance of upper-bounded LRTA\*? To see this, we examine the value of  $u(s)$  of the initial state  $s$  at the beginning of the second trial in the learning episodes, which was obtained with the online update scheme described in Section 4.5 to make  $u(s)$  finite.

In the Gridworld domain, the initial value of  $u(s)$  in the second trial was a factor of 13 larger than the optimal solution cost on average, yet upper-bounded LRTA\* reduced the overall cost for convergence and stabilized the convergence process compared with

Table 6  
Performance indices for stability. The figures are means over 100 instances in each domain

(a) Gridworld					
	IAE ( $\times 10^6$ )	ISE ( $\times 10^{10}$ )	ITAE ( $\times 10^{11}$ )	ITSE ( $\times 10^{13}$ )	SOD ( $\times 10^6$ )
$\delta = 2$	7.6	<b>3.2</b>	19.4	<b>5.9</b>	<b>2.8</b>
$\delta = 4$	8.6	4.1	23.9	9.5	3.5
$\delta = 10$	9.7	5.9	28.8	17.5	4.5
LRTA*	9.4	9.4	31.3	32.8	5.1
FALCONS	<b>6.7</b>	389.4	<b>5.5</b>	21.7	3.6
(b) Eight Puzzle					
	IAE ( $\times 10^6$ )	ISE ( $\times 10^9$ )	ITAE ( $\times 10^8$ )	ITSE ( $\times 10^{11}$ )	SOD ( $\times 10^3$ )
$\delta = 2$	4.1	<b>1.1</b>	8.8	<b>1.0</b>	<b>9.3</b>
$\delta = 4$	5.8	1.7	14.8	2.5	16.6
$\delta = 10$	7.7	3.0	19.8	5.7	30.5
LRTA*	8.2	4.0	20.7	8.3	38.3
FALCONS	<b>3.6</b>	3.5	<b>2.6</b>	1.9	16.2

LRTA\*. Therefore, we can expect that upper bounds obtained from prior knowledge can be effective as well, even if they are typically several times larger than the optimal solutions.

In the Fifteen Puzzle, by contrast, the scheme of Section 4.5 yielded the value of  $u(s)$  722 times larger than the optimal solution cost, on average over the 32 problem instances solved. As this value is too large to be effective as an upper bound, the algorithm behaved exactly like LRTA\* in earlier trials. It turns out that these instances were optimally solved not because the initial values of  $u(\cdot)$  provided a good estimate of  $h^*(\cdot)$ , but because their values lowered in later trials as a result of normal value-update operations in Algorithm 2. Thus, upper-bounded LRTA\* can still be effective in some domains in which the first-trial solutions do not provide effective upper bound estimates, or no prior knowledge is available on upper bounds.

### 5.5. Combining the weighted and the upper-bounded LRTA\* methods

Weighted LRTA\* and upper-bounded LRTA\* provide two different ways of generalizing LRTA\*, but the techniques used in them are compatible. We can create a merged algorithm which is identical to upper-bounded LRTA\*, except that an  $\varepsilon$ -admissible initial heuristic function  $h_0(\cdot)$  is used instead of an admissible one. The resulting method, called  $\varepsilon\delta$ -LRTA\*, reduces the overall amount of learning at the sacrifice of the quality of converged solutions, and stabilizes the convergence process with the help of upper bound heuristic estimates. Fig. 4 shows a typical learning process of  $\varepsilon\delta$ -LRTA\* (with  $\varepsilon = 0.4$  and  $\delta = 2$ ; single episode) in the  $100 \times 100$  Gridworld of Fig. 1. For ease of comparison, a learning process of LRTA\* (which plots the same data as in Fig. 2) is also shown in the same graph.

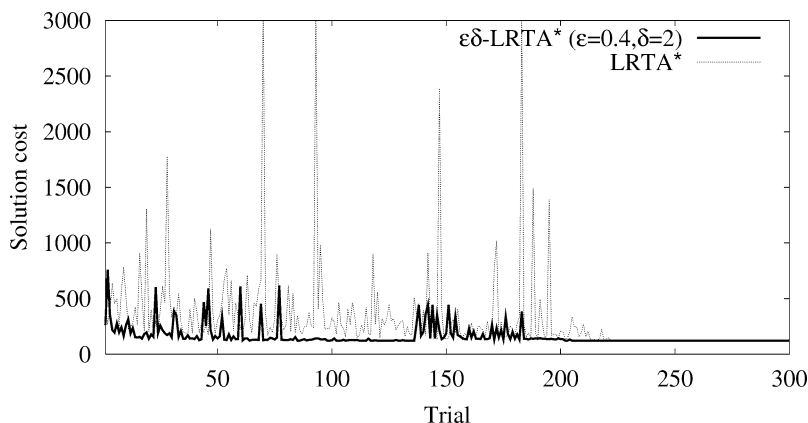


Fig. 4. Typical learning processes of LRTA\* and  $\epsilon\delta$ -LRTA\* (single episode).

## 6. Related work

### 6.1. Weighted off-line search algorithms

As explained earlier, LRTA\* and weighted LRTA\* differ only in the class of the initial heuristic functions used and are algorithmically identical. In particular, weighted LRTA\* subsumes LRTA\* when  $\epsilon = 0$ . A similar relation can be observed between the two well-known off-line search algorithms, namely, Hart et al.'s A\* [13] and Pohl's *heuristic path algorithm (HPA)* [28,29] (also known as *weighted A\** [23]). HPA uses the evaluation function  $f(x) = (1 - W)g(x) + Wh(x)$  to choose which states to expand next, where  $g(x)$  is the cost of the current path from the initial state to a state  $x$ , and  $h(x)$  is an admissible heuristic estimate to the goals (but unlike real-time search,  $h(x)$  is stationary). By substituting  $W = (1 + \epsilon)/(2 + \epsilon)$ , we see that this is equivalent to the A\* algorithm (which uses  $f(x) = g(x) + h(x)$  for evaluation) with  $\epsilon$ -admissible  $h(\cdot)$ . It has been proven that HPA terminates after finding an  $\epsilon$ -optimal solution [6]. Moreover, HPA often yields a drastic reduction in the amount of search [23], and is successfully used in Bonet and Geffner's HSP2 planner [4].

If LRTA\* is a real-time counterpart of the off-line search algorithm A\*, weighted LRTA\* can be regarded as a real-time version of Pohl's algorithm, both in terms of configurations and effects. However, while they are conceptually similar, it should be noted that real-time search and off-line search rest on different search models and their properties do not necessarily transfer to each other. For example, in weighted LRTA\*, the set of expanded states after convergence need not be identical to that of HPA with the same  $\epsilon$ , and unlike HPA, the completeness of weighted LRTA\* is not guaranteed in an infinite state space (see [22]).

Harris's bandwidth search [12] is another off-line heuristic search algorithm that makes use of non-admissible heuristic functions. The difference between the bandwidth search and Pohl's HPA is that while the latter constructs a heuristic function by multiplying an admissible heuristic function by  $(1 + \epsilon)$ , the bandwidth search assumes that in every state,

the overestimated heuristic value is within some constant additive factor of  $\epsilon$  of the exact value (i.e.,  $\forall x. h(x) \leq h^*(x) + \epsilon$ ). We have shown in Section 3.4 that a composition of a similar extension with LRTA\* (weighted LRTA\* with  $\epsilon$ -additive weight) is also possible.

## 6.2. Korf's RTA\*

In his pioneering real-time search paper [22], Korf introduced the real-time A\* (RTA\*) algorithm as well as LRTA\*. The key difference between LRTA\* and RTA\* is that the latter updates the heuristic value for the problem solver's current state  $x$  by the *second* lowest value among the quantities  $k(x, y) + h(y)$  of successor states  $y$ . Consequently, the heuristic values may overestimate the exact costs, even if they are initially admissible. In many cases, this allows RTA\* to find better solution than LRTA\* in the first problem solving trial (see Section 5.3.1). However, since there is no bound on the value of  $h(\cdot)$  in each state, the solutions obtained with RTA\* are not convergent under the repeated problem solving model. Later, Edelkamp and Eckerle [8] modified RTA\* to allow learning over repeated problem solving trials. Their algorithms, CRTA\* and SLRTA\*, use techniques similar to the ones described in Section 4.5.

## 6.3. Russell and Wefald's LDTA\*

The perspective of real-time search as an agent architecture was first pursued by Russell and Wefald [31]. They investigated the rationality of real-time search agents especially in terms of the trade-off between reactivity and deliberativeness in a single problem solving trial. They also discuss learning with their LDTA\* algorithm, but in a quite different sense from the convergence of LRTA\*. In their case, learning is intended to acquire meta-level knowledge for controlling how look-ahead search is performed. These algorithms are probably not easily comparable because of the different objectives. In fact, Russell and Wefald do not refer to LRTA\* in their work, but instead use RTA\* as the baseline for comparison.

## 6.4. Furcy and Koenig's FALCONS

Similarly to upper-bounded LRTA\*, Furcy and Koenig's FALCONS algorithm [9] uses an extra heuristic function in addition to  $h(\cdot)$ . Unlike upper-bounded LRTA\*, the new heuristic function of FALCONS maintains lower bounds for the cost from the *initial* state to each state in the state space. With the help of this new heuristic function, it achieved a significant acceleration of convergence, as we saw in Section 5.

Although our proposed methods sped up the convergence as well, it should be noted that FALCONS and these methods were designed with different objectives in mind. The goal of FALCONS is to speed up convergence while retaining optimality of converged solutions, whereas weighted LRTA\* intends to trade off optimality for resource-boundedness; speedup was not the primary goal of weighted LRTA\*, but a by-product of resource-boundedness consideration. In fact, the convergence process of FALCONS is quite distinct from that of LRTA\* and weighted LRTA\*, in that it performs an enormous amount of exploration in earlier trials. This strategy brings about faster convergence, but is

accompanied by poor performance in earlier trials.<sup>9</sup> FALCONS can find optimal solutions, but because of this fact, it is unlikely that it can escape from intractability present in most AI problems. We expect the weighting technique to be applicable to FALCONS to cope with these problems.

The main objective of upper-bounded LRTA\* was not speedup either. Indeed, it can sometimes make convergence slower than LRTA\* (and FALCONS) to achieve a more stable convergence process. Both FALCONS and upper-bounded LRTA\* use additional heuristic estimates. However, FALCONS does not suffer from the major limitation of upper-bounded LRTA\*; i.e., it is equally applicable to directed as well as undirected state spaces.

## 7. Summary and future directions

In this paper, we have discussed the behavior of real-time search problem solvers during convergence. The flexibility of real-time search algorithms is attractive for autonomous agents, but when it comes to learning capability, the behavior of the algorithms is not always acceptable. In particular, the LRTA\* algorithm performs excessive exploration, and incurs instability of solution quality during convergence. We have argued that this unfavorable behavior exemplifies the lack of rationality consideration in the algorithm. As a remedy, we have proposed and implemented two techniques, both of which are designed to make the problem solver behave more rationally and to allow for fine-grained control over the convergence process.

The first of the proposed methods, weighted LRTA\* adds small weights to initial heuristic estimates to avoid spending too much effort on minor performance improvements. A significant speedup in convergence process as well as saving in memory are achieved at the sacrifice of the quality of converged solutions. It appeals to problems in which optimality of solutions is dispensable, or in which optimality has to be given up because of the inherent intractability of the task.

The second method, upper-bounded LRTA\*, is applicable to undirected state spaces. After an upper bound on the solution cost is obtained, it guarantees stable performance improvements and eventually converge to optimal solutions (when  $\delta \geq 2$ ). This property shows that upper-bounded LRTA\* can effectively balance future investments and the present problem solving efficiency. Our experimental result shows that upper-bounded LRTA\* can optimally solve the Fifteen Puzzle with less memory compared with LRTA\* and FALCONS. This does not mean all large scale problems can be solved with upper-bounded LRTA\*, but at least suggest that it is a promising approach.

Future research directions include: overcoming inability to deal with directed state space that is inherent in upper-bounded LRTA\*; application of weighting techniques to other real-time search algorithms, possibly in non-deterministic domains; and developing a better scheme for combining the weighted and the upper-bounded LRTA\* methods with

---

<sup>9</sup> A newer version called eFALCONS [11] addresses this problem by using a total of four estimates. We did not test the algorithm because the reported improvement over the original FALCONS remained within 2–5% in terms of the travel cost, despite the large overhead in memory usage.

automatic parameter adjustment. Another topic of interest is whether these methods, when combined with deeper (e.g., minimin [22]) look-ahead search, can solve larger problems such as the Twenty-four or Thirty-five Puzzles.

## Acknowledgements

We would like to thank Edson T. Miyamoto for his thorough proofreading and suggestions. We are also grateful to David Furcy for answering our questions concerning FALCONS, and to the anonymous reviewers for detailed and helpful comments.

## Appendix A. Proofs

### A.1. Proofs for weighted LRTA\*

In a given problem solving trial of weighted LRTA\*, recall that by time  $t$ , we mean the moment of time immediately after the  $t$ th iteration of step (1) in Algorithm 1, and time  $t = 0$  denotes the moment before the first iteration of step (1). Let  $x_t$  be the state occupied by the problem solver at time  $t$ , and, for any state  $x \in X$ , let  $h_t(x)$  be the value of heuristic estimate  $h(x)$  at time  $t$ .

**Lemma A.1** (Lemma 3.3). *For each time instant  $\tau \geq 0$ , the following relation (A.1) holds.*

$$\sum_{t=1}^{\tau} k(x_{t-1}, x_t) \leq \sum_{x \in X} h_{\tau}(x) - h_{\tau}(x_{\tau}) - \sum_{x \in X} h_0(x) + h_0(s). \quad (\text{A.1})$$

**Proof.** The statement of the lemma is obviously true for  $\tau = 0$ , so we concentrate on the case where  $\tau > 0$ .

First note that according to rules (1) and (2) in Algorithm 1, the following equation holds for every time  $t \geq 1$ .

$$h_t(x_{t-1}) \geq \min_{y \in \text{Succ}(x_{t-1})} [k(x_{t-1}, y) + h(y)] = k(x_{t-1}, x_t) + h_{t-1}(x_t). \quad (\text{A.2})$$

On the other hand, the update accompanying the  $t$ th move does not alter the heuristic estimates for states other than  $x_{t-1}$ . Thus,

$$h_t(x) = h_{t-1}(x) \quad (\text{A.3})$$

for each  $x \in X - \{x_{t-1}\}$ . By Assumption 2.5, we have  $x_t \neq x_{t-1}$  for each time  $t \geq 1$ . It follows that  $h_t(x_t) = h_{t-1}(x_t)$  and hence Eq. (A.2) can be restated as

$$h_t(x_{t-1}) \geq k(x_{t-1}, x_t) + h_t(x_t). \quad (\text{A.4})$$

The fact that  $x_{t-1}$  is the only state whose heuristic estimate is updated on the  $t$ th move yields

$$\begin{aligned}
\sum_{x \in X} h_t(x) &= \sum_{x \in X} h_{t-1}(x) - h_{t-1}(x_{t-1}) + h_t(x_{t-1}) \\
&\geq \sum_{x \in X} h_{t-1}(x) - h_{t-1}(x_{t-1}) + k(x_{t-1}, x_t) + h_t(x_t)
\end{aligned} \tag{A.5}$$

for all  $t \geq 1$ , where the inequality follows from Eqs. (A.3) and (A.4). Substituting

$$H_t = \sum_{x \in X} h_t(x) - h_t(x_t)$$

and rearranging terms, we can restate Eq. (A.5) as

$$k(x_{t-1}, x_t) \leq H_t - H_{t-1}.$$

Summing this inequality over  $t = 1, 2, \dots, \tau$  yields

$$\sum_{t=1}^{\tau} k(x_{t-1}, x_t) \leq H_{\tau} - H_0,$$

which is the statement of the lemma. Note that  $s = x_0$ .  $\square$

## A.2. Proofs for upper-bounded LRTA\*

The state space considered in this subsection is undirected, in the sense of Assumption 4.1.

**Proposition A.2** (Proposition 4.3). *Suppose we perform a series of updates of  $u(\cdot)$  according to rule (7), with each update using an arbitrary pair of neighboring states. If the following statements initially hold, then they continue to hold at any subsequent moment.*

- (1) For every state  $x \in X$ ,  $u(x)$  is an upper bound for  $h^*(x)$ , or,  $u(x) \geq h^*(x)$ .
- (2)  $u(\cdot)$  is anti-consistent. That is, for any non-goal state  $x \in X - G$  in which  $u(x)$  is finite, there exists a neighbor  $y \in \text{Ngh}(x)$  of  $x$  such that

$$u(x) \geq k(x, y) + u(y).$$

**Proof.** All the proofs are by induction on the number  $j$  of updates by rule (7). For each state  $x$  and  $j = 0, 1, \dots$ , let  $u_j(x)$  denote the value of  $u(x)$  before  $(j + 1)$ -st update. The induction bases for  $j = 0$  are given. We show that if  $u_{j-1}(\cdot)$  meets the above conditions, then they hold for  $u_j(\cdot)$  as well. Assume that in the  $j$ th update, the estimate for state  $x$  is updated in reference to a neighbor  $y$ , or equivalently,

$$u_j(x) = k(x, y) + u_{j-1}(y). \tag{A.6}$$

Since  $x$  is the only state whose estimate is modified on the  $j$ th update, we have for any state  $z \neq x$ ,

$$u_j(z) = u_{j-1}(z).$$

- (1) Assume we have  $u_{j-1}(z) \geq h^*(z)$  for all  $z \in X$  as the induction hypothesis, and in particular  $u_{j-1}(y) \geq h^*(y)$ . Plugging this inequality into (A.6) yields

$$u_j(x) \geq k(x, y) + h^*(y).$$

On the other hand, by Bellman's optimality equation, we have

$$h^*(x) = \min_{z \in \text{Ngh}(x)} k(x, z) + h^*(z) \leq k(x, y) + h^*(y).$$

Combining the above two inequalities yields  $u_j(x) \geq h^*(x)$ .

- (2) The change of the value from  $u_{j-1}(x)$  to  $u_j(x)$  in state  $x$  may only violate the constraints between  $x$  and its neighboring states. Thus, it remains to show that there exists a state  $z \in \text{Ngh}(x)$  such that

$$u_j(x) \geq k(x, z) + u_j(z), \quad (\text{A.7})$$

and for each state  $w \in \text{Ngh}(x)$  in which  $u_{j-1}(w)$  is finite and

$$u_{j-1}(w) \geq k(w, x) + u_{j-1}(x), \quad (\text{A.8})$$

we have

$$u_j(w) \geq k(w, x) + u_j(x). \quad (\text{A.9})$$

Inequality (A.7) follows from (A.6) by substituting  $z = y$ . Since  $u(\cdot)$  is nonincreasing, we have  $u_{j-1}(x) \geq u_j(x)$ . Substituting this and  $u_j(w) = u_{j-1}(w)$  in inequality (A.8) yields (A.9).  $\square$

**Lemma A.3.** *Every  $u$ -path is acyclic.*

**Proof.** Suppose on the contrary that there exists a cyclic  $u$ -path  $(x_0, x_1, \dots, x_n)$ , where  $x_0 = x_n$ . Then by the definition of  $u$ -path, for all  $j = 0, \dots, n-1$ , we have

$$k(x_j, x_{j+1}) + u(x_{j+1}) = \min_{y \in \text{Ngh}(x_j)} [k(x_j, y) + u(y)].$$

According to the anti-consistency of  $u(\cdot)$  given by Proposition 4.3(2), the RHS in the above equation is upper-bounded by  $u(x_j)$ . Hence, we have

$$k(x_j, x_{j+1}) + u(x_{j+1}) \leq u(x_j)$$

for all  $j = 0, \dots, n-1$ . Since  $k(\cdot, \cdot)$  is positive by Assumption 2.2, we have  $u(x_{j+1}) < u(x_j)$  for all  $j = 0, \dots, n-1$ . Chaining this inequality over  $j = 0, \dots, n-1$  yields  $u(x_n) < u(x_0)$ , but this contradicts the supposition that  $x_0 = x_n$ .  $\square$

**Proposition A.4** (Proposition 4.5). *If  $u(\cdot)$  initially satisfies the conditions of Proposition 4.3 and is updated exclusively by rule (7), then at any moment and for any state  $x \in X$  such that  $u(x)$  is finite, the cost of any  $u$ -path is at most  $u(x)$ .*

**Proof.** According to Lemma A.3, every  $u$ -path is acyclic, and since the number  $|X|$  of states is finite by Assumption 2.1, the length of any  $u$ -path is finite and always terminates at a goal.

Let  $\pi = (x_0, x_1, \dots, x_n)$  be a  $u$ -path from a state  $x$ , where  $x = x_0$  and  $x_n \in G$ . By the definition of  $u$ -paths, we have for all  $1 \leq j \leq n$ ,

$$k(x_{j-1}, x_j) + u(x_j) = \min_{y \in \text{Ngh}(x_{j-1})} k(x_{j-1}, y) + u(y). \quad (\text{A.10})$$

The anti-consistency of  $u(\cdot)$  given by Proposition A.2(2) states that for each state in  $x_j$ , there must be a nonempty set of neighbors  $y$  such that  $u(x_j) \geq k(x_j, y) + u(y)$ , which must include  $x_{j+1}$  according to Eq. (A.10). Therefore,

$$u(x_{j-1}) \geq k(x_{j-1}, x_j) + u(x_j)$$

holds for all  $1 \leq j \leq n$ . Summing this inequality over  $j = 1, \dots, n$  and substituting  $x = x_0$  yields

$$u(x) \geq \sum_{j=1}^n k(x_{j-1}, x_j) + u(x_n) = \sum_{j=1}^n k(x_{j-1}, x_j),$$

where the last equality follows from the fact that  $x_n$  is a goal state. Noting that the RHS restates the cost of  $\pi$ , we see that  $u(x)$  upper bounds the cost of any  $u$ -path from  $x$ .  $\square$

### A.3. Convergence of LRTA\* with cutoffs

Consider a repeated application of LRTA\* to a state space of Section 2.1. For each trial  $i = 1, 2, \dots$ , let  $\tau(i)$  be the number of moves the problem solver has made in the  $i$ th trial, and  $d(i)$  be the cost of the path traversed by the problem solver in the same trial. We denote by  $x_j^i$  the state at which the problem solver has arrived by the  $j$ th move in the  $i$ th trial (hence  $0 \leq j \leq \tau(i)$ ), and for any  $x \in X$  denote by  $h_j^i(x)$  the heuristic estimate  $h(x)$  at the same instant. Using these definitions, we have

$$d(i) = \sum_{j=1}^{\tau(i)} k(x_{j-1}^i, x_j^i).$$

In the following, we set a deadline for arriving at a goal in each trial. Unlike the repeated problem solving scenario described in Section 2.2, a trial may be cut off before a goal is reached. However, the cutoff should not be too restrictive, as it is hopeless to expect the problem solver to ever reach a goal if the cutoff threshold is less than the cost of the optimal solutions. This requirement is guaranteed by making the explicit assumption on the cost spent in each trial as follows.

**Assumption A.5.** For each trial  $i$ , the cost  $d(i)$  of the path traversed in the trial satisfies

$$d(i) \geq h^*(s),$$

where the equality holds iff  $x_{\tau(i)} \in G$ , or equivalently, the problem solver reaches a goal in the trial through an optimal path.

**Lemma A.6.** For each trial  $n = 1, 2, \dots$ , the following inequality holds.

$$\sum_{i=1}^n d(i) \leq \sum_{x \in X} h_{\tau(n)}^n(x) - \sum_{x \in X} h_0^1(x) + \sum_{i=1}^n [h_0^i(s) - h_{\tau(i)}^i(x_{\tau(i)}^i)].$$

**Proof.** First note that since the heuristic estimates at the end of a trial is reused as the initial heuristics in the next trial, we have for any trial  $i = 1, 2, \dots$  and any state  $x \in X$ ,

$$h_{\tau(i)}^i(x) = h_0^{i+1}(x). \quad (\text{A.11})$$

Now for each trial  $i = 1, 2, \dots$ , Lemma A.1 holds. Hence,

$$d(i) \leq \sum_{x \in X} h_{\tau(i)}^i(x) - h_{\tau(i)}^i(x_{\tau(i)}) - \sum_{x \in X} h_0^i(x) + h_0^i(s).$$

Summing this inequality over  $i = 1, 2, \dots, n$  yields

$$\begin{aligned} \sum_{i=1}^n d(i) &\leq \sum_{x \in X} h_{\tau(n)}^n(x) + \sum_{i=1}^{n-1} \sum_{x \in X} [h_{\tau(i)}^i(x) - h_0^{i+1}(x)] - \sum_{x \in X} h_0^1(x) \\ &\quad + \sum_{i=1}^n h_0^i(s) - \sum_{i=1}^n h_{\tau(i)}^i(x_{\tau(i)}). \end{aligned}$$

Since the second term in the RHS is 0 by Eq. (A.11), and  $h_t^i(x) \geq 0$  for any  $i, t$  and  $x$ , the statement of the lemma follows.  $\square$

**Proposition A.7** (Proposition 4.13). *LRTA\* converges to optimal solutions in the following scenario. Each trial may be terminated as soon as the cost of the path traversed so far in the trial exceeds a certain cutoff, regardless of whether a goal state has been reached. After a trial is terminated this way, the problem solver is reset to the initial state and a new trial is started as usual. The cutoff may vary between trials, as long as either (i) it is strictly greater than  $h^*(s)$ , or (ii) it is greater than or equal to  $h^*(s)$  and an additional update operation of rule (1) is performed in the non-goal state at which the trial is cut off.*

**Proof.** From Lemma A.6, we have

$$\begin{aligned} &\sum_{i=1}^n [d(i) - h^*(s)] \\ &\leq \sum_{x \in X} h_{\tau(n)}^n(x) - \sum_{x \in X} h_0^1(x) + \sum_{i=1}^n [h_0^i(s) - h_{\tau(i)}^i(x_{\tau(i)})] - \sum_{i=1}^n h^*(s) \\ &\leq \sum_{x \in X} h^*(x) - \sum_{x \in X} h_0^1(x), \end{aligned} \quad (\text{A.12})$$

for every trial  $n$ , where  $h_t^i(x) \leq h^*(x)$  for any  $x \in X$ ,  $t \geq 0$ , and  $i \geq 1$  (which follows from Lemma 3.2 as a particular case of  $\varepsilon = 0$ ) is used to derive the last inequality. By Assumption A.5,  $\sum_i [d(i) - h^*(s)]$  is a series of nonnegative terms, and since the series is upper bounded as given by (A.12), it is convergent. Thus we have

$$\lim_{n \rightarrow \infty} [d(n) - h^*(x_0^n)] = 0,$$

which implies that there exists a trial  $m$  such that  $d(n) = h^*(s)$  for all trial  $n \geq m$ , as the state space is finite. The proposition follows from Assumption A.5, in particular, that  $d(n) = h^*(s)$  iff it traverses an optimal path and arrives at a goal in trial  $n$ .  $\square$

## References

- [1] A.G. Barto, S.J. Bradtke, S.P. Singh, Learning to act using real-time dynamic programming, *Artificial Intelligence* 72 (1–2) (1995) 81–138.
- [2] D.P. Bertsekas, J.N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, Prentice Hall, Englewood Cliffs, NJ, 1989.
- [3] B. Bonet, H. Geffner, Learning sorting and decision trees with POMDPs, in: *Proc. Fifteenth International Conference on Machine Learning (ICML'98)*, 1998, pp. 73–81.
- [4] B. Bonet, H. Geffner, Planning as heuristic search, *Artificial Intelligence* 129 (1–2) (2001) 5–33.
- [5] P. Dasgupta, P.P. Chakrabarti, S.C. DeSarkar, Agent search in a tree and the optimality of iterative deepening, *Artificial Intelligence* 71 (1994) 195–208.
- [6] H.W. Davis, A. Bramanti-Gregor, J. Wang, The advantages of using depth and breadth components in heuristic search, in: *Proceedings of the Third International Symposium on Methodologies for Intelligent Systems*, Turin, Italy, North-Holland, Amsterdam, 1988, pp. 19–28.
- [7] R.C. Dorf, R.H. Bishop, *Modern Control Systems*, 7th Edition, Addison-Wesley, Reading, MA, 1995.
- [8] S. Edelkamp, J. Eckerle, New strategies in learning real time heuristic search, in: *On-line Search: Papers from AAAI Workshop*, Providence, RI, AAAI Press, 1997, pp. 30–35.
- [9] D. Furcy, S. Koenig, Speeding up the convergence of real-time search, in: *Proc. AAAI-2000*, Austin, TX, 2000, pp. 891–897.
- [10] D. Furcy, S. Koenig, Speeding up the convergence of real-time search: empirical setup and proofs, Technical Report GIT-COGSCI-2000/01, College of Computing, Georgia Institute of Technology, Atlanta, GA, 2000.
- [11] D. Furcy, S. Koenig, Combining two fast-learning real-time search algorithms yields even faster learning, in: *Proc. Sixth European Conference on Planning (ECP-01)*, Toledo, Spain, 2001.
- [12] L.R. Harris, The heuristic search under conditions of error, *Artificial Intelligence* 5 (3) (1974) 217–234.
- [13] P.E. Hart, N.J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost path, *IEEE Trans. Systems Sci. Cybernet. (SSC)* 4 (2) (1968) 100–107.
- [14] T. Ikeda, H. Imai, Enhanced  $A^*$  algorithms for multiple alignments: Optimal alignments for several sequences and  $k$ -opt approximate alignments for large cases, *Theoret. Comput. Sci.* 210 (2) (1999) 341–374.
- [15] T. Ishida, *Real-Time Search for Learning Autonomous Agents*, Kluwer Academic, Dordrecht, 1997.
- [16] T. Ishida, R.E. Korf, Moving-target search, in: *Proc. IJCAI-91*, Sydney, Australia, Morgan Kaufmann, San Mateo, CA, 1991, pp. 204–210.
- [17] T. Ishida, R.E. Korf, Moving-target search: A real-time search for changing goals, *IEEE Trans. Pattern Anal. Machine Intelligence (PAMI)* 17 (6) (1995) 609–619.
- [18] T. Ishida, M. Shimbo, Improving the learning efficiencies of real-time search, in: *Proc. AAAI-96*, Vol. 1, Portland, OR, 1996, pp. 305–310.
- [19] S. Koenig, The complexity of real-time search, Technical Report CMU-CS-92-145, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1992.
- [20] S. Koenig, Mini-max real-time search, *Artificial Intelligence* 129 (1–2) (2001) 165–197.
- [21] R.E. Korf, Depth-first iterative-deepening: An optimal admissible tree search, *Artificial Intelligence* 27 (1985) 97–109.
- [22] R.E. Korf, Real-time heuristic search, *Artificial Intelligence* 42 (2–3) (1990) 189–211.
- [23] R.E. Korf, Linear-space best-first search, *Artificial Intelligence* 62 (1993) 41–78.
- [24] T. Miura, T. Ishida, Stochastic node caching for efficient memory-bounded search, in: *Proc. AAAI-98*, Madison, WI, 1998, pp. 450–459.
- [25] N. Mizuno, T. Ishida, Evaluation on learning efficiencies of real-time search, *J. Japan. Soc. Artificial Intelligence* 10 (2) (1995) 306–313, In Japanese. English translation available in [15, Chapter 1].
- [26] A.W. Moore, C.G. Atkeson, Prioritized sweeping: Reinforcement learning with less data and less time, *Machine Learning* 13 (1993) 103–130.
- [27] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley, Reading, MA, 1984.
- [28] I. Pohl, First results on the effect of error in heuristic search, in: *Machine Intelligence*, Vol. 5, Edinburgh University Press, Edinburgh, UK, 1970, pp. 219–236.
- [29] I. Pohl, Heuristic search viewed as path finding in a graph, *Artificial Intelligence* 1 (1970) 193–204.

- [30] D. Ratner, M. Warmuth, The  $(n^2 - 1)$ -puzzle and related relocation problems, *J. Symbolic Comput.* 10 (1990) 111–137.
- [31] S.J. Russell, E. Wefald, *Do the Right Thing: Studies in Limited Rationality*, MIT Press, Cambridge, MA, 1991.
- [32] M. Shimbo, T. Ishida, Towards real-time search with inadmissible heuristics, in: W. Horn (Ed.), *Proc. Fourteenth European Conference on Artificial Intelligence (ECAI'2000)*, Berlin, Germany, IOS Press/Ohmsha, 2000, pp. 609–613.
- [33] H.A. Simon, *The Sciences of the Artificial*, 3rd Edition, MIT Press, Cambridge, MA, 1996.
- [34] T. Yoshizumi, T. Miura, T. Ishida, A\* with partial expansion for large branching factor problems, in: *Proc. AAAI-2000*, Austin, TX, 2000, pp. 923–929.