Propagating Updates in Real-Time Search: FALCONS(*k*)

Carlos Hernández and Pedro Meseguer
Institut d'Investigaci´o en Intel.ligència Artificial
Consejo Superior de Investigaciones Científicas
Campus UAB, 08193 Bellaterra, Spain.
{chernan|pedro}@iiia.csic.es

Abstract

We enhance real-time search algorithms with bounded propagation of heuristic changes. When the heuristic of the current state is updated, this change is propagated consistently up to k states not necessarily distinct. Applying this idea to FALCONS, we have develop the new FALCONS(k), an algorithm that keeps the good theoretical properties of FALCONS and improves its performance. We provide experimental results on benchmarks for real-time search, showing the benefits of our approach.

1. Introduction

Real-time search interleaves planning and action execution in an on-line manner. This allows to face search problems in domains with incomplete information, where it is impossible to perform the classical search approach: planning first the whole solution off-line, and then executing such solution. For these domains, interleaving planning and action execution is needed: planning explores some local search space selecting the best action from that limited exploration, and action execution carries out that action, causing changes in the world. The process repeats until achieving a solution. To assure completeness, real-time search has to record in a hash table those actions that have been executed. Often, there is an upper bound on the computing time that the planning phase can take.

Real-time search is also useful for domains with complete information whose search spaces are too large to be explored in practice. In such cases, real-time search is a suitable alternative to the impractical off-line search. It is able to find a solution within reasonable time, although usually it is not the optimal one [9].

A number of algorithms have been proposed to perform real-time search. They are based on the following strategy. Assuming an initial admissible heuristic, the algorithm explores a search space local to the current state and updates its heuristic accordingly, backing up costs. Then, the algorithm moves to the next state where exploraition/updating is repeated. The process is iterated until finding a goal state. Some of these algorithms are able to learn: solving repeatedly the same problem instance performance improves and they may converge to optimal paths.

LRTA* is the most popular real-time search algorithm, it was proposed in the seminal work of Korf [11]. This algorithm works on a search space where every state x has a heuristic estimate $h(x)$ of the cost from x to a goal. LRTA* is able to learn, it converges to optimal paths. FALCONS [3] is an algorithm proposed to speed up the convergence of LRTA*. In FALCONS every state x has two heuristic estimates, $h(x)$ of the cost from x to a goal and $g(x)$ of the cost from start state to x. FALCONS uses the heuristic estimation $g(x) + h(x)$ like in A* [12], to select the next state, unlike LRTA* that only use $h(x)$.

Instead of updating the heuristic of one (the current) state only, we have proposed to propagate the heuristic change consistently up to k states [7]. We call this idea bounded propagation, and we have applied it to LRTA* [7, 6], producing the LRTA*(k) algorithm. Experimental results show that LRTA*(k) causes large benefits on first solution, convergence and solution stability with respect to LRTA*, at the extra cost of longer planning steps. In this paper, we apply bounded propagation to FALCONS, in order to propagate the heuristic change consistently up to k states in $g(x)$ and $h(x)$, producing the new FALCONS(k) algorithm. Experimental results show that bounded propagation causes the same kind of benefits in first solution, convergence and solution stability.

The paper structure is as follows. In Section 2 we revise existing work on real-time search, with special emphasis on FALCONS. In Section 3 we present the idea of bounded propagation. In Section 4, we describe the new FALCONS(k) algorithm, showing its correctness and completeness. In Section 5, we provide experimental results of FALCONS(k) on classical real-time benchmarks. Finally, in Section 6 we extract some conclusions from this work.

Proceedings of the XXV International Conference of the Chilean Computer Science Society (SCCC'05)
0-7695-2491-5/05 $20.00 © 2005 IEEE

2. Real-Time Search

The state space is defined as (X, A, c, s, s_{goal}), where (X,A) is a finite graph, $c : A \rightarrow [0,\infty)$ is a cost function that associates each arc with a finite cost, $s \in X$ is the start state, and $s_{goal} \in X$ is the goal state. X is a finite set of states, and $A \subset X \times X - \{(x, x)|x \in X\}$ is a finite set of arcs. Each arc (v,w) represents an action whose execution causes the agent to move from v to w. The successors of a state x are $Succ(x) = \{y|(x, y) \in A\}$. The predecessors of a state x are $Pred(x) = \{y|(y, x) \in A\}$. A path $(x_0, x_1, x_2, \ldots)$ is a sequence of states such that every pair $(x_i, x_{i+1}) \in A$. The cost of a path is the sum of costs of the actions in that path. A heuristic function $h : X \rightarrow [0,\infty)$ associates with each state x an approximation $h(x)$ of the cost of a path from x to a goal. A heuristic function $g : X \rightarrow [0,\infty)$ associates with each state x an approximation $g(x)$ of the cost of a path from the initial state to x. The exact cost $h^*(x)$ is the minimum cost to go from x to a goal, and the exact cost $g^*(x)$ is the minimum cost to go from the initial state to x. The f value of state x denotes an estimate of cost $f^*(x) = g^*(x) + h^*(x)$ of a shortest path from the initial state to the goal through state x. h is admissible iff $\forall\, x \in X$, $h(x) \leq h^*(x)$. h is consistent iff $h(x_{goal}) = 0$ and $0 \leq h(x) \leq c(x,w) + h(w)$ for all states x with $x \neq x_{goal}$ and $w \in Succ(x)$. A path $(x_0, x_1, \ldots, x_n)$ with $h(x_i) = h^*(x_i)$, $0 \leq i \leq n$ is optimal.

LRTA* works as follows. From the current state x, it performs lookahead at depth d, and updates, according to the value-update rule, $h(x)$ to the $\max\{h(x), \min[c(x, v) + h(v)]\}$, where v is a frontier state and $c(x, v)$ is the cost of going from x to v. Then, it moves to a state y, successor of x, with minimum $c(x, y) + h(y)$. This state become the current state and the process iterates, until finding a goal state. This process is called a trial. LRTA* is a correct and complete algorithm, that eventually converges to optimal paths when solving repeatedly the same instance, keeping the heuristic estimates of the previous trial. The updating rule of LRTA* assures admissibility, provided the original heuristic was admissible. Since the updated heuristic maintains its admissibility, it can be reused in the next trial.

FALCONS adds the use of $g(x)$, a heuristic estimation of the cost of go from the initial state to x, for each state x of the space of states. Thus, FALCONS in a similar way to A*, uses the heuristic estimations $g(x)$ and $h(x)$ to select the next state, unlike LRTA* that only uses $h(x)$. In addition, LRTA* and FALCONS have different value-update rules. The value-update rule in FALCONS provides more informed but still admisible estimates of h and g values, by making better use of information in the neighborhood of the current state. In the case of $h(x)$, the value-update is the $\max\{h(x), \min_{v \in Succ(x)}[c(x, v) + h(v)], \max_{v \in Pred(x)}[h(v)-c(v, x)]\}$, for $g(x)$ is similar. The updating rule of FALCONS maintains the values of $h(x)$ and $g(x)$ admissible and locally consistent. FALCONS is a correct and complete algorithm, that eventually converges to optimal paths when solving repeatedly the same instance, keeping the heuristic estimates of the previous trial. With respect to performance, FALCONS accelerates convergence but it may perform a large amount of exploration in earlier trials.

The FALCONS algorithm with lookahead at depth 1 (the case considered in this paper) appears in Figure 1. Like in [3], we assume that the initial heuristic values are admissible. If $h(x,s_{goal})$ denotes $h(x)$ with respect to goal s_{goal}, then the values of h_0 and g_0 are: $h_0(x) := h(x, s_{goal})$ and $g_0(x) := h(s, x)$ for all states x.

Procedure FALCONS initializes the heuristic estimates h and g of every state to h_0 and g_0 respectively. Then it repeats the execution of FALCONS-trial until convergence (h and g do not change). At this point, an optimal path has been found. Procedure FALCONS-trial performs a solving trial on the problem instance. It initializes the current state x with the start s, and executes the following loop until finding a goal. First, it performs lookahead from x at depth 1, updating its heuristic estimates accordingly (call to functions FALCONS-LookaheadUpdateh1 and FALCONS-LookaheadUpdateg1 for h and g respectively). Second, it selects state y of $Succ(x)$ with minimum value of $f(y)$ as next state (break ties in favor of a successor w with the smallest value of $c(x,w)+h(w)$; break remaining ties arbitrarily, but systematically). Third, it executes an action that passes from x to y. At this point, y is the new current state and the loop iterates. Note that the heuristic estimators computed in a trial are used as initial values in the next trial.

Function FALCONS-LookaheadUpdateg1 performs lookahead from x at depth 1 for g values. If x is a initial state, then $g(x)$ maintains his value and the function return false. State y is a $Pred(x)$ with minimum value of $g(y) + c(y,x)$. State z is a $Succ(x)$ with maximum value of $g(z) - c(x,z)$. The variable e is used to keep the higher value between $g(y) + c(y,x)$ and $g(z) - c(x,z)$. Updating $g(x)$ with e guarantees the local consistency of the heuristic values. If $g(x)$ is smaller than e, then $g(x)$ takes the value of e and the function returns true, else the function returns false.

Function FALCONS-LookaheadUpdateh1 performs lookahead from x at depth 1 for h values. If x

is a goal state, then $h(x)$ maintains his value and the function return false. State y is a $Succ(x)$ with minimum value of $h(y) + c(x,y)$. State z is a $Pred(x)$ with maximum value of $h(z) - c(z,x)$. The variable e is used to keep the higher value between $h(y) + c(y,x)$ and $h(z) - c(z,x)$. Updating $h(x)$ with e guarantees the local consistency of the heuristic values. If $h(x)$ is smaller than e, then $h(x)$ takes the value of e and the function returns true, else the function returns false.

```
procedure FALCONS(X, A, c, s, s_goal)
  for each x ∈ X do h(x) ← h0(x); g(x) ← g0(x);
  repeat
    FALCONS-trial(X, A, c, s, s_goal);
  until h and g do not change;

procedure FALCONS-trial(X, A, c, s, s_goal)
  x ← s;
  while x ≠ s_goal do
    dummy ← FALCONS-LookaheadUpdateh1(x, s_goal);
    dummy ← FALCONS-LookaheadUpdateg1(x,s);
    y ← argmin_{w∈Succ(x)} f(w), where f(w) := max(g(w) +
    h(w), h(s)); Break ties in favor of a successor w
    with the smallest value of c(x,w)+h(w).
    Break remaining ties arbitrarily (but systematically).
    x ← y;

function FALCONS-LookaheadUpdateg1(x,s): boolean;
  if x = s then g(x) ← g(s) return false;
  y ← argmin_{w∈Pred(x)}[g(w) + c(w,x)];
  z ← argmax_{w∈Succ(x)}[g(w) - c(x,w)];
  e ← 0;
  if g(y) + c(y,x) ≥ g(z) – c(x,z) then e ← g(y) + c(y,x);
  else e ← g(z) – c(x,z);
  if g(x) < e then g(x) ← e; return true;
  else return false;

function FALCONS-LookaheadUpdateh1(x,s_goal):
    boolean;
  if x = s_goal then h(x) ← h(s_goal) return false;
  y ← argmin_{w∈Succ(x)}[h(w) + c(x,w)];
  z ← argmax_{w∈Pred(x)}[h(w) - c(w,x)];
  e ← 0;
  if h(y) + c(x,y) ≥ h(z) - c(z,x) then e ← h(y) + c(x,y);
  else e ← h(z) - c(z,x);
  if h(x) < e then h(x) ← e; return true;
  else return false;
```

Figure 1. The FALCONS algorithm.

In addition to the mentioned LRTA* and FALCONS, there are more algorithms for real-time search. RTA* [11] is an algorithm with an effective value-update rule, able to find better solution in the first trial but without converging to optimal routes. The weighted and bounded versions of LRTA* [14] speed up convergence and improve solution stability, but sacrifice optimality. HLRTA* [15] is a hybrid between RTA* and LRTA*. It finds better solutions than LRTA* in the first trial and converges to optimal paths. eFALCONS [4] is a hybrid between HLRTA* and FALCONS. It converges as FALCONS, performing a smaller amount of actions in earlier trials, although it may be greater than LRTA*. A new version of LRTA* [10] improves convergence by increasing lookahead depth, causing to increase the planning time per step. γ-Trap [1] uses adaptive lookahead depth and offers control on the exploration vs. exploitation trade-off, but sacrifices optimality. Other approaches consider search with moving target [8] and cooperative agents [9], [5].

3. Bounded Propagation

As search progresses, heuristic values of visited states are updated. Using the information obtained at the lookahead phase, we can better estimate the cost of reaching a goal from a visited state, and this new information is stored in the heuristic value of that state. This is a general strategy in real-time search algorithms. So far, most algorithms limit heuristic updating to the current state (except the LCM algorithm [13], without much influence in the field). Recently, we have proposed to propagate consistently the change of heuristic estimate of the curent state up to k states, not necessarily distinct. We call this idea bounded propagation, since changes are propagated up to a bound or limit of k states per step. The propagation occurs on predecessors of the state that changes its heuristic estimate. If one of these predecessors changes, this is again propagated on its own predecessors. This process is iterated with a limit k considered states. Propagation improves the heuristic quality while keeping it admissible, so search will find better heuristic estimates in the future states that will help to find a solution sooner.

We have applied bounded propagation to LRTA*, producing the LRTA*(k) algorithm (in fact, LRTA* is just a particular case of LRTA*(k) with $k = 1$). LRTA*(k) performance improves greatly with respect to LRTA*, in terms of first solution quality, convergence and solution stability [7]. However, bounded propagation requires longer planning steps, since propagating to k states is computationally more expensive than propagating to one (the current) state. Nevertheless, benefits are important and the extra requirements on planning time are moderate, so if the application can accommodate longer planning steps,

the use of bounded propagation is strongly recommended. Precise results depends on the parameter k (the higher k the more benefits at the cost of longer planning steps) and the specific problem considered.

In LRTA*(k), heuristic values are propagated consistently, that is, the new value of the heuristic estimate is $h(x) := \min_{v \in Succ(x)}[c(x, v) + h(v)]$. If it happens that $h(w)$ changes, $w \in Succ(x)$ but w is not the state with the minimum $[c(x, v) + h(v)]$, $v \in Succ(x)$, no matter the change in $h(w)$, $h(x)$ will not change because the minimum of its successors has not changed its heuristic estimate. That state is called a support for $h(x)$.

Formally, state y is support of $h(x)$, written $y := supp(x)$, iff $y = \text{argmin}_{v \in Succ(x)}[c(x, v) + h(v)]$. The previous paragraph describes a simple property of bounded propagation: if state y changes its heuristic estimate, only those states x predecessors of y such that y is their support could change its heuristic estimate. A predecessor state z not supported by y will not change: z is supported by other state and as far this state does not change its heuristic estimate, z will not change. Bounded propagation can benefit from this property, by propagating those states which are supported by the state that has changed its heuristic estimate. The use of supports requires a table that records for each expanded state its corresponding support.

Since propagation is limited up to k states, it is meaninful to consider which states are the most adequate to be updated. Originally, we decided to limit propagation to states already expanded in the current trial. Other alternatives are discussed in [6].

4. FALCONS(k)

FALCONS(k) is the algorithm that combines FALCONS with bounded propagation. The rationale for this combination is as follows. First, it is reasonable to expect that FALCONS would benefit from bounded propagation in the same way that LRTA* does, improving first solution quality, convergence and solution stability. Second, it has been observed experimentally that FALCONS may perform a large amount of exploration in earlier trials. Since bounded propagation improves first solution, its combination with FALCONS could be quite beneficial. Experimental results, reported in Section 5, clearly justify this combination.

In order to propagate consistently the heuristic values in FALCONS(k), the propagation must be coherent with the value-update rule, like in LRTA*(k). In FALCONS(k), heuristic values are propagated consistently, that is, the new value of the heuristic estimate for $h(x)$ (for $g(x)$, is analogous) is $h(x) := \max\{\min_{v \in Succ(x)}[c(x, v) + h(v)], \max_{v \in Pred(x)}[h(v) - c(v, x)]\}$. If it happens that h($w$) changes, $w \in Succ(x)$ but w is not the state with the minimum $[c(x, v) + h(v)]$, v

---

procedure FALCONS(k)(X, A, c, s, s_{goal})
  for each $x \in X$ **do**
    $h(x) \leftarrow h_0(x)$; $g(x) \leftarrow g_0(x)$;
    $suppminh(x) \leftarrow null$; $suppming(x) \leftarrow null$;
  repeat
    FALCONS(k)-trial(X, A, c, s, s_{goal});
  until h and g do not change;

procedure FALCONS(k)-trial(X, A, c, s,s_{goal})
  $x \leftarrow s$;
  while $x \neq s_{goal}$ **do**
    FALCONS-LookaheadUpdategK(x,s,k,$path$);
    FALCONS-LookaheadUpdatehK(x, s_{goal} ,k,$path$);
    $y \leftarrow \text{argmin}_{w \in Succ(x)} f(w)$, where $f(w) := \max(g(w) + h(w), h(s))$;
    Break ties in favor of a successor w with the smallest value of $c(x,w)+h(w)$. Break remaining ties arbitrarily (but systematically).
    $x \leftarrow y$;

procedure FALCONS-LookaheadUpdategK (x,s,k,$path$)
  $Q \leftarrow \langle x \rangle$;
  $cont \leftarrow$ k – 1;
  while Q $\neq \varnothing$ **do**
    $v \leftarrow$ extract-first(Q);
    if FALCONS-LookaheadUpdateg1(v,s) **then**
      for each $w \in Succ(v)$ **do**
        if $cont > 0 \wedge v = suppming(w) \wedge (g(v) + c(v,w)) > g(w)$ **then**
          $Q \leftarrow$ add-last(Q,w);
          $cont \leftarrow cont - 1$;
      for each $w \in Pred(v)$ **do**
        if $cont > 0 \wedge (g(v) - c(w,v)) > g(w)$ **then**
          $Q \leftarrow$ add-last(Q,w);
          $cont \leftarrow cont - 1$;

procedure FALCONS-LookaheadUpdatehK (x,s,k,$path$)
  $Q \leftarrow \langle x \rangle$;
  $cont \leftarrow$ k – 1;
  while Q $\neq \varnothing$ **do**
    $v \leftarrow$ extract-first(Q);
    if FALCONS-LookaheadUpdateh1(v) **then**
      for each $w \in Pred(v)$ **do**
        if $cont > 0 \wedge v = suppminh(w) \wedge (h(v) + c(w,v)) > h(w)$ **then**
          $Q \leftarrow$ add-last(Q,w);
          $cont \leftarrow cont - 1$;
      for each $w \in Succ(v)$) **do**
        if $cont > 0 \wedge (h(v) - c(v,w)) > h(w)$ **then**
          $Q \leftarrow$ add-last(Q,w);
          $cont \leftarrow cont - 1$;

```
function FALCONS-LookaheadUpdateg1(x,s): boolean;
  if x = s then g(x) ← g(s) return false;
  y ← argmin_{w∈Pred(x)}[g(w) + c(w,x)];
  z ← argmax_{w∈Succ(x)}[g(w) - c(x,w)];
  suppming(x) ← y ;  e ← 0;
  if g(y) + c(y,x) > g(z) – c(x,z) then e ← g(y) + c(y,x);
  else e ← g(z) – c(x,z);
  if g(x) < e then g(x) ← e; return true;
  else return false;

function FALCONS-LookaheadUpdateh1(x, s_goal): boolean;
  if x = s_goal then h(x) ← h(s_goal) return false;
  y ← argmin_{w∈Succ(x)}[h(w) + c(x,w)];
  z ← argmax_{w∈Pred(x)}[h(w) - c(w,x)];
  suppminh(x) ← y ;  e ← 0;
  if h(y) + c(x,y) > h(z) - c(z,x) then e ← h(y) + c(x,y);
  else e ← h(z) - c(z,x);
  if h(x) < e then h(x) ← e; return true;
  else return false;
```

Figure 2. The FALCONS(*k*) algorithm.

$\in$ *Succ*(x), no matter the change in $h(w)$, $h(x)$ will not change because the minimum of its successors has not changed its heuristic estimate. The state y with minimum $[c(x, v) + h(v)]$, $v \in$ *Succ*(x) is called support minimum for $h(x)$. Formally, state y is support minimum of $h(x)$, written y := *suppminh*(x), iff $y := \text{argmin}_{v \in Succ(x)}[c(x, v) + h(v)]$. If *suppminh*($x$) changes then $h(x)$ can change. If it happens that $h(w)$ changes, $w \in$ *Pred*(x), $h(x)$ will change when $h(w) - c(w,x) > h(x)$.

The FALCONS(k) algorithm appears in Figure 2. The main differences with FALCONS appear in the FALCONS-Trial procedure, where the call to FALCONS-LookaheadUpdateh1 and FALCONS-LookaheadUpdateg1 are replaced by procedure FALCONS-LookaheadUpdatehK and procedure FALCONS-LookaheadUpdategK respectively. These procedures performs the updating of the current state plus bounded propagation. FALCONS-LookaheadUpdateh1 returns true when h has changed as consequence of propagation, false otherwise. FALCONS-LookaheadUpdateg1 returns true when g has changed as consequence of propagation, false otherwise. Bounded propagation is done in the FALCONS-LookaheadUpdatehK and FALCONS-LookaheadUpdategK procedures. In FALCONS-LookaheadUpdatehK, if x is the current state of search, it initializes queue Q with capacity for k elements. After it enters the following loop. While Q is not empty, it extracts the first state v from Q and performs h updating. If this updating has caused some change in $h(v)$, it has to be propagated, so the predecessors of v are entered in Q, provided the following conditions: there is room ($cont > 0$), a predecessor w has some chance to modify its heuristic estimator (if $v = suppminh(w) \wedge h(v) + c(w,v) > h(w)$). And the successors of v are entered in Q, provided the following conditions: there is room ($cont > 0$), a successor w has some chance to modify its heuristic estimator (if $h(v) - c(v,w) > h(w)$). The procedure FALCONS-LookaheadUpdategK works in similar way. FALCONS(k) uses two supports by expand state, one for h and one for g. The allocation of the supports is made so that it is consistent with the value-update rule of FALCONS.

FALCONS is correct, complete and converge to optimal path in a finite problem state space with positive actions costs, in which the space state is safely explorable, and starting with non-negative admissible initial heuristic values [3]. FALCONS(k) inherits the good properties of FALCONS. To prove this, if neccesary guarantee that bounded propagation of heuristic changes maintains the admissibility.

Lemma 1 Let $x \in X$ and h admissible. Update $h(x)$ with (1) $h(s_{goal})$ if $x = s_{goal}$ or (2) $\max(p,q,r)$ where $p = h(x)$, $q = \min_{v \in Succ(x)}[c(x, v) + h(v)]$ and $r = \max_{v \in Pred(x)}[h(v) - c(v, x)]$ if $x \neq s_{goal}$ otherwise, implies that $h(x) \leq h^*(x)$.

Proof. In case (1), there is nothing to prove. In the case (2) we have three alternatives:

1. When the maximum value is p, the update is $h(x) \leftarrow h(x)$, so there is nothing to prove.
2. When the maximum value is q, the update is $h(x) \leftarrow \min_{v \in Succ(x)}[c(x, v) + h(v)]$. There is an optimal path from x to a goal that passes through a sucessor w. Therefore, $h^*(x) = c(x,w) + h^*(w) \geq c(x,w) + h(w) \geq \min_{v \in Succ(x)}[c(x, v) + h(v)] = h(x)$. So $h^*(x) \geq h(x)$, that is, the heuristic remains admissible.
3. When the maximum value is r, the update is $h(x) \leftarrow \max_{v \in Pred(x)}[h(v) - c(v, x)]$. Let us see that this update maintains admissibility. Since h values are admissible, the goal distance of any predecessor v of x is at least $h(v)$. This implies that the goal distance of x is at least $h(v) - c(v,x)$. Since this is true for any predecessor v, we can writte $h^*(x) \geq \max_{v \in Pred(x)}[h(v) - c(v, x)] = h(x)$. So the heuristic remains admissible.

Lemma 2 Let $x \in X$ and g admissible. Update $g(x)$ with $g(s)$ if $x = s$ or $g(x)$ with $\max(p,q,r)$ where $p = g(x)$, $q = \min_{v \in Pred(x)}[g(v) + c(v,x)]$ and $r = \max_{v \in Succ(x)}[g(v) - c(x,v)]$ if $x \neq s$ otherwise, implies that $g(x) \leq g^*(x)$.

Proof. It is analog of Lemma 1.

From this result, convergence of FALCONS(k) to optimal paths is guaranteed in the same terms as FALCONS, because bounded propagation maintains the admissibility. Thus, Theorem 3 of [3] is also valid for FALCONS(k) and the new algorithm inherits the good theoretical properties of FALCONS.

5. Experimental Results

We compare the performance of FALCONS(k), for different values of k, with RTA* (first trial only), FALCONS, LRTA* and LRTA*(k) (first trial, convergence and stability). In FALCONS(k) and LRTA*(k) we maintain the table of supports and heuristics values between trials [6]. It is important to see that FALCONS(k) can only make until k updates in g and h and LRTA(k) until k updates in h. As benchmarks we use these four-connected grids where an agent can move one cell north, south, east or west:

- **Grid35**, grids of size 301x301 with a 35% of obstacles placed randomly. In this type of grid heuristics tend to be only slightly misleading.
- **Grid70**, grids of size 301x301 with a 70% obstacles placed randomly. In this type of grid heuristics could be misleading.
- **Maze**, acyclic mazes of size 181x181 whose corridor structure was generated with depth-first search. Here heuristics could be very misleading.

Results are averaged over 1000 different instances. In grids of size 301x301 the start and goal state are chosen randomly assuring that there is a path from the start to the goal. In mazes, the start is (0,0), and the goal is (180,180). As initial heuristic we use the Manhattan distance. Results for FALCONS(k), FALCONS and RTA* appear in Table 1. For LRTA* and LRTA*(k) results appear in Table 2. The results are presented in percentage with respect to the value of row 100%. The values are in terms of solution cost ($\times 10^3$), number of expanded states ($\times 10^3$) and time per step ($\times 10^{-6}$ seconds), for the first trial; convergence (trials $\times 10^3$ to converge); and stability indexes [13].

FALCONS(k) is an algorithm that work on directed domains. The benchmarks used are undirected, therefore, predecessors and successors of one state are the same. This situation is considered when implementing the algorithm, to insert only once each neighbor of a state in Q and to not repeat work.

In the first trial, FALCONS(k) improves the performance of FALCONS and LRTA* in terms of the solution cost for all values of k. In order to surpass or to equal the performance of LRTA*(k) and RTA*, FALCONS(k) needs a higher k and therefore higher time of planning by step. The performance of FALCONS(k) and LRTA*(k) they are not totally comparable respect to k value, because, FALCONS(k) makes more work for a fixed value of k (update g and h), given this situation, the experimental result shows that the positive effect of k is better in LRTA*(k) that in FALCONS(k) in the first trial, for exemple, with k = 500 LRTA*(k) can makes up to 500 updates in h and FALCONS(k) can makes up to 500 updates in h and up to 500 updates in g. FALCONS(k) makes more work, but, the travel cost of LRTA*(k) is better or equal than FALCONS(k). With unbounded k FALCONS(k) is a little better than LRTA*(k) in the three benchmarks. With great values of k, the FALCONS(k) performance improves or equals the LRTA*(k) performance in all types of grids, but, it needs greater computation. If the agent has enough planning time per step, it can use a high k and thus increases largely the solution quality. Otherwise, low values of k improve solution quality and planning time per step remain reasonable.

In convergence, FALCONS(k) obtains optimal solutions with less cost than FALCONS and LRTA* for the all values of k tested on the three benchmarks. The results obtained by LRTA*(k) and FALCONS(k) are similar for all the values of $k > 1$ (considering that FALCONS(k) makes more work than LRTA*(k)). Exists one slight advantage of FALCONS(k) in Grid35 and Grid70. In MAZE the results are very similar. We can see that the advantages of the mechanisms of update and neighbor selection of FALCONS respect to LRTA*, tends to disappear with the bounded propagation. Considering trials to convergence, FALCONS(k) requires fewer trials than LRTA* and fewer trials than FALCONS from k = 15 in Grid35 and Grid70 and from k = 6 in Maze. Comparing with LRTA*(k), the the amount of trials to convergence are very similar for all the values of k tested. With respect to averange planning time per step, the result shows that FALCONS(k) requires more computation by step than LRTA*(k) (this was expected).

The number of expanded states is the amount memory shown in the tables. In convergence we can see that, in FALCONS(k), the number of expanded states tends to increase a bit when k increases. The amount memory required in LRTA*(k) and LRTA* are similar. In the first trial the number of expanded states decrease in Grid35 and Maze when k increases and it remains relatively constant in Grid70. The amount memory required in LRTA*(k) is similar. RTA* uses less memory that FALCONS(k) for most k tested. To measure solution stability we computed the indices IAE, ISE, ITAE, ITSE, and SOD [13]. IAE provides the sum of the error in convergence. ISE provides the

Grid35												
k	Cost	Memory	Time/Steps	Cost	Trials	Memory	Time/Steps	IAE	ISE	ITAE	ITSE	SOD
100%	961.6	12.8	0.61	4279.8	1.13	40.2	0.63					
1(FALCONS)	100%	100%	100%	100%	100%	100%	100%	3.7E+06	3.9E+12	8.0E+08	9.5E+12	1.1E+06
6	4%	35%	169%	65%	133%	110%	154%	2.1E+06	1.6E+10	9.3E+08	2.1E+12	6.0E+05
15	2%	30%	223%	40%	83%	111%	198%	1.3E+06	6.2E+09	3.7E+08	7.9E+11	3.7E+05
500	1%	42%	564%	7%	13%	111%	627%	2.4E+05	1.4E+09	1.1E+07	2.8E+10	7.9E+04
unbounded	1%	44%	869%	2%	3%	104%	2118%	6.5E+04	4.9E+08	6.2E+05	2.3E+09	1.9E+04
RTA*	3%	36%	54%									
Grid70												
100%	77.8	1.47	0.55	640.2	0.17	1.65	0.58					
1(FALCONS)	100%	100%	100%	100%	100%	100%	100%	4.8E+05	1.9E+10	4.1E+07	6.2E+11	1.3E+05
6	63%	102%	159%	56%	107%	101%	139%	2.0E+05	3.4E+09	5.9E+06	1.3E+10	4.8E+04
15	25%	100%	213%	33%	60%	101%	176%	1.1E+05	8.9E+08	2.3E+06	4.8E+09	3.5E+04
500	8%	101%	716%	5%	9%	101%	546%	1.9E+04	6.6E+07	8.1E+04	1.6E+08	4.1E+03
unbounded	3%	101%	3267%	0.8%	1.2%	99%	3309%	2.2E+03	3.7E+06	2.6E+03	4.0E+06	2.2E+01
RTA*	29%	100%	60%									
MAZE												
100%	622.3	8.9	0.6	21593.9	0.8	13.76	0.6					
1(FALCONS)	100%	100%	100%	100%	100%	100%	100%	1.9E+07	2.2E+12	7.5E+09	5.0E+14	6.7E+06
6	16%	82%	187%	53%	93%	109%	155%	8.9E+06	9.7E+11	1.6E+09	1.2E+14	5.2E+06
15	13%	82%	261%	31%	48%	112%	203%	5.2E+06	4.8E+11	5.8E+08	4.0E+13	3.1E+06
500	4%	82%	1188%	5%	7%	113%	673%	9.3E+05	5.0E+10	1.9E+07	7.5E+11	4.9E+05
unbounded	2%	82%	12182%	0.2%	0.4%	112%	25915%	2.6E+04	4.2E+08	4.7E+04	7.6E+08	1.0E+04
RTA*	2%	82%	53%									

Table 1. Results for the first trial (left), convergence (middle) and stability (right) for FALCONS(k), FALCONS and RTA*. Averange over 1000 instances.

Grid35												
k	Cost	Memory	Time/Steps	Cost	Trials	Memory	Time/Steps	IAE	ISE	ITAE	ITSE	SOD
1(LRTA*)	7%	37%	55%	151%	226%	111%	57%	5.2E+06	5.6E+10	3.8E+09	1.3E+13	1.7E+06
6	2%	27%	102%	70%	152%	112%	97%	2.2E+06	9.3E+09	1.2E+09	2.5E+12	6.9E+05
15	1%	29%	134%	42%	93%	112%	125%	1.3E+06	5.1E+09	4.4E+08	8.8E+11	4.1E+05
500	1%	46%	322%	8%	15%	111%	398%	2.5E+05	1.3E+09	1.3E+07	3.1E+10	8.2E+04
unbounded	1%	47%	427%	2%	3%	104%	1239%	6.8E+04	4.9E+08	6.2E+05	2.3E+09	1.8E+04
Grid70												
1(LRTA*)	188%	99%	65%	123%	182%	100%	70%	5.1E+05	2.4E+10	2.0E+07	9.1E+10	1.2E+05
6	65%	98%	110%	56%	102%	100%	103%	2.0E+05	3.2E+09	6.0E+06	1.5E+10	5.3E+04
15	35%	99%	144%	33%	60%	101%	126%	1.2E+05	1.0E+09	2.3E+06	5.0E+09	3.3E+04
500	7%	102%	461%	5%	10%	101%	358%	1.8E+04	6.4E+07	8.2E+04	1.7E+08	5.2E+03
unbounded	4%	107%	3101%	1%	3%	100%	1506%	2.3E+03	4.7E+06	2.6E+03	4.9E+06	1.7E+01
Maze												
1(LRTA*)	94%	93%	59%	129%	197%	100%	60%	2.2E+07	5.4E+12	6.9E+09	1.2E+15	1.3E+07
6	27%	85%	102%	53%	93%	107%	97%	8.7E+06	1.1E+12	1.6E+09	1.5E+14	5.3E+06
15	15%	83%	140%	30%	48%	112%	129%	5.0E+06	4.8E+11	5.6E+08	4.0E+13	3.0E+06
500	4%	81%	577%	5%	7%	113%	488%	8.0E+05	4.3E+10	1.4E+07	5.7E+11	4.2E+05
unbounded	2%	82%	4842%	0.2%	0.4%	113%	11359%	2.7E+04	4.4E+08	4.7E+04	7.8E+08	1.1E+04

Table 2. Results for the first trial (left), convergence (middle) and stability (right) for LRTA*(k) and LRTA*. Averange over 1000 instances.

square of the sum of the error in convergence, it penalizes large overshoots. ITAE and ITSE are two time-weighted versions of IAE and ISE, that impose large penalties on sustained errors. SOD sums up the difference in solution costs between two consecutive trials when the solution worsens. If SOD is equal to 0 convergence is monotonic. FALCONS(k) outperforms LRTA* for all values of k, and outperforms FALCONS, except for index ITAE on Grid35 with k = 6, where FALCONS obtains better results. Comparing with LRTA*(k) the values are similar for all k values tested in all indices for the three benchmarks.

FALCONS was superior to LRTA* in number of trials to convergence. This superiority disappears when comparing FALCONS(k) and LRTA(k): the initial advantange of FALCONS (k = 1) decreases substantially as k increases. For medium k, both algorithms exhibit a similar performance. Considering the cost of the first trial, something similar happens: FALCONS(k) is no longer more costly than LRTA(k) for medium k.

6. Conclusions

As in the case of LRTA*, bounded propagation of heuristic changes causes great benefits when applied to FALCONS, at the extra cost of longer planning steps. It is important to see that the advantages of the mechanisms of update and neighbor selection of FALCONS(k) respect to LRTA*(k) remain valid with k = 1 only. With greater k the advantages disappear. FALCONS(k) solves the FALCONS drawback of making too many steps in early trials. Future research includes the combination with other heuristic techniques and to make more efficient use of k in the propagation.

7. Acknowledgments

It work is partially supported by the Spanish REPLI project TIC-2002-04470-C03-03. 2003. We thank the Universidad Católica de la Santísima Concepción of Chile, CONICYT and anonymous reviewers, whose comments greatly improved the paper.

8. References

[1] V. Bulitko, "Learning for adaptive real-time search", *The Computing Research Rep.* (CoRR): cs.DC/0407017, 2004.

[2] S. Edelkamp and J. Eckerle, "New strategies in learning real time heuristic search", *In Proc. AAAI Workshop on On-Line Search*, pages 30–35, 1997.

[3] D. Furcy and S. Koenig, "Speeding up the convergence of real-time search", *In Proc. AAAI*, 2000.

[4] D. Furcy and S. Koenig, "Combining two fast-learning real-time search algorithms yields even faster learning", *In Proc. 6th European Conference on Planning*, 2001.

[5] M. Goldenberg, A. Kovarksy, X. Wu, and J. Schaeffer, "Multiple agents moving target search", *In Proc. 18th IJCAI*, 2003.

[6] C. Hernández and P. Meseguer, "Improving convergence of LRTA*(k)", *In Proc. IJCAI Workshop on Planning and Learning in a Priori Unknown or Dynamic Domains*, 2005.

[7] C. Hernandez and P. Meseguer, "LRTA*(k)", *In Proc. 19th IJCAI*, 2005.

[8] T. Ishida and R. E. Korf, "Moving target search", *In Proc. 12th IJCAI*, 1991.

[9] K. Knight, "Are many reactive agents better than a few deliberative ones?", *In Proc. 13th IJCAI*, 1993.

[10] S. Koenig, "A comparison of fast search methods for real-time situated agents", *In Proc. 3rd AAMAS*, 2004.

[11] R. E. Korf, "Real-time heuristic search", *Artificial Intelligence*, 1990, 42(2-3):189–211.

[12] N. Nilsson, *Problem Solving Methods in Artificial Intelligence*, McGraw-Hill, 1971.

[13] J. Pemberton and R. E. Korf, "Making locally optimal decisions on graphs with cycles", *Technical Report 920004*, Computer Science Dep. UCLA, 1992.

[14] M. Shimbo and T. Ishida, "Controlling the learning process of real-time heuristic search", *Artificial Intelligence*, 2003, 146(1):1–41.

[15] P. E. Thorpe, "A hybrid learning real-time search algorithm" *Master's thesis*, Computer Science Dep., UCLA, 1994.