Annales Univ. Sci. Budapest., Sect. Comp. 17 (1998) 33-48

A MORE EFFECTIVE VERSION OF ALGORITHM A

T. Ásványi and T. Gregorics (Budapest, Hungary)

Abstract. The algorithm A is one of the most important search techniques of the artificial intelligence. This heuristic search algorithm always finds a solution even in infinite problem space if there exists a solution at all. When the heuristic information applied in this algorithm satisfies a special property, i.e. it is admissible, then the solution found is optimal. (We call it algorithm A*.) In this paper a more effective version of algorithm A is presented and analysed. At first a new version of the general graph-search algorithm will be defined. The well-known heuristic search algorithms (algorithm A, algorithm A*, algorithm A^c) can be introduced in the same way as for the original graph-searching. We will show that these algorithms preserve their main properties, except the algorithm A* finds only nearly optimal solution. Finally some conditions on finding the optimal solution are going to be given.

1. Introduction

A number of problems in the Artificial Intelligence (AI) area can be related to the general problem of finding a path through a space of problem states from the initial state to any goal state. In this *state-space representation* any problem can be treated as a directed graph. Obtaining a solution to such a problem means finding a path in this *representation graph* from the start node to a goal node.

Several search techniques have been developed, which use heuristic information, i.e. special knowledge available from the problem domain in order to solve this search problem in efficient way. Among the heuristic *graph-searching algorithms* we have the class of the *algorithm A*. When the heuristic information built into the algorithm A satisfies the condition of admissibility

then it always finds an optimal (cheapest) solution. The name of this algorithm is algorithm A*. Furthermore, we make a strict distinction between algorithm A and algorithm A*. A less-widely known property of algorithm A is that it can find the solution (path from the start node to a goal node) even in infinite representation graph, if a solution exists. [3].

In this paper a more effective version of algorithm A is presented and analysed. Before introducing this new version we give a brief overview of the general graph-search technique and their well-known subclasses. The basic terminology and notations are also presented. An eager version of graph-searching is defined and its properties are proved in the third chapter. In the fourth chapter the algorithm A, A* and A^c of the new version are introduced. It will be shown that their properties are preserved, except that algorithm A* finds only nearly optimal solution. In the fifth chapter the conditions on finding the optimal solution are going to be given.

2. Overview of the graph-searching

The classical graph-search algorithm is going to be defined. Depth-first-search, breadth-first-search, uniform-cost-search, Algorithm A, A* and A^c can be defined as special cases of it, changing its evaluation function (f) for getting algorithms with different desired features [2].

2.1. Notations

The proofs of the theorems presented later in this paper require a special terminology. Now the main notations are defined without explanations. More detailed description can be found in [2].

$R = (N, A)$ - R is a directed graph, N is the set of nodes, A is the set of arcs. We suppose that R is a "δ-graph". (There are only a finite number of arcs outgoing from each node and every arc has its own cost, which is higher than a given positive δ value.)

s, T - s is the start node and T is the set of the goal nodes.

(n, m) - directed arc from node n to node m.

$(n_0, \ldots, n_k)$ - the directed path from node n_0 to node n_k through $n_1, \ldots, n_{k-1}$. The length of this path is k by definition.

$n_0 \rightarrow n_k$ - a directed path from node n_0 to node n_k.

$P(n, m)$ - the set of the directed paths from node n to node m.

$d^*(n)$ - the length of the shortest path from s to n.
$N(K)$ - the set of nodes n where $d^*(n) \leq K$. $N(K)$ is a finite set, because R is a "δ-graph".
$c(n,m)$ - cost of the directed arc from node n to node m.
$k(n,m)$ - cost of the directed path from node n to node m, by definition it is the sum of the costs of the arcs of it.
$P(K,n,m)$ - the set of the directed paths from node n to node m where $k(n,m) \leq K$. $P(K,n,m)$ is a finite set, because R is a "δ-graph".
$k^*(K,n,m)$ - cost of the cheapest path from node n to node m where $k(n,m) \leq K$, that is $k^*(K,n,m) := min\{k(n,m)|(n \rightarrow m) \in \in P(K,n,m)\}$.
$k^*(n,m)$ - cost of the cheapest path from node n to node m, that is $k^*(n,m) := min\{k(n,m)|(n \rightarrow m) \in P(n,m)\}$, if $P(n,m)$ is not empty. If there is no path from n to m then $k^*(n,m) := \infty$.
$g^*(n)$ - $g^*(n) := k^*(s,n)$.
$h^*(n)$ - $h^*(n) := min\{k^*(n,t)|t \in T\}$. If there is no path from n to T then $h^*(n) = \infty$.
$f^*(n)$ - the cost of the optimal solution through n, i.e. $f^*(n) := g^*(n)+ +h^*(n)$.
$g(n)$ - is the cost of the cheapest path found from s to node n. $g(n) \geq g^*(n)$.
$p(n)$ - is the pointer referring to the parent of n on that path.
$f(n)$ - is the value of the evaluation function of node n.
G - is the search graph.
$OPEN$ - is set of the open nodes.
Γ - The expansion of a node n is the set $\Gamma(n)$. It means the set of its successors, according to the directed arcs going out from node n.

2.2. General graph-search algorithm

The general graph-search algorithm (called GS here) [1][2][3] with minor formal modifications is the following:

```
function GS( s: node) return node U {fail} is
  G, OPEN, M: node_set;
  m: node;
begin
1. g(s) := 0 ; p(s) := null;
2.  G :={s}; OPEN := {s};
3.  loop
4.     n := min_f(OPEN); OPEN := OPEN \ {n};
```

```
5.    if n is a goal state then return n;
6.      M :=Γ(n);            /* expansion */
7.      for m in M loop
8.        if m∉ G or else g(n) + c(n, m) < g(m) then
9.          g(m) := g(n) + c(n, m); p(m) := n ;
10.           OPEN := OPEN U {m}; G := G U {m};
11.         end if;
12.       end loop;
13.     if OPEN is empty then return fail; end if;
14.   end loop;
end GS;
```

This algorithm discovers the representation graph of any problem starting from the start node s step by step. The subgraph that has been traversed at any stage is called search graph (G). It contains two kinds of nodes; the closed nodes and open nodes. The node that has already been expanded is the closed node. The one that has been generated but not expanded or put back into OPEN (GS.8-11) and not expanded from that time is the open node. Every node has three important attributes besides of the description of the actual state, namely the values of *g (cost function)*, *p (pointer)*, and *f (evaluation function)*. Let $g(n)$ be the cost of the cheapest path found by algorithm from s to node n (see GS.8-9). Let $p(n)$ be the pointer referring to the parent of n on that path. Let $f(n)$ be the value determining the selection of the node to be expanded (see GS.4). The algorithm terminates if a goal node is selected for expansion or there are no open nodes.

Taking stricter and stricter conditions on the evaluation function f we get several classes of graph-search algorithms. When $f(n) = g(n) + h(n)$ for any node n, where $h(n)$ is a nonnegative estimate of the cost required to get from n to any goal node, then the algorithm is called algorithm A. In other words, $f(n)$ is an estimate of the total cost of the cheapest solution path going through node n. We know that the algorithm A always finds a path from the start node to a goal node, if such path exists [2, 3]. When $h(n)$ is a lower bound to the cost of the minimal path from n to the nearest member of the goal nodes, then we call it algorithm A*. It is a well-known fact that this algorithm is admissible, i.e. it always finds an optimal (cheapest) path from the start node to a goal node, if such path exists [1]. An algorithm A* is a consistent algorithm (A^c), if the heuristic function h satisfies the monotone restriction (that is $h(n) - h(m)$ is smaller than or equal to the cost of all arcs going from n to m). It does not expand any node two or more times [4][5].

3. An eager graph-search algorithm

Now we are going to define an eager version of GS, which is more effective than the original one. We will show that its main properties remain true; it terminates in finite representation graph, and finds a solution if it exists.

3.1. The new version

The eager version of GS called GE is defined as follows:

```
function GE( s: node) return node U fail is
  G, OPEN, M: node_set;
  m: node;
begin
1.  g(s) := 0 ; p(s) := null;
2.  if s is a goal state then return s; end if;
3.    G := { s }; OPEN := { s };
4.    loop
5.      n := min_f(OPEN); OPEN := OPEN \ { n };
6.      M :=Γ(n);            /* expansion */
7.      for m it in M loop
8.        if m∉G or else g(n) + c(n, m) < g(m) then
9.          g(m) := g(n) + c(n, m); p(m) := n ;
10.           if m is a goal state then return m; end if;
11.           OPEN := OPEN U { m }; G := G U { m };
12.       end if;
13.   end loop;
14.   if OPEN is empty then return fail; end if;
15. end loop;
end GE;
```

In GS a goal node is recognized, when it is selected from the *OPEN* set. In our version the solution is found eagerly: when a node is generated, first it is checked, if it is a goal node. If it is not, it can be processed. The two algorithms are the same except the place of the termination condition, therefore the set of expansions done and the set of nodes generated by the eager version are subsets of the appropriate sets of GS.

GE terminates, when the first goal node is generated. GS terminates when a goal node is taken out from OPEN. A whole "level" of nodes is expanded and the next whole "level" is generated between the two actions by GS, when the first goal node is generated and when it is selected for expansion. The space complexity of a graph-search algorithm depends on the number of nodes generated by the algorithm. The time complexity of it depends on the number of expansions performed by it. Therefore, depending on the speed of the expansion of G, GS might need even two or more times more space, and even two or more times more computing time than GE.

3.2. The properties of the GE

The algorithm GE preserves the most important properties of GS.

Lemma 3.1. *The number of expansions is finite for any $n \in N$ node.*

Proof. If $n = s$ then n is expanded only once, because its initial g value is minimal. If $n \neq s$ then n has some g value, for example $g(n) = K$, when it is generated for the first time. Therefore $g(n)$ cannot be reduced and n cannot be put back into $OPEN$ more than $|P(K, s, n)|$ times. ($P(K, s, n)$ is finite.)

Consequence 3.2. *GE always terminates if the accessible part of the state-space-graph (from s) is finite.*

Lemma 3.3. (An invariant of GE.4 and GE.14) *If $n \in N$ has not yet been expanded then for any optimal $s \rightarrow n$ path there is an m element of that path, so that:*

a) $m \in OPEN$;

b) each of the elements on $s \rightarrow n$ preceding m is closed and has an optimal g value;

c) m also has its optimal g value ($g(m) = g^*(m)$).

Proof. First $m = s$. Later we can assume that there are some closed nodes on that optimal $s \rightarrow n$ path and s is closed. Let m be the first open node on $s \rightarrow n$. Let m be the first open node on $s \rightarrow n$. Let z be the node exactly preceding m.

Each node on $s = n_0, n_1, ..., n_k = z$ is closed and their g value is optimal: Let us assume indirectly, that $\exists j \in 0..k : n_j$ is closed, $g(n_j) > g^*(n_j)$ and $\forall i \in 0...j-1 : g(n_i) = g^*(n_i)$. Then $j \neq 0$, because $g(s) = 0 = g^*(s)$ (GE.1). When n_{j-1} was expanded last time, its g value was optimal. So n_j got its optimal g value (GE.9), unless n_j had got its optimal g value before. This consequence contradicts with the assumption.

Therefore, when z was expanded last time, m got its optimal g value, unless m had already got its optimal g value before.

Theorem 3.4. *If there is a path from s to a goal node and from s the accessible part of the state-space-graph is finite, then GE terminates by finding a goal state.*

Proof. If s is a goal node, it is trivial. If s is not a goal node, it is enough to prove, that some time a goal state is chosen at GE.10. Otherwise the algorithm must terminate at GE.14. Supposing this indirectly, let n be a goal state and $s \rightarrow n$ be an optimal path from s to n. When it terminates, n has not been selected for expansion. (Otherwise, GE should have had terminated at GE.10.) Because of the previous lemma, there is always an $m \in OPEN$ on $s \rightarrow n$ at GE.14. Therefore GE cannot terminate at GE.14 and cannot terminate at all. This result contradicts with 3.2.

4. The eager version of algorithm A

The algorithm A and its subclasses (algorithm A*, algorithm A^c) are the most important members of graph searching algorithms. They can always find a solution even in infinite problem space, if there exists the solution. In other hand they are heuristic algorithms so they take advantage of the fact that most problem spaces provide, at relatively small computational cost, some information (heuristic) that distinguishes among nodes in terms of likelihood leading to a goal state.

Now we will introduce the algorithm A, A*, A^c of our eager version, and prove their main properties.

4.1. Algorithm AE

Algorithm AE is derived in the same way from GE, as algorithm A is derived from GS.

Definition 4.1.1. *Algorithm AE is defined by f(n):=g(n) + h(n), where the only restriction on the heuristic function h is the following:* $\forall n \in \in N : h(n) \geq 0$.

Lemma 4.1.2. *If algorithm AE does not terminate, each of the open nodes is expanded in a finite number of steps.*

Proof. For any $n \in N$: $f(n) = g(n) + h(n) \geq g(n) \geq g^*(n) \geq d^*(n)\delta$, so $f(n) \geq d^*(n)\delta$. Let m be an element of $OPEN$. $K := f(m)$. If $\forall n \in \in OPEN \setminus \{m\} : f(n) > K$ then m is going to be expanded. If $d^*(n) > K/\delta$ then $f(n) > K$.

$N(K/\delta)$ is finite (see 2.1.). According to Lemma 3.1, there is a finite number of occasions, when a node of the set $N(K/\delta)$ is expanded by GE. Therefore, there have been only a finite number of expansions before m is expanded.

Theorem 4.1.3. *If there is a path from s to a goal node, algorithm AE terminates and finds a solution. (This solution is not necessarily the optimal one.)*

Proof. Let us assume that n is a goal node that there is an $s \rightarrow n$ path, but AE does not terminate. Let $s = m_0, m_1, ..., m_k = n$ be an optimal path from s to n. According to Lemma 3.3, there is always an m_i $(0 \leq i \leq n)$ - at GE.4 and GE.14 - in $OPEN$, because n is never chosen to be expanded. (Otherwise, AE should have had terminated at GE.10, because a node can be put into $OPEN$ only at GE.11). In the beginning, m_0 is expanded. Then m_1 is put into $OPEN$. According to the previous lemma, m_1 is also expanded in a finite number of steps. Then m_2 is put into $OPEN$ and so on. At last, m_{k-1} is expanded, $m_k = n$ is generated and AE terminates contradicting our assumption.

Because AE terminates, it can terminate at GE.10 or GE.14. Let us assume indirectly that it terminates at GE.14. When it terminates, n has not been selected for expansion. Therefore (see 3.3), there is an $m_i \in OPEN$ $(0 \leq i \leq k)$, so AE cannot terminate at GE.14. This contradiction proves that algorithm AE terminates at GE.10 finding a solution.

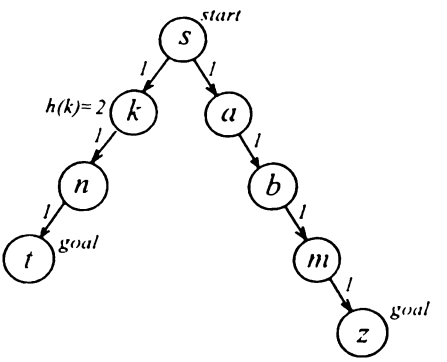
4.2. Algorithm AE*

Algorithm AE* comes from algorithm AE as algorithm A* comes from algorithm A. Our exact results on its behaviour are shown below.

Definition 4.2.1. *An algorithm AE is AE*, if* $\forall n \in N : \; h(n) \leq h^*(n)$ *(h is admissible).*

Algorithm A* is admissible, but, unfortunately, algorithm AE* does not have this nice feature, i.e. algorithm AE* does not find the optimal solution. Let us see the next example, where all arcs have unit cost. The heuristic value of the nodes are 0, except $h(k) = 2$. Let t and z be the goal nodes. It is clear, that the algorithm may find z instead of t.

In spite of this fact, the algorithm can be recommended to be used, because it inherits its increased effectivity from GE and its solution is nearly optimal. The next theorem shows that the cost of the solution found by AE* is near to the value $f^*(s)$.



Lemma 4.2.2. *If node n is selected by algorithm AE^* for expansion, then* $f(n) \leq f^*(s)$.

Proof. We can suppose that $f^*(s) < \infty$. Let $s \rightarrow t$ be an optimal solution path. At GE.4, immediately before n is selected, there is an $m \in s \rightarrow t$ that $m \in OPEN$ and $g(m) = g^*(m)$ (see 3.3). n is selected to be expanded, therefore:

$$f(n) \leq f(m) = g(m) + h(m) = g^*(m) + h(m) \leq g^*(m) + h^*(m) =$$

$$= f^*(m) = f^*(s).$$

Theorem 4.2.3. *Let α be a positive constant so that*

$$\forall t \in T, \quad \forall (n,t) \in A : \; h(n) + \alpha \geq c(n,t).$$

Let t be the goal node found by algorithm AE^* with h. Then

$$g(t) \leq f^*(s) + \alpha.$$

Proof. The node t is found by algorithm AE*, when one of its parents is expanded. Let it be n. Because of Lemma 4.2.2 and GE.9:

$$f^*(s) + \alpha \geq f(n) + \alpha = g(n) + h(n) + \alpha \geq g(n) + c(n,t) = g(t).$$

We can notice that $\alpha \leq max\{c(n,t) \mid (n,t) \in A, \; t \in T\}$. This means that usually the solution found by algorithm AE* is not much worse than the one found by algorithm A*. The previous example (see in 4.2) shows that there is a case when the cost of the solution found is exactly $f^*(s) + \alpha$. Other examples can be seen at the end of the fifth section.

4.3. Algorithm AE^c

Algorithm AE^c comes from algorithm AE as algorithm A^c comes from algorithm A: The heuristic function h must satisfy the monotone restriction. Therefore h is admissible, so A^c is admissible, too, and AE^c is a special case of AE* as A^c is a special case of A*, but the only result on the optimality of AE^c is Theorem 4.2.3. Anyway, it inherits its increased effectivity from GE, and it does not expand any node twice, like A^c. Therefore it can be recommended to be used. Our exact results on its behaviour are the following:

Definition 4.3.1. *Algorithm AE* is consistent (we call it algorithm AE^c), if h satisfies the monotone restriction, i.e. $\forall(n,m) \in A : \; h(n) - h(m) \leq c(n,m)$ (m is a successor of n).*

The next lemma is needed for the following theorem and it will be also used later.

Lemma 4.3.2. *Let us be given an optimal $s \rightarrow t$ path and the nodes n, m on that path. If m is a descendant of n and h satisfies the monotone restriction, then (see [4])*

$$g^*(n) + h(n) \leq g^*(m) + h(m).$$

Theorem 4.3.3. *If AE^c selects a node n for expansion, then $g(n) = g^*(n)$.*

Proof. We can suppose that n has not yet been expanded. If we can prove that $g(n) = g^*(n)$ for the first time, $g(n)$ cannot be reduced later, so n will not be put back into $OPEN$. Therefore it cannot be expanded several times. According to Lemma 3.3, $\exists m \in s \rightarrow n$ optimal path that $m \in OPEN$, $g(m) = g^*(m)$. We know (see 4.3.2) that $g^* + h$ is non-decreasing on $s \rightarrow n$. Therefore

$$f(m) = g(m) + h(m) = g^*(m) + h(m) \leq g^*(n) + h(n) \leq g(n) + h(n) = f(n).$$

If $g^*(n) \neq g(n) \Rightarrow g*(n) < g(n) \Rightarrow f(m) < f(n) \Rightarrow n$ cannot be selected to be expanded $(n, m \in OPEN)$, but the algorithm selects n to expand, therefore $g(n) = g^*(n)$.

Consequence 4.3.4. *Algorithm AE^c does not expand any node twice (or more).*

5. Optimal solution and algorithm AE

If we want to make sure, that algorithm AE^* or AE^c finds optimal solution, then we need some extra conditions on its heuristic function detailed below.

5.1. Algorithm AE^+

The first algorithm called AE^+ that finds optimal solution is an algorithm AE^* with a relatively strict condition on the heuristic function: the value of the heuristic function of the nodes exactly preceding goal nodes gives exact estimate of the distance to the goal.

Definition 5.1.1. *The algorithm AE^+ is an algorithm AE^* where $\forall t \in T$, $\forall (n,t) \in A$: $h(n) = c(n,t)$.*

AE^+ always finds an optimal solution, if some solution exists. AE^+ needs an extra condition compared to A^*, but it terminates earlier, so it needs less time and space to run. For example the 8-puzzle with P or W heuristics [2] provides AE^+.

Theorem 5.1.2. *If there is a path from s to a goal node then algorithm AE^+ terminates by finding an optimal solution path.*

Proof. AE^+ terminates by finding a solution path because it is an algorithm AE. When a goal state (namely t) is found, one of its predecessors (namely n) has been selected to be expanded and

$$g(t) = g(n) + c(n,t) \quad (GE.9),$$
$$f(n) \leq f^*(s) \qquad (\textbf{Lemma 4.2.2}),$$
$$f^*(s) \leq f^*(t) \quad \text{and} \quad h^*(t) = 0 \quad (t \in T).$$

Therefore

$$g(t) = g(n) + c(n,t) = g(n) + h(n) = f(n) \leq f^*(s) \leq f^*(t) =$$
$$= g^*(t) + h^*(t) = g^*(t) \leq g(t) \Rightarrow g(t) = f^*(s).$$

5.2. Algorithm AE^m

The second algorithm called AE^m that finds optimal solution is an algorithm AE^c with an extra condition on its heuristic function: difference between

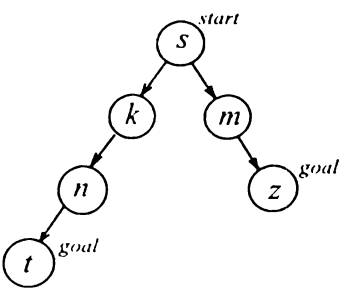
the heuristic value of the nodes exactly preceding goal nodes and the distance to the goal is a constant.

Definition 5.2.1. *The algorithm* AE^m *is an algorithm* AE^c *where* $\exists \alpha \geq \geq 0 : \forall t \in T, \forall (n,t) \in A : h(n) + \alpha = c(n,t)$.

For example the 8-puzzle with $h = P$, $h = W$ or $h \equiv 0$ heuristics [2] provides AE^m.

Theorem 5.2.2. *If there is a solution, algorithm* AE^m *finds an optimal one.*

Proof. Let us suppose that it finds a solution that is not optimal. Let it be $s \rightarrow z$. Let m be the parent of z on $s \rightarrow z$. Let $s \rightarrow t$ be an optimal solution path. It is clear that $s \neq t$ (GE.2). Let n be the parent of t on $s \rightarrow t$. Node z is found when m is expanded. That time n has not been expanded. (Otherwise t had been found.) Therefore (based on Lemma 3.3), there is an open node (k) on $s \rightarrow n$, that $g^*(k) = g(k)$.



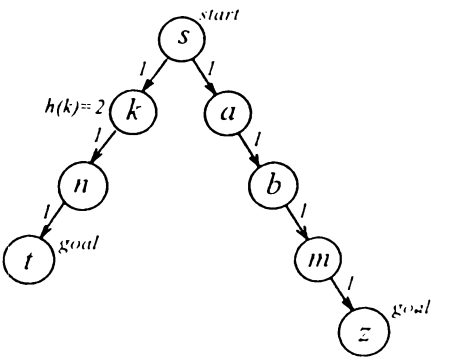
$f(m) \leq f(k) = g(k) + h(k)$,	(m is selected to be expanded)
$g(k) + h(k) = g^*(k) + h(k)$,	(k is on the optimal path $s \rightarrow t$)
$g^*(k) + h(k) \leq g^*(n) + h(n)$,	(because of Lemma 4.3.2)
$g^*(n) + h(n) = g^*(n) + c(n,t) - \alpha$,	(based on the extra condition of 5.2.1)
$g^*(n) + c(n,t) - \alpha = g^*(t) - \alpha$,	(n is on the optimal path $s \rightarrow t$)
$g^*(t) - \alpha < g(z) - \alpha = g(m) + c(m,z) - \alpha$,	(according to the indirect supposal)
$g(m) + c(m,z) - \alpha = g(m) + h(m) = f(m)$,	(based on the extra condition of 5.2.1)

$\Rightarrow f(m) < f(m) >$

This contradiction proves the theorem.

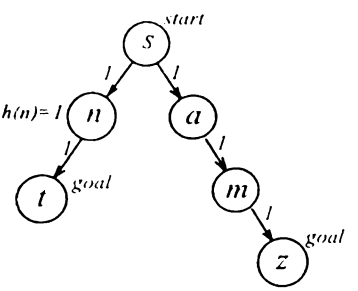
The examples presented below show that the conditions of algorithm AE^m are necessary to find an optimal solution.

If h is only admissible with the extra condition of 5.2.1:



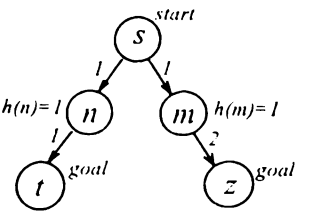
All arcs have unit cost. The heuristic value of the nodes are 0, except $h(k) = 2$. t and z are the goal nodes. It is clear, that the algorithm may find z instead of t.

If h is only monotone without the extra condition of 5.2.1:



All arcs have unit cost. The heuristic values of the nodes are 0, except $h(n) = 1$. t and z are the goal nodes. It is clear, that the algorithm may find z instead of t.

If h is the only constant on the nodes being parents of goal nodes, instead of the extra condition of 5.2.1:



All arcs have unit cost, except $c(m, z) = 2$. The heuristic values of the nodes are 0, except $h(n) = 1$, $h(m) = 1$. t and z are the goal nodes. It is clear, that the algorithm may find z instead of t.

Conclusions

In the previous sections we introduced and described a new version of the heuristic search algorithms. Now we will summarize our results.

Our first remark was that a goal node can be recognized by the graph-search algorithm when it is generated. This eager version of the graph-search algorithm (GE) might need even two or more times less space, or even two or more times less computing time than the original version. It has been showed that the new version always terminates and can find a solution (if there exists one) in finite representation graph. These properties are equivalent to the ones of the original version.

Secondly the algorithm A of the new version (AE) was introduced in the same way as algorithm A is derived from the original version. It has been proved that almost every property remains true. Algorithm AE always finds a solution even in infinite representation graph, if the solution exists. The well-known subclasses of algorithm A (namely algorithm A*, algorithm A^c) can be also derived from algorithm AE. They are called AE* and AE^c. The only difference between the eager versions and the original ones is that the eager versions (AE* and AE^c) cannot find optimal solution.

Although the main purpose of this paper is to show the condition of finding optimal solution a very interesting extra additional result has been given. We proved that the cost of the solution found by algorithm AE* is never greater than the sum of the optimal cost and the cost of the most expensive arc going to the goal. In practice it means that the cost of the solution is near to the optimal cost.

At last the criteria of finding optimal solution was presented. Two algorithms were defined. The algorithm AE^+ is a subclass of algorithm AE^* where the value of the heuristic function of the nodes exactly preceding goal nodes gives exact estimate of the distance to the goal. The algorithm AE^m is subclass of algorithm AE^c where difference between the heuristic value of the nodes exactly preceding goal nodes and the distance to the goal is a constant. Although these conditions seem to be very strict, many problems have the heuristic function that satisfies them. The extra condition of AE^m is weaker than the one of AE^+. In order to decide that an algorithm AE is algorithm AE^c, it is sufficient to see that the heuristic function satisfies the condition of the monotone restriction and takes zero on each goal node [4]. Many times it easier to find a heuristic function of this kind than heuristics that are lower bound on h^* (generally not known). Therefore algorithm AE^m may be used better than algorithm AE^+. There is an interesting conclusion on the non-heuristic graph-search algorithms. If there is a constant c, so that $\forall t \in T$, $\forall (n,t) \in A : \; c(n,t) = c$, then the eager version of the uniform-cost strategy, which is algorithm AE with $h \equiv 0$, finds optimal solution. The eager version of breadth-first-search also finds optimal solution, because it is a special case of the uniform-cost strategy, so that $\forall (n,m) \in A : \; c(n,m) = 1$.

In summary, the algorithm AE can be recommended for use, because it inherits its increased effectivity from GE. In addition, using admissible heuristic function the solution found is nearly optimal, and taking one of the extra conditions on the heuristic the new version of algorithm A finds optimal solution.

References

[1] **Hart P., Nilsson N.J. and Raphael B.,** A Formal Basis for the Heuristic Determination of Minimum Cost Paths, *IEEE Trans. System, Man and Cybernet.*, **4** (2) (1968), 100-107.

[2] **Nilsson N.J.,** *Principles of Artificial Intelligence,* Springer, 1982.

[3] **Gregorics T., Fekete I. and Varga L.Zs.,** Corrections to Graph-Search Algorithms, *Fourth Conference of Program Designers, ELTE, Budapest, June 1-3, 1988,* ed. A.Iványi, 25-30.

[4] **Gregorics T.,** Another Introduce to Consistent Algorithms, *Proc. First Seminar on Artificial Intelligence, Visegrád, January 23-25, 1989,* ed. I.Fekete and S.Nagy, 137-144.

[5] **Rich E. and Knigth K.,** *Artificial Intelligence,* McGraw-Hill Inc., 1991.

[6] **Pearl J. and Korf R.E.,** Search Techniques. *Ann. Rev. Comput. Sci.*, (2) (1987), 451-467.

T. Ásványi
Dept.of General Comp. Science
Eötvös Loránd University
VIII. Múzeum krt. 6-8.
H-1088 Budapest, Hungary
asvanyi@ludens.elte.hu

T. Gregorics
Dept.of General Comp. Science
Eötvös Loránd University
VIII. Múzeum krt. 6-8.
H-1088 Budapest, Hungary
greti@ludens.elte.hu