Acta Univ. Sapientiae, Informatica, **6,** 2 (2014) 190–205

DOI: 10.1515/ausi-2015-0003

Remarks on the A^{**} algorithm

Tibor GREGORICS
Eötvös Loránd University, Faculty of Informatics
Budapest
email: gt@inf.elte.hu

Abstract. The A^{**} algorithm is a famous heuristic path-finding algorithm. In this paper its different definitions will be analyzed firstly. Then its memory complexity is going to be investigated. On the one hand, the well-known concept of better-information will be extended to compare the different heuristics in the A^{**} algorithm. On the other hand, a new proof will be given to show that there is no deterministic graph-search algorithm having better memory complexity than A^{**}. At last the time complexity of A^{**} will be discussed.

1 Introduction

The A^{**} algorithm is one of the graph-search algorithms that can be used to solve path-finding problems.

Path-finding problems are the tasks that can be modeled by a directed arc-weighted graph (this is the so-called representation graph). Let $R = (N, A, c)$ denote a directed arc-weighted graph where N is the set of nodes, $A \subseteq N \times N$ is the set of directed arcs and $c : A \mapsto \mathbb{R}$ is the weight function. The graphs of our interest have got only finite outgoing arcs from their nodes and there is a global positive lower limit ($\delta \in \mathbb{R}$) on the weights. These graphs are the δ-graphs.[5] Thus a path-finding problem can be represented by the triple (R, s, T) where R is a δ-graph, s denotes the start node and T denotes the set of goal nodes. The solution of this problem is a path from the start node to some goal node that can be denoted by $s \to t$ where $t \in T$.

Computing Classification System 1998: I.2.8
Mathematics Subject Classification 2010: 68T20
Key words and phrases: path-finding problem, graph-search algorithms, heuristic function, A^* algorithm, A^{**} algorithm

The cost of a path can be calculated as the summation of the cost of the arcs on this path. We will denote the smallest cost path from n to m as $n \rightarrow^* m$. This path is called as optimal path. In many path-finding problems the optimal path from the start node to some goal node is needed to be found. The value $h^*(u)$ shows the optimal cost from the node u to any goal node. The function $h^* : N \mapsto \mathbb{R}$ is called as optimal remaining cost function. The value $g^*(u)$ gives the optimal cost from the start node to the node u and the function $g^* : N \mapsto \mathbb{R}$ is named as optimal leading up cost function. We can calculate the optimal cost of the path going from the node s to any goal node via the node u in the way $f^*(u) = g^*(u) + h^*(u)$ where $f^* : N \mapsto \mathbb{R}$ is the optimal cost function. We remark that the value $f^*(s)$ denotes the cost of the optimal solution.

Graph-search algorithms try to find a path from the start node to a goal node and during their process they always record the sub-graph of the representation graph that has been discovered. This is the search graph (G). The nodes of G whose children are known are the so-called closed nodes. The other nodes of G that wait for their expansion are the open nodes. Sometimes a node may be open and closed at the same time if it has been expanded but its children (more precisely the optimal paths form the start node to its children) are not completely known. Let OPEN denote the set of the open nodes at any time. In every step the most appropriate open node will be selected and expanded, i.e. its children must be generated or regenerated. The general graph-search algorithm [5, 2, 3] evaluates the open nodes with an evaluation function $f : \text{OPEN} \mapsto \mathbb{R}$ and chooses the open node with the lowest f value for expansion.

```
procedure Graph-search
    G := ({s}, ∅) : OPEN := {s} : π(s) := nil : g(s) := 0
    while OPEN ≠ ∅ loop
        n := min_f(OPEN)
        if goal(n) then return there is a solution endif
        foreach m ∈ Children(n) loop
            if m ∉ G or g(m) > g(n) + c(n, m) then
                π(m) := n : g(m) := g(n) + c(n, m)
                OPEN := OPEN ∪ m
            endif
        endforeach
        G := G ∪ {(n, m)|m ∈ Children(n)}
        OPEN := OPEN ∪ {(n, m)|m ∈ Children(n)}
    endwhile
    return there is no solution
end
```

The cheapest paths from the start node to the nodes of G, which are found so far, must be recorded with their cost. These costs are shown by the function $g \in N \mapsto \mathbb{R}$. (It is clear that $g(u) \geq g^*(u)$ for all node u of N.) The algorithm also maintains a pointer $\pi \in N \mapsto N$ that marks the best parent of each discovered node (except the start node). The best parent of any node is the one which is along the cheapest discovered path driving from s to that node. The recorded paths form a directed spanning tree in the search graph where the root node is the start node. We will denote a recorded path driving from s to u as $s \rightarrow^{\pi} u$ and $c^{\pi}(s, u)$ will symbolize its cost.

The computation of the evaluation function of a graph-search algorithm can contain some extra knowledge about the problem. This is the so-called heuristic function $h : N \mapsto \mathbb{R}$ that estimates the remaining optimal path cost from a node to any goal node, i.e. $h(u) \approx h^*(u)$ for all nodes u of N.

The most famous heuristic graph-search algorithm is *the* A^* *algorithm*. The evaluation function of this algorithm is $f = g + h$ where the cost function g is calculated by the algorithm and the heuristic function h is derived from the problem. The A^* algorithm uses a nonnegative and admissible heuristic function. The admissibility means that the heuristic function gives a lower limit on the remaining optimal path cost from any node to a goal node, i.e. $h(u) \leq h^*(u)$ for all node u of N.

2 Definitions of the A^{**} algorithm

The A^{**} algorithm can be treated as a modification of the A^* algorithm. During the execution of A^*, it can occur that the evaluation function value of an open node n $(g(n) + h(n))$ might be smaller than the value $g(u) + h(u)$ of some node u on the recorded path from s to n. It signs that the estimation $h(n)$ is too low for the remaining path-cost hence

$$h(n) < g(u) + h(u) - g(n) = h(u) - c^{\pi}(u, n) \leq h^*(u) - c^{\pi}(u, n) \leq h^*(n).$$

The A^{**} *algorithm* has been introduced by Dechter and Pearl to correct this failure of the heuristic function.[1] Its evaluation function gives the maximum of the value $g(u) + h(u)$ for all node u on the recorded path $s \rightarrow^{\pi} n$, i.e.

$$f(n) = \max_{u \in \{s \rightarrow^{\pi} n\}} [g(u) + h(u)]$$

where h is a non-negative admissible heuristic function. Additionally, the selection breaks ties arbitrarily but in favor of goal nodes.

Corresponding to this definition, the evaluation function value of all open nodes must be always recomputed after each expansion since the recorded paths can be changed. However, the computational cost of the recalculation of the evaluation function value of all open nodes with their recorded path is too high. Russell and Norvig mention an alternative way to implement the A^{**} algorithm. [6] They suggest that the evaluation function value of u is calculated by the following recursive formula:

$$f(u) := \max(g(u) + h(u), f(v))$$

when a better path is found to a node u after the expansion of its parent node v. Initially, $f(s) := h(s)$. The next theorem shows that the original A^{**} algorithm and this latter alternative one work in the same way.

Theorem 1 *Both versions of the A^{**} algorithm contain the same open nodes with the same evaluation function values in each step.*

Proof. We prove this result using induction on the number of the expansions.

Initially both versions add the start node into OPEN with the same evaluation function value ($f(s) = h(s)$). Let us suppose that the statement of the theorem is true in the i^{th} step when the open node n is selected and expanded. Let k be an arbitrary open node after this expansion. We will show that the evaluation function value of k is independent of any version of A^{**}.

If there is no cheaper path from s to k via n, then the node k has to be in OPEN before the expansion of n and neither of the versions modify its evaluation function value.

Otherwise, we must distinguish two cases: either the node k or some of its ancestor is a child of the node n.

If k is a child of n and either k is not in G or $g(n) + c(n, k) < g(k)$ hold, both versions recalculate the value $f(k)$ after the expansion of n. According to the original version,

$$f(k) = \max_{u \in \{s \to^{\pi} k\}} [g(u) + h(u)]$$

where $s \to^{\pi} k$ is the new recorded path via the node n. According to the alternative version,

$$f(k) = \max(f(n), g(k) + h(k)).$$

But the recorded path $s \to^{\pi} n$ is not changed during the expansion of n thus the value $f(n)$ remains the same, and, by the induction hypothesis, this values

in both versions are identical, i.e.

$$f(n) = \max_{u \in \{s \to^{\pi} n\}} [g(u) + h(u)].$$

Thus the new evaluation function values of k in both versions are identical because of

$$\max_{u \in \{s \to^{\pi} k\}} [g(u) + h(u)] = \max\Big(\max_{u \in \{s \to^{\pi} n\}} [g(u) + h(u)], g(k) + h(k) \Big).$$

If k were in OPEN before the expansion of n, and one of its ancestors (let m denote it) were a child of n, and a cheaper path were discovered form s to this very ancestor via n, then the alternative version would not recalculate $f(k)$ but the original version would. Let $f^{old}(k)$ and $f^{new}(k)$ be the earlier and the new value of k maintained by the original version. By the induction hypothesis, $f(k) = f^{old}(k)$. It should be shown that $f^{new}(k) = f^{old}(k)$ because in this case $f(k) = f^{new}(k)$ is followed. Let α denotes the recorded path from s to m before the expansion of n. Let $g^{\alpha}(u)$ denote the cost of α from s to u before the expansion of n, and in this case $g(n) + c(n, m) < g^{\alpha}(m)$. It is obvious that

$$\begin{aligned} f^{old}(k) &= \max\Big(\max_{u \in \{s \to^{\alpha} m\} \setminus \{m\}} [g^{\alpha}(u) + h(u)], g^{\alpha}(m) + h(m), \\ &\qquad \max_{u \in \{m \to^{\pi} k\} \setminus \{m\}} [g(u) + h(u)] \Big), \\ f^{new}(k) &= \max\Big(\max_{u \in \{s \to^{\pi} n\}} [g(u) + h(u)], g(n) + c(n, m) + h(m), \\ &\qquad \max_{u \in \{m \to^{\pi} k\} \setminus \{m\}} [g(u) + h(u)] \Big). \end{aligned}$$

We know that $f(n) = \max_{u \in \{s \to^{\pi} n\}} [g(u) + h(u)]$ and the following inequations hold:

- $\max_{u \in \{s \to^{\alpha} m\} \setminus \{m\}} [g^{\alpha}(u) + h(u)] \leq f(n)$ because the path $s \to^{\alpha} m$ was discovered before the path $s \to^{\pi} m$,
- $g^{\alpha}(m) + h(m) \leq f(n)$ because the path $s \to^{\alpha} m$ was discovered before the path $s \to^{\pi} n$,
- $\max_{u \in \{m \to^{\pi} k\} \setminus \{m\}} [g(u) + h(u)] \leq f(n)$ because the path $m \to^{\pi} k$ was discovered before the path $s \to^{\pi} m$.

On the one hand, it follows from the above inequations that $f^{old}(k) \leq f(n)$ and since the algorithm selects n for expansion instead of k, the inequation $f(n) \leq f^{old}(k)$ must hold. Ergo, $f^{old}(k) = f(n)$. On the other hand, $f(n) \leq f^{new}(k)$ because n is on the recorded path $s \rightarrow^{\pi} k$, and $f(n) \geq f^{new}(k)$ is also true because the path $m \rightarrow^{\pi} k$ were discovered before the path $s \rightarrow^{\pi} n$. It follows that $f^{new}(k) = f(n)$. Thus $f^{new}(k) = f^{old}(k)$. □

The main consequence of this theorem is that the following lemmas and theorems that are proved on only the original version of A** are valid for both versions of A**.

Lemma 2 *The value f(m) calculated by the A** algorithm is proportional to the depth of* m.

Proof. Let $d(m)$ denote the length of the recorded path from start node to the node m. Let $d^*(m)$ denote the length of the shortest path from start node to the node m. It is obvious that $d^*(m) \leq d(m)$. By respecting the definition of A**, $f(m) \geq g(m) + h(m)$ for all open node m. We know that $h(m) \geq 0$ and the cost of the arcs are greater and equal to a positive δ. Thus we have

$$f(m) \geq g(m) + h(m) \geq g(m) > d(m) \cdot \delta \geq d^*(m) \cdot \delta.$$

□

From this lemma, it follows that the A** algorithm can find a solution if there exists a solution even if the heuristic function is non-admissible and only non-negative. This proof is analogous to the proof of the correctness of the A* algorithm. [5]

Lemma 3 *When the A** algorithm selects a node* n *for expansion, the inequation* $f(n) \leq f^*(s)$ *holds.*

Proof. If there is no solution, then $f^*(s)$ can be considered infinite. If there exists a solution, there must be a node m on the optimal solution path at the time of the selection of n so that m is in OPEN and an optimal path from s to m is recorded by algorithm, i.e. $g(u) = g^*(u)$ for all nodes u of this path. Let v denote the node of this path where

$$\max_{u \in \{s \rightarrow^{\pi} m\}} [g(u) + h(u)] = g(v) + h(v)$$

hold. The A** algorithm selects the node n instead of the node m, so $f(n) \leq f(m)$ must hold. Based on the admissible property of the heuristic function we

get

$$f(n) \leq f(m) = \max_{u \in \{s \to^{\pi} m\}} [g(u) + h(u)] = g(v) + h(v) \leq g^*(v) + h^*(v) = f^*(s).$$

□

The consequence of this lemma is that the A^{**} algorithm can find optimal solution if there exists a solution. This proof is analogous to the proof of the optimality of the A^* algorithm. [5]

At last an interesting property of the A^{**} algorithm will be proved.

Theorem 4 *If the* A^{**} *algorithm selects the node* m *after the node* n *for expansion, then* $f(n) \leq f(m)$.

Proof. This statement is enough to prove with two nodes that are expanded directly after each other: firstly the node n, then the node m.

If m has already been in OPEN just before n is expanded but this expansion does not find a cheaper path from s to m, then the value of $f(m)$ does not change. But $f(n) \leq f(m)$ has to hold because the algorithm selects the node n instead of the node m.

If m is not in OPEN before the expansion of n but it gets into there after n is expanded, or if m has already been in OPEN just before n is expanded and this expansion finds a cheaper path from s to m, then m must be a child of n and the recorded path π from s to m drives via n. It is obvious that

$$\max_{u \in \{s \to^{\pi} n\}} [g(u) + h(u)] \leq \max_{u \in \{s \to^{\pi} m\}} [g(u) + h(u)]$$

because n is on the path π. Thus, by respecting the definition of the evaluation function of A^{**}, $f(n) \leq f(m)$ holds. □

An important consequence of this theorem is that if A^{**} expands the same node twice, its second evaluation function value will be greater than or equal to the first one.

3 Memory complexity of the A^{**} algorithm

The memory requirement of a graph-search algorithm depends basically on the size of its search graph. This size can be mesuared by the number of the expanded nodes of this search graph. These are the so-called closed nodes. The size of the search graph is the biggest when the algorithm terminates thus the memory requirement is given with the number of the closed nodes at

termination. Because of this, we assume that the path-finding problems of our focus has got a solution, thus most of the famous graph-search algorithms – specially the A^{**} algorithm – must terminate.

Let CLOSED_S denote the set of closed nodes of the graph-search algorithm S at termination. Let X and Y be arbitrary graph-search algorithms. We can say that X is *better* than Y in a given path-finding problem if $\mathsf{CLOSED}_X \subset \mathsf{CLOSED}_Y$, and X is *not worse* than Y in a given path-finding problem if $\mathsf{CLOSED}_X \subseteq \mathsf{CLOSED}_Y$.

3.1 Comparison of different heuristics in the A^{**} algorithm

At first we will compare two A^{**} algorithms, namely A_1 and A_2 using different heuristic funtions. Let h_1 and h_2 be admissible and non-negative heuristic functions deriving from the same problem. We say – based on the work of Nils Nilsson [5] – that the A_2 algorithm using h_2 is *more informed* than the A_1 algorithm using h_1 if, for all nongoal nodes n, $h_2(n) > h_1(n)$. We would expect intuitively that the more informed algorithm would need to expand fewer nodes to find a minimal cost path. Indeed, analogously to the similar result of the A^* algorithm, we can only prove that A_2 does not expand a node that A_1 does not either.

Theorem 5 *If A_1 and A_2 are two versions of A^{**} such that A_2 is more informed than A_1, then A_2 is not worse than A_1.*

Proof. We prove this result, following to Nilsson [5], using induction on the depth of a node in the spanning tree of the search graph of A_2 at termination.

First, if A_2 expands the node having zero depth, in this case this node must be the start node, then so must A_1. (If s is a goal node, neither algorithm expand any nodes.)

Continuing the inductive argument, we assume (the induction hypothesis) that A_1 expands all the nodes expanded by A_2 having depth d or less in the A_2 search graph. We must now prove that any node n expanded by A_2 and of depth $d+1$ in the A_2 search graph is also expanded by A_1. Let us suppose the opposite of this, namely that there is a node m having depth $d+1$ expanded by A_2 but it is not expanded by A_1. (We remark that this node m may not be a goal node since it is expanded by a graph-search algorithm.)

It is trivial that m had to get into the OPEN for A_2 if A_2 expanded it. According to Lemma 3, since A_2 has expanded node m, we have $f_2(m) \leq f^*(s)$ where $f_2(m)$ is the evaluation function value of m at its expansion.

According to the induction hypothesis, since A_2 has found a path from s to m where all ancestors of m are below the level d, these ancestors are also expanded by A_1. Thus, firstly, node m must be in OPEN for A_1. Let $f_1(m)$ denote the evaluation function value of m at the termination of A_1. It is obvious that $f_1(m) \geq f^*(s)$. Secondly, the recorded path from s to m in the A_1 search graph does not cost more than the path discovered by A_2, that is, $g_1(m) \leq g_2(m)$. Thirdly, for each ancestor v of node m on the recorded path from s to m in the A_1 search graph, $f_1(v) \leq f^*(s)$ since they have been expanded by A_1. Thus, by respecting the definition of the evaluation function of A^{**}, $g_1(v) + h_1(v) \leq f^*(s)$. Because of this, $f_1(m) = g_1(m) + h_1(m)$ follows from the inequation $f_1(m) \geq f^*(s)$.

Summarizing the consequences we get that

$$f_2(m) \leq f^*(s) \leq f_1(m) = g_1(m) + h_1(m) \leq g_2(m) + h_1(m) < g_2(m) + h_2(m) \leq f_2(m),$$

but this is a contradiction. □

3.2 Comparison of the A^{**} algorithm and other admissible graph-search algorithms

In this analysis we will use the natural definition of "equally informed" allowing the algorithms compared to have access to the same heuristic information while placing no restriction on the way they use it. Accordingly, we assume that the heuristic function is a part of the parameters that specify path-finding problem-instances and correspondingly, we will represent each problem-instance by the quadruple $P = (R, s, T, h)$ where $R = (N, A, c)$ is the representation δ-graph, s is the start node, T is the set of goal nodes, and h is the heuristic function. If the heuristic function is non-negative and admissible, then these problem-instances are called *admissible problems*. Moreover we suppose that these problems have got solutions.

As the A^{**} algorithm always finds the optimal solution in an admissible problem, our aim is to compare the A^{**} algorithm to other graph-search algorithms that can also find optimal solution. These algorithms are called admissible graph-search algorithms, or shortly admissible algorithms. Famous graph-search algorithms, as the A^* algorithm, the A^{**} algorithm, the B algorithm [4], or uniform-cost search [5], belong to this algorithm class.

In order to decide if an algorithm X dominates an algorithm Y, several criteria can be used. According to one of these criteria, X dominates Y relative to a set of problems if, in every problem, X is not worse than Y. Additionally,

if Y does not dominate X, i.e. in at least one problem X is better than Y, then we say that X strictly dominates Y relative to that set of problems.

However, now we do not have to compare two algorithms but two algorithm classes. First, the A** algorithm is a non-deterministic algorithm because its OPEN set can contain several nodes with the same evaluation function value and in order to select the best one, we need some rules to break these ties. By collecting all possible tie-breaking-rules we can get a set of deterministic A** algorithms instead of the original non-deterministic one. In this sense A** can be treated as an algorithm class. Secondly, we must compare this algorithm class with all members of the admissible algorithms, i.e. the class of the admissible algorithms. Thus we must extend the concept of the dominance of algorithm to algorithm classes.

The algorithm class X is said to *dominate* the algorithm class Y relative to a set of problems if in every problem for all deterministic versions y of Y there exists a member x of X so that x is not worse than y. X *strictly dominates* Y if X dominates Y and Y does not dominate X. (This definition corresponds to the Type-1 criterion of the paper of Dechter and Pearl. [1])

Dechter and Pearl have thoroughly analyzed the memory complexity of the A* algorithms and they have shown that neither algorithm A* nor any other admissible graph-search algorithm dominate all admissible algorithms. They have proven that the A** algorithm strictly dominates the A* algorithm relative to the admissible problems. Furthermore, they have mentioned the following theorem. (See it as Theorem 10 in Dechter and Pearl's paper. [1])

Theorem 6 *There is no admissible problem where all versions of A** expand a node skipped by a deterministic admissible graph-search algorithm.*

Unfortunately, the proof of this theorem has got a little mistake in Dechter and Pearl's paper. In their argumentation, a new admissible problem must be constructed from an elder one but the cost of the new arc adding to the elder one may be zero. In our assumption, however, all arc-costs of the problem of interest must be greater than a positive real number. This condition guarantees that the length of the optimal solution path is finite. At first I tried to improve the original proof but I could not. A new proof is presented here.

Proof. Let us suppose indirectly that P_1 is an admissible problem where all versions of A** expand an extra node skipped by a given deterministic admissible graph-search algorithm Y.

If n denotes the extra node expanded by a version of A** but skipped by Y and C_1^* is the optimal solution cost in the problem P_1, then it is obvious that

$f(n) \leq C_1^*$ at the time of the expansion of n (see Lemma 3). In addition, there must be at least one version of A^{**} (it will be called a^{**}) so that $f(n) < C_1^*$. If this version did not exist, the evaluation function value of each extra node would be identical to C_1^*. But it must be an optimal solution path found by Y avoiding the extra nodes on order to Y could find optimal solution. The set OPEN of all versions of A^{**}always records a node u from this optimal solution path with $f(u) \leq C_1^*$. It means that a version of A^{**} can be constructed in a way that it always prefers the node u even its own extra node, but it is a contradiction.

We are going to construct a new problem P_2 from the search graph (G) maintained by a^{**} over the problem P_1 at its termination appended by a new arc (n, t) where t is a new goal node. (This graph contains all nodes of P_1 expanded by Y.) Let the cost of this new arc be $(C_1^* - f(n))/2$. This is obviously a positive value. The start node of P_2 is the same one of P_1.The heuristic function h is identical to the heuristic function of P_1 except that $h(t) = 0$. Let C_2^* denote the optimal cost from s to t.

It is easy to see that $C_2^* \leq g(n) + c(n, t) \leq f(n) + c(n, t) < C_1^*$ (where $g(n)$ is the cost of the path from s to n, found by a^{**}) so that the only optimal solution path of P_2 goes through the node n to the node t.

Now we will show that P_2 is an admissible problem. The heuristic function was admissible in the problem P_1 thus $h(u) \leq h_1^*(u)$ for all node u of G where $h_1^*(u)$ is the optimal path-cost from u to a goal node of G. Let $h_2^*(u)$ be the optimal path-cost from the node u to a goal node of $G \cup (n, t)$. Since only one new goal node was added it is conceivable that $h_2^*(u) < h_1^*(u)$ for a node u if there is a path from the node u to the node t in P_2. To discover this path, the node u must have been expanded by algorithm a^{**} during the solution of P_1 so it must be expanded during the solution of P_2. By respecting the definition of A^{**}, $f(u) \geq g(u) + h(u)$ for all open node u. If $h(u) > h_2^*(u)$, then we would have

$$f(u) \geq g(u) + h(u) > g(u) + h_2^*(u) \geq g_2^*(u) + h_2^*(u) = f_2^*(u) \geq C_2^*$$

but this contradicts that $f(u) \leq C_2^*$ (Lemma 3). Thus for all node u of G, $h(u) \leq h_2^*(u)$ as well for the new goal node t since $h(t) = 0$ by definition.

In searching P_2, algorithm Y must behave in the same way as in problem P_1, it must avoid expanding n and halting with cost C_1^*, which is higher than that found by a^{**}. This contradicts Y being an admissible algorithm and it should find the optimal solution. □

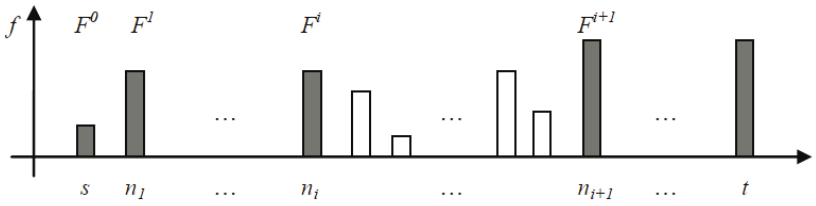

Figure 1: Execution diagram and its thershold values

4 Time complexity of the A^{**} algorithm

The running time of a graph-search algorithm depends on the number of its iteration and on the time of one iteration. The latter factor can be estimated on the basis of only the knowledge of the problem wanted to be solved, and the first factor depends primaly on the strategy of the graph-search. Thus we will analyze just this first factor. This will be given as a function of the number of closed nodes at termination. (We suppose that the problems of our interest have got a solution thus our algorithms terminate.) Hereinafter k will denote the number of closed nodes at termination.

It is clear that the best running time is k since at least k iterations are needed to expand k nodes if every node is expanded only once. But in many problems a node may expand several times. It is known that, in the worst case, the time complexity of the A^* algorithm is 2^{k-1} but there is a modification of A^*, this is algorithm B; its running time is at most $\frac{1}{2}k^2$ in the worst case. [4]

An excellent tool to present the execution of a graph-search algorithm and to calculate its time complexity is the execution diagram (Figure 1). This diagram enumerates the expanded nodes with their current evaluation function values in the order of their expansions (the same node may occur several times). It is trivial that the first value is $f(s)$ and the last one is the value of the goal node found. (The goal node is selected for expansion but not expanded.) A monotone increasing subsequence F^i $(i = 1, 2, \dots)$ is constructed from the values of the diagram so that it starts with the first value $f(s)$ and then the closest non less one must always be selected. The selected values are called threshold values and the nodes belonging to these values are called threshold nodes. Let n^i denote the i^{th} threshold node where $F^i = f(n^i)$ is its threshold value. The set of nodes that are expanded between the i^{th} and the $(i+1)^{th}$ threshold nodes is called the i^{th} ditch.

Let us introduce some further notations:

- OPEN^i – the OPEN set just before the i^{th} threshold node is expanded by A^*
- $g^i(u)$ – the value $g(u)$ just before the i^{th} threshold node is expanded by A^*
- $f^{A^*}(u)$ – the evaluation function value $f(u)$ according to A^*
- $f^{A^{**}}(u)$ – the evaluation function value $f(u)$ according to A^{**}

Theorem 7 *Consider an admissible path-finding problem where the threshold values of the execution diagram of the A^* algorithm form a strictly monotone increasing number sequence. Algorithm A^{**} expands the threshold nodes of A^* in the same order and with the same threshold values if these algorithms use the same tie-breaking-rule.*

Proof. We prove this theorem using induction on the threshold nodes of the diagram of the A^* algorithm. The first threshold node is the start node and this same node is expanded by A^{**} at first. It is clear that $f^{A^*}(s) = f^{A^{**}}(s)$. Let us suppose that until the i^{th} threshold node the statement is true, that is, at the expansion of n^i both algorithms maintain the same search graphs, the same sets OPEN, the same cost function (g) values, the same parent pointers (π) and $F^i = f^{A^*}(n^i) = f^{A^{**}}(n^i)$. (This is the induction hypothesis.) We will show that between n^i and n^{i+1}, the A^* algorithm and the A^{**} algorithm expand the same nodes. It follows that the induction statement is also true for the $(i+1)^{th}$ threshold node.

Let us describe the nodes expanded by A^* in the i^{th} ditch. A node m belongs to this ditch if it gets into OPEN after the expansion of n^i and $f^{A^*}(m) < F^i$. The condition of m getting into OPEN is that there exists a path $s \to m$ via the node n^i so that it is found after the expansion of n^i and all nodes on this path between n^i and m are also in this ditch and the cost of this path is cheaper than all other path $s \to m$ discovered before. It also means that, for all nodes u on that path between n^i and m, $g(u) + h(u) < F^i$ holds. [3]

On the one hand, if the node m of the i^{th} ditch is put into OPEN by the A^{**} algorithm, then it must be expanded before the next threshold since

$$\begin{aligned} f^{A^{**}}(m) &= \max_{u \in s \to^{\pi} m} [g(u) + h(u)] = \\ &= \max\Big(\max_{u \in s \to^{\pi} n^i} [g(u) + h(u)], \max_{u \in n^i \to^{\pi} m} [g(u) + h(u)] \Big) = \\ &= \max\Big(f^{A^{**}}(n^i), \max_{u \in n^i \to^{\pi} m} [g(u) + h(u)] \Big) = \\ &= F^i < F^{i+1}. \end{aligned}$$

On the other hand, let us suppose now indirectly that A^{**} expands a node before the expansion of n^{i+1} that does not belong to the i^{th} ditch of A^*. Let us take the first such node in the order of the expansions of A^{**}. This node will be denoted by m. It is obvious that m must be put into OPEN by A^*. If $f^{A^{**}}(m) < F^{i+1}$, then the node m should be expanded by the A^* algorithm before n^{i+1}, thus m would belong to the i^{th} ditch of A^*. If $f^{A^{**}}(m) \geq F^{i+1}$, then all nodes of the i^{th} ditch would precede the expansion of m, thus n^{i+1} would be put into OPEN, too. The only chance of the expansion of m preceding the expansion of n^{i+1} is that $f^{A^{**}}(m) = F^{i+1}$. It is clear that $f^{A^{**}}(m) = g(m) + h(m)$. In this case the node m would be put into OPEN by A^* after the expansions of the i^{th} ditch and $f^{A^*}(m) = F^i$ would follow from $f^{A^{**}}(m) = g(m) + h(m)$. If the tie-breaking-rule of A^{**} selected the node m instead of n^{i+1}, then A^* would do the same, that is n^{i+1} would not be the next threshold node of A^*. This is a contradiction. □

There are two interesting side effects of this proof. First, the A^* algorithm and the A^{**} algorithm expand the same nodes between two neighboring threshold nodes. Secondly, the evaluation function values of the nodes expanded by A^{**} in the i^{th} ditch are equal to the i^{th} threshold value F^i.

We underline that the difference of execution diagrams of algorithms A^* and A^{**} under the constraints of the previous theorem is that how many times a node is expanded between two neighboring thresholds and how much its evaluation function value is. The number of the expansions of A^{**} partly depends on the order of the expansions of the nodes having the same evaluation function value. If the version of A^{**} is introduced, whose tie-breaking rule prefers the node having less costly recorded path from s, it can be prevented that the same node is expanded several times in one ditch because, after the expansion of a node, algorithm cannot find a better path to this node in this ditch. Thus the running time of this version is not worse than any version of A^*.

But what can we say when the threshold values of the execution diagram of A^* is not strictly monotone but just monotone? In this case some neighboring threshold values may be equal. In the ditches after these thresholds the A^{**} algorithm expands the nodes with the same evaluation function value. Because our tie-breaking-rule defined in the previous paragraph may prefer n^{i+1} for expansion to some node of the i^{th} ditch and this node will be expanded later or never, thus the number of the expansions of this tie-breaking-rule may be less than the number of the expansions of the best version of the A^* algorithm. On the Figure 2 our tie-breaking-rule of the A^{**} algorithm expands every node

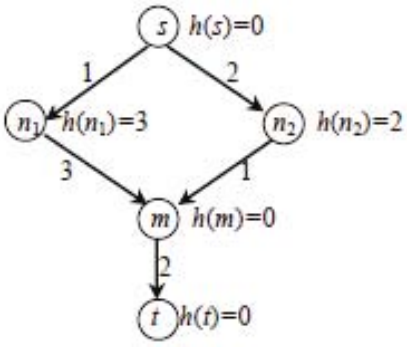

Figure 2: Example where the A^{**} algorithm is faster than algorithm B

only once but the best version of the A^* algorithm, this is algorithm B [4], expands the node m twice.

5 Summary

The A^{**} algorithm was defined by Dechter and Pearl [1] but it was introduced in a bit different form by Russell and Norviq. [6] I have shown that these versions do not differ (Theorem 1). The fact that the A^{**} algorithm is an admissible graph-search algorithm is a well-known property. But I have mentioned that, if the heuristic function applied by algorithm is not admissible but only non-negative, then A^{**} always finds a not necessarily optimal solution (if there exists a solution). I have proved that the evaluation function values of the nodes expanded by A^{**} form a monotone increasing number sequence (Theorem 4).

Then we have extended the the concept of "better-informed" derived from Nilsson [5] onto the A^{**} algorithm. It allows us to compare the memory complexity of two algorithms A^{**} using different heuristics (Theorem 5). Perhaps it is much more important to compare the memory requirement of A^{**} with other graph-search algorithm, especially the A^* algorithm using the same heuristics. All of this has been done in the excellent work of Dechter and Pearl. However there is a statement in that paper about the A^{**} algorithm whose proof has got a mistake. I have given another proof of that statement (Theorem 6).

At last I have shown (Theorem 7) that the time complexity, more precisely, the number of the expansions of a certain version of the A^{**} algorithm is not worse than the versions of A^*.

References

[1] R. Dechter, J. Pearl, Generalized best-first search strategies and optimality of A*, *Journal ACM*, **32,** 3, (1985) 505–536. ⇒192, 199, 204

[2] I. Fekete, T. Gregorics, L. Zs. Varga, Corrections to graph-search algorithms. *Proc. of the Fourth Conference of Program Designers*, ELTE, Budapest, June 1–3, 1988. ⇒191

[3] T. Gregorics, Which of graphsearch versions is the best? *Annales Univ. Sci. Budapest., Sect. Comput.* **15** (1995) 93–108. ⇒191, 202

[4] A. Martelli, On the complexity of admissible search algorithms. *Artificial Intelligence*, **8,** 1 (1977) 1–13. ⇒198, 201, 204

[5] N. J. Nilsson, *Principles of Artificial Intelligence.* Springer-Verlag, 1982. ⇒190, 191, 195, 196, 197, 198, 204

[6] S. J. Russell, P. Norvig, *Artificial Intelligence. A Modern Approach.* Prentice Hall Inc., 1995. ⇒193, 204

Received: October 7, 2014 • Revised: October 30, 2014