Annales Univ. Sci. Budapest., Sect. Comp. 52 (2021) 163–176

TIME COMPLEXITY OF A^{**}

Tibor Gregorics (Budapest, Hungary)

Communicated by András Benczúr

(Received April 30, 2021; accepted August 5, 2021)

Abstract. This paper focuses on the time complexity of three famous heuristic graph-search algorithms, namely the algorithm A^*, B and A^{**}, and analyzes their executions. The concept of the super-threshold will be introduced which draws attention to certain steps of the executions of the graph-search algorithms. It will be shown that the states of the executions of A^*, B and A^{**} at these moments are identical and the executions of these algorithms can differ at most between two such adjacent moments. Thereafter a secondary tie-breaking rule for the algorithm A^{**} will be defined in order for this algorithm to be better than the other two.

1. Introduction

Few years ago I made some remarks on the algorithm A^{**} [1] in the paper [2]. At the end of that work the time complexity of A^{**} was compared to the algorithm A^* [4] under a special condition. In the last paragraph I mentioned that our results may also be valid in a general case and A^{**} with an extra tie-breaking rule may be better than the algorithm B [5], which is famous for being better than A^*. However, I think I owe accurate proof and the time is ripe for completing that work.

Firstly, a short overview of heuristic graph-search will be presented. After the general version of the graph-search algorithm is outlined, the algorithms A^*, B and A^{**} will be defined as special heuristic graph-searches. (Section 2)

Secondly, my earlier results about the time complexity of A^{**} will be summarized with a few new remarks. (Section 3 and Section 4)

Key words and phrases: Heuristic graph-search, algorithm A^*, A^{**} and B.
2010 Mathematics Subject Classification: 68T20, ACM I.2.8.
https://doi.org/10.71352/ac.52.163

Thirdly, the concept of the super-threshold will be defined which can help to compare the executions of A^{**}, A^* and B over the same arbitrary path-finding problems. It will be shown that these algorithms achieve the same states from time to time during their executions. (Section 5)

Fourthly, it will be proved that A^{**} with an extra tie-breaking rule is never worse than A^* or B with respect to the number of expansions (i.e. the number of iterations), and some path-finding problems will be demonstrated where A^{**} terminates with fewer expansions than the other two algorithms. (Section 6)

2. Basic concepts and related works

A^{**} was introduced by Dechter and Pearl [1] when they analyzed the memory complexity of A^*. These algorithms are graph-searches which gradually discover a δ-graph going out from a given start node and trying to find a path to any goal node. A δ-graph is a directed arc-weighted (not necessarily finite) graph where the number of the outgoing arcs of each node is finite and the cost of each arc is greater than or equal to a given positive constant number δ.

A graph-search algorithm stores all the paths outgoing from the start node (denoted by s) that are discovered during the search. The graph formed by these paths and stored by the graph-search is the search graph that is denoted by G. The algorithm maintains a so-called parent pointer function (denoted by π), which shows one parent node of each discovered node. Since there may be several discovered paths from the start node to the same node, one path is always highlighted by the values of the parent pointer function. In order to find the minimal cost path, the algorithm uses a cost function (denoted by g). For every discovered node m, $g(m)$ gives the cost of the preferably cheapest discovered path from s to m. The functions π and g are calculated by the graph-search algorithms themselves.

The last nodes of the discovered paths, whose sequels have not been known yet or enough, are stored separately. These last nodes are called open nodes. The set of the open nodes is denoted by $OPEN$. In each step, these open nodes are compared according to a so-called evaluation function (denoted by f) and the one which has got the smallest evaluation function value, i.e. which is the best open node, is selected for expansion. It means that if the selected node (n) is a goal node, the algorithm terminates, otherwise the children of the selected node ($Children(n)$) are generated.

The general graph-search algorithm, which is the ancestor of all special variants, is the following:

```
procedure Graph-search
    G := ({s}, ∅) ; OPEN := {s} ; π(s) := nil ; g(s) := 0
    while OPEN ≠ ∅ loop
        n := arg min{f(m)|m ∈ OPEN}
        if goal(n) then return solution is shown by π 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 ∪ (Children(n), {(n, m)|m ∈ Children(n)})
    endwhile
    return there is no solution
end
```

(The signs $\notin$ and $\cup$ are not the usual set theoretic operators. The notation $m \notin G$ shows that the node m is not in the set of the nodes of the search graph G. The operator $\cup$ must be applied on graphs rather than on sets: more precisely, it must be applied separately on the nodes and on the arcs.)

Different graph-search algorithms apply different evaluation functions. For example, A^*, which was published by Hart, Nilsson and Raphael in 1968 in [4], uses the sum of two functions. By definition,

$$f^{A^*}(m) ::= g(m) + h(m)$$

where m is an open node, g is the cost function, and h is a heuristic function. The value $h(m)$ estimates the cost of the optimal path from the node m to the goal nodes. The function h has to be admissible, i.e. $h(m)$ has to be a non-negative lower bound on the remaining optimal cost.

Many researchers have investigated the complexity of A^* and have given its improved modifications. Perhaps the most remarkable modification is B defined by Martelli in 1977 in [5]. This algorithm alternates between two different evaluation functions depending on a permanently changing threshold value (this value is denoted by F). One of these evaluation functions is the function g and the other one is the function $g + h$ (this is the evaluation function of A^*).

$$f^B(m) ::= \begin{cases} g(m) & \text{if } g(m) + h(m) < F \\ g(m) + h(m) & \text{if } g(m) + h(m) \geq F \end{cases}$$

Initially, the threshold value F is the $g + h$ value of the start node and in each step, if there is no open node whose $g + h$ value is below the current threshold value, this threshold value must be overwritten by the $g + h$ value of the currently selected open node.

A^{**}, which is in the focus of this paper, is another modification of A^*. Its evaluation function takes some ancestors of the current node into consideration so that the value $f(m)$ is equal to the maximum of the values $g(k)+h(k)$, where g is the cost function, h is an admissible heuristic function, and k is a node via the path recorded by π from the start node to the node m ($s \rightarrow^{\pi} m$). [1]

$$\begin{aligned} f^{A^{**}}(m) &::= max_{u \in s \rightarrow^{\pi} m}\ g(u) + h(u) = \\ &= max\{g(u) + h(u) | u \in s \rightarrow^{\pi} m\} \end{aligned}$$

Because of this definition the evaluation function values of the nodes expanded by A^{**} form a monotone increasing sequence in the order of their expansions [2].

In addition, the definition of A^{**} includes a tie-breaking rule. If there are several best open nodes and the set of the best open nodes contains a goal node, A^{**} selects the goal node for expansion and terminates immediately. This rule tries to manage the non-deterministic behavior of the graph-search algorithm with resolving ties, however, in order to transform it into a deterministic form, additional tie-breaking rules may be required.

Russel and Norviq, in their famous book [6] suggested the following evaluation function:

$$f(m) ::= max\{g(m) + h(m), f(\pi(m))\}$$

together with the rule that the value $f(m)$ is always recalculated whenever a better (a cheaper) path to m is found. It can be observed that the evaluation function of Dechter and Pearl and the evaluation function of Russel and Norviq are similar. However, it is obvious that the computational cost of Russel and Norviq's function is better. I have shown that the execution of A^{**} is almost identical to the execution of the graph-search algorithm of Russel and Norviq if these algorithms use the same tie-breaking rules.[2] The only difference between these algorithms is that the version of Russel and Norviq stores the evaluation function values of the non-open nodes, too.

It can be proved that all algorithms mentioned above always find an optimal solution path even in infinite δ-graphs if there exists a solution path.

3. Execution diagram and the time complexity

The time complexity of a graph-search algorithm depends on the cost of one iteration and the number of iterations. One iteration consists of two parts: the selection of the best open node and its expansion. These are dependent on the computation cost of the evaluation function (in this regard, there are no significant difference between A^*, A^{**} and B) and the structure of the

representation graph of the path-finding problem (this is independent of the graph-search algorithm). Thus the comparison of the time complexity of the mentioned algorithms may be limited to the measurement of the number of expansions (i.e. the number of iterations) with respect to the number of the expanded nodes.

These expansions can be described in the so called execution diagram (Figure 1) whose horizontal axis enumerates the nodes that are expanded by the algorithm in the order of their expansions. The same node can occur several times in this diagram because a node can be expanded more than once by a graph-search algorithm. The vertical axis shows the f values of the expanded nodes. Using this diagram different graph-search algorithms can be compared according to the number of their iterations. A monotone increasing sub-

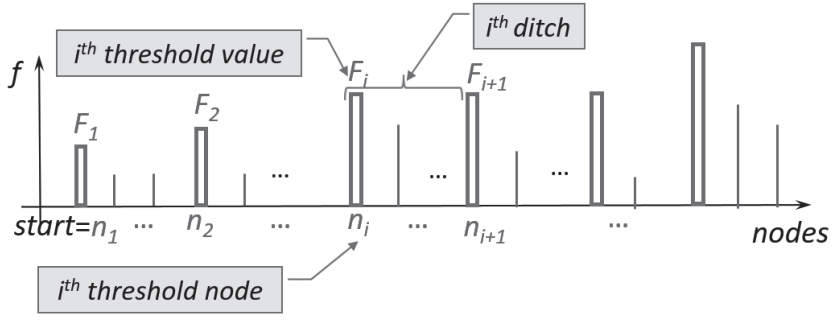

Figure 1. Execution diagram and its thresholds
This diagram contains the f values of the expanded nodes in the order of the expansions of a graph-search algorithm. A monotone increasing subsequence can be selected from these values. These values are the threshold values.

sequence can be constructed from the values of the diagram so that it starts with the first value and then the next one is always the closest value which is greater than or equal to the previous selected one. These selected values are called threshold values, the nodes belonging to them are the threshold nodes, and a sequence of the neighboring nodes of this diagram starting with a threshold node and including all nodes before the next threshold is called a ditch. In the Figure 1, n_i is the i^{th} threshold node, F_i is the i^{th} threshold value, and the section between the expansions of n_i and n_{i+1} including n_i is the i^{th} ditch.

We remark that a node may be a threshold node at most only once at its first expansion if the evaluation function is decreasing [3] as it holds on A^*, A^{**} and B.

4. When thresholds of A^* are strictly monotone increasing

In [2] the executions of A^* and A^{**} have been compared above the problems whose threshold values in the execution diagram deriving from the execution of A^* form a strictly monotone increasing sequence. It was proved that A^{**} expands all the threshold nodes of A^* with the same threshold values and in the same order. (Figure 2)

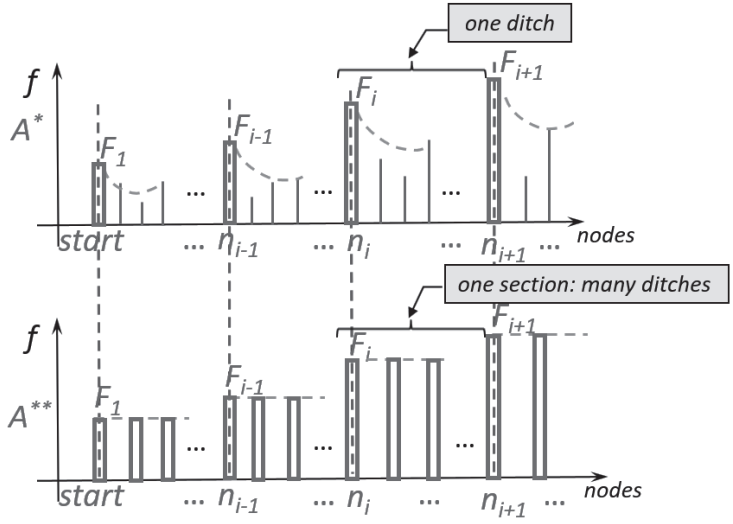

Figure 2. Execution diagrams of A^* and A^{**}
Both diagrams can be divided into the same sections
along the strictly monotone increasing thresholds of A^*.

There were two interesting side effects of the proof of this theorem. Firstly, A^* and A^{**} expand the same nodes between two neighboring thresholds of A^*. Secondly, the f values according to A^{**} (shortly the $f^{A^{**}}$ values) of all nodes are equal to F_i between the expansions of n_i and n_{i+1} because the $f^{A^{**}}$ values of the nodes form a monotone increasing sequence but the $f^{A^{**}}$ values of the nodes expanded before n_{i+1} are not allowed to be greater than F_i. It follows that the execution of A^{**} consists of many one length ditches.

The executions of A^* and A^{**} over the same problem can be divided into the same kind of sections along the thresholds of A^*. In the corresponding sections the same nodes are expanded by these algorithms. The difference between these corresponding sections is the number of times that a node is expanded and the order of expansions of the nodes belonging to these sections.

One section is analogous to one ditch in the execution of A^* thus the first f^{A^*} value, which is a threshold, is greater than the rest. Meanwhile, the $f^{A^{**}}$ values of the execution of A^{**} are identical alongside in each section thus a section consists of many one-length ditches in the diagram of A^{**} (see in Figure 2). That is why an appropriate tie-breaking rule is able to change the order of expansions, moreover, the number of expansions inside the sections of A^{**}.

Let A^{**} use the cost function g as a secondary tie-breaking rule. (The primary tie-breaking rule of A^{**} prefers the goal nodes.) It means that, if there are several best open nodes and none of them is a goal node, the best open node which has got the smallest g value must be selected for expansion. Since the evaluation function values of the expanded nodes inside every section are equal, the usage of this secondary tie-breaking rule is as if the evaluation function of A^{**} were exchanged for the cost function g inside the sections.

This was the basic idea of Martelli when he introduced B [5]. It is a well-known fact that the thresholds of B are equal to the thresholds of A^*, and when these thresholds are strictly monotone increasing, A^{**} also expands them in the same order as it has been shown in [2]. In a section (in a ditch of B and also A^*), B uses the cost function g as an evaluation function in the same way as A^{**} does it with my secondary tie-breaking rule. It follows that B and A^{**} with the secondary tie-breaking rule expand the same nodes in the same order. Therefore their time complexity, i.e. the number of their iterations is the same as well. Since Martelli has proved that B expands a node at most once in a section, this time complexity is quadratic with respect to the number of the expanded nodes in the worst case. Meanwhile, A^* can produce an exponential running time [5].

5. When thresholds of A^* are non-strictly monotone increasing

What can we say about the time complexity of A^{**} over the problems whose threshold values deriving from the execution of A^* are non-strictly monotone increasing? In this case a new concept must be introduced.

Definition 5.1. A strictly monotone increasing subsequence can be constructed from the values of the execution diagram of a graph-search algorithm so that it starts with the first value and then the next one is always the closest value which is greater than the previously selected one. These selected values are called super-threshold values, the nodes belonging to them are the super-threshold nodes, and a sequence of the neighboring nodes of this diagram starting with a super-threshold node and including all nodes before the next super-threshold is called a super-ditch.

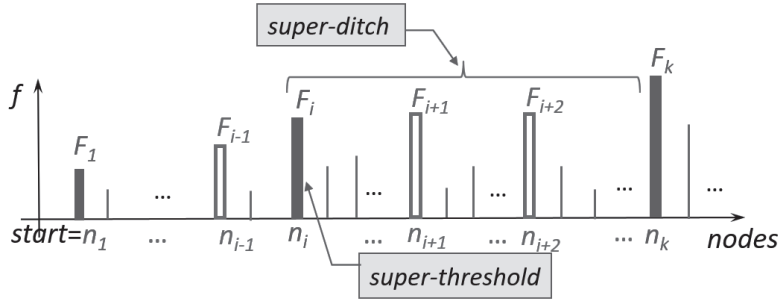

Figure 3. Super-thresholds
A super-ditch is a maximal-length section of
the neighboring ditches having the same threshold value.

It can be observed that a super-ditch is a maximal-length section of the neighboring ditches having the same threshold values. The super-ditch starting with the expansion of n_i can be named the super-ditch of n_i. The f values of the expansions in this super-ditch are less than or equal to F_i. Since the f values of the nodes expanded by A^{**} form a monotone increasing sequence [2], the values of the super-ditch of n_i are equal to F_i in the execution of A^{**}.

Lemma 5.1. *Let a path-finding problem be given with its graph representation. Let us suppose that when A^* and A^{**} expands the node n_i with the F_i value, this node is a super-threshold node in the executions of both algorithms and these algorithms maintain the same search graphs (G_i), the same open sets ($OPEN_i$), the same recorded paths (π_i) and the same cost function (g_i) values of the nodes of G_i. In this case the next super-threshold will be the same in the executions of these algorithms, and these algorithms will maintain the same search graphs, the same open sets, the same recorded paths and the same cost function values at the execution of this next super-threshold.*

Proof. Let $f_i^{A^*}(m)$ and $f_i^{A^{**}}(m)$ denote the f value of the node m ($m \in \in OPEN_i$) according to A^* or A^{**} at the expansion of the super-threshold n_i. We have that $f_i^{A^*}(n_i) = f_i^{A^{**}}(n_i) = F_i$.

Let us describe the set of the nodes expanded by A^* in the super-ditch of n_i. This set is denoted by SD_i. Every node which gets into the open set and whose f value is less than or equal to F_i is expanded before the next super-threshold whose f value is greater than F_i. Thus a node m belongs to SD_i if it is either already in $OPEN_i$ and $f_i^{A^*}(m) = F_i$ or it will get into the open set during the expansions of the nodes of SD_i and $f^{A^*}(m) \leq F_i$. In the latter case there must exist a path α from some node n of $OPEN_i$ to m through SD_i. If m was in G_i, the cost of the path from s to m including α

$(s \to n \to^{\alpha} m)$ must be cheaper than the cost of the former recorded path from s to m, i.e. $g_i(n) + c^{\alpha}(n,m) < g_i(m)$. (The cost of the path $n \to^{\alpha} m$ is denoted by $c^{\alpha}(n,m)$.) Because of $f^{A^*}(m) = g_i(n) + c^{\alpha}(n,m) + h(m)$ (see the definition of A^*), we have $g_i(n) + c^{\alpha}(n,m) + h(m) \leq F_i$. Formally

$$
\begin{aligned}
(5.1)\quad SD_i \quad = \{m \in N \mid \quad & \text{either } m \in OPEN_i \text{ and } f_i^{A^*}(m) = F_i \\
& \text{or } \exists n \in OPEN_i \text{ and } \exists \alpha \in \{n \to m\} \text{ so that} \\
& \qquad \forall k \in \alpha - \{m\} : k \in SD_i \\
& \qquad \text{and} \quad g_i(n) + c^{\alpha}(n,m) < g_i(m) \text{ if } m \in G_i \\
& \qquad \text{and} \quad g_i(n) + c^{\alpha}(n,m) + h(m) \leq F_i \ \}.
\end{aligned}
$$

We prove first that A^{**} expands all nodes of SD_i in the super-ditch of n_i. Let m be a node in the set $SD_i \cap OPEN_i$. If this node is n_i, it is expanded by A^{**} (see the conditions of this lemma). Otherwise, the inequality $f_i^{A^{**}}(m) \geq F_i$ is hold because the node n_i is selected instead of m. We also have $f_i^{A^{**}}(m) \leq F_i$ because $f_i^{A^{**}}(m) = max_{u \in s \to^{\pi_i} m} \{g_i(u) + h(u)\} = max_{u \in s \to^{\pi_i} m} f_i^{A^*}(u)$ where $f_i^{A^*}(m) = F_i$ (see (5.1)) and the other nodes in the path $s \to^{\pi_i} m$ has also been expanded by A^* in the super-ditch of n_i thus $f_i^{A^*}(u) \leq F_i$. To summarize the above, $f_i^{A^{**}}(m) = F_i$ and it follows that m is expanded by A^{**} in the super-ditch of n_i.

Now let us suppose that every node in SD_i which can be reached from a node of $OPEN_i$ via a k steps path $(k > 0)$ through SD_i is expanded by A^{**} in the super-ditch of n_i. (This is the induction hypothesis.) Let us have a node m in SD_i so that there exists a node n in $OPEN_i$ and a path α from n to m through SD_i whose length is $k+1$.

On the one hand, m gets into the open set of A^{**} since the path α is discovered after the expansion of n_i (see the induction hypothesis) and either this path is the only one that leads to m, or this path is cheaper than all paths discovered before the expansion of n_i (see the definition of (5.1)).

When the path α is discovered, we have

$$
(5.2) \qquad g^{A^{**}}(u) \leq g_i(n) + c^{\alpha}(n,u) \text{ for all } u \in n \to^{\alpha} m
$$

where $g^{A^{**}}$ denotes the g cost value computed by A^{**}.

On the other hand,

$$\begin{aligned}
f^{A^{**}}(m) &= max_{u \in s \to^{\pi} m}\{g^{A^{**}}(u) + h(u)\} = \\
&= max\{max_{u \in s \to^{\pi_i} n}\{g^{A^{**}}(u) + h(u)\}, \\
&\qquad max_{u \in n \to^{\alpha} m}\{g^{A^{**}}(u) + h(u)\}\} = \\
&\qquad \text{where } n \in OPEN_i \cap \{s \to^{\pi} m\} \\
&= max\{f_i^{A^{**}}(n), max_{u \in n \to^{\alpha} m}\{g^{A^{**}}(u) + h(u)\}\} = \\
&\qquad \text{since} f_i^{A^{**}}(n) = F_i \text{ because } n \in SD_i \cup OPEN_i \\
&= max\{F_i, max_{u \in n \to^{\alpha} m}\{g^{A^{**}}(u) + h(u)\}\} \leq \\
&\qquad \text{see } (5.2) \\
&\leq max\{F_i, max_{u \in n \to^{\alpha} m}\{g_i(n) + c^{\alpha}(n,u) + h(u)\}\} \leq F_i
\end{aligned}$$

(In the last step we used that each node in the path α is in SD_i.) Therefore m is expanded by A^{**} in the super-ditch of n_i.

We finish the proof by showing that there are no nodes outside SD_i which are expanded by A^{**} in the super-ditch of n_i. Let us suppose indirectly that there exist such nodes and take the first one according to the sequence of expansions of A^{**}. This node is denoted by m. In order for m to be expanded in the super-ditch of n_i it must get into the open set and $f^{A^{**}}(m) = F_i$ must hold. It means that either m is in $OPEN_i$, or there is a path α from a node n of $OPEN_i$ to m through SD_i (see indirect assumption) so that if m is in G_i, then $g_i(n)+c^{\alpha}(n,m) < g_i(m)$. But $g_i(n)+c^{\alpha}(n,m)+h(m) \leq f^{A^{**}}(m)$ because of the definition of A^{**}, thus $g_i(n) + c^{\alpha}(n,m) + h(m) \leq F_i$ holds. Therefore the node m must belong to SD_i and this is a contradiction.

All in all, A^{**} expands the same nodes like A^* in the super-ditch of n_i thus these algorithms store the same information about the problem when they must select the next super-threshold node. ■

Theorem 5.1. *A^{**} expands the same super-threshold nodes with the same super-threshold values in the same order like A^* or B.*

Proof. We prove this theorem using induction on the super-threshold nodes of the execution diagram of A^*. It is enough to examine the connection of A^* and A^{**} since B expands the same threshold nodes with the same threshold values in the same order like A^* [5], and these thresholds include the super-thresholds.

The first super-threshold node is the start node and this same node is expanded by A^{**} at first. It is clear that $f^{A^*}(s) = f^{A^{**}}(s) = F_1$. Let us suppose that until the expansion of the super-threshold node n_i the statement is true, that is, at the expansion of n_i both algorithms maintain the same search graphs (G_i), the same open sets ($OPEN_i$), the same recorded paths (π_i), the same cost function values of the nodes of G_i and $F_i = f_i^{A^*}(n_i) = f_i^{A^{**}}(n_i)$.

(This is the induction hypothesis.) We have shown in the previous lemma (Lemma 5.1) that these algorithms expand the same nodes in the super-ditch of n_i thus after this process they store the same information about the problem and they expand the same next super-threshold node. ■

Corollary 5.1. *The only difference of the executions of A^*, A^{**}, and B is how many times a node is expanded in a super-ditch and what is the order of their expansions.*

6. A^{**} is better than A^* or B

Now we are going to focus on the differences between A^{**} and the other two algorithms.

Theorem 6.1. *A^{**} using the cost function g as a secondary tie-breaking rule expands a node at most once in a super-ditch.*

Proof. Let us suppose indirectly that A^{**} expands the node m twice in the super-ditch of n_i. It means that there are two different paths (α and β) from the start node through one-one node of $OPEN_i$ to m and both paths have been discovered in the super-ditch of n_i. It can only happen that the more expensive path is discovered at first. Let us suppose that the cost of β is cheaper than the cost of α (If there were more than two paths, let α and β be the cheapest ones.)

When the node m is expanded at first after the path α has been discovered, $f^{A^{**}}(m) = max_{u \in s \to^{\alpha} m} \{g(u) + h(u)\} = F_i$ holds, where $g(m)$ is the cost of α. At the same time on the path β there must be a node m' which is in the open set and $f^{A^{**}}(m') = max_{u \in s \to^{\beta} m'} \{g(u) + h(u)\}$. We have $f^{A^{**}}(m') = F_i$ since the node m' will be expanded in the current super-ditch and there are no other cheaper path from the start node to m' which is discovered in this super-ditch.

It can be seen that when the node m is expanded at first, m and m' are in the open set and $f^{A^{**}}(m') = f^{A^{**}}(m)$. It means that the secondary tie-breaking rule selects the node m for expansion instead of the node m', i.e. $g(m) \leq g(m')$ must hold. But $g(m')$ is cheaper than the cost of whole β and the cost of β is cheaper than the cost of α (see the indirect assumption), thus $g(m') < g(m)$. This is a contradiction. ■

The Lemma 6.1 shows that A^{**} expands a node only once in a super-ditch. Comparing this with the Corollary 5.1, we can say that the number of expansions of A^{**} can never be larger than the number of expansions of A^* and B. It follows the next corollary.

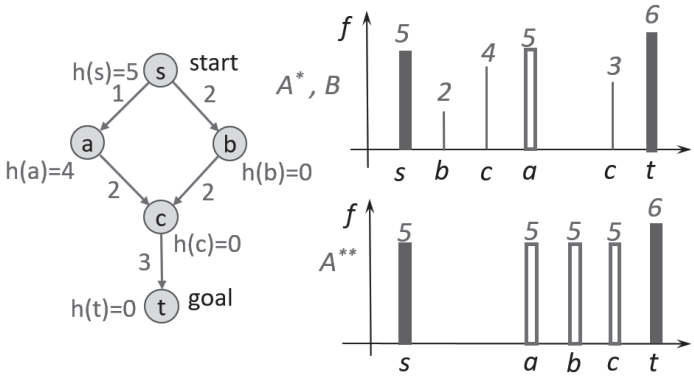

Figure 4. Example where A^{**} is better than A^* and B
The node c is expanded by A^{**} only once since A^{**} expands the node a before the node b and the node c because of its secondary tie-breaking rule.

Corollary 6.1. *The time complexity of A^{**} can never be worse than the time complexity of A^* and B.*

Now we are going to show that it can be better. In the following examples, A^{**} terminates earlier than the other algorithms because of its tie-breaking rules.

In the first example (Figure 4), the node c is expanded twice by A^* and B in two different ditches, but these ditches form only one super-ditch, where A^{**} with my secondary tie-breaking rule expands this node only once.

In the second example (Figure 5), the node a and the node b race in the open set after the first expansion of each algorithm. In the second step, A^* and B expand b, A^{**} expands a (because of the different evaluation functions). However, all algorithms put the goal node t in the open set. In the third step, A^* and B expands a (because its f value is less than the f value of the goal node t), but A^{**} terminates since the f value of b and t is the same and, in this case, the primary tie-breaking rule works and the goal node is selected for expansion.

7. Summary

The concept of super-ditch (Definition 5.1) made it possible to divide the executions of the graph-search algorithms into successive sections (super-ditches) and compare the states of the executions of some famous algorithms at the boundary of these sections. It has been shown that the states of the executions

of A^{**}, A^* and B are the same at the end of every super-ditch (Theorem 5.1). The only difference between the executions of these algorithms is the number of times that a node is expanded in the corresponding super-ditches and the order of expansions of the nodes belonging to these super-ditches (Corollary 5.1).

Adding another tie-breaking rule to the algorithm A^{**}, it expands a node at most once in a super-ditch. (Theorem 6.1) Thus the time complexity, more precisely, the number of the expansions of A^{**} is never larger than the number of the expansions of A^* and B. (Corollary 6.1).

At the end, two examples (Figure 4, Figure 5) were presented where A^{**} with our secondary tie-breaking rule is faster than the other algorithms. Thus we can say that A^{**} is better than A^* and B with respect to their time complexity.

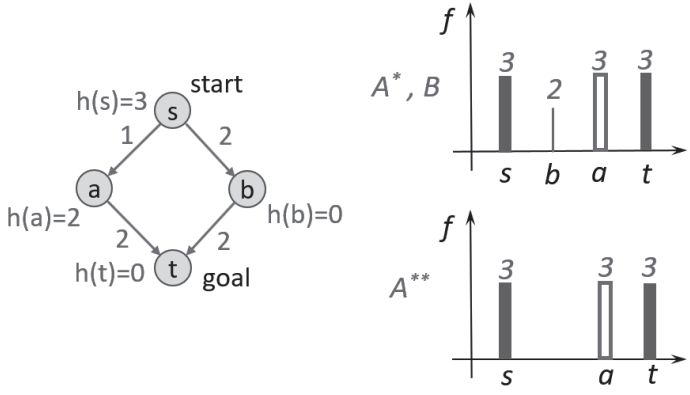

Figure 5. Example where A^{**} is better than A^* and B
A^{**} terminates earlier because of its primary tie-breaking rule.

References

[1] **Dechter, R., J. Pearl,** Generalized best-first search strategies and optimality of A^*, *Journal of the Association for Computing Machinery*, **32(3)** (1985), 505–536.

[2] **Gregorics, T.,** Remarks on the A^{**} algorithm, *Acta Sapientiae Informatica*, **6(2)** (2014), 190–205.

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

[4] **Hart, P., N.J. Nilsson, and B. Raphael,** A formal basis for the heuristic determination of minimum cost paths, *IEEE Trans. System, Man and Cybernet*, **4(2)** (1968), 100–107.

[5] **Martelli, A.** On the complexity of admissible search algorithms, *Artificial Intelligence*, **8(1)** (1977) 1–13.

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

T. Gregorics
Eötvös Loránd University
Budapest
Hungary
gt@inf.elte.hu