A^* as an Optimal Resource Allocation Policy in Path Finding Problems

Antti Autere
Dept. of Computer Science and Engineering, Helsinki University of Technology,
P.O.Box 5400, FIN-02015 HUT, FINLAND
email: aau@cs.hut.fi

Abstract

Consider a problem of finding a path from a start to a goal node on a graph. Suppose that we have divided the problem into smaller problems by dividing the search graph into subgraphs. Nodes and edges in a subgraph form subsets of the nodes and the edges in the original search graph.

Assume that we have either specialized algorithms for searching the subgraphs or only a single algorithm expanding nodes in every subgraph. Usually it is impossible to know *a priori* which algorithm finds solution paths fastest or which subproblem is the easiest to solve.

In this paper, we will examine how A^* can be used as a resource allocation policy for node expansions among the subgraphs. Moreover, we will discuss when A^* is the optimal algorithm for that purpose.

Keywords: Resource Allocation, Path Finding, A^*

Introduction

Let $G = (N, E)$ be a locally finite graph, where N is a set of nodes and E is a set of edges between the nodes. The nodes represent system states and the edges transitions from one state to another. For example, a system state can be a robot manipulator in a particular position in its work space. A transition is a movement of the robot from one position to another.

A positive cost c is associated with every edge in E. A search problem refers to finding a path, a concatenation of nodes and edges, from a start node to a goal node on G. A path has a cost that is the sum of its edge costs.

Subproblems

Suppose that we have several different algorithms for solving a path finding problem. They can share a common search graph or have their own search graphs. In the latter case, for example, the algorithms may use different representations and data structures for the problem. It is also possible that the search graphs have common nodes and edges, e.g., in bidirectional search using the same graph but different start and goal nodes.

It may be possible to divide a search problem into a set of smaller subproblems. This can be based on the knowledge and previous experience of an expert person on that problem domain. Every subproblem can have a specialized algorithm for solving it or there can be a single algorithms for all the subproblems. The subproblems can be solvable either independently of each other or not. If they are independently solvable and the goal of the original problem has the form $goal = g_1 \ and \ g_2 \ and \ ...$, where the g_is are the goals of the subproblems, then we usually call the original problem decomposable.

Let us now formalize the above cases by using search graphs. Assume that the original path finding problem has a search graph G. We model the subproblems by subgraphs G_i of G: $G = G_1 \cup G_2 \cup G_3 \cup ... \cup G_r$. In addition to search graphs, path finding problems have also start nodes s and sets of goal nodes Γ. Hence both the original and the subproblems are actually triples (G_i, s, Γ) but we denote them here by G and G_i, respectively.

We will use the following operations between graphs. Let $G_i = (N_i, E_i)$. First, $G_k \subseteq G_j$ iff $N_k \subseteq N_j$ and $E_k \subseteq E_j$. Second, let the original search graph be $G = G_k \cup G_j$. Then the union of the two subgraphs G_k and G_j is $G_k \cup G_j = (N_k \cup N_j, E_k \cup E_j \cup E_{k,j})$. The set of edges $E_{k,j}$ connects nodes $n_k \in N_k$ and $n_j \in N_j$ in G. Finally, the union of several graphs, and the intersection of two and more graphs are defined analogously.

An intersection of G_i and G_j $(i \neq j)$ is not necessarily empty. Every G_i can have a "private" search algorithm A_i for expanding its nodes or there can be only a single algorithm expanding all the nodes in G. Expanding the nodes in G_i can generate successors in G_j $(i \neq j)$. Hence a final solution path can have nodes in several G_is. The subproblems can also be independently solvable.

If several algorithms solving the same problem use their own search graphs G_i, then every G_i itself represents the original problem. In this case, calling the G_is subgraphs or subproblems is somewhat misleading. Despite of this, we will formally write $G = \bigcup_{i=1}^{r} G_i$. Here G is actually a union of different representations of the same problem.

A simple example where $G = G_1 \cup G_2$ and $G_1 = G_2 = G$ is a bidirectional search: there are two algorithms A_1 and A_2 searching G_1 and G_2, respectively. G_1 and G_2 have different start and goal nodes. The goal of G_1 is the start of

G_2 and vice versa. The search stops if either algorithm finds its goal node or A_i tries to expand a node that has already been expanded by A_j ($j \neq i$). Both the algorithms can also be instances of a single one, for example A^*.

Usually we may be able to say which nodes belong to which subgraph without starting any search. An example of this is a path finding on a d-dimensional rectangular grid. Nodes are in coarser or finer subgrids. This corresponds to searching the original grid with different resolutions. Here a subgraph equals to a subgrid of one resolution. The subgraphs are defined before the search process begins. We have used this method in robot path planning, see [1].

Alternatively, we can decide in which subgraph a node is only after an algorithm has generated it. The generated nodes are included in the subgraphs according to criteria that are implemented in the algorithms themselves. The criteria, for example, can use information about the generated nodes and their immediate predecessors. For more details, see [1].

Resource Allocation

Let a search problem on a graph G be divided into subproblems: $G = \bigcup_{i=1}^{r} G_i$. Assume that we know how to expand nodes in every G_i, which is programmed in the search algorithms.

Assume that the subproblems G_i are not independently solvable or we have less than r computing machines available. Then we have to define a strategy that tells us which subproblem G_i is assigned to which machine at a time before any nodes are expanded. Let us call this strategy a *resource allocation policy* among the subgraphs G_i, or the subproblems. In this paper, we assume that only one machine is available for all the G_is.

One resource allocation policy is to first search for a solution on the graphs $\bigcup_{i=1}^{r_1} G_i$, $r_1 < r$. If a solution is not found then search a bigger set of graphs $\bigcup_{i=1}^{r_2} G_i$, $r_1 < r_2 \leq r$, etc. This strategy can be called *greedy* or a *depth-first search* among the graphs G_i. If the original graph G has infinite number of nodes, then it may happen that this method fails to find a solution path even if it exists.

Another resource allocation policy is to search all the the graphs G_i ($i = 1, 2, .., r$) at the same time. We assign to each node n_i^j in G_i a weight w_i^j. An algorithm can minimize the numbers or *total weights* $W_i = \sum_{j=1}^{K_i} w_i^j$ of the expanded nodes in every G_i in order to find a solution path on one of them. K_i is the number of the expanded nodes in G_i at a given time. This is done by next expanding a node in G_j that has a minimum total weight W_i so far. The strategy corresponds to a *breadth-first search* among the graphs G_i.

Suppose that we do not know which subproblem G_i is the easiest to solve. Alternatively, we do not know which algorithm is best in solving a given problem. Then we may wish to use the algorithms in such a way that the number of the nodes that they have expanded together in G is minimized when a solution path is found. In this sense, the above depth-first and breadth-first strategies can hardly be compared with each other *a priori*.

Let us modify the breadth-first strategy: if node expansions in G_k generate paths that "seem to lead towards a goal", then give computing resources to explore more nodes in G_k before start searching any other G_j ($j \neq k$). One way to implement this strategy is that we estimate the number of the nodes still to be expanded in every G_i before a solution is found. The estimate, say H, works here similarly as a heuristic function h in the A^* algorithm. In the modified resource allocation policy, the sets of the expanded nodes in different G_is correspond to paths in the A^*.

It turns out that the new modified strategy is optimal over the breadth-first strategy. The notion of optimality will be defined below. Moreover, in some cases the new strategy is the best one available. This is possible because it has the same structure as the A^* and "inherits" its optimality properties.

In the next section, we will give a summary of the optimality properties of A^*. After this we will define the new resource allocation policy and show that it is equal to A^*. In the rest of the paper, we will examine situation where the new algorithm is the best available and how to calculate its heuristic H by using the heuristic h related to A^*. The rest of the paper is divided into two subsections. In the first subsection, we allow paths to have nodes only in one subgraph, e.g., when the subproblems are independently solvable. In the second subsection, paths can have nodes in different subgraphs, a situation which is more complex. We will also present a pseudo code of the new algorithm.

Optimality Properties of A^*

In brief, A^* (originally in [3]; see also [4] and [5]) is an ordered best-first graph search algorithm that always expands the "most promising" node n based on the function: $f(n) = g(n) + h(n)$. $g(n)$ is the cost of the cheapest path from the start node to n. The *heuristic* $h(n)$ is an estimate of the cost of the cheapest path from n to any goal node. A pseudo code for A^* is, e.g., in [5] p. 64-65.

A^* always finds a cheapest path from the start node to a goal node if it exists when the heuristic underestimates the actual cost of the cheapest path from any node to the goal ($h \leq h^*$). In this case, both h and A^* are called *admissible*.

Let the domain of problem instances on which A^* is admissible be denoted by I_{AD}, see [2] (p. 96):

$$I_{AD} = \{(G, s, \Gamma, h) \mid h \leq h^* \text{ on } G\}, \qquad (1)$$

where $G = (N, E)$ is a locally finite graph. $s \in N$ is the start node and $\Gamma \subseteq N$ is a set of goal nodes.

A heuristic $h(n)$ is *consistent* if it satisfies the triangle inequality: $h(n) \leq k(n, m) + h(m)$ for any node n and its descendants m, see [5] (p. 82). $k(n, m)$ is the cost of the cheapest path from n to m. Consistency implies admissibility but not vice versa.

A heuristic $h(n)$ is *monotone*, if it satisfies: $h(n) \leq c(n, n') + h(n')$ for every n and its immediate successors n'. Monotonicity and consistency are equivalent properties [5] (p. 83).

Let C^* be the cost of an optimal path on G. When A^* has found the optimal path using an admissible heuristic, it has *surely expanded* any node reachable by a strictly C^*-bounded path. A path is strictly C^*-bounded if every node

n along that path satisfies $f(n) < C^*$. On the other hand, if A^* uses a monotone heuristic, then it has surely expanded the set $\{n \mid f(n) = g^*(n) + h(n) < C^*\}$, cf. [5] (p. 84).

The nodes for which $f(n) = C^*$ may or may not be expanded depending on the so called tie-breaking rule. Taking this into account we will next define the notion: "algorithm 1 is optimal over algorithm 2":

Definition 1. [2] (p. 96): An algorithm A is said to be *optimal* over a class **A** of algorithms relative to a set I of problem instances if in each instance of I, every algorithm in **A** will expand all the nodes surely expanded by A in that problem instance.

Theorem 1. cf. [2] (Theorem 3, p. 98): A^* is optimal over any algorithm which is admissible on I_{AD} and provided a consistent heuristic h.

If the heuristic is not consistent (can be admissible though), then no such optimal algorithm exists [2] (p. 98). In case of a consistent heuristic the expression "is optimal over" or "dominates" is synonymous with "largely dominates", see [5] (p. 85).

Theorem 2. [5] (p. 81 and 85): If $h_2 \geq h_1$ and both are admissible (monotone), then A_2^* using h_2 is optimal over or dominates (largely dominates) A_1^* using h_1.

A^* as a Resource Allocation Policy

Consider a path finding problem and its search graph $G = \bigcup_{i=1}^{r} G_i$ where the intersection of the G_is is not necessarily empty. Let there be algorithms $A_1, A_2, .., A_r$ expanding nodes in $G_1, G_2, .., G_r$, respectively. Some or all the algorithms can be identical. For example the bidirectional search, in "Introduction", has $r = 2$ and $A_1 = A_2$ except their starting and goal nodes that are reversed. Furthermore, there can also be a single algorithm for expanding all the nodes in G.

Assume that A_i has expanded every node in a subgraph $G_i(\tau) \subseteq G_i$ at "a time" τ. Let $N_i(\tau) \subseteq N_i$ be the set of the expanded nodes at τ and $\mid N_i(\tau) \mid$ be the number of the nodes in $N_i(\tau)$.

The breadth-first strategy for allocating computing resources among the G_is, in "Introduction", is: at τ, choose the $G_j(\tau)$ for which $\mid N_j(\tau) \mid$ is minimum and expand one successor to a node in $N_j(\tau)$. Clearly, if a solution path is first found in $G_k(\tau^*)$, then $\mid N_k(\tau^*) \mid$ is minimized.

Let $N_i(\tau^1)$ and $N_i(\tau^2)$ be the sets of the nodes that an algorithm A_i has expanded at τ^1, and $\tau^2 > \tau^1$. Assume that $\mid N_i(\tau^1) \mid = l$. If $N_i(\tau) = N_i(\tau^1)$ where $\tau^1 \leq \tau < \tau^2$ and $N_i(\tau^2)$ contains only one more node than $N_i(\tau^1)$, then let us simply write $N_i(\tau^1) = N_i(l)$ and $N_i(\tau^2) = N_i(l+1) = N_i'(l)$. The set N_i' is called a *successor* to the set N_i.

Now, imagine that every set $N_i(l)$ forms a node and there is an edge between $N_i(l)$ and $N_i'(l) = N_i(l+1)$ for all $l = 1, 2, ...$ For example, if A_i expands a starting node s in G_i at τ^1, then $N_i(1)$ contains only s. Thus $N_i(1) = \{s\}$ is another node. Its successor node $N_i'(1) = N_i(2)$ contains two nodes in G_i expanded by A_i, say, s and n_1: $N_i(2) = \{s, n_1\}$. Similarly, $N_i'(2) = N_i(3) = \{s, n_1, n_2\}$ after A_i has expanded n_2 etc.

Definition 2. Let a graph $G = \bigcup_{i=1}^{r} G_i$ and $N_i(l)$ be the set of the expanded nodes in $G_i \subseteq G$ at τ^l. Call $N_i(l)$ a node. $N_i'(l) = N_i(l+1)$ is the only successor to $N_i(l)$ if $\mid N_i(l+1) \mid = \mid N_i(l) \mid +1 = l+1$.

Let there be an edge between $N_i(l)$ and $N_i'(l)$ with an associated cost $C(N_i(l), N_i'(l)) > 0$ for any i and l. There is no edge allowed from N_i to N_j if $i \neq j$. Let there also be a dummy node $N(0)$ with edges from $N(0)$ to every $N_i(1)$. $C(N(0), N_i(1)) = a_i \geq 0$.

A concatenation of the nodes and the edges, starting from $N(0)$, form a list called an *abstract* or a *meta path* and is denoted by MP_i $(i = 1, .., r)$. The cost of MP_i is the sum of its edge costs. *End of Def.* 2

In general, it is sufficient that only one of the subgraphs G_i contains the start node and one contains the goal node.

If the costs $C(N_i(l), N_i'(l)) = 1$ for every i and l, then minimizing the cost of $MP_i(l)$ equals to minimizing $\mid N_i(l) \mid = l$. For example, if a goal is found first in G_k containing l^* expanded nodes, then the cost of the optimal meta path MP_k^* is l^*.

Based on Definition 2. the above breadth-first search strategy among the G_is can be interpreted as one finding a cheapest meta path MP_i starting from $N(0)$.

From now on, N_i represents $N_i(l)$ for all l. Let a graph $MG = \bigcup_{i=1}^{r} MP_i$. Actually, MG is a tree with a root $N(0)$. Analogously to I_{AD} in Equation (1) let MI_{AD} be:

$$MI_{AD} = \{(MG, s, \Gamma, H) \mid H \leq H^* \text{ on } MG\}. \qquad (2)$$

Assume again that $C(N_i, N_i') = 1$ for all i. Assign to each node N_i along MP_i a heuristic $H(N_i(l))$. $H(N_i(l))$ is an estimate of the number of nodes to be expanded in G_i, after l, before a solution path on G is found. Similarly, $H^*(N_i(l))$ is the minimum number of nodes still to be expanded in G_i before a goal is found. If $H(N_i) \leq H^*(N_i)$ for all N_i, then H is called admissible.

The breadth-first strategy for finding a cheapest meta path $MP_i \subseteq MG$ has $H(N_i) = 0 \leq H^*(N_i)$. Hence it is admissible on the domain MI_{AD} of problem instances. However, if we can estimate an admissible $H \geq 0$, then an A^* using it is also admissible on MI_{AD}. Then Theorem 2. implies that the A^* is optimal over the breadth-first strategy relative to MI_{AD}.

If $H(N_i)$ is also consistent then we can reformulate Theorem 1.

Theorem 3. Let A^* be the resource allocation policy for node expansions among the subgraphs G_i $(i = 1, .., r)$, or among the algorithms A_i. Assume that the A^* uses the nodes and the meta paths in Definition 2. and is provided a consistent heuristic H, in equation (2). Then the A^* largely dominates (or is optimal over) any resource allocation policy that is admissible on MI_{AD}, if provided the same H, in a sense of Definition 1.

Proof. Follows directly from Theorem 1. □

Suppose that we wish to use the algorithms A_i searching the graph $G = \bigcup_{i=1}^{r} G_i$ in such a way that the number of the nodes that they *together* expanded in $\bigcup_{i=1}^{r} G_i$ is minimum after a solution path is found. Then Theorem 3. says that the A^* defined above is the best resource allocation method among the ones that use the same consistent heuristic H. If H is admissible, then the A^* is optimal over the breadth-first strategy by Theorem 2.

Estimating the Heuristic H by h

Let a search graph $G = \bigcup_{i=1}^{r} G_i$ and algorithms A_i be as before.

Suppose that we can not estimate H explicitly or node expansions in G require different amount of computational work. The work is measured by the edge costs of G. Let us associate with every node n a cost or a *weight* $w(n) = c(m, n)$, the cost of an edge between n and its immediate predecessor m. If there are many paths leading to n, then we define $w(n) = c(m, n)$ where m is the father of n when n was found at the first time. Let us arbitrary assign $w(s_i) = a_i \geq 0$ to the first node s_i in N_i.

Assume that we can calculate a heuristic $h(n)$ for all the nodes n in G. If a goal node and n_i are in a subgraph G_i, then $h(n_i)$ estimates the cost of the path from n_i to the goal. If G_i does not contain any goal node, then $h(n_i) = 0$ or $H(N_i)$ is estimated in some other way.

Let us allocate computing resources for node expansions among the graphs G_i, or among the algorithms A_i, such that the number of the weighted nodes expanded in $\bigcup_{i=1}^{r} G_i$ is minimized. This can be done by the A^* algorithm minimizing the costs of the meta paths in Definition 2. From now on, let us call the A^* used in resource allocation A^*_{MG}: it searches the graph MG in MI_{AD} defined in equation (2). Now, the edge cost between a node N_i and its successor N'_i is $C(N_i, N'_i) = c(n, n') = w(n')$, $n' \in N_i$.

Paths with Nodes in a Single Subgraph

Assume that all the subgraphs G_i have the start and goal nodes. If every G_i has a solution path, then the subproblems represented by the G_is can be solved independently of each other. The intersection of G_j and G_k $(j \neq k)$ may or may not be empty. The bidirectional search, in "Introduction", is an example of the latter case $(G_1 = G_2)$.

Let us now consider the following case: if a node in G_i is expanded then all its successors will be only in G_i and not in any other subgraph.

Theorem 4. Suppose that A^*_{MG} uses the nodes and the meta paths in Definition 2. Assume that every path candidate P_i is on G_i $(P_i \subseteq G_i)$. Let $C(N_i, N'_i) = w(n'_i) = c(n_i, n'_i) > 0$, where n_i and its successor n'_i both are in G_i. If $h(n_i)$ is monotone for all n_i in all G_i, then A^*_{MG} using a function

$$F(N_i) = G(N_i) + H(N_i)$$
$$= \sum_{j=1}^{K_i} w(n_i^j) + min\{h(n_i^j) \mid n_i^j \in N_i \, \forall j\}$$

is admissible on MI_{AD}. N_i has K_i expanded nodes at a given time. In other words, A^*_{MG} minimizes $\sum_{j=1}^{K_k} w(n_k^j)$ after a goal has been found in G_k. Furthermore, A^*_{MG} is the optimal resource allocation policy for node expansions among the G_is according to Definition 1.

Proof. Let us expand a node n'_i, a successor to n_i. If $h(n'_i) \geq H(N_i)$ then $H(N'_i) = H(N_i)$ by the definition of H. Assume that $h(n'_i) < H(N_i)$. This implies $H(N'_i) = h(n'_i) \geq h(n_i) - c(n_i, n'_i)$. The latter inequality follows from the monotonicity of h. Hence $H(N'_i) \geq H(N_i) - C(N_i, N'_i)$ since $H(N_i) = min\{h(m_i) \mid m_i \in N_i\}$ and $C(N_i, N'_i) = c(n_i, n'_i)$. Thus H is monotone, consistent and admissible.

Hence A^*_{MG} is admissible in MI_{AD} and largely dominates (is optimal over) any admissible algorithm in MI_{AD} by Theorem 1. □

The proof of Theorem 4. essentially says that the minimum of monotone heuristics is also monotone.

Theorem 4. do not require that the algorithms A_i, expanding nodes in G_is, must themselves use the monotone heuristics h. Only the resource allocation algorithm A^*_{MG} utilizes it. The A_is can be any path searching algorithms, not necessary admissible ones. Recall that there can also be a single algorithm for expanding all the nodes in G.

Let us further discuss the bidirectional search on the graph G in "Introduction". If we can calculate monotone heuristics h_1 and h_2 for the algorithms A_1 and A_2, then A^*_{MG} is the optimal resource allocation policy between A_1 and A_2 in a sense of Definition 1 by Theorem 4.

The bidirectional search proceeds as follows. First, A_1 and A_2 expand their starting nodes s_1 and s_2. Hence $N_1(1) = \{s_1\}$ and $N_2(1) = \{s_2\}$. The OPEN set of A^*_{MG} has now two nodes $N_1(1)$ and $N_2(1)$. $F(N_1(1)) = F(N_2(1)) = 1 + h_1(s_1)$ assuming that $h_1(s_1) = h_2(s_2)$ and $w(s_1) = w(s_2) = 1$. Suppose that A^*_{MG} selects first $N_1(1)$. It means that one of the successors s'_1 to s_1 is expanded by A_1. After this A^*_{MG} generates a successor to $N_1(1)$, $N_1(2) = \{s_1, s'_1\}$, and calculates $F(N_1(2)) = G(N_1(2)) + H(N_1(2)) = 1 + c(s_1, s'_1) + min\{h_1(s_1), h_1(s'_1)\}$. Thus the OPEN set of A^*_{MG} now contains nodes $N_1(2)$ and $N_2(1)$ since $N_1(1)$ is now placed on CLOSED. Next, A^*_{MG} selects a node from OPEN for which the F-value is minimum etc. Both the A_is have their private mechanisms of choosing the next node in N_i to be expanded. They can use the h_is or not. They can also be A^* algorithms if wanted.

Paths with Nodes in Many Subgraphs

Let us delete an assumption of Theorem 4: "every path candidate $P_i \subseteq G_i$". Hence solution paths can contain nodes in several subgraphs G_i $(i = 1, 2, .., r)$. Subgraphs G_j and G_k $(j \neq k)$ can have common nodes. It is not necessary, however, that the start node and goal nodes are in every G_i.

In this section, the expansion of a node in G_i can produce successors that are also in G_j, $i \neq j$. The resource allocation algorithm A^*_{MG} then chooses on which set N_j $(j = 1, .., r)$ to place the successors. Let us first extend Definition 2.

Definition 3. All the sets N_i in Definition 2. contain *both* expanded *and* open, unexpanded, nodes in G_i. The predecessor of N_i contains no open nodes. $| N_i |$ is the number of the expanded nodes in N_i at a given time.

Let us now assign a *type* to every node: n has a type k if n is in G_k and it is denoted by n_k. It is possible that the type of the node is known before any algorithm has generated it, namely, if we know the division $G = \bigcup_{i=1}^{r} G_i$ *a priori*. On the other hand, a decision on which set N_k a node n' is placed may depend on the type of its predecessor n and is found out after the expansion of n. In the latter case, the type of the node is *implicitly defined* whereas in the former case it is *explicitly defined.*

As an example, let us discuss the search graph defined on a d-dimensional rectangular grid and mentioned in "Introduction", see [1] for more details. The nodes on the grid, G, are vectors $n \in Z^d$ whose components are integers. For example, the neighboring nodes in G are $n = (n_1, n_2, ..., n_d)$ and $n' = (n_1, n_2+1, ..., n_d)$. $G = G_1 \cup G_2 \cup G_4 \cup G_8 \cup .. \cup G_{max}$ defines a hierarchy of grids as follows. The nodes in G_i $(i = 1, 2, 4, .., max)$ are vectors $\{n = (n_1, n_2, ..., n_d) \mid gcd(n_1, n_2, ..., n_d) = i\}$, where $gcd(\cdot)$ denotes the greatest common divisor of the arguments. Thus the types of the nodes are explicitly defined. The subgraphs G_i do not have any common nodes. However, there are edges between nodes in different G_is such that the expansion of a node in G_i can produce successors that are in G_j, $i \neq j$.

Let us discuss an example of the implicit problem division [1]. Denote now by G^i the graph G_i in the graph hierarchy of the previous paragraph. Here the subgraphs G_i do not refer to the graph hierarchy but are defined as follows. First, assume that A^*_{MG} has chosen a set N_i. Second, suppose that a node n_i in G_i has been expanded. Let n_i be in G^j and its successor n' be in G^k in the graph hierarchy. If $k \geq j$ then A^*_{MG} places n' on N_i and sets i as the type of n'. If $k < j$ then n' is placed on N_k and its type is k. Hence the type of a successor depends on the type of its father. We can imagine that every N_i contain nodes in G^i, in the graph hierarchy, from which trees of partial paths start. Nodes in this forest of trees in N_i are in $G^k, k \geq i$.

In general, A^*_{MG} works as follows. After a node n_i in N_i is expanded A^*_{MG} places its every successor n'_k on N_k where the type $k = 1, 2, .., r$ can be different from i. The successors are now open nodes in N_k. The function $F(N_i)$ is calculated as in Theorem 4.

A pseudo code of A^*_{MG} is shown below. In the code, "N(i)" is N_i and "n.type" is the type of n. OPEN refers to the set of sets N(i) that contain at least one open unexpanded, node.

ALGORITHM A^*_{MG}:

```
(1) Choose the first node m, create a set
    N(m.type) and place it on OPEN
(2) Place m on N(m.type)
    and calculate F(N(m.type))
(3) Choose N(i) containing open nodes
    from OPEN for which F(N(i)) is minimum
(4) IF no such N(i) is found on line (3)
(5) THEN exit with failure
(6) ELSE expand an open node n in N(i)
(7) IF n is a goal THEN exit successfully
(8) IF N(i) has no more open nodes
    THEN remove N(i)
(9) FOR all the successors n' to n
(10)  IF N(n'.type) does not already exist
(11)  THEN
(12)   Create and place N(n'.type) on OPEN
(13)  END IF
(14)  Place n' on N(n'.type)
      if it is not already there and
      calculate F(N(n'.type))
(15)END FOR
(16)Go to line (3)
```

There is no CLOSED set despite of the fact that A^*_{MG} is actually A^*. This is for clarity. In this paper, we are mostly interested in cases where the heuristic $H(N_i)$ is monotone and thus no reopenings are necessary, see e.g. [5] (p. 83). If N_i does not have any open nodes, then line (8) "closes" it. The meta paths are not explicitly formed, since N_i and its only successor $N^!_i$ are not implemented as separate sets: $N^!_i = N_i \cup n_i$ after the insertion of n_i, in line (14).

Node expansions in the sets N_i and successor generations, on line (6), can be done by different algorithms A_i if wanted. There can also be only a single algorithm for expanding all the nodes in G. Line (14) can be replaced by "(14') Place n' on N(n'.type) if it is not already in any N(i)...". Then one node is only in one N_i. In general, a node can have many types if the types are implicitly defined.

Perhaps the simplest case here is that if the expansion of a node n_i in G_i produces a successor n_j in G_j $i \neq j$, then n_j is the first node in N_j. While N_j does not contain any nodes, we can think that "$H(N_j) = 0$". When A^*_{MG} has placed the first node n_j on N_j, then $H(N_j)$ is monotone if $h(n_j)$ is monotone (if G_j has a goal node). Hence Theorem 4. holds here. If A^*_{MG} finds the goal node in G_j first, then it has minimized the number of the expanded (weighted) nodes in G_j with the start node n_j.

The above case showed that, in general, it is possible that search processes on different subgraphs can start at different times. If N_j gets its first node after the start node s in G has been expanded, then we have to set $F(N_j) \geq min\{F(N_i) \mid i \neq j\}$ where every N_i already contains at least one node. This is to keep the values F monotone.

Suppose that on line (8) of the pseudo code, N_k has no open nodes left. Then A^*_{MG} deletes N_k. Assume next that a node n_k in G_k is generated. Now A^*_{MG} has to create a new set N_k for n_k. Let us call this "*Property A* ".

If A^*_{MG} did not recreated N_k and placed n_k on the old N_k, then problems would occur. Namely, if A^*_{MG} finds later a goal node first in G_k, then it does not necessarily minimize the number of the (weighted) expanded nodes in G_k. An example of this can be found easily. In other words, A^*_{MG} is not be admissible on MI_{AD} in equation (2). Then we cannot tell much of its optimality either. This does not mean, however, that an algorithm like A_{MG} without "Property A" would necessarily behave badly in real situations.

It may happen that A^*_{MG} recreates N_k so late that it contains only a few nodes before A^*_{MG} finds a goal in G_k. Now A^*_{MG} has actually minimized the (weighted) expanded nodes in a subgraph of G_k. This may be disappointing since we originally wanted to minimize the nodes in G_k. However, this effect can be compensated somewhat by setting the starting value $G(N_k)$ and thus $F(N_k)$ high enough, as we already discussed above.

Let us next generalize Theorem 4. If a goal node is in a subgraph G_i, then $h(n_i)$ estimates the distance from n_i to it. If G_i does not contain any goal node, then $h(n_i) = 0$ or $H(N_i)$ is estimated in some other way.

Theorem 5. Assume that A^*_{MG} uses the nodes and the meta paths in Definitions 2. and 3. Let it be possible for path candidates on G to have nodes in different subgraphs G_i $(i = 1, 2, ..r)$. Furthermore, let $h > 0$ be monotone if G_i contains a goal node and $h = 0$ if G_i has no goal nodes.

Assume that "Property A" holds. Moreover, assume that $h(n'_j) \geq H(N_j) - C(N_j, N'_j)$ where n'_j is the immediate successor to n_i for all $i \neq j$ when A^*_{MG} places n'_j on N_j. $C(N_j, N'_j) = w(n'_j) = c(n_i, n'_j)$.

Now A^*_{MG} using the $F(N_i)$ in Theorem 4. is admissible on MI_{AD}. A^*_{MG} is also the optimal resource allocation policy for node expansions among the G_is according to Definition 1.

Proof. Let A^*_{MG} expand a node $n_i \in N_i$ and place its successor n'_j as an open node on N_j. If $i = j$ then Theorem 4. holds since h is monotone. If $i \neq j$ and n'_j is the first node in N_j then Theorem 4. holds, too.

Assume that $i \neq j$ and n'_j is not the first node in N_j. If $h(n'_j) \geq H(N_j) = min\{h(m_j) \mid m_j \in N_j\}$ then $H(N'_j) = H(N_j)$ and H is monotone. On the other hand, if $h(n'_j) < H(N_j)$ then $H(N'_j) = h(n'_j) \geq H(N_j) - C(N_j, N'_j)$, which follows from the last assumption above.

Theorem 1. implies the rest. □

A necessary assumption of Theorem 5. is: $h(n'_j) \geq H(N_j) - C(N_j, N'_j)$ when A^*_{MG} places n'_j, a successor to n_i, on N_j. This is true if $i = j$ because of the monotonicity of h, as Theorem 4. already showed. However, the validity of the assumption is not at all clear when $i \neq j$. The assumption means that A^*_{MG} can not place all the nodes on N_j whose father is not in N_j. Only those nodes are accepted whose h values are not small enough. Usually, we can check this fact only after the generation of n'_j when A^*_{MG} has determined its type j.

A partial solution to the above problem is to let the algorithms A_i avoid node expansions that violate the assumption as long as it is possible. However, at some point we may have to violate the assumption unless the solution path cannot be found in another way.

Another solution is that when A^*_{MG} observes that the assumption does not hold then it creates a new set whose first node is the one that otherwise would have caused the problem. However, we may now face the problems discussed a few paragraphs above.

The violation of the above assumption means that A^*_{MG} has a heuristic H that is not monotone. If we can keep H admissible, then A^*_{MG} is optimal over the breadth-first strategy (having $H = 0$) by Theorem 2. It can happen that $h(n'_j) << H(N_j)$ so that $H(N_j)$ is not even admissible. If this happens, then nothing can be said about the optimality of A^*_{MG}.

Conclusions

Suppose that we have a division of the path finding problem into subproblems. This is done by dividing the original search graph into subgraphs. In general, every subgraph do not have to contain the start and goal nodes. Hence a solution path can have nodes in several subgraphs. In other words, the smaller subproblems are not required to be solvable independently of each other. Every subproblem can have a different algorithm for expanding its nodes.

In this paper, we showed that A^* can be used as a resource allocation policy for node expansions among the subgraphs. In some cases, A^* is the optimal resource allocation algorithm. This requires the possibility of underestimating the number of the nodes still to be expanded in each subgraph before a solution path is found, see Theorem 3.

If every path candidate can be only on a single subgraph, then one method of the estimation is to use a monotone heuristic assigned to the nodes in the original graph. The monotonicity guarantees the existence of an optimal resource allocation policy (A^*), see Theorem 4.

If path candidates can have nodes in several subgraphs, then the situation is more complicated. Additional assumptions are needed for the existence of an optimal resource allocation policy, see Theorem 5. It is not always possible to know *a priori* whether these constraints is satisfied or not.

References

[1] Antti Autere. Hierarchical a^* based path planning - a case study. In *2nd Int. ICSC Symposium on Engineering of Intelligent Systems (EIS'2000)*, Univ. of Paisley, Scotland, U.K., June 27 - 30 2000. ISBN 3-906454-21-5, ISBN 3-906454-23-1.

[2] Rina Dechter and Judea Pearl. The optimality of a revisited. In *3rd AAAI Natl. Conf. on AI, Washington D.C.*, 1983.

[3] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Systems Science and Cybernetics*, 2:100–107, 1968.

[4] N. J. Nilsson. *Principles of Artificial Intelligence*. Palo Alto, Calif.:Tioga, 1980.

[5] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.