Helsinki University of Technology Laboratory for Theoretical Computer Science
Research Reports 98
Teknillisen korkeakoulun tietojenkäsittelyteorian laboratorion tutkimusraportti 98
Espoo 2005 HUT-TCS-A98

EXTENSIONS AND APPLICATIONS OF THE A^* ALGORITHM

Antti Autere

TEKNILLINEN KORKEAKOULU
TEKNISKA HÖGSKOLAN
HELSINKI UNIVERSITY OF TECHNOLOGY
TECHNISCHE UNIVERSITÄT HELSINKI
UNIVERSITE DE TECHNOLOGIE D'HELSINKI

Helsinki University of Technology Laboratory for Theoretical Computer Science
Research Reports 98
Teknillisen korkeakoulun tietojenkäsittelyteorian laboratorion tutkimusraportti 98
Espoo 2005 HUT-TCS-A98

EXTENSIONS AND APPLICATIONS OF THE A^* ALGORITHM

Antti Autere

Dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the Department of Computer Science and Engineering, for public examination and debate in Auditorium T2 at Helsinki University of Technology (Espoo, Finland) on the 16 of December, 2005, at 12 o'clock noon.

Helsinki University of Technology
Department of Computer Science and Engineering
Laboratory for Theoretical Computer Science

Teknillinen korkeakoulu
Tietotekniikan osasto
Tietojenkäsittelyteorian laboratorio

Distribution:
Helsinki University of Technology
Laboratory for Theoretical Computer Science
P.O.Box 5400
FI-02015 TKK
Tel. +358-0-451 1
Fax. +358-0-451 3369
E-mail: lab@tcs.hut.fi

ISBN 951-22-7974-9
ISSN 1457-7615

Multiprint Oy
Helsinki 2005

ABSTRACT: In this thesis we investigate path finding problems, that is, planning routes from a start node to some goal nodes in a graph. Such problems arise in many fields of technology, for example, production planning, energy-aware message routing in large networks, resource allocation, and vehicle navigation systems. We concentrate mostly on planning a minimum cost path using the A^* algorithm.

We begin by proving new theorems comparing the performance of A^* to other (generalized) path finding algorithms. In some cases, A^* is an optimal method in a large class of algorithms. This means, roughly speaking, that A^* explores a smaller region of the search space than the other algorithms in the given class.

We develop a new method of improving a given (static) heuristic for A^* dynamically, during search. A heuristic controls the search of A^* so that unnecessary branches of the tree of nodes that A^* visits are pruned. The new method also finds an optimal path to any node it visits for the first time so that every node will be visited only once. The latter is an important property considering the efficiency of the search.

We examine the use of A^* as a higher level method to allocate resources among several path finding algorithms. In some cases, the A^* is an optimal resource allocation method, which means that the number of the nodes the path finding algorithms together visit is minimized.

As applications of A^*, we have developed new hierarchical algorithms for robot point-to-point path planning tasks, and new algorithms for power-aware routing of messages in large communication networks. The new algorithms are more robust than some older ones to which we compare. Moreover, one of the message routing algorithms produces higher average lifetimes of the network than those of the widely quoted *max-min* zP_{min} algorithm.

KEYWORDS: Path finding, heuristic algorithms, best-first search, A^*, resource allocation, robotics, motion planning, message routing

CONTENTS

PREFACE

The work reported in this thesis began in 1995 during the research projects of the Robotics Group at the Laboratory of Information Processing Science and TAI Research Centre of Helsinki University of Technology. The group has since evolved to be the Industrial Information Technology Laboratory. I am indebted to the members of the Robotics Group, Professor Juha Tuominen, Ph.D. Pekka Isto, Mr. Johannes Lehtinen, Mr. Pasi Eronen, Mr. Andy Nurminen, and Mr. Jaakko Väyrynen for brainstorming atmosphere and inspiring times. Especially, I would like to thank Mr. Johannes Lehtinen and Mr. Jaakko Väyrynen for implementing several algorithms tested in this thesis, running simulations, and making valuable comments during the development process.

The thesis was finished at the Laboratory for Theoretical Computer Science of Helsinki University of Technology. I would like to thank the people at the laboratory for their time and help in practical matters. I am grateful to Professor Martti Mäntylä for introducing me to Professor Pekka Orponen who later became my thesis instructor and supervisor. I am very much obliged to Pekka Orponen for his patience to listen, try to understand, and criticize my arguments — especially my attempts to prove the same theorems again and again. These conversations that often lasted for many hours convincingly showed how difficult it is to explain oneself when one has only a limited understanding of what he is talking about. It also became clear, at least until then, that doing everything wrong for most of the time is the essence of learning something new.

I am grateful to my pre-examiners Professors Tapio Elomaa and Ville Leppänen for their time and comments that improved the manuscript.

The funding and other resources for this research have come from numerous sources. Laboratory of Information Processing Science, the Academy of Finland, European Commission, Industrial Information Technology Laboratory, Laboratory for Theoretical Computer Science, Helsinki University of Technology, the Finnish IT center for science (CSC), and Helsinki Graduate School in Computer Science and Engineering are gratefully acknowledged.

Finally, I would like to thank my parents and friends, especially Sirpa, for their support and encouragement in times when life was not easy. Such support and friendship has been and will always remain beyond measure.

Otaniemi, November 2005

Antti Autere

1 INTRODUCTION

This thesis considers path finding problems in graphs. The path finding problem is to plan a route, an alternating sequence of nodes and edges, from a start node to some goal nodes in the graph.

The neighborhood relation, modeled as edges in the graph, constrains possible movements, and path finding problems can be seen as a special case of constraint satisfaction problems. Usually every edge has an associated number modeling the distance or other cost measures between neighboring nodes. We may wish to minimize (maximize) the cost of the path satisfying the constraints, that is, the sum of the edge costs of feasible paths. Then the problem becomes an instance of a linear programming problem.

During the 1960s, direct linear programming formulations of path finding problems were replaced by more specific techniques to better exploit the structure of the underlying search graphs, see e.g. [19, 54, 32, 58]. The classical method of this category is Dijkstra's shortest path algorithm [19]. It is an example of breadth-first search strategies that are guaranteed to find a minimum cost solution path if it exists in the graph (with strictly positive edge costs). Other strategies are, e.g., depth-first and backtracking methods. The latter ones may miss existing optimal solution paths in graphs with infinite number of nodes.

Dijkstra's shortest path algorithm searches every potential path candidate in the graph towards "every direction" around the start node. This can be very time consuming. Sometimes we may be able to describe where a goal is approximately located, or which parts of the graph are most promising to search. This information may not be explicitly present in the structure of the search graph. How can this additional information, or our intuition of the task, be given as advice to a path finding strategy to make it more effective?

In this thesis, we call the above additional advice *heuristic information*. Among the most popular methods of exploiting heuristic information to prune the search is the *informed best-first strategy*. The general philosophy of this strategy is to use the heuristic information to assess the merit latent in every candidate search avenue and then continue the exploration along the direction of highest merit [18]. A simple example is the following. The problem is to search for a minimum cost path from a start node to a goal node in a two dimensional rectangular grid with some obstacles. A heuristic information may include the direction where the goal is located or its distance from any node in the grid without the obstacles.

The most studied informed best-first strategy is the A^* algorithm. It is a generalization of Dijkstra's shortest path algorithm, where heuristic information is exploited. Let the edges in the above grid have unit costs. Dijkstra's algorithm searches the grid towards every direction forming a "Manhattan ball" around the start node. Path candidates of A^* search, with heuristic information, go mostly towards the location of the goal. Figure 2.1 in the next chapter shows an example of this situation. We will discuss Figure 2.1 in more detail later.

Most often A^* visits a smaller set of nodes in the search graph than the Dijkstra method. However, a drawback of A^* search is still its slowness in many problems. This is because good or effective heuristic information to guide the search is often very hard to get. Usually, more greedy algorithms that do not search every path candidate have been found to work fast and well enough. The latter algorithms can, as pointed out earlier, miss solution paths when they exist. Moreover in many applications, we may not be interested in minimizing path costs and can develop faster constraint satisfaction algorithms. But then, comparisons of the efficiency of the latter algorithms to A^* must almost always be empirical.

A strong point of A^*, and related informed best-first strategies, is that theoretical results exist both concerning the effectiveness and characterizing good heuristic information. However, these results need the assumption of path cost minimization. If this goal is not of interest, then theoretical comparisons of algorithms with A^* are hard to get. On the other hand, if we have a path cost minimization problem such that an optimal solution path must be found if it exists, then A^* and its variants are among the best algorithms available.

Nowadays minimum cost path finding problems are important, for example, in production planning, in energy-aware message routing in large communication networks, and in vehicle navigation systems, see e.g. [44, 48, 25, 29].

1.1 CONTRIBUTIONS AND ORGANIZATION OF THE THESIS

Chapter 2 provides theoretical results concerning A^* and, more generally, informed best-first strategies. It is basically in the form of definitions and theorems that will be needed in subsequent chapters. Notions and concepts are defined at the time they are needed. Proofs of the most important theorems are also included.

Section 2.12.3 discusses how A^* could be used to minimize real valued functions. Now, the search is done in the space of complete solutions. The "goodness" of a solution is measured by the value of the function to be minimized.

Section 2.12.4 presents an algorithm that is a combination of two search strategies: a "base" algorithm and a "probe". The number of node expansion of the composed algorithm is at most $(1 + \beta)$ times the number of node expansions of the base algorithm, for example A^*, where β is a parameter defined by the user. The probe can be a simple and possibly a greedy search, to look around the search space to see whether a goal can be found in the "neighborhood" explored by the base algorithm.

The material in Sections 2.12.3 and 2.12.4 appears for the first time in this thesis.

Chapter 3 compares the performance of A^* to that of other (generalized) path finding algorithms in cases where their guiding heuristics are not necessarily admissible. Most theorems in the literature concern only admissible heuristics.

At first, we define more precisely what we mean by a path finding algorithm. Then we introduce generalizations of consistent heuristics that are not necessarily admissible. The rest of the chapter contains new theorems that characterize the sets of nodes explored by the generalized path finding algorithms by using the set of nodes explored by A^*. One of the optimality theorems for A^* guided by nonadmissible heuristics is a generalization of the results published earlier. Other theorems need additional assumptions compared to their counterparts in the literature.

The material in this chapter appears for the first time in this thesis.

Chapter 4 defines a new method of improving a given (static) admissible heuristic function for A^*. The improvements are done by estimating successive lower bounds for the cost of an optimal path dynamically, during the search. Thus, the evaluation function of the new algorithm at some time instant during the search is a function of the whole search tree at that time instant. We show that the new algorithm guided by the improved heuristic is optimal over A^*. This means, roughly speaking, fewer node expansions compared to those of A^*. Moreover, we also show that when the new algorithm expands a node for the first time, it has found an optimal path to the node. It follows that no reexpansions of nodes are necessary, which is an important property considering the efficiency of the search.

Chapter 4 presents a revised version of the conference paper [2].[1]

Chapter 5 examines how A^* can be used to allocate computing resources among several search algorithms solving the same path finding problem. The high-level A^* algorithm uses abstracted paths that are composed of sets of nodes expanded by the low-level algorithms during the search process. In some situations, A^* is an optimal resource allocation policy. This means, roughly speaking, that the number of the nodes expanded by all the low-level search algorithms together is minimized. As an example of a resource allocation problem, we discuss so called bidirectional search.

At the end of the chapter, we show that A^* that searches on a *tree* and is guided by an admissible heuristic is the optimal path finding algorithm. This theorem together with the theorems in Chapter 3 can be used to generalize the optimality results of this chapter.

Chapter 5 presents a revised and enlarged version of the conference paper [3].[2]

Chapter 6 introduces and tests four hierarchical robot path planning algorithms using five simulated robot workcells. Two of the tested algorithms use A^* in resource allocation, as in Chapter 5, among the searches in several resolutions of the robot's configuration space. The methods with resource allocation use different "step sizes" at the same time. The decision how a robot's collision free configuration is expanded depends on how close that configuration is to the obstacles. When a configuration q is far away from ob-

[1] Portions of this document were first published as Antti Autere, A Dynamically Improving Heuristic for A^*, *Int. ICSC Congress on Intelligent Systems and Applications (ISA'2000)*, with permission from ICSC.

[2] Portions of this document were first published as Antti Autere, A^* as an Optimal Resource Allocation Policy in Path Finding Problems, *The 14th International FLAIRS Conference (FLAIRS-2001)*, copyright 2001, AAAI, with permission from AAAI.

stacles, then its successors are also far away from q. Instead, if a configuration q is near to an obstacle surface, then its successors are also near q.

The simulations suggest that the algorithms using A^* as a resource allocation policy, on average, find paths faster and consume less memory than the other hierarchical algorithms. Detailed pseudocodes are also given that implement the node generation process of the hierarchical path planners.

Chapter 6 is a revised version of the journal paper [4].[3]

Chapter 7 presents three new online A^*-based algorithms for power-aware routing of messages in large communication networks where future message sequences are not known. The goal is to maximize the average lifetime of a network in the sense of the average number of messages that can be sent through it. A message cannot be routed if its every path candidate in the network contains at least one node with less energy than is needed for the node to relay, that is receive and send, the message further. For achieving this goal the algorithms use different optimization criteria.

The new algorithms are simpler and their running times are shorter than those of the widely quoted *max-min* zP_{min} algorithm. In addition, they are not as sensitive to parameter settings as *max-min* zP_{min}. We show empirically that one of the algorithms produces longer average lifetimes than *max-min* zP_{min}. The other two algorithms perform similarly as *max-min* zP_{min}.

Chapter 7 is an enlarged version of reference [5].[4]

1.2 LIST OF PUBLICATIONS

The published material of this thesis is based on the following references:

Antti Autere, A Dynamically Improving Heuristic for A^*, *Int. ICSC Congress on Intelligent Systems and Applications (ISA'2000)*, Wollongong, NSW Australia, December 12–15, 2000. ICSC Academic Press.

Antti Autere, A^* as an Optimal Resource Allocation Policy in Path Finding Problems, *The 14th International FLAIRS Conference*, Key West, Florida USA, May 21–23, 2001. AAAI Press.

Antti Autere, Hierarchical A^* Based Path Planning — A Case Study, *Knowledge Based Systems*, 15(1–2), 2002. Elsevier.

Antti Autere, New Online Power-Aware Algorithms in Wireless Networks, *Int. Conference on Software, Telecommunications and Computer Networks (SoftCOM 2004)*, Split–Dubrovnik, Croatia, Venice Italy, October 11–13, 2004. University of Split.

[3] Portions of this document are reprinted from Knowledge Based Systems, Vol. 15, No. 1–2, Antti Autere, Hierarchical A^* based Path Planning — A Case Study, pp. 53–66, with permission from Elsevier.

[4] Portions of this document were first published as Antti Autere, New Online Power-Aware Algorithms in Wireless Networks, *Int. Conference on Software, Telecommunications and Computer Networks (SoftCOM 2004)*, with permission from SoftCOM.

2 BACKGROUND

In the following, we assume a directed or undirected graph $G = (V, E)$, where V denotes the set of vertices (nodes), and E denotes the set of edges between two vertices.

In addition, G has the following properties. There is a special node $s \in V$ called the *start node* and a non-empty set $\Gamma \subseteq V$ of *goal nodes*. Let $\text{neigh}(n) \subseteq V$ denote the set of *neighbors* to a node $n \in V$, that is, the nodes that are connected to n by an edge. For every n, $\text{neigh}(n)$ contains finite number of nodes, that is, G is *locally finite*. G can have an infinite number of nodes, though. Every edge in E between any two nodes n and m in G has a cost $c(n, m) \geq \delta > 0$. A *path* is a sequence of edges. A *solution path* is a path from the start node s to any of the goal nodes. In the sections concerning the A^* algorithm, the *cost of a path* is the sum of the costs of the edges which comprise the path. We let C^* denote the cost of any minimum cost solution path.

2.1 THE A^* ALGORITHM

A^* (originally in [32]; see also [55] and [57]) is a best-first graph search algorithm that always expands the "most promising" node n based on the *evaluation function*:

$$f(n) = g(n) + h(n). \tag{2.1}$$

The function $g(n)$ is the cost of the cheapest path from the start node to n among the paths found so far. The *heuristic* $h(n)$ is an estimate of the cost of the cheapest path from n to a goal node.

The A^* algorithm maintains two sets of nodes: CLOSED and OPEN. OPEN is sorted in ascending order of the evaluation function f. A^* repetitively removes from OPEN and places in CLOSED a node n for which $f(n)$ is minimum. If n is a goal node, then a solution is found. Otherwise, A^* generates all the neighboring nodes of n in $\text{neigh}(n)$, except the one that A^* has already found on the path from s to n. This set of nodes is called the *successors* to n and is denoted by $\text{succ}(n) \subseteq V$; $\text{succ}(n) \subseteq \text{neigh}(n)$. For every $n' \in \text{succ}(n)$, the value of the evaluation function $f(n')$ is calculated. Then A^* places every n' that is not already in OPEN or CLOSED in OPEN.

Figure 2.1 shows an example of the search process of A^* on a rectangular grid with three obstacles. The start node on the left is labeled by 0, and the goal on the right is labeled by 33. Black dots are expanded nodes in CLOSED (all their successors have been generated before the goal has been found), and white circles are generated nodes in OPEN. The labels of the nodes represent their g -values, measured by the Manhattan distance from the start node. The cost of an optimal path from the start node to the goal is 33. A traversal tree connects the nodes. The heuristic h guiding the search process of A^* is the Manhattan distance to the goal measured on a complete grid with no obstacles. From Figure 2.1, it can be seen that the heuristic prunes the search compared to the breadth-first search strategy: The path candidates of A^* go mostly to the right from the start node. In the breadth-first strategy, path candidates would go towards every direction forming a "Manhattan ball" around the start node. Note however, that nodes in Figure 2.1 form big heaps in front of the black obstacles.

The following presents the pseudocode for A^*, [57, pp. 64–65].

ALGORITHM A^*

```
(1)   Put the start node s into OPEN.
(2)   IF OPEN is empty THEN exit with failure.
(3)   Remove from OPEN and place in CLOSED a node n
        for which f is minimum.
        (Resolve ties for minimal f value,
        but always in favor of any goal node.)
(4)   IF n is a goal node THEN exit successfully
        with the solution obtained by
        tracing back the pointers from n to s.
(5)   ELSE expand n, generating all its successors,
        and attach to them pointers back to n.
      FOR every successor n' of n DO
(5.1)   IF n' is not already in OPEN or CLOSED
        THEN estimate h(n'),
          and calculate f(n') = g(n') + h(n'),
          where g(n') = g(n) + c(n,n') and g(s) = 0,
          and put n' into OPEN.
(5.2)   IF n' is already in OPEN or CLOSED
        THEN direct its pointer along the path
          yielding the lowest g(n').
(5.3)   IF n' required pointer adjustment and was in CLOSED
          THEN reopen it.
      END FOR
(6)   GO TO step 2.
```

During the search when A^* explores the search graph G, it builds up a *traversal tree* T composed of the nodes in OPEN or CLOSED and pointers between them. T is a subgraph of G. The leaf nodes of T are either in OPEN or in CLOSED. The latter leaf nodes were selected for expansion but no successors could have been generated to them. The interior nodes are also in either CLOSED or in OPEN . The reason for an interior node to be in OPEN is found in step (5.3) in the pseudocode. The set of path candidates that A^* has found so far includes the paths obtained by tracing back the pointers from every leaf node to the start node s.

Dijkstra's shortest path algorithm [19] is a special case of the A^* algorithm where $h(n) = 0$ for every node n.

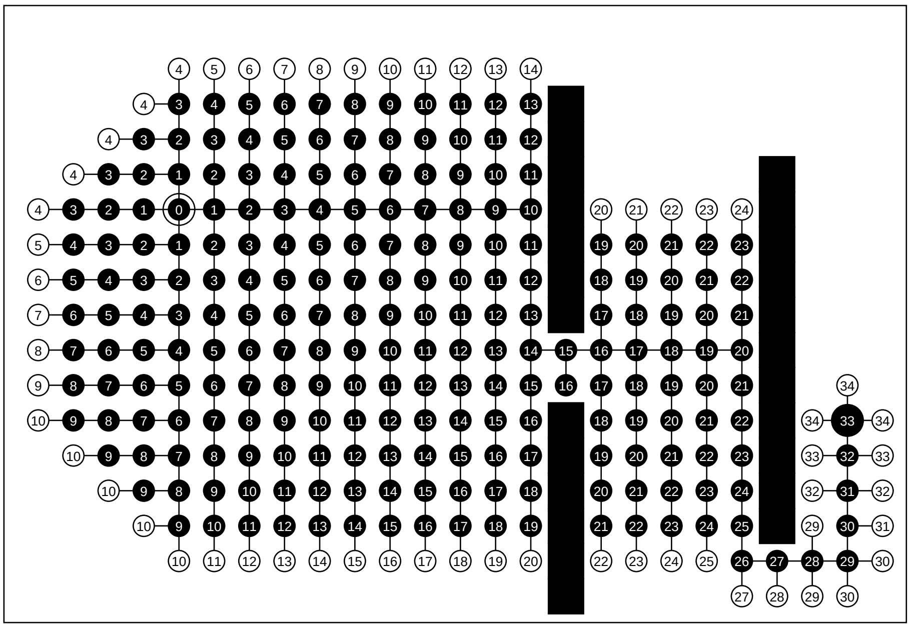

Figure 2.1: A 2-D grid with three obstacles: A^* [2].

2.2 TERMINATION AND COMPLETENESS OF A^*

Definition 2.2.1 *[57, p. 75] A search algorithm is* **complete** *if it terminates with a solution when one exists.*

A^* always terminates on finite graphs with strictly positive edge costs. The reason is that the number of acyclic paths in each finite graph is finite and with every node expansion A^* adds new edges to its traversal tree. Each newly added edge represents a new acyclic path and so the reservoir of paths must eventually be exhausted [57, p. 76].

Lemma 2.2.1 *[57, p. 77] If a solution path $P_{s-\gamma}$ exists, then OPEN cannot be empty and A^* cannot exit with failure, in line (2) in the pseudocode, before $P_{s-\gamma}$ is discovered.*

Proof: Assume that A^* exits with failure. Then there would be a last node $n' \in P_{s-\gamma}$ in OPEN which is expanded and found to generate no new successors (at least one node, the start node s, from $P_{s-\gamma}$ has surely entered OPEN). This, however, contradicts the assumption that n' lies on a solution path since every such node, except a goal node, has at least one successor also lying on a solution path. $\square$

It follows that A^* is complete on finite graphs [57, pp. 76–77].

Theorem 2.2.2 *[57, p. 77] If the cost of every infinite path in a locally finite graph is unbounded, then A^* is complete on that graph.*

Proof: Consider the set of nodes on the solution path $P_{s-\gamma}$; all these nodes are assigned finite f values, and at all times at least one of them is in OPEN.

If A^* does not return a solution, then it does not terminate at all. The latter follows from Lemma 2.2.1: A^* cannot exit with failure since the solution path $P_{s-\gamma}$ is assumed to exist. If A^* does not terminate, then it must be chasing an infinite path. Then, however, the infinite path has a *bounded* cost since otherwise all the nodes on $P_{s-\gamma}$ would have been expanded. This is a contradiction. □

Note that the proof does not actually require that all edge costs are strictly positive; it is enough to assume that the cost of every infinite path is unbounded. We will return to this subject later, in Section 3.2. In the following, however, we assume that all the edge costs are strictly positive.

2.3 ADMISSIBILITY OF A^*

Definition 2.3.1 *[57, p. 75] An algorithm is* **admissible** *if it is guaranteed to return an optimal solution whenever a solution exists.*

Definition 2.3.2 *[57, p. 77] A heuristic function h is* **admissible** *if it underestimates the cost of an optimal solution path from n to the goal nodes in Γ, $h(n)^*$, i.e.,*

$$h(n) \leq h^*(n) \ \ \forall n. \tag{2.2}$$

Before proving that A^* guided by an admissible heuristic is admissible, we present two lemmas. The lemmas tell us properties of nodes on an optimal path $P^*_{s-\gamma}$. Let us first introduce some more terminology.

From now on, we use the phrases 'an OPEN node' and 'a node in OPEN' as synonyms. The phrase 'the shallowest OPEN node on a path' means the node n that is in OPEN and whose g -value on that path (from s to n) is the smallest. Let f^* and g^* (such as h^* above) denote the optimal, that is, the minimum values of the variables, respectively.

Lemma 2.3.1 *[57, pp. 77–78] At any time before A^* guided by an admissible heuristic terminates, there exists an OPEN node n' on $P^*_{s-\gamma}$ with $f(n') \leq C^*$, where C^* is the cost of $P^*_{s-\gamma}$.*

Proof: Consider any optimal path $P^*_{s-\gamma}$;

$$P^*_{s-\gamma} = s, n_1, n_2, \ldots, n_k, n', \ldots, \gamma. \tag{2.3}$$

Let n' be the shallowest OPEN node on $P^*_{s-\gamma}$ (there is at least one OPEN node on $P^*_{s-\gamma}$ because γ is not CLOSED until termination), that is, all ancestors of n' are in CLOSED (see line (3) in the pseudocode of A^*). Since the path $s, n_1, n_2, \ldots, n_k, n'$ is optimal, it must be that the pointer assigned to n' is towards $n_k \in P^*_{s-n'}$ (see line (5.2) in the pseudocode of A^*). It follows that $g(n') = g^*(n')$. Using the admissibility of h, we obtain

$$f(n') = g^*(n') + h(n') \leq g^*(n') + h^*(n') = f^*(n') = C^*. \tag{2.4}$$

□

Lemma 2.3.2 *[57, p. 78] (Corollary of Lemma 2.3.1): Let n' be the shallowest OPEN node on an optimal path $P^*_{s-n''}$ to any arbitrary node n'', not necessarily in Γ. Then*

$$g(n') = g^*(n') \tag{2.5}$$

and the pointer path from n' to s will remain unaltered through the search.

Proof: It follows directly from the proof of Lemma 2.3.1, where the equality $g(n') = g^*(n')$ was established without using the fact that n' is in the role of a goal node. □

Theorem 2.3.3 *[57, p. 78] A^* guided by an admissible heuristic h is admissible.*

Proof: Suppose A^* terminates with a goal node $\gamma \in \Gamma$ for which $f(\gamma) = g(\gamma) > C^*$. When γ was chosen for expansion, it satisfied

$$f(\gamma) \leq f(n) \ \ \forall n \in \text{OPEN}. \tag{2.6}$$

The latter follows from line (3) in the pseudocode of A^*. This means that, immediately prior to termination, all the nodes in OPEN satisfied $f(n) > C^*$. This, however, contradicts Lemma 2.3.1 which guarantees the existence of at least one OPEN node n' with $f(n') \leq C^*$. Therefore the terminating γ must have $g(\gamma) = C^*$, which means that A^* returns an optimal path. □

2.4 CONDITIONS FOR NODE EXPANSION BY A^*

In this section, we will examine properties of nodes expanded and not expanded by A^*. In the following, we assume that the heuristic used by A^* is admissible.

Theorem 2.4.1 *[57, p. 79] No node expanded by A^* can have an f value exceeding C^*:*

$$f(n) \leq C^* \ \ \forall n \text{ expanded}. \tag{2.7}$$

Proof: Follows directly from Lemma 2.3.1. □

Theorem 2.4.2 *[57, p. 79] Every node in OPEN for which $f(n) < C^*$ will eventually be expanded by A^*.*

Proof: Suppose, at some stage, n is found in OPEN with $f(n) < C^*$. A^* terminates with $f(\gamma) = C^*$ after selecting γ for expansion from OPEN . The fact that γ, not n, was selected means either that n was expanded before γ or that $f(n)$ has in the meantime been modified so as to exceed C^*. But f can only be modified downward, see line (5.2) in the pseudocode of A^*. Thus n has to be expanded before γ. □

Theorems 2.4.1 and 2.4.2 constitute a necessary and a sufficient condition for A^* to expand nodes, respectively. The nodes m for which $f(m) = C^*$ may or may not be expanded by A^*. This remains to be decided by the *tie-breaking rule* employed in the particular implementation of A^*; the tie-breaking rule decides in which order the nodes with the same f value are expanded, see also line (3) in the pseudocode of A^*.

Theorems 2.4.1 and 2.4.2, giving simple conditions for node expansions, have two weaknesses. First, the function $g(n)$, in $f(n) = g(n) + h(n)$, is not only a property of node n but also depends on which path to n A^* has found most recently. Second, Theorem 2.4.2 requires that n resides in OPEN. Whether or not a given node ever enters OPEN depends not on n itself but on the behavior of A^* while exploring the paths leading to n. The following results overcome these difficulties. Now, we write $g_P(n)$ and $f_P(n)$ to emphasize that these function values depend also on the most recently found path to n.

Definition 2.4.1 *[57, p. 80] A path P is* **C-bounded** *relative to f if every node n along this path satisfies $f_P(n) \leq C$. Similarly, if a strict inequality holds for every n along P, then P is* **strictly C-bounded**.

Definition 2.4.1 is also valid for more general evaluation functions f_P than the additive one, $f_P(n) = g_P(n) + h(n)$, that A^* uses.

Now we formulate a necessary and a sufficient condition for A^* to expand a node. These are important tools in the analysis of the performance of A^*.

Theorem 2.4.3 *[57, pp. 80–81] A sufficient condition for A^* to expand a node n is that there exists some strictly C^*-bounded path P from the start node s to n.*

Proof: Assume to the contrary that there exists a strictly C^*-bounded path P from s to n and at termination its final node n has not yet been expanded. Let n' be the shallowest OPEN node on P (at termination). Since all ancestors of n' are in CLOSED, the g value that A^* assigns to n' cannot be more costly than $g_P(n')$; hence

$$f(n') = g(n') + h(n') \leq g_P(n') + h(n') < C^*. \tag{2.8}$$

But then n' should have been chosen for expansion instead of the goal node γ for which $f(\gamma) = C^*$. That contradicts our assumption that an OPEN node n' could exist on P and, hence, n must be expanded. □

Theorem 2.4.4 *[57, p. 81] A necessary condition for A^* to expand a node n is that there exists a C^*-bounded path from the start node s to n.*

Proof: If A^* expands a node n, then n must be in OPEN and $f(n) \leq C^*$ (Theorem 2.4.1). Consider a pointer path PP that A^* has assigned to n at the time of its expansion (see line (5) in the pseudocode of A^*). Each ancestor n' of n along PP has been chosen for expansion (perhaps more than once) some time in the past at which time it satisfied $g_{P'}(n') + h(n') \leq C^*$ (from Theorem 2.4.1). Its current $g(n')$ value along PP cannot be higher than its $g_{P'}(n')$ value at the time of expansion. Consequently every node n' along PP currently satisfies $f(n') \leq C^*$, meaning that PP itself is C^*-bounded. □

Based on Theorems 2.4.3 and 2.4.4 nodes in a graph can be divided into two sets: *nodes surely expanded by A^** and *nodes surely skipped by A^**.

Definition 2.4.2 *[17] The set of nodes* **surely expanded by** A^* *is a set of nodes* n *to which there exists a strictly* C^**-bounded path* P *from the start node* s*, i.e., for all nodes* n' *along* P*:* $f(n') < C^*$.

Definition 2.4.3 *The set of nodes* **surely skipped by** A^* *is a set of nodes* n *to which there does not exist a* C^**-bounded or a strictly* C^**-bounded path from* s*, i.e., the nodes* n *for which* $f(n) > C^*$.

Now, we can take the union of the sets in Definitions 2.4.2 and 2.4.3 and then its complement. This is a set of nodes that A^* may or may not expand, that is, nodes n to which there exists a C^*-bounded and not a strictly C^*-bounded path from s: $f(n) = C^*$. The expansion of these nodes depends on the tie-breaking rule of a particular implementation of A^*.

2.5 THE EFFECTIVENESS OF PATH FINDING ALGORITHMS

In this section, we clarify what we mean by saying that "a path finding algorithm A_1 is better or more efficient than another one A_2". We identify three different criteria for effectiveness. The two first ones measure *the number of expansions of distinct nodes*, that is, if a node is expanded many times only the first expansion is recorded.

Definition 2.5.1 *[57, p. 75] An algorithm* A_1 **dominates** A_2 *if every node expanded by* A_1 *is also expanded by* A_2.

Definition 2.5.2 *An algorithm* A_1 **largely dominates** A_2 *if every node surely expanded by* A_1 *is also expanded by* A_2*, cf. [17] and [57, p. 85].*

In case of two A^* algorithms, A_1^* and A_2^*, Definition 2.5.2 allows some nodes n for which $f_1(n) = C^*$ to be expanded by A_1^* and possibly skipped from expansion by A_2^*. How many such nodes an algorithm expands before finding a solution depends on its tie-breaking rule, cf. Section 2.4. Usually it is assumed that the number of these nodes is small. This does not always have to be the case, however.

The above definitions are rather strong because they require that in order for A_1 to be superior over A_2, A_1 must pass two difficult tests:

(1) to expand a *subset* of nodes rather than a *smaller number* of nodes,

(2) to outperform A_2 in *every* problem instance rather than in the *majority* of instances.

Unfortunately, there seems to be no easy way of loosening the above requirements without having to make statistical assumptions of problem instances and their relative likelihoods. This is because if in some problem instance A_2 skips even one node that is expanded by A_1, then one could immediately construct an infinite set of instances where A_2 outperforms A_1. This can be done by appending to the skipped node a variety of trees with negligible costs (and low heuristic values) [18].

The third criterion measures the **total number of node expansions**. Remember that A^* can reopen closed, expanded, nodes and expand them again later which this criterion takes into account. This criterion actually measures the **number of iterations** between lines (2) and (6) in the pseudocode of A^*. In the literature, there is no common name reserved for this criterion.

2.6 PRUNING POWER OF ADMISSIBLE HEURISTICS IN A^*

The power of the heuristic estimate h is measured by the amount of pruning induced by h and depends on the accuracy of this estimate. If h estimates the path cost precisely ($h(n) = h^*(n)\ \forall n$), then A^* will only expand nodes lying along optimal paths. On the other hand, if no heuristic at all is used ($h(n) = 0\ \forall n$), a breadth-first search will expand exhaustively all nodes reachable from s by a path costing less than C^*. The more common cases lie somewhere between these two extremes.

Theorem 2.6.1 *Let two A^* algorithms A_1^* and A_2^* be guided by admissible heuristics h_1 and h_2, respectively. If $h_1(n) > h_2(n)$ for every nongoal node n, then A_1^* dominates A_2^*, cf. [57, p. 81].*

Proof: Assume that A_1^* expands a node n. From Theorem 2.4.4 it follows that there must exist a C^*-bounded path P from s to n if judged by h_1. Since $h_1(n') > h_2(n')$ for every node n' along P, P is strictly C^*-bounded when judged by h_2. Hence, by Theorem 2.4.3, A_2^* must also expand n. □

If the relation $h_1(n) > h_2(n)$ for every nongoal node n holds between two admissible heuristics, then we say that h_1 is *more informed* than h_2.

Theorem 2.6.2 *Let two A^* algorithms A_1^* and A_2^* be guided by admissible heuristics h_1 and h_2, respectively. If $h_1(n) \geq h_2(n)$ for every node n, then A_1^* largely dominates A_2^*, cf. [57, p. 85].*

Proof: If n belongs in the set of nodes surely expanded by A_1^*, then there must exist a strictly C^*-bounded path P from s to n if judged by h_1. Since $h_1(n') \geq h_2(n')$ for every node n' along P, P is also strictly C^*-bounded when judged by h_2. Hence, by Theorem 2.4.3, A_2^* must also expand n. □

Notice the difference between Theorems 2.6.1 and 2.6.2: In Theorem 2.6.2 there may be some nodes that A_1^* expands but A_2^* doesn't. Theorem 2.6.2 is more useful of the two because it allows situations where $h_1(n) = h_2(n)$ for some n. These situations arise, for example, if an admissible heuristic is formed as a maximum of several admissible heuristics.

Theorem 2.6.1, however, can allow $h_1(n) = h_2(n)$ if the two algorithms A_1^* and A_2^* have the same tie-breaking rule that is purely structural, that is, it does not depend on the values of g and h, see [57, p. 112] exercise 3.1. An example of a purely structural tie-breaking rule is: If there are several nodes with the same f value to be expanded next, then always expand first the leftmost node in the search tree (here, of course, we assume that the search trees of the two algorithms are constructed in the same way).

2.7 A^* AND MONOTONE (CONSISTENT) HEURISTICS

In this section, we will show that under certain conditions A^* never reopens a node from CLOSED, and consequently the work related to re-expansions can be saved. Also some other useful properties follow.

The condition that enables A^* to forgo reopenings is a certain property of the heuristic h. The heuristic should be "geometric" in nature, which means that it should satisfy *the triangle inequality*.

Definition 2.7.1 *[57, p. 82] Let a node m be a descendant of a node n.*[1] *A heuristic function $h(n)$ is* **consistent** *if it satisfies:*

$$h(n) \leq k(n,m) + h(m), \tag{2.9}$$

for all pairs of nodes n and m, where $k(n,m)$ denotes the cost of a cheapest path from n to m.

Definition 2.7.2 *[57, p. 83] A heuristic function $h(n)$ is* **monotone** *if it satisfies*

$$h(n) \leq c(n,n') + h(n') \ \ \forall n, n';\ n' \in \text{succ}(n), \tag{2.10}$$

where $\text{succ}(n)$ *denotes the set of successors to n.*

Clearly, consistency implies monotonicity but it follows from a proof by induction that monotonicity also implies consistency:

Theorem 2.7.1 *[57, p. 83] Monotonicity and consistency are equivalent properties.*

It is also simple to prove that consistency (monotonicity) implies admissibility.

Theorem 2.7.2 *[57, p. 83] Every consistent heuristic is also admissible.*

Proof: We replace m in the equation of Definition 2.7.1 by any goal node $\gamma \in \Gamma$, obtaining

$$h(n) \leq k(n,\gamma) + h(\gamma) \ \ \forall n. \tag{2.11}$$

Now, since $h(\gamma) = 0$ and $k(n,\gamma) = h^*(n)$ for some goal node $\gamma \in \Gamma$, we have the admissibility condition $h(n) \leq h^*(n)$. $\square$

It follows from the next theorem that an A^* using a consistent (monotone) heuristic never reopens nodes from CLOSED.

Theorem 2.7.3 *[57, pp. 83–84] An A^* guided by a consistent heuristic finds optimal paths to all expanded nodes, i.e.,*

$$g(n) = g^*(n) \ \ \forall n \in \text{CLOSED}. \tag{2.12}$$

[1] The node m is any node in the subtree of the traversal tree of A^* starting from n.

Proof: Assume that A^* selects for expansion a node for which $g(n) > g^*(n)$. Consider an optimal path P^*_{s-n} from s to n. If n is the only OPEN node on P^*_{s-n}, then obviously all the ancestors of n have been expanded and from Lemma 2.3.2 it follows that $g(n) = g^*(n)$. Otherwise, if n is not the only OPEN node on P^*_{s-n}, then let n' be the shallowest OPEN node on P^*_{s-n}. Lemma 2.3.2 states that $g(n') = g^*(n')$ and therefore, using consistency, we have

$$f(n') = g^*(n') + h(n') \leq g^*(n') + k(n', n) + h(n). \tag{2.13}$$

The sum $g^*(n') + k(n', n)$ is equal to $g^*(n)$ because n' is an ancestor of n along P^*_{s-n} and so

$$f(n') \leq g^*(n) + h(n). \tag{2.14}$$

Now the assumption $g(n) > g^*(n)$ implies $f(n') < f(n)$ and hence n' should have been expanded before n. Thus $g(n) = g^*(n)$. □

Theorem 2.7.4 *[57, p. 84] Monotonicity implies that the f values of the sequence of nodes expanded by A^* are non-decreasing.*

Proof: Let n_2 be expanded immediately after n_1. If n_2 resided in OPEN while n_1 was expanded, then $f(n_1) \leq f(n_2)$ follows from the node selection rule of A^*. If n_2 did not reside there, then n_2 must be a successor to n_1 for which we have $g(n_2) = g(n_1) + c(n_1, n_2)$ and for which monotonicity dictates:

$$f(n_2) = g(n_1) + c(n_1, n_2) + h(n_2) \geq g(n_1) + h(n_1) = f(n_1). \tag{2.15}$$

□

Note that Theorem 2.7.4 guarantees only implication. Later in Section 2.12.1, we will encounter a variant of A^* that always produces a non-decreasing sequence of f values but its heuristic does not have to be monotone (consistent).

Monotonicity simplifies the necessary and sufficient conditions for node expansions and the definition of the corresponding sets, cf. Theorems 2.4.3, 2.4.4 and Definitions 2.4.2, 2.4.3. This is mainly due to Theorem 2.7.3 stating that $g(n) = g^*(n)$ when A^* expands node n for the first time. This means that the following conditions indeed are functions of n only.

Theorem 2.7.5 *[57, p. 84] If h is monotone, then the necessary condition for expanding node n is given by*

$$g^*(n) + h(n) \leq C^* \tag{2.16}$$

and the sufficient condition by the strict inequality

$$g^*(n) + h(n) < C^*. \tag{2.17}$$

Proof: The necessary condition follows by combining Theorems 2.4.1 and 2.7.3. The sufficient condition is based on the non-decreasing nature of f

values along an optimal path P^*_{s-n} from s to n. If n' is the parent of n along P^*_{s-n}, then from Theorem 2.7.4 it follows that

$$f(n') = g^*(n') + h(n') \leq g^*(n) + h(n) = f(n). \tag{2.18}$$

This implies that if n satisfies $g^*(n) + h(n) < C^*$, all its ancestors along P^*_{s-n} must also satisfy this inequality. Thus P^*_{s-n} is strictly C^*-bounded and, according to Theorem 2.4.3, n will be expanded by A^*. $\square$

The sets of nodes that A^* surely expands and never expands, Definitions 2.4.2 and 2.4.3, can now be easily defined by using the above conditions for node openings. The set of nodes that A^* may or may not expand is $\{n \mid g^*(n) + h(n) = C^*\}$.

Finally, let us present an easily provable theorem that we will need later.

Theorem 2.7.6 *The maximum of two monotone heuristics is monotone.*

2.8 ON GENERALIZED BEST-FIRST ALGORITHMS

This section introduces some generalizations, or extensions, of the A^* algorithm. In the following, let us again assume a locally finite graph G with a start node, a non-empty set of goal nodes, and strictly positive edge costs. Now the cost of a path in G can be *an arbitrary function of possible weights assigned to the nodes and branches along that path*.

A *general best-first* strategy (GBF) [18, pp. 505–506] searches G by constructing a tree T of selected paths of G using the elementary computational operation of *node expansion*, that is, generating all successors of a given node. GBF starts searching from s and will expand a leaf node of T that is "most promising" measured by an *evaluation function* f. GBF will maintain in T all previously encountered paths that still may be candidates for the best solution path. GBF stops when no such candidate is available for node expansion. The solution is then the best solution path found so far, or a failure if no such path has been found.

GBF has a more general evaluation function than $f = g + h$ of A^*. For example, the evaluation functions can include multiplicative, maximum, most frequent, last, and average costs along paths, etc. In cases of additive cost, $f(n)$ can be a function of the path from s to n, for example, $f(n) = \max_{n'}\{g(n') + h(n')\}$, where n' is a node on the path to n. The latter evaluation function is used, for example, in the A^{**} algorithm discussed in Section 2.12.1. In Chapter 4, we will encounter an f that is a function of the search tree.

Usually, f is restricted to be *order preserving*, [18, p. 506]. An evaluation function is order preserving if, for any two paths P_1 and P_2, leading from s to n, and for any common extension P_3 of those paths, it holds that:

$$f(P_1) \geq f(P_2) \Rightarrow f(P_1P_3) \geq f(P_2P_3). \tag{2.19}$$

Order preservation is a version of the so-called *principle of optimality* in dynamic programming [22] that simply states that every subpath of an optimal path is also optimal. If the above is true, then there is no need to keep in the search tree T multiple paths to the same node; each time a node n is generated that already is in the search tree, we maintain only the lower f or g path to it, see for example line (5.2) in the pseudocode of A^* and line (5.2') below.

We call the GBF with the above order preserving evaluation function BF*. This section introduces some notions and properties of BF* algorithms from [57, 18]. The notions, lemmas and theorems are somewhat analogous to those of A^* in Sections 2.2– 2.4. We will use these results to compare the performance of A^* to that of other BF* algorithms, GBF strategies, and other generalized search procedures in Section 2.9 and Chapter 3.

The pseudocode of the BF* algorithm is identical to that of A^* in Section 2.1, except that the evaluation function is now simply f (not $g + h$) and is calculated just above line (5.1), and line (5.2) is replaced with line (5.2') (g is replaced with f) [18, p. 506]:

```
(5.2’) IF n’ is already on OPEN or CLOSED
       THEN direct its pointer along the path yielding
       the lowest f(n’) value.
```

Despite the difference between lines (5.2) and (5.2'), A^* is a BF* algorithm since it does not matter whether f or g is used in redirecting pointers on line (5.2) ($f(n) = g(n) + h(n)$ in A^* is order preserving). *Depth-first* strategies can be obtained from BF* by setting $f(n') = f(n) - 1,\ f(s) = 0$, where the node n' is a successor to n.[2] If a locally finite graph to be searched contains an infinite number of nodes, then the depth-first strategy may not terminate even if a solution path with a bounded cost exists.

In the following, let us assume that $f \geq 0$. We start the study of BF* algorithm without assuming any relationships between the cost C, defined on complete solution paths, and the evaluation function f, defined on partial paths. The next results and more can be found in references [57, 18]. The presentation follows [18] and we refer only to it.

In locally finite graphs, the set of solution paths is countable [18, p. 509], hence they can be enumerated:

$$P_1^S, P_2^S, \ldots, P_j^S, \ldots \tag{2.20}$$

and we can use the notation $f_j(n)$ to represent $f_{P_j^S}(n)$. Let a non-negative real number M be

$$M = \min_j \{\max_{n \in P_j^S} \{f_j(n)\}\}. \tag{2.21}$$

We call M the *minmax value related to the evaluation function* f or the *minmax value for algorithm A (using f)*. For example, let M be the minmax value for A^*. If the heuristic h is admissible ($h(n) \leq h^*(n)\ \forall n$) then $M = C^*$, the cheapest cost of any solution path, by Theorem 2.4.1. If the heuristic can be nonadmissible, then $M \geq C^*$.

[2] Here we let the f-values become negative.

2.8.1 Termination and Completeness

We state the following lemma without a proof. The proof is found in [18, pp. 509–510].

Lemma 2.8.1 *[18, pp. 509–510] At any time before BF* terminates, there exists in OPEN a node n' that is on some solution path and for which $f(n') \leq M$.*

Lemma 2.8.1 is analogous to Lemma 2.3.1 in Section 2.3. The next theorem is a counterpart of Theorem 2.2.2.

Theorem 2.8.2 *[18, p. 510] If there is a solution path and f is such that $f_P(n)$ is unbounded along any infinite path P, then BF* terminates with a solution, i.e., BF* is complete.*

Proof: In any locally finite graph, there is only a finite number of paths with finite length. If BF* does not terminate, then there is at least one infinite path along which every finite-depth node will eventually be expanded. That means that f must increase beyond bounds and, after a some time τ, no OPEN nodes on any given solution path will ever be expanded. However, from Lemma 2.8.1, $f(n') \leq M$ for some OPEN node n' along a solution path, which contradicts the assumption that n' will never be chosen for expansion. □

The condition of Theorem 2.8.2 may not hold for cost measures such as the maximum of the edge costs along the path. In this situation, as well as the case of depth-first strategies ($f \leq 0$), termination cannot be guaranteed on graphs having infinite number of nodes. Then the termination must be controlled, for example, by using succeeding increasing but finite depth bounds. An example is the depth-first iteratively deepening A^* algorithm IDA^* in [40].

2.8.2 Solution Quality and Admissibility of BF^*

We have not yet specified any relation between the cost C of solution paths, and the evaluation function f of partial paths or solution candidates. Since f is a tool to guide the search toward the cheapest solution paths, we define that f is monotonic with C when evaluated on complete solution paths: If P and Q are two solution paths with $C(P) > C(Q)$, then $f(P) > f(Q)$. So, let

$$f(s, n_1, n_2, \ldots, n) = \begin{cases} \psi(C(s, n_1, n_2, \ldots, n)) & n \in \Gamma, \\ F(s, n_1, n_2, \ldots, n) & n \notin \Gamma, \end{cases} \tag{2.22}$$

where $\psi\colon C(\cdot) \rightarrow \mathcal{R}^+$ is an increasing function of $C(\cdot)$ over the positive reals. No constraints, however, are assumed concerning the relation between C and f on nongoal nodes; $F(\cdot)$ is an arbitrary function of the path $P = s, n_1, n_2, \ldots, n$. The evaluation function $f = g + h$ used by A^* is an example of the above function: For a goal node $\gamma \in \Gamma$, $h(\gamma) = 0$ implies

$f = C$ on solution paths, whereas on other paths f is not necessarily related to C, since h could take on arbitrary values.

We now give two results concerning the cost of a solution path using the above relationship.

Theorem 2.8.3 *[18, p. 511] BF* is $\psi^{-1}(M)$ -admissible, i.e., the cost of the solution path found by BF* is at most $\psi^{-1}(M)$.*

Proof: Let BF* terminate with a solution path $P_j^S = s, \ldots, \gamma$ where $\gamma \in \Gamma$. From Lemma 2.8.1, it follows that BF* cannot select for expansion any node n having $f(n) > M$, where M is the minmax value related to f. This includes the node $\gamma \in \Gamma$ and, hence, $f_j(\gamma) \leq M$. But the above constraint on f implies that $f_j(\gamma) = \psi(C(P_j^S))$ and so, since ψ and ψ^{-1} are monotonic, $C(P_j^S) \leq \psi^{-1}(M)$. $\square$

Theorem 2.8.3 is useful in calculating the cost of the solution path a nonadmissible algorithm finds as a function of the cost of the existing optimal path, i.e., *the degree of suboptimality*, as we will see later in Section 2.11. Karp and Pearl [38] have used Theorem 2.8.3 in studying search on graphs with random costs where they have used estimates of M to establish probabilistic bounds on the degree of suboptimality.

Theorem 2.8.3 can also be used to check for admissibility, when $C(P^S) = C^*$: We only need to verify the equality $\psi^{-1}(M) = C^*$. For studying admissibility, however, we preferably use the next corollary.

Corollary 2.8.4 *(of Theorem 2.8.3) [18, pp. 512–513] If in every graph searched by BF* there exists at least one optimal solution path along which f attains its maximal value on the goal node, then BF* is admissible.*

Proof: Let BF* terminate with a solution path $P_j^S = s, \ldots, t$ and let $P^* = s, \ldots, \gamma$ be an optimal solution path such that $\max_{n \in P^*} f_{P^*}(n) = f_{P^*}(\gamma)$. By Theorem 2.8.3, we know that $f_j(t) \leq M$. Moreover, we have $M \leq \max_{n \in P_i^S} f_i(n)$ for every solution path P_i^S. In particular, taking $P_i^S = P^*$, we obtain

$$f_j(t) \leq M \leq \max_{n \in P^*} f_{P^*}(n) = f_{P^*}(\gamma). \tag{2.23}$$

However, from the constraint on f we know that f is monotonically increasing in C when evaluated on complete solution paths; thus

$$f_j(t) = \psi(C(P_j^S)) \leq f_{P^*}(\gamma) = \psi(C^*), \tag{2.24}$$

implying $C(P_j^S)) \leq C^*$. $\square$

Dechter and Pearl [18] demonstrate the use of the above corollary by calculating the range of admissibility of a weighted evaluation function $f_w = (1-w)g + wh$, $0 \leq w \leq 1$ [58]. Here $\psi(C) = (1-w)C$ for $w < 1$. It remains to examine which values of $w < 1$ will force f_w to get its maximum value at the end of an optimal path P^*. For more details, see [18, p. 513].

The above corollary can also be used to check whether a given combination of g and h, $f = f(g, h)$, would result in an admissible heuristic in problems of minimizing additive cost measures. For more details, see again [18, p. 513].

2.8.3 Conditions for Node Expansion

To save space, we state here only two results without proofs. For a more thorough treatment of the subject, see [57, 18]. The next theorem follows directly from Lemma 2.8.1. Its counterpart is Theorem 2.4.1, where C^* plays the role of the minmax value M.

Theorem 2.8.5 *[18, p. 514] Any node expanded by BF* has $f(n) \leq M$ immediately before its expansion.*

Let us define a *strictly M-bounded path* analogously to Definition 2.4.1 in Section 2.4 (replace C by M). The following theorem, a sufficient condition for BF* to expand a node, is a counterpart of Theorem 2.4.3.

Theorem 2.8.6 *[18, p. 514] Any node reachable from s by a strictly M-bounded path will be expanded by BF*.*

In later sections, we will use the above result to compare the performance of A^* to that of other generalized path finding algorithms.

2.9 MORE OPTIMALITY PROPERTIES OF A^*

In this section, we show that A^* is an "effective" path searching algorithm. We first compare the performance of A^* to that of a class of GBF strategies that are admissible and *equally informed* as A^*. An algorithm is equally informed as A^* if it has access to the same heuristic information h as A^* but can use h in any way it likes. The admissibility requires that the algorithm finds a least-cost solution, like A^*, when $h(n) \leq h^*(n)\ \forall n \in G$. We denote this class of algorithms $\mathbf{A}_{ad}$. If $h(n) > h^*(n)$ then solutions found by algorithms in $\mathbf{A}_{ad}$ and A^* may be of different cost.

Additionally, we assume that every algorithm A in $\mathbf{A}_{ad}$ uses the primitive step of *node expansion*, and A only expands nodes that it has generated before. Furthermore, A always begins the expansion process at the start node s. GBF (and BF*) strategies in Section 2.8 satisfy these requirements. The requirements exclude, however, bidirectional searches or algorithms that simultaneously grow search trees from several "seed nodes" across G.

We assume that a heuristic $h(n)$ is assigned to the nodes of G and the value $h(n)$ is available to each path finding algorithm that generates n. Thus h is "static" in the sense that its values do not depend on the search process of an algorithm. Hence we can actually set $h(n)$ as one of the parameters that specify problem instances. Let us denote such a problem instance by the quadruple $I = (G, s, \Gamma, h)$. In particular, we are interested in two sets of problem instances where the heuristic h is admissible or consistent (monotone), [18, p. 522]:

$$\mathbf{I}_{AD} = \{(G, s, \Gamma, h) \mid h(n) \leq h^*(n)\ \forall n \in G\}, \tag{2.25}$$

$$\mathbf{I}_{CON} = \{(G, s, \Gamma, h) \mid h \text{ is consistent on } G\}. \tag{2.26}$$

Now we require that consistency (or monotonicity) holds for *any* pair of nodes in G. Since consistency (monotonicity) implies admissibility but not vice versa $\mathbf{I}_{CON} \subseteq \mathbf{I}_{AD}$.

In the following, the nodes *surely expanded by algorithm* A means the set of nodes that A is always *guaranteed* to expand before it finds a solution path. We recall from Theorem 2.4.3, Definitions 2.4.1 and 2.4.2, that if the algorithm is A^*, then the set of nodes surely expanded by A^* has a characterizing property: *All the nodes n to which there exists a strictly C^*-bounded path belong to this set.* We also saw that a similar property holds for nodes surely expanded by BF* in Theorem 2.8.6.

We are now more precise when we define the notion of *dominance*, cf. Definition 2.5.2.

Definition 2.9.1 *[17] An algorithm A is* **optimal** *over a class* $\mathbf{A}$ *of algorithms relative to a set* $\mathbf{I}$ *of problem instances if in each instance of* $\mathbf{I}$*, every algorithm in* $\mathbf{A}$ *will expand all the nodes surely expanded by A in that problem instance.*

Dechter and Pearl [18] refine the concept of optimality by noting that A^* does not stand for just one but a whole family of algorithms, each defined by the tie-breaking rule chosen. They identified four different optimality criteria. We, however, do not go into details and later use mostly the optimality in Definition 2.9.1.

The next theorem guarantees that A^* is an optimal algorithm according to Definition 2.9.1 over the class $\mathbf{A}_{ad}$ of algorithms relative to problem instances $\mathbf{I}_{CON}$, that is, where the heuristic h is consistent (monotone). We also present the proof since we will prove new theorems in later sections by modifying the proof's structure. Let $\mathbf{N}_f^{C^*} = \mathbf{N}_{g+h}^{C^*}$ denote the set of nodes surely expanded by A^*.

Theorem 2.9.1 *Any algorithm that is admissible on $\mathbf{I}_{AD}$, i.e., any algorithm in $\mathbf{A}_{ad}$ will expand, in every instance $I \in \mathbf{I}_{CON}$, all nodes surely expanded by A^*, cf. [18, pp. 522–524] and [17].*

Proof: Let $I = (G, s, \Gamma, h)$ be some problem instance in $\mathbf{I}_{CON}$ and assume that n is surely expanded by A^*, i.e., $n \in \mathbf{N}_{g+h}^{C^*}$. Therefore there exists a path P_{s-n} such that

$$g(n') + h(n') < C^* \quad \forall n' \in P_{s-n}. \tag{2.27}$$

Let B be an algorithm in $\mathbf{A}_{ad}$, namely, halting with cost C^* in I, and assume that B does not expand n. Moreover, assume that if there is only one least-cost solution path in G, then it does not go via n. If the only least-cost path goes via n, then B must also expand n in order to be in $\mathbf{A}_{ad}$.

We now construct a new graph G', see Figure 2.2, by adding to G a goal node t with $h(t) = 0$ and an edge from n to t with nonnegative cost $c(n, t) = h(n) + \Delta$, where

$$\Delta = \frac{1}{2}(C^* - D) > 0 \quad \text{and} \quad D = \max\{f(n') \mid n' \in \mathbf{N}_{g+h}^{C^*}\}. \tag{2.28}$$

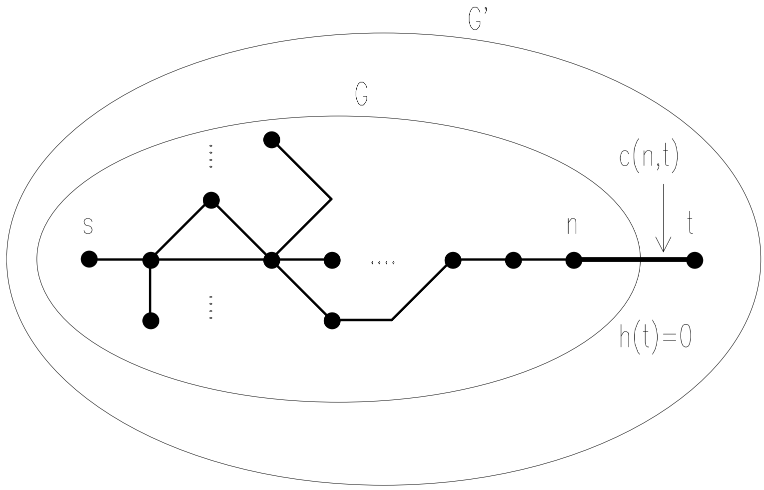

Figure 2.2: The graph G' is constructed from G by including a new solution path going via n to the new goal node t.

This construction creates a new solution path P^* with a cost $C(P^*) < C^*$ and, simultaneously (owing to the consistency of h on I), retains the consistency (and admissibility) of h on the new instance $I' = (G', s, \Gamma \cup \{t\}, h)$. To establish the consistency of h on I', we note that since we kept the h values of all the nodes in G unchanged, consistency will continue to hold between any pair of nodes previously in G. It remains to verify consistency on pairs involving the new goal node t, which amounts to establishing the inequality $h(n') \leq k(n', t)$ for every node n' in G. Now, if at some node n' we have $h(n') > k(n', t)$, then we should also have

$$h(n') > k(n', n) + c(n, t) = k(n', n) + h(n) + \Delta \qquad (2.29)$$

in violation of the consistency of h on I. Thus the new instance I' is also in $\mathbf{I}_{CON}$.

In searching G', algorithm A^* will find the extended path P^* with cost $C(P^*) < C^*$ because

$$f(t) = g(n) + c(n, t) = f(n) + \Delta \leq D + \Delta < D + 2\Delta = C^* \qquad (2.30)$$

and, so, t is reachable from s by a path strictly bounded by C^*, which ensures its selection.

Algorithm B, on the other hand, if it avoids expanding n, must behave the same as in problem instance I, halting with cost C^*, which is higher than that found by A^*. This contradicts the supposition that B is both admissible on I' and avoids the expansion of node n. $\square$

The above proof makes it tempting to conjecture that A^* is also optimal relative to instances in which h is admissible but not necessarily consistent, that is, instances in $\mathbf{I}_{AD}$. However, in the above theorem, if h is admissible but not consistent, then, after adding the goal node t to G we cannot guarantee that h will remain admissible on the new instance I' [18, p. 524]. Moreover, we can construct an algorithm that is admissible on $\mathbf{I}_{AD}$ and in some problem instances, it will outperform A^*. For more details, see [17] and [18, pp.

524–525]. However, if the search graph is a tree, then it will turn out that A^* is the optimal algorithm relative to $\mathbf{I}_{AD}$, see Section 5.4.

Is there an optimal algorithm over $\mathbf{A}_{ad}$ relative to $\mathbf{I}_{AD}$, in the following sense? An optimal algorithm, if it exists, must skip in some problem instances in $\mathbf{I}_{AD}$ at least one node surely expanded by A^*, while it is not allowed to expand any node that is surely skipped by A^*. In references [17] and [18, pp. 525–526], Dechter and Pearl show that this is impossible relative to a subset of $\mathbf{I}_{AD}$, for which there exists at least one optimal solution path along which h is not fully informed, i.e., $h(n) < h^*(n)$ for every non-goal node n on that path. Let us denote this subset by $\mathbf{I}^-_{AD}$. Note that in $\mathbf{I}^-_{AD}$, *all* nodes expanded by A^* equals to the set of nodes surely expanded by it.

Theorem 2.9.2 *[18, pp. 525–526] and [17] If an algorithm B in $\mathbf{A}_{ad}$ does not expand a node that is surely expanded by A^* in some problem instance in $\mathbf{I}^-_{AD}$, then, in that very problem instance, B must expand a node that is avoided by every tie-breaking rule in A^*.*

Theorem 2.9.2 states that although there does not exist an optimal algorithm in the above sense, A^* is not a bad choice: *"The only way to gain one node from A^* is to relinquish another".*

We now compare A^* with two subclasses of $\mathbf{A}_{ad}$: $\mathbf{A}_{gc}$ and $\mathbf{A}_{bf}$. $\mathbf{A}_{gc}$ denotes the class of GBF algorithms (Section 2.8) that are *globally compatible* with A^*, that is, they return optimal solutions whenever A^* does, even in cases where the heuristic is not admissible ($h(n) > h^*(n)$ for some nodes $n \in G$). It is easy to see that $A^* \in \mathbf{A}_{gc}$ and $\mathbf{A}_{gc} \subseteq \mathbf{A}_{ad}$. $\mathbf{A}_{bf}$ stands for the class of BF* algorithms (Section 2.8), guided by a path-dependent evaluation function, which are admissible if $h(n) \leq h^*(n)\ \forall n \in G$. It is again easy to see that $A^* \in \mathbf{A}_{bf}$ and $\mathbf{A}_{bf} \subseteq \mathbf{A}_{ad}$. However, no inclusion relation holds between $\mathbf{A}_{gc}$ and $\mathbf{A}_{bf}$, despite that BF* algorithms are a special class of GBF algorithms. This is because algorithms in $\mathbf{A}_{bf}$ may return non-optimal solutions on the same instances where A^* returns optimal solutions, that is, when the heuristic is not admissible, see [18, pp. 520, 523].

Theorem 2.9.3 *[18, p. 528] Any algorithm in $\mathbf{A}_{gc}$ will expand, in every instance $I \in \mathbf{I}_{AD}$, all nodes surely expanded by A^*.*

The proof is analogous to that of Theorem 2.9.1. A new graph G' is again constructed from G by adding to G a new goal node, as in Figure 2.2, to obtain a contradiction. For more details, see [18, p. 528].

Theorem 2.9.3 states that if we restrict the class of algorithms from $\mathbf{A}_{ad}$, for which there exists no optimal algorithm, to the class $\mathbf{A}_{gc}$, then A^* is an optimal algorithm in that class in the sense of Definition 2.9.1.

The next theorem guarantees that A^* is an optimal algorithm over the class $\mathbf{A}_{bf}$. We present also the proof since we will modify it when we prove new theorems in later sections.

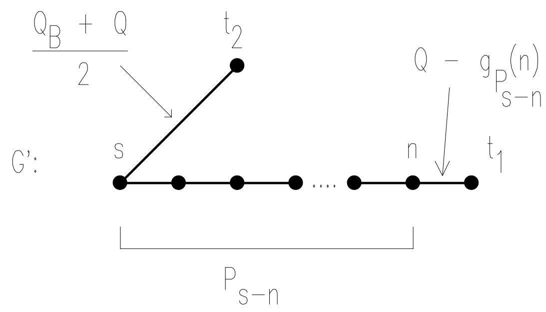

Figure 2.3: The graph G' constructed from G with two solution paths going to the new goal nodes t_1 and t_2.

Let a BF* algorithm in $\mathbf{A}_{bf}$ be guided by an evaluation function $f_B(n) = f_P$ that is a function of the path P leading to the node n, that is, a function of the nodes, the edge-costs and the heuristic values of the nodes along P:

$$f_P = f(s, n_1, n_2, \ldots, n) = f(\{n_i\}, \{c(n_i, n_{i+1})\}, \{h(n_i)\} \mid n_i \in P). \tag{2.31}$$

Theorem 2.9.4 *Let B be an admissible BF* algorithm, i.e. B in $\mathbf{A}_{ad}$, using the above evaluation function f_B such that for every problem instance $I \in \mathbf{I}_{AD}$, f_B satisfies*

$$f_B(\gamma) = f_{P_{s-\gamma}} = f(s, n_1, n_2, \ldots, \gamma) = C(P_{s-\gamma}) \ \ \forall \gamma \in \Gamma. \tag{2.32}$$

Then B expands every node in $\mathbf{N}^{C^*}_{g+h}$, i.e., every node surely expanded by A^*, cf. [18, pp. 529–530].

Proof: Let $\mathbf{N}^{C^*}_{f_B}$ stand for the set of all the nodes that are reachable by some strictly C^*-bounded path relative to f_B. Let $I = (G, s, \Gamma, h) \in \mathbf{I}_{AD}$ and assume $n \in \mathbf{N}^{C^*}_{g+h}$ but $n \notin \mathbf{N}^{C^*}_{f_B}$, i.e., there exists a path P_{s-n} such that for every $n' \in P_{s-n} : \ g_P(n') + h(n') < C^*$ and, for some $n' \in P_{s-n} : \ f_B(n') \geq C^*$. Let

$$Q = \max_{n' \in P_{s-n}} \{g(n') + h(n')\} \ \text{ and } \ Q_B = \max_{n' \in P_{s-n}} \{f_B(n')\}. \tag{2.33}$$

Obviously, $Q < C^*$ and $Q_B \geq C^* \Rightarrow Q_B > Q$. Define G' to include only the path P_{s-n} with two additional goal nodes t_1 and t_2 as described in Figure 2.3. The cost on edge (s, t_2) is $(Q_B + Q)/2$; the cost on edge (n, t_1) is $Q - g_{P_{s-n}}(n)$; t_1 and t_2 are assigned $h = 0$, while all other nodes retain their old h values. $I' = (G', s, \Gamma \cup \{t_1, t_2\}, h) \in \mathbf{I}_{AD}$ since $\forall n' \in P_{s-n} : g(n') + h(n') \leq Q$, which implies that $h(n') \leq Q - g(n') = h^*_{I'}(n')$.

Clearly the optimal path in G' is P_{s-t_1} with cost Q. However, the evaluation function f_B satisfies

$$Q < f_B(t_2) = C(P_{s-t_2}) = \frac{Q_B + Q}{2} < Q_B. \tag{2.34}$$

Since $M_{P_{s-t_1}} = \max_{q \in P_{s-t_1}} \{f_B(q)\} \geq Q_B$, we have $f_B(t_2) < M_{P_{s-t_1}}$. Hence B halts on the suboptimal path P_{s-t_2}, contradicting its admissibility. Thus we have proved that $\mathbf{N}^{C^*}_{g+h} \subseteq \mathbf{N}^{C^*}_{f_B}$, cf. Lemma 6 in [18, p. 529].

Now, let M be the minmax value related to the evaluation function f_B on G of I, defined in Section 2.8. It is easy to see that $M \geq C^*$. From that and Theorem 2.9.3 we get

$$\mathbf{N}_{g+h}^{C^*} \subseteq \mathbf{N}_{f_B}^{C^*} \subseteq \mathbf{N}_{f_B}^{M}, \tag{2.35}$$

and Theorem 2.8.6 states that all the nodes in $\mathbf{N}_{f_B}^{M}$ are expanded by the admissible BF* algorithm, cf. Theorem 12 (a) in [18, pp. 529–530]. □

2.10 RELAXED MODELS AND ADMISSIBLE HEURISTICS

This section describes one method of generating admissible heuristics, namely utilizing *relaxed models*.

Relaxed models are a well-known source of admissible heuristics [57, 72, 53, 30, 62, 61]. They are abstract problem descriptions generated by ignoring constraints that are present in base-level problems [30]. The intuitive reason that abstractions generate admissible heuristics is because they add short-cut solution paths by simplifying the original problem [61]. Several authors emphasize, however, that relaxed problems should be easily solvable compared with their original counterparts in order to speed up the overall computation time, [72, 57, 53, 30].

Let a search problem be (G, c, s, Γ, h), where $G = (V, E)$ as before, $s \in V$ is the start node, $\Gamma \subseteq V$ is a set of goal nodes, and h is a heuristic function. Moreover, the cost function between any two nodes in V is $c \colon V \times V \to \mathcal{R}^+$. The following defines a relaxed model of a problem instance.

Definition 2.10.1 *An* **abstracting transformation** $\phi \colon G \to G'$ *removes certain details (e.g. constraints) from the original problem* (G, c, s, Γ, h) *and produces a* **relaxed problem** *or a* **relaxed model** (G', c', s, Γ', h') *iff* ϕ *reduces all costs and preserves all the goals* $\gamma \in \Gamma$*:*

$$c'(\phi(n), \phi(m)) \leq c(n, m) \quad \forall n, m \in V \tag{2.36}$$

$$\phi(\gamma) \in \Gamma' \quad \forall \gamma \in \Gamma, \tag{2.37}$$

where $c(n, m)$ *and* $c'(\phi(n), \phi(m))$ *are the costs of the shortest paths between the corresponding nodes, cf. [61].*

The next theorem shows that heuristics generated by optimizations over relaxed models are monotone and thus admissible.

Theorem 2.10.1 *[72, 57] Assign every node* n *in* G *of the original problem a heuristic* $h(n)$ *that is the cost of the optimal solution of a relaxed problem instance with start node* $\phi(n)$ *in* G'*. Then* $h(n)$ *is monotone for all* n *in* G*.*

Proof: [57, p. 116] Suppose $h(n)$ and $h(n')$ are the heuristics assigned to nodes n, n' in G ($n \neq n'$), respectively. These heuristics are minimum costs from the corresponding nodes $\phi(n), \phi(n')$ in G' to a goal $\phi(\gamma) \in \Gamma'$. Thus $h(n) \leq c'(\phi(n), \phi(n')) + h(n')$, where $c'(\phi(n), \phi(n'))$ is the relaxed optimal cost. Otherwise $c'(\phi(n), \phi(n')) + h(n')$, instead of $h(n)$, would constitute the optimal cost from $\phi(n)$ to $\phi(\gamma)$ when $\phi(n) \neq \phi(n')$. Monotonicity follows from Definition 2.10.1: $c(n, n') \geq c'(\phi(n), \phi(n'))$, and from the condition $h(\gamma) = 0$. $\square$

To satisfy the conditions of Definition 2.10.1, it suffices that n and m are *neighboring nodes* in G.

2.11 RELAXING ADMISSIBILITY

Sometimes the requirement of finding an admissible or monotone heuristic is too restrictive. In the following, we examine some extensions to A^* and prove properties that hold also when the heuristic is not admissible. The properties concern the quality of the solution (its cost) and the number of iterations of the algorithms. All the algorithms are special cases of the BF* algorithm in Section 2.8, hence their completeness is guaranteed by Theorem 2.8.2.

The material in this and the following section is not very closely related to the work in subsequent chapters. We will, however, refer to this material later.

Pohl [60] introduces an evaluation function:

$$f(n) = g(n) + h(n) + \epsilon \left(1 - \frac{d(n)}{d_s}\right) h(n), \tag{2.38}$$

where $d(n)$ is the depth of node n, d_s is the depth of the shallowest optimal goal node and ϵ is a parameter. Now, if $h(n) \leq h^*(n)$, then $h(\gamma) = 0$ for $\gamma \in \Gamma$ implying that $\psi(C) = C$ in Theorem 2.8.3. We can bound M, the minmax value related to f, along any optimal solution path P^* for which $g(n) = g^*(n)$:

$$\begin{aligned} M \leq \max_{n \in P^*} f_{P^*}(n) &\leq \max_{n \in P^*} \left(g^*(n) + h^*(n) + \epsilon h^*(n) \left(1 - \frac{d(n)}{d_s}\right)\right) \\ &= C^* + \epsilon h^*(s) = C^*(1+\epsilon). \end{aligned} \tag{2.39}$$

On the other hand, Theorem 2.8.3 states that the search terminates with cost $C_t \leq M$. Hence $C_t \leq C^*(1+\epsilon)$. This property was first shown by Pohl [60].

Harris [31] shows that the extra cost of the solution path found by A^* relates to the amount by which h^* is overestimated as follows. If $h(n) - h^*(n) \leq e$, $\forall n$ in G, then A^* produces a solution path with a cost $C_t \leq C^* + e$. Theorem 2.8.3 and the assumption $h(\gamma) = 0$ for $\gamma \in \Gamma$ implies that $\psi(C) = C$. We can now bound M by considering $\max_{n \in P} f_P(n)$ along any solution path for which $g(n) = g^*(n)$:

$$\begin{aligned} M &\leq \max_{n \in P} f_P(n) \leq \max_{n \in P} (g^*(n) + h(n)) \\ &\leq \max_{n \in P} (g^*(n) + h^*(n) + e) = C^* + e. \end{aligned} \tag{2.40}$$

Theorem 2.8.3 states that the search terminates with cost $C_t \leq M$. Hence $C_t \leq C^* + e$.

Pearl and Kim [56] generalize Harris' study and examine a search process where the uncertainty of the estimation of h^* is expressed in the form of a probability density function. The heuristic and thus the evaluation function $f = g + h$ are treated as random variables. We, however, do not go into details of this research.

Pearl and Kim [56] also introduce an algorithm called A^*_ϵ, which uses two sets: OPEN and FOCAL. FOCAL is a subset of OPEN containing nodes n:

$$\text{FOCAL} = \{n \mid f(n) \leq (1+\epsilon) \min_{n' \in \text{OPEN}} f(n')\}, \tag{2.41}$$

where $\epsilon > 0$ is a parameter. The operation of A^*_ϵ is identical to that of A^* except that A^*_ϵ selects the node from FOCAL with the lowest h_2 value, where $h_2(n)$ is a *second* heuristic estimating the computational effort required to complete the search starting from n. $h_2(n)$ can be almost any ranking function and it can be completely independent of h. However, if a goal node is in FOCAL, then A^*_ϵ always chooses it.

Following the previous analyses, if $h(\gamma) = 0$ for $\gamma \in \Gamma$, then $\psi(C) = C$. If h is admissible, then we can bound M along any optimal solution path P^*:

$$M \leq \max_{n \in P^*} f_{P^*}(n) \leq \max_{n \in P^*}(1+\epsilon) f_{P^*}(n) = (1+\epsilon)C^*. \tag{2.42}$$

Hence, by Theorem 2.8.3, $C_t \leq (1+\epsilon)C^*$.

2.12 MORE EXTENSIONS TO A^*

This section presents some extensions to A^*. This materal is not very closely related to the work in subsequent chapters and, thus, we only briefly introduce four algorithms and their properties. We will, however, refer to some of these algorithms later.

2.12.1 A^{**}

Here we introduce a variant of A^*, A^{**} [18], that dominates A^*. The dominance relation is, however, different from those discussed above.

A^{**} uses an evaluation function

$$f'(n) = \max\{g(n') + h(n') \mid n' \text{ on the current path to } n\}. \tag{2.43}$$

A^{**} chooses for expansion the node with the lowest f' value in OPEN (breaking ties arbitrarily, but in favor of goal nodes) and adjusts pointers along the path having the lowest g value. Despite the evaluation function f', A^{**} operates similarly as A^*. A^{**}, however, is not a best first algorithm (BF*), like A^* is, since it uses one function f' for ordering nodes for expansion and a different function g for redirecting pointers.

Clearly the f' values of the sequence of nodes expanded by A^{**} is nondecreasing. From this fact and the fact that A^{**} explores all the potential path candidates in turn, similarly as A^*, it is easy to see that A^{**} finds a least-cost path whenever $h(n) \leq h^*(n)\ \forall n$, that is, A^{**} is admissible relative to $\mathbf{I}_{AD}$. A detailed proof of the admissibility of A^{**} is very much alike that of A^*, see Theorem 2.3.3. Note that if the heuristic h is consistent (monotone), then A^{**} and A^* operate identically.

From the definition of f' it follows that all paths that are strictly bounded below C^* relative to f of A^* are also strictly bounded below C^* relative to f'. Therefore, both algorithms have exactly the same set of surely expanded nodes, $\mathbf{N}_f^{C^*} = \mathbf{N}_{f'}^{C^*}$, and this set is expanded before any node outside this set. Hence A^{**} is not optimal over A^* in the sense of Definition 2.9.1.

We just mention here that A^{**} dominates A^* in the following sense.

Theorem 2.12.1 *For every tie-breaking rule of A^* and for every problem instance $I \in \mathbf{I}_{AD}$, there exists a tie-breaking rule for A^{**} that expands a subset of the nodes expanded by A^*, cf. [18, p. 534–535].*

Since $\mathbf{N}_f^{C^*} = \mathbf{N}_{f'}^{C^*}$, the above theorem concerns only nodes n for which $f(n) = C^*$.

2.12.2 Algorithms making $O(|N|^2)$ Iterations at Worst

Above we were concerned with the sets of distinct nodes expanded by A^*. We saw that if the heuristic is not monotone (consistent), then A^* can expand the same node many times. Here we examine how many nodes A^* and some of its variants *totally* expand in the worst case, that is, the number of the *iterations* of the algorithm (for A^*, the iterations between lines (2)–(6) in its pseudocode in Section 2.1).

Martelli [52] denotes by $|N|$ the size of the graph G where $|N| < \infty$ is not the number of the nodes in G but the number of the nodes in the set $\bar{N} = \{n \mid f(n) \leq C^*\}$. A^* will expand nodes only in $\bar{N}$ but not necessarily all the nodes in $\bar{N}$ (Theorems 2.4.3 and 2.4.4). For example, A^* guided by a monotone heuristic never re-expands nodes. Hence it expands totally $O(|N|)$ nodes.

In [18, pp. 524–526], Dechter and Pearl show that if the heuristic is admissible and not monotone, then there is no optimal algorithm as measured by the number of distinct nodes expanded. Hence it is easy to believe that no optimal algorithm exists as measured by the number of nodes totally expanded. Mérõ [50] has proved this by constructing example graphs.

Martelli [52] proves the following theorem. Its proof is quite long and is omitted here.

Theorem 2.12.2 *[52] For all $|N|$ there exists a search graph G_N of size $|N|$, with positive costs and estimates which are lower bounds on the minimal cost ($h(n) \leq h^*(n)$ for each n in G_N), on which A^* runs for $O(2^{|N|})$ steps.*

Martelli [52] presents a modified A^* algorithm, called B, to improve the performance in Theorem 2.12.2 to a worst case performance of $O(|N|^2)$ in the case where h is admissible. Later, Bagchi and Mahanti [7] show that B

makes $O(|N|^2)$ iterations at worst also when the heuristic is nonadmissible. In the latter case, the size of the graph G is measured as the number of the nodes $|N| < \infty$ in a set $\bar{N}_M$: $\bar{N}_M = \{n \mid f(n) \leq M\}$, where M is the minmax value of BF* algorithms in Section 2.8. Moreover, B always returns a cheaper or at least as cheap a solution path as A^*.

Bagchi and Mahanti [7] also present another $O(|N|^2)$ algorithm called C that always, independently of the admissibility of h, returns a cheaper or at least as cheap a solution path as B.

Following the ideas of Mérõ [50], Mahanti and Ray [51] develop an algorithm called D that modifies a given (static) heuristic dynamically, during the search. D makes also $O(|N|^2)$ iterations at worst when the heuristic is admissible or nonadmissible. Moreover, D never expands more nodes than C. The cost of the solution path returned by D equals to that returned by C. D is a modification of A^* where line (3) and the calculation of the heuristic on line (5.1) in the pseudocode of A^* are replaced by the following lines:

```
(3'):Find nodes from OPEN with the smallest f value, select
     among them a node n whose g value is the smallest.
     (Resolve ties arbitrarily, but always in favor of
     any goal node). Place n in CLOSED.
```

(*Heuristic calculation on line 5.1*): If an expanded node n has no successors, then set $h(n) = \infty$. Let n' be the successor to n for which $e = c(n, n') + h(n')$ is minimal. If $e > h(n)$, then set $h(n) \leftarrow e$. Otherwise for each successor n_i to n, if

$$h(n_i) < h(n) - c(n, n_i) \tag{2.44}$$

holds, then set

$$h(n_i) \leftarrow h(n) - c(n, n_i), \tag{2.45}$$

where $c(n, n_i)$ is a strictly positive cost of the edge between n and n_i.

The above steps always produce an admissible (dynamic) heuristic if the original (static) heuristic is admissible. Mahanti and Ray [51] prove that in this case D finds an optimal path if it exists. If the original heuristic, however, is nonadmissible but vanishes at the goal nodes, then Mahanti and Ray prove that D always finds a cheaper or as cheap a solution path as A^*. However, D can expand more nodes than A^*.

2.12.3 On Function Minimization by A^*

In this section, we discuss how A^* could be used to minimize positive real valued functions. The minimization problem can be constrained or unconstrained. Now, the search is done in the space of complete solutions. The "goodness" of a solution is measured by using the values of the function to be minimized. Here we believe that short paths, lists of feasible solutions, correlate with the effectiveness of the search whose goal is to find a solution near to the minimum value of the function. The heuristic to be introduced in Section 3.1 can be used as part of the guiding heuristic of the search. In addition, similar information that is frequently used in Branch-and-Bound methods can be utilized.[3]

Let the function to be minimized be $F : X \to \mathcal{R}_0^+$, where X is a set and $\mathcal{R}_0^+$ is the set of positive real numbers. The metric d between any two points n and m in X is defined:

$$d(n, m) = |F(n) - F(m)|. \tag{2.46}$$

If X is originally a *real metric space*, then d often differs from the original metric, usually defined as a distance between n and m.

Let us discretize X so that the result is a locally finite graph G. Nodes in G are (discretized) points in X and edges in G connect the nodes that are immediate neighbors. The cost of an edge between the neighboring nodes n and m in G is:

$$c(n, m) = \begin{cases} d(n, m) & \text{if } d(n, m) \geq \epsilon, \\ \epsilon & \text{otherwise,} \end{cases} \tag{2.47}$$

where $\epsilon > 0$ is a small positive constant. If $d(n, m) < \epsilon$, then we assume that the difference between $F(n)$ and $F(m)$ cannot any more be recognized by the computer. In the latter case, we set the cost $c(n, m) \leftarrow \epsilon > 0$. In this way, we have defined a locally finite graph whose edge costs are strictly positive. Hence, we can use the A^* algorithm, its variants or extensions to find paths in G.

Let us assume that we know a lower bound $\tilde{F}$ for the global minimum of F, say F^*, ($\tilde{F} \leq F^*$). Our task is to find a solution with a value close enough to F^*, say F_G^* ($F_G^* \geq F^*$), in the discretized space (in the graph G) effectively by using a start point $s \in G$. Here we are not concerned how a good starting point(s) is found. Let us use A^* to find F_G^*. Now, the path finding problem is defined in the space of *complete* solutions. Let us use the following heuristic h for A^*:

$$h(n) = F(n) - \tilde{F}. \tag{2.48}$$

If $\tilde{F} < F^* \leq F_G^*$, then $h(F_G^*) > 0$. Does h satisfy the triangle inequality among the nodes in G? The answer is affirmative.

Theorem 2.12.3 *The heuristic $h(n) = F(n) - \tilde{F}$ satisfies the triangle inequality $h(n) \leq k(n, n_1) + h(n_1)$ for all pairs of nodes n and n_1 in G, where n_1 is a descendant of n.*

Proof: Let n' be a successor to n in G. Then

$$\begin{aligned} h(n) &= F(n) - \tilde{F} = F(n) - F(n') + F(n') - \tilde{F} \\ &\leq |F(n) - F(n')| + h(n') \leq c(n, n') + h(n'). \end{aligned} \tag{2.49}$$

A proof by induction shows that also $h(n) \leq k(n, n_1) + h(n_1)$ for all pairs of nodes n and n_1 in G, cf. Theorem 2.7.1. □

[3] The material in Sections 2.12.3 and 2.12.4 appears for the first time in this thesis.

Let the global minimum of $F(t)$ be $F^*(t^*)$. Then $h(t^*) = F^*(t^*) - \tilde{F} \geq 0$. If the heuristic does not vanish at the goal it still satisfies the triangle inequality and hence is a $h_\triangleleft$ heuristic, to be defined in the Section 3.1.

We can use A^* with the above heuristic to search for a solution whose value is less or equal to F^*_G.[4] A^* finds shortest paths to all expanded nodes and no re-expansions need to be done by Theorem 3.1.1. Also the other results in Chapter 3 can be utilized if needed. If we have to find a *global* minimum of F in G by using only the above heuristic, then we, unfortunately, have to expand all the nodes in G at worst.

If we have a constrained optimization task, then A^* is allowed to expand only *feasible* nodes, that is, the ones that satisfy the constraints. Of course, we assume that a feasible start node can be found, too.

In addition to the above $h_\triangleleft$, we may have other information available. For example, we may be able to calculate *lower bounds* for some nodes n in G. A lower bound C_n is the lowest F value we can possibly attain when we continue our search from n. Lower bounds are usually obtained by solving simplified, easily solvable, versions of the original problem and are commonly used in Branch-and-Bound methods, see e.g. [45]. We can take the lower bounds into account by maintaining the lowest *upper bound* $\bar{F}$ on the minimum of F obtained so far, and set $h_\triangleleft(n) = \infty$ if $C_n > \bar{F}$. If we can obtain the lower bounds, then we may find the global optimum of F in G without expanding all the nodes in G.

The lower bounds can also be taken into the tie-breaking rule of the A^*: *Among all the nodes with the same value of the evaluation function f, select and expand first a node n for which the lower bound C_n is minimum.* The following explains the advantage of using the above tie-breaking rule. Let there be any two nodes n and m with the same f value and let $C_m < C_n$. If we first expand m, then it can happen that there are successors w to m for which $C_m \leq F(w) < C_n$ and we can set $h(n) = \infty$ and thus prune the search space. Otherwise, if we first expand n, then it never happens that any successor v to n has $F(v) \leq C_m$ by the definition of the lower bounds: $F(v) \geq C_n > C_m$ for all v.

If we can obtain lower bounds, then we can also use as the heuristic $h(n) = F(n) - \min\{C_k\}$, where the minimum is taken over all the nodes in OPEN every time a new node enters OPEN. If a lower bound for a node m is not known, then set $C_m = F(m)$. The above h values are "dynamic" in the sense that when a new lower bound is obtained, then the h values for all the nodes in G are recalculated. It is easy to see that the above h also satisfies the triangle inequality among the nodes in G at all times during the search by looking at the proof of Theorem 2.12.3. The above h can be called a "dynamic" or an "adaptive" version of the heuristic in Theorem 2.12.3.

We could, of course, search the minimum of F by an algorithm that always selects and expands a node n for which C_n is minimum regardless of its f value. This algorithm is greedy in the sense that it does not penalize long

[4]The exact value F^*_G is not assumed to be given but, instead, the user is supposed to stop the search when she or he is satisfied with the goodness of the solution.

paths from the start node as A^* does. We could also combine several optimization algorithms similarly as proposed in [3], or in Chapter 5.

2.12.4 Regulating the Number of Node Expansions

In the context of A^*_ϵ in Section 2.11 ([56]), we saw the use of two different heuristics h and h_2 in the same algorithm returning a solution path of length $C_t \leq (1+\epsilon)C^*$. Here we present an algorithm, also using two heuristics, that expands at most $(1+\beta)E_X$ nodes, where E_X is the number of nodes expanded by algorithm X. The parameter β is a given by the user. X can be, for example, the A^* algorithm.

Consider the following algorithm. Let X and Y be any BF* algorithms guided by heuristics h_X and h_Y, respectively. For simplicity, let us assume that both X and Y use a common OPEN and CLOSED sets.

ALGORITHM X_β

```
(1) REPEAT
(2)     FOR a while DO
(3)         Run algorithm X with heuristic hX.
(4)     END FOR
(5)     Set E = the number of expanded nodes that have not
        already been in CLOSED, in the above FOR-loop.
(6)     Expand Beta*E nodes not already in CLOSED by
        another algorithm Y with heuristic hY.
(7) UNTIL a goal node is found
```

On lines (3) and (6), if a goal node is found, then exit immediately without continuing the execution of the search algorithms any more. On line (2), the "FOR a while DO" -sentence means that the user can specify how long algorithm X runs; the number of the nodes to be expanded can be used as a criterion, but also other criteria may be considered. Algorithms X and Y can re-expand nodes that previously were in CLOSED but the number of re-expansions is not recorded here.

An interpretation of X_β is the following. Y is in a role of a "probe" that tries to find a goal, possibly located nearby, in some easy "clever" or perhaps greedy way. X on the other hand is a "base" algorithm, for example, A^* that "does not leave any stone unturned".

Theorem 2.12.4 *Algorithm X_β expands at most $(1+\beta)E_X$ nodes, where E_X is the number of nodes expanded by algorithm X.*

Proof: Assume that X has expanded E_i new nodes during iteration i of the REPEAT–UNTIL -loop ($i = 1, 2, \ldots$). The fact that X_β has expanded at worst $(1+\beta)E_X$ nodes after, say k, iterations of the REPEAT–UNTIL -loop can easily be seen in the following summation:

$$\sum_{i=1}^{k}(1+\beta)E_i = (1+\beta)E_X. \tag{2.50}$$

$\square$

Clearly, algorithm X_β can also be designed with any two search algorithms, not necessarily BF* algorithms.

In robot point-to-point path planning, for example, Y can be an algorithm that generates partial paths towards a negative gradient of an artificial potential field generated in the robot's configuration space, see Chapter 6.

2.13 DISCUSSION

Many variants, modifications, and extensions to A^* have been published over the years. In this chapter, we have already seen A^{**} (Dechter and Pearl [18]), a class of generalized best-first algorithms BF^* (Dechter and Pearl [18]), algorithms improving the worst case behavior of A^* such as B (Martelli [52]), C (Bagchi and Mahanti [7]), and D (Mahanti and Ray [51]), a dynamic weighting of the evaluation function (Pohl [60]), a heuristic with bounded error (Harris [31]), and A^*_ϵ (Pearl and Kim [56]). Other works include, for example, iterative-deepening-A^* IDA^* (Korf [40]), restricted memory algorithms such as $MREC$ (Sen and Baghi [66]), MA^* (Chakrabarti et al. [10]), SMA^* (Russel [65]), $RFBS$ (Korf [42]), and algorithms for dynamic environments such as D^* (Stentz [70]) — not to mention the works dealing with bidirectional search methods.

Farreny [28] proposes a formal generalization for various works with heuristic search in state space graphs. He identifies five dimensions for generalizations: (1) The notion of length to measure the paths between nodes, (2) the characteristics of the state graphs dealt with, (3) the choices of the nodes to expand, (4) the kinds of updating rules to realize, and (5) the properties of the evaluation functions that guide the search. Consequently, Farraney discusses algorithm families, which include many successors of A^* such as B, A^*_ϵ, C, D, BF^*, IDA^*, and A^{**}. He employs this formalization to present general theorems about the completeness and the admissibility/sub-admissibility of the algorithms.

In Chapters 3–7, we will also present some extensions to A^* and their applications. (A) We study nonadmissible heuristics, present a dynamically improving heuristic for A^*, and use A^* to control the allocation of computing resources among search algorithms. (B) We apply A^* to robot point-to-point path planning and power-aware routing of messages in wireless communication networks.

3 COMPARISONS OF A^* TO OTHER PATH PLANNING STRATEGIES

Let us define more precisely what we mean by a path finding problem and a path finding algorithm.

Let $G = (V, E)$ be a directed or undirected locally finite graph, where V is a set of nodes and E is a set of edges between the nodes. Let every edge in E have an associated strictly positive real number called *edge cost*. Let $s \in V$ denote the *start node* and $\Gamma \subseteq V$ the *set of goal nodes*. A *heuristic function* $h(n)$ attaches a nonnegative real number to every node n in V. The heuristic $h(n)$ estimates a path cost from a node n in G to a goal node $\gamma \in \Gamma$.

An *instance of path finding problems* is denoted by

$$I = (C_I, G, s, \Gamma, h), \tag{3.1}$$

where C_I includes *context information* that is not explicitly present in the structure of G or in h. For example, C_I could include details of the application area of I, etc. We assume that all the data of I (as well as C_I) is given at the same time as input to a *path finding algorithm*. Moreover, we assume that the possibly infinite graph G is not explicitly coded but is provided, for example, by giving rules to generate successors to a node at hand.

The nodes in G being searched are at any time divided into four sets, cf. [57, p. 34]:

- Nodes that have been *expanded*.
- Nodes that have been *explored*.
- Nodes that have been *generated* but not yet explored or expanded.
- Nodes that have not yet been generated.

Node generation means computing the representation code of a node from that of its parent. The new successor node is then said to be generated and its parent explored or expanded. *Node expansion* consists of generating *all* the immediate successor nodes for a given parent node and no more. *Node exploration* refers to gathering or dismissing information concerning the neighborhood structure of a given node. An example of node exploration is generating at least one successor node to a given node. Also, a connected graph of nearby nodes can be generated "starting" from a given node. If a node is expanded, then it is explored too, but not vice versa.

Usually, nodes that have been expanded are called *closed*, and nodes that have been generated and are awaiting expansion or exploration are called *open*. Furthermore, a path finding algorithm does not know anything about the nodes that are not yet generated unless the context information C_I gives such a knowledge.

A *search procedure* or a *path finding algorithm* is a prescription for determining the order in which nodes are to be generated. We allow that the order is partial so that many nodes can be generated concurrently, cf. [57, p. 34]. From now on, we assume that a search procedure satisfies the following constraints:

1. *It is deterministic.*

2. *It solves one problem instance at a time.*

3. *It uses the computational steps of node exploration and node generation.*

4. *It explores only nodes that it has generated in the same problem instance.*

5. *It begins the search from the start node of a problem instance. It stops when it has explored or generated a goal node in the same instance or has no new candidates for exploration.*

For example, bidirectional searches or algorithms that start searching and/or search from several "seed" nodes are forbidden by Constraints 4 and 5. As a second example, any BF* algorithm defined in Section 2.8 (including A^*) clearly satisfies the above constraints. In particular, BF* algorithms always expand one node at a time. Furthermore, greedy algorithms can generate a subset of the successor nodes for a given parent node.

Constraints 1–5 do not imply that a search procedure must explore nodes one after another; it can explore a set of nodes concurrently. However by Constraint 4, all the explored nodes have to be already generated. Hence algorithms satisfying Constraints 1–5 are not necessarily GBF strategies as discussed in Section 2.8 ([18, p. 506]).

Let a *search history* of a path finding algorithm A satisfying Constraints 1–5 be a set $\mathbf{I}_A$ of all problem instances $I = (C_I, G, s, \Gamma, h)$ that A has solved previously. There can be infinitely many previous instances in $\mathbf{I}_A$ and the search graph G of I can have infinitely many nodes. Hence, A can have an infinite memory.

Now, let G be a graph of an instance I currently being searched by A. Let us define the *current search history concerning* I as the graph $H = (N_H, E_H)$, where N_H is composed of all the nodes explored or generated by A in G before a given "time", and E_H contains a subset of edges between the above nodes such that H is *connected*. Moreover, any edge $e \in E_H$ has an associated edge cost that is the same as $e \in E$ of G has, and any node $n \in N_H$ has an associated nonnegative heuristic value that is the same as $n \in V$ of G has. In a case of A^*, for example, the graph H is a tree, where edges connect any expanded node n to its successors in $\mathrm{succ}(n)$ such that the edges are directed from the nodes in $\mathrm{succ}(n)$ to n.

In the following sections, we will compare the performance of A^* to that of other path finding algorithms. The proofs of several theorems use the following strategy, already presented in the proof of Theorem 2.9.1. A problem instance I is assumed, where A^* expands a node n and another algorithm, say A, does not explore n. Then a new instance $I' \neq I$ is constructed from I such that A^* also expands n and A does not explore n in I'. This is possible if $C_I = C'_I$, that is, the context information of I and I' is the same. In the following, we will assume the above.

The search history $\mathbf{I}_A$ from problem instances previously solved by A *is not of any use* to A in the proofs of the following theorems. This is because I' will be constructed in such a way that A has "at most" generated n both in I and in I' and hence does not know anything about the successor nodes of n in I or in I' (Constraint 4). The conclusions of the theorems are based on what happens to the successors of n in I' (A^* will expand those nodes). Thus, we will not explicitly write the search history $\mathbf{I}_A$ anymore although the computations of the following algorithms may depend on it in principle.

Constraint 1 implies that there is a function that completely determines which set of nodes is to be explored next. The nodes in that set are assumed to be explored concurrently. Let us call the above process the *exploration function* Φ. Φ is the core of a path finding algorithm satisfying Constraints 1–5: It selects the set of generated nodes $S \subseteq N_H$ to be explored next and explores them. Nodes in S have to be generated before their exploration by Constraint 4. Let us write abstractly:

$$\Phi : ((N_H, E_H), I) \mapsto (N_{H'}, E_{H'}), \tag{3.2}$$

where both $H = (N_H, E_H)$ and $H' = (N_{H'}, E_{H'})$ are connected and $N_H \subseteq N_{H'}$. When the search starts, $H = (\{s\}, \emptyset)$ (Constraint 5). Note that H is not necessarily a subgraph of H' since E_H can contain edges that are not in $E_{H'}$, see for example line (5.2) in the pseudocode of A^* in Section 2.1.

Let us call the set of path finding algorithms satisfying Constraints 1–5 and having an exploration function Φ *deterministic path finding strategies* (DP). In general, we do not require the DP strategies to be *complete*, that is, to find a solution path whenever one exists. [1]

A DP strategy can have an *evaluation function*

$$f(n) = f(n; H, I), \tag{3.3}$$

which attaches a nonnegative real value to any node $n \in N_H$ it has generated. Let $S \subseteq N_H$, as above, denote the generated nodes that are awaiting exploration. We do not specify here how the nodes in S are selected, except that the selection process is deterministic (Constraint 1). S can also contain nodes that have been explored (or expanded) before, see line (5.3) in the pseudocode of A^* in Section 2.1. A DP strategy that always explores nodes $n \in S$ with the minimum $f(n)$ value is called a *deterministic best-first path finding strategy* (DPBF).

DP and DPBF strategies explore nodes, that is, they do not have to generate *all* the successors for the selected node. In that sense, DP and DPBF strategies are more general than the algorithms studied by Farreny [28]. On the

[1] Farreny [28] studies complete algorithms.

other hand, Farreny uses a more general definition of the path length than the sum of edge costs.

A DPBF strategy can search G by constructing a tree T of selected path candidates in G by always expanding a leaf node of T with a minimal f value. The algorithm thus expands one node at a time and maintains T. Such search methods are similar to the GBF strategies in Section 2.8. The evaluation function of the A^* algorithm is: $f_{A^*}(n) = g(n) + h(n)$ for any generated node n. Thus f_{A^*} depends *only* on the most recently found path to n and the heuristic value $h(n)$. A^* is an example of BF* algorithms, and a BF* algorithm is an example of GBF and DPBF strategies.

If there are many nodes awaiting exploration with the same minimum f value and we do not want to explore them concurrently, then a so called *tie-breaking rule*, coded in the search procedure, determines the node to be explored next.[2]

In this chapter, we compare the performance of A^* guided by a possibly nonadmissible heuristic to that of some other path finding algorithms.

3.1 HEURISTICS SATISFYING THE TRIANGLE INEQUALITY

Here we introduce generalizations of consistent heuristics that are not necessarily admissible. The A^* algorithm guided by such a heuristic always finds optimal paths to all the nodes it expands, in spite of the nonadmissibility of the heuristic.

In Section 2.7, we defined and discussed heuristics that satisfy the triangle inequality, see Definitions 2.7.1 and 2.7.2 (consistent and monotone heuristics), where:

$$h(n) \leq k(n, n') + h(n') \tag{3.4}$$

for all pairs of nodes n and n', where n' is a descendant of n. The cost of an optimal path from n to n' is denoted by $k(n, n')$.

In Definitions 2.7.1 and 2.7.2, it is implicitly assumed that h vanishes at the goal nodes ($h(\gamma) = 0, \forall\gamma \in \Gamma$). Here we introduce a heuristic $h_\triangleleft$ that *only* satisfies the triangle inequality, *but is not necessarily zero* at the goal nodes, that is, $h_\triangleleft(\gamma) > 0$ for some $\gamma \in \Gamma$. It follows that $h_\triangleleft$ can be nonadmissible. In addition, we assume that $h_\triangleleft$ satisfies Eq. (3.4) for *any* pair of nodes n and n'. Next, we will show that some properties concerning A^* with (admissible) monotone or consistent heuristics remain the same even if A^* uses the possibly nonadmissible $h_\triangleleft$.

Do nonadmissible heuristics satisfying the triangle inequality exist? Let us give an example. We do not, however, argue how good or bad this heuristic would be in practice.

Assume that there are many goal nodes in Γ and we can easily calculate the distance $d(n, \cdot)$ between any node n and only a *subset* Γ_d of the goal nodes in Γ. Hence, we do not know which goal node in Γ is closest to the

[2] We could have included the tie-breaking rule in f as well but we will follow the convention used in the literature.

start node. Let us choose one node γ_d from Γ_d and use it for the calculation of the distance function $d(n, \gamma_d)$ ($\forall n \in G$) and use d as the heuristic. The function $d(n, \gamma_d)$, since it is a distance function, satisfies the triangle inequality. However, we do not know whether γ_d is closest to the start node, hence we can have a nonadmissible $h_{\triangleleft}(n) = d(n, \gamma_d)$. Clearly, every goal node γ_c that is closer to the start node than γ_d, as measured by d, satisfies $h_{\triangleleft}(\gamma_c) = d(\gamma_c, \gamma_d) > 0$.

Theorem 3.1.1 *An A^* guided by $h_{\triangleleft}$ finds optimal paths to all expanded nodes (cf. Theorem 2.7.3 and Theorem 10 in [57, pp. 83–84]).*

Proof: The proof is exactly the same as that of Theorem 2.7.3 (Theorem 10 in [57, pp. 83–84]). The proof needs Lemma 2.3.2 guaranteeing that the shallowest OPEN node n in an optimal path has $g(n) = g^*(n)$. Neither the proof of Theorem 2.7.3 nor Lemma 2.3.2 uses the fact that the heuristic should vanish at the goal nodes. □

It follows from Theorem 3.1.1 that A^* using $h_{\triangleleft}$ always finds an optimal solution to a goal node if one exists. If there are several goal nodes, then A^* halts at the first goal γ it finds. However, the path to γ may not be the globally optimal one since the heuristic may be nonadmissible. In general, A^* has to find all the goal nodes before deciding on which solution path is the globally optimal one. However, no node is expanded twice, hence A^* using $h_{\triangleleft}$ makes at worst $O(|N|)$ iterations, see Section 2.12.2.

Similarly, by looking at the proof of Theorem 2.7.4 (Theorem 11 in [57, p. 84]), we notice that it does not use the fact $h(\gamma) = 0$ either. Hence *the f values of the sequence of nodes expanded by A^* using $h_{\triangleleft}$ are non-decreasing.*

Recall that $\mathbf{I}_{AD}$ denotes the set of problem instances, where the heuristic h is admissible as defined in Section 2.9. Furthermore, let us denote by $\mathbf{I}_{\triangleleft} = \{(G, s, \Gamma, h_{\triangleleft})\}$ the set of problem instances with the above heuristic $h_{\triangleleft}$. Obviously $\mathbf{I}_{CON} \subseteq \mathbf{I}_{\triangleleft}$, where $\mathbf{I}_{CON}$ was also defined in Section 2.9.

Let a DPBF strategy B be guided by a heuristic h. Moreover, assume that the evaluation function of B satisfies $\forall m_1, m_2 \in G$ of I:

$$h(m_1) > h(m_2) \Rightarrow f_B(m_1; H, I(\cdot, h)) > f_B(m_2; H', I(\cdot, h)) \qquad (3.5)$$

independent of the current search histories H and H' (concerning $I = (G, s, \Gamma, h)$). The above requirement characterizes greedy or depth-first algorithms, where distances to goals are emphasized. Finally, assume that B can find an optimal solution if $h(n) \leq h^*(n)\ \forall n$, that is, in all the problem instances in $\mathbf{I}_{AD}$: $B \in \mathbf{A}_{ad}$, where $\mathbf{A}_{ad}$ is here assumed to contain all admissible DP strategies equally informed as A^*, cf. Section 2.9.

Let the set of nodes surely expanded by A^* be denoted by $\mathbf{N}^M_{g+h}$, where M is the minmax value related to the evaluation function $f = g + h$ of A^*, defined in Section 2.8. In other words, the nodes n in $\mathbf{N}^M_{g+h}$ are reachable by a strictly M-bounded path: $f(n) < M$, see also Theorem 2.8.6.

Now, we are ready to prove the following theorem that is a counterpart of Theorem 2.9.1 (cf. Theorem 8 [18, pp. 522-524] that holds for any DP strategy $B \in \mathbf{A}_{ad}$).

Theorem 3.1.2 *Let $\Delta = \max_{\gamma \in \Gamma}\{h_\lhd(\gamma)\}$. Then any DPBF strategy B that is admissible on $\mathbf{I}_{AD}$, i.e., $B \in \mathbf{A}_{ad}$ will explore, in every instance $I_\lhd \in \mathbf{I}_\lhd$, all the nodes in $\mathbf{N}^M_{g+h} \cap \{n \mid h_\lhd(n) \geq \Delta\}$ provided that $\forall m_1, m_2 \in G$ of any $I \in \mathbf{I}_\lhd \cup \mathbf{I}_{CON}$:*

$$h(m_1) > h(m_2) \Rightarrow f_B(m_1; H, I(\cdot, h)) > f_B(m_2; H', I(\cdot, h)) \tag{3.6}$$

independent of the current search histories H and H' concerning I.

Proof: Let $I_\lhd = (G, s, \Gamma, h_\lhd)$ be some problem instance in $\mathbf{I}_\lhd$. If $\Delta = 0$, then $I_\lhd \in \mathbf{I}_{CON}$, and by Theorem 2.9.1 we are done. Assume that $\Delta > 0$.

Assume that A^* expands a node n in $\mathbf{N}^M_{g+h} \cap \{n \mid h_\lhd(n) \geq \Delta\}$ of $I_\lhd$. Moreover, assume that B does not explore n.

Let us create a new graph G_1. G_1 is the same as G except two modifications. First, the heuristic values of the nodes are changed:

$$h_1(m) = \max\{0, h_\lhd(m) - \Delta\} \tag{3.7}$$

for all the nodes in G_1 except n. Set $h_1(n) = h_\lhd(n) - \Delta + \epsilon > 0$, where

$$0 < \epsilon < M - g^*(n) - h_\lhd(n), \tag{3.8}$$

where M is the minmax value for A^* on G, and $\epsilon > 0$ by Theorem 2.8.6. The value $h_1(n) > 0$ by the assumption of the theorem: $h_\lhd(n) - \Delta \geq 0$. The second modification is that all the edges between n and its neighbors n' in G_1 are directed from n' to n.

Now, $h_1(\gamma) = 0$ for any goal node $\gamma \in G_1$ because of the first modification. The heuristic h_1 is monotone (consistent) since it is a maximum of two monotone heuristics 0 and $h_\lhd - \Delta$ (Theorem 2.7.6) for all nodes in G_1 except n. If n' is a neighbor of n, then $h_1(n)$ and $h_1(n')$ satisfy the triangle inequality because of the second modification. Denote the new problem instance by $I_1 = (G_1, s, \Gamma, h_1)$. It follows that $I_1 \in \mathbf{I}_{CON}$.

Since A^* expands n in G: $f(n) = g^*(n) + h_\lhd(n) < M$ by Theorems 2.8.6 and 3.1.1. From the construction of h_1, it follows that the minmax value M_1 for A^* in G_1 satisfies $M \geq M_1 \geq M - \Delta$. A^* will expand n also in G_1 since $f_1(n) = g^*(n) + h_\lhd(n) - \Delta + \epsilon < M - \Delta$ because of the definition of $\epsilon > 0$ and by Theorem 2.7.3.

Since B does not explore n in G, it follows that $f_B(m; H, I(\cdot, h_\lhd)) \leq f_B(n; H', I(\cdot, h_\lhd))$ and hence $h_\lhd(m) \leq h_\lhd(n)$ for all nodes m that B explores in G.[3] Then it follows that $h_1(m) < h_1(n)$ and thus

$$f_B(m; \cdot, \max\{0, h_\lhd(m) - \Delta\}) < f_B(n; \cdot, h_\lhd(n) - \Delta + \epsilon) \tag{3.9}$$

and B, being deterministic, will explore all the nodes m in G_1 without exploring n if I_1 is given as input to B.

Now, since $I_1 \in \mathbf{I}_{CON}$ we can utilize the proof of Theorem 2.9.1 directly to create an instance $I' \in \mathbf{I}_{CON}$, where A^* will find an optimal solution path whereas B will not if I' is given as input to B. This contradicts the assumption that $B \in \mathbf{A}_{ad}$ and avoids the exploration of node n. □

[3] Here and in subsequent theorems, we assume that the value $f_B(n; \cdot)$ exists despite that B may not yet have generated n.

The above theorem states that greedy DPBF strategies always explore the nodes that A^* surely expands in regions of graphs, where the heuristic is not small. Usually, the heuristic is small in the vicinity of goal nodes. If $h(\gamma)$ is close to zero for every $\gamma \in \Gamma$, then the set $\mathbf{N}_{g+h}^{M} \cap \{n \mid h(n) \geq \Delta\}$ may be close to $\mathbf{N}_{g+h}^{M}$. The above theorem is not as strong as Theorem 2.9.1 (Theorem 8 [18, pp. 522-524]). There seems to be a price to pay when we allow nonadmissible heuristics to exist in the proofs.

3.2 OTHER NONADMISSIBLE HEURISTICS

To what kind of algorithms and how can the performance of A^* guided by any possibly nonadmissible heuristic be compared? We give an answer here. Let

$$\mathbf{I} = \{(G, s, \Gamma, h)\} \tag{3.10}$$

be a set of problem instances, where the heuristic h may be nonadmissible.

Let the A^* algorithm solve problems in $\mathbf{I}$. A^* is an example of a BF* algorithm and hence is a DP strategy. Let another DP strategy B also solve problems in $\mathbf{I}$; B needs not to be a BF* algorithm, or a DPBF strategy. Now, assume that B can find an optimal solution path whenever A^* does, that is, B is globally compatible with A^*: $B \in \mathbf{A}_{gc}$, where $\mathbf{A}_{gc}$ is here assumed to contain all DP strategies equally informed as A^* and globally compatible with A^*, cf. also Section 2.9.

To prove new theorems, we could apply the same strategy as was done in Theorem 3.1.2 by defining a new, here admissible, heuristic $h_{AD}(r) = \max\{0, h(r) - \Delta\}$, where $\Delta = \max_{m \in G}\{h(m) - h^*(m)\}$. We will prove Theorem 5.4.2 in Chapter 5 by using the above.

However, we will choose a simpler strategy in proving the following theorems. The next theorem can be seen as a counterpart of Theorem 2.9.3 (cf. also Theorem 11 in [18, p. 528] that also holds for any DP strategy $B \in \mathbf{A}_{gc}$). The proof is a modification of the proof of Theorem 11 in [18].

Let the set of nodes surely expanded by A^* be denoted by $\mathbf{N}_{g+h}^{M}$, as before. Let C^* be the cost of an optimal path and let $g_{P_{s-n}}(n)$ denote the cost of the path P_{s-n}, from the start node s to n.

Theorem 3.2.1 *Let $\mathbf{I} = \{(G, s, \Gamma, h)\}$. Then any DP strategy $B \in \mathbf{A}_{gc}$ will explore, in every instance $I \in \mathbf{I}$, all the nodes in $\mathbf{N}_{g+h}^{M} \cap \{n \mid g_{P_{s-n}}(n) < C^*\}$, where $g_{P_{s-n}}(n)$ is the cost of the path P_{s-n} in G found by A^*.*

Proof: Let $I = (G, s, \Gamma, h)$ be a problem instance in $\mathbf{I}$. Let a node n in G, expanded by A^*, be in $\mathbf{N}_{g+h}^{M} \cap \{n \mid g_{P_{s-n}}(n) < C^*\}$ and not explored by B. Obviously $C^* \leq M$, the minmax value for A^* in G.

Let us create a new graph G', as in Figure 3.1, by adding to G a goal node t with $h(t) = 0$ and an edge from n to t with nonnegative cost $0 < c(n, t) < C^* - g_{P_{s-n}}(n)$. The graph G' has an extended path P_{s-t}^* with cost $C(P_{s-t}^*) = g_{P_{s-n}}(n) + c(n, t) < C^*$. Let $I' = (G', s, \Gamma \cup t, h)$ be a new problem instance.

Algorithm A^* searching G' will find t since $C(P_{s-t}^*) < C^* \leq M$ by Theorem 2.8.6 and, thus, finds the new optimal path P_{s-t}^*. Algorithm B, if given

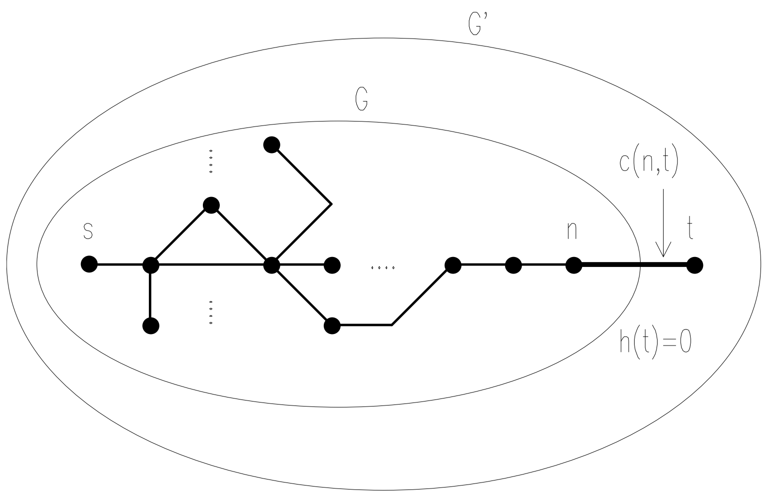

Figure 3.1: The graph G' is constructed from G by including a new solution path going via n to the new goal node t.

I' as input, will search I' in exactly the same way as it would have searched I since B is deterministic; the only way B can reveal any difference between I' and I is by exploring n. Since it does not, it will not find the solution path P^*_{s-t}. This contradicts its property of being in $\mathbf{A}_{gc}$. □

If the heuristic h is admissible, then $M = C^*$ and the above theorem is the same as Theorem 2.9.3 (Theorem 11 in [18, p. 528]), provided that B expands nodes. If h can be nonadmissible in some nodes, then B does not necessarily explore all the nodes surely expanded by A^*.

The next corollary follows directly from the above theorem.

Corollary 3.2.2 *Let $\mathbf{I} = \{(G, s, \Gamma, h)\}$. Suppose A^* finds an optimal path P^* with cost C^* such that*

$$\max_{n' \in P^*} \{g(n') + h(n')\} < C^*, \tag{3.11}$$

except at the goal node. In these problem instances $I \in \mathbf{I}$, any DP strategy $B \in \mathbf{A}_{gc}$ will explore all the nodes surely expanded by A^*.

The following theorem concerns DP strategies in a subclass of $\mathbf{A}_{gc}$, and it is a variant of Theorem 3.2.1.

Theorem 3.2.3 *Let $\mathbf{I} = \{(G, s, \Gamma, h)\}$. Assume that D is a DP strategy and always finds at least as cheap a solution path as A^* does. Then D will explore, in every instance $I \in \mathbf{I}$, all the nodes in $\mathbf{N}^M_{g+h} \cap \{n \mid g_{P_{s-n}}(n) < C_D\}$, where C_D is the cost of the solution path in G found by D, and $g_{P_{s-n}}(n)$ is the cost of the path P_{s-n} in G found by A^*.*

Proof: Let $I = (G, s, \Gamma, h)$ be a problem instance in $\mathbf{I}$. Let a node n in G, expanded by A^*, be in $\mathbf{N}^M_{g+h} \cap \{n \mid g_{P_{s-n}}(n) < C_D\}$ and not explored by D. Clearly $C_D \leq M$ since the cost of the path found by A^* is at least as high as C_D.

Let us create a new graph G', as in Figure 3.1, by adding to G a goal node t with $h(t) = 0$ and an edge from n to t with nonnegative cost $0 < c(n,t) < C_D - g_{P_{s-n}}(n)$. The graph G' has an extended path P'_{s-t} with cost $C(P'_{s-t}) = g_{P'_{s-n}}(n) + c(n,t) < C_D$. Let $I' = (G', s, \Gamma \cup t, h)$ be a new problem instance.

Algorithm A^* searching G' will find t since $C(P'_{s-t}) < C_D \leq M$ and, thus, finds the new path P'_{s-t}. Algorithm D, if given I' as input, will search I' in exactly the same way as it would have searched I since D is deterministic; the only way D can reveal any difference between I' and I is by exploring n. Since it does not, it will not find the solution path P'_{s-t}, but will halt with cost C_D. This contradicts its property of always finding at least as cheap a solution path as A^* does. $\square$

In Section 2.12.2 we shortly presented examples of algorithms like the above DP strategy D (one of the algorithms was, somewhat misleadingly, also called D).

The next corollary follows from the above Theorems 3.2.1 and 3.2.3.

Corollary 3.2.4 *Let B and D be the same algorithms as in Theorems 3.2.1 and 3.2.3. Then*

$$\mathbf{N}^M_{g+h} \cap \{n \mid g_{P_{s-n}}(n) < C^*\} \subseteq \mathbf{N}^M_{g+h} \cap \{n \mid g_{P_{s-n}}(n) < C_D\}. \qquad (3.12)$$

The above corollary states that D will explore all the nodes that B *"at least"* explores in the same problem instance. This is because $C^* \leq C_D$. The corollary is somehow in accordance with the principle that in order to get solutions with better quality (cheaper costs) one has to pay something extra.

Let us modify Theorem 3.2.1. To do it, we alter the search graph G. The edges of G were assumed to have strictly positive costs.

Let us look at Theorem 2.2.2 (Theorem 1 in [57, p. 77]). Its proof does not require that the edge costs of G are strictly positive. It is enough to demand that the cost of every infinite path in G is unbounded. A similar condition if imposed on any best-first strategy would render it complete [57, p. 77]. In other words, some edge costs in a locally finite graph can be negative and still any best-first strategy, such as A^*, finds a solution path if it exists.

Let us denote by $\check{G}$ a locally finite graph such that the cost of any infinite path in $\check{G}$ is unbounded. Thus some edges of $\check{G}$ can have negative costs. Assume that a DP strategy B can find an optimal solution path whenever A^* does in $\check{G}$: $B \in \check{\mathbf{A}}_{gc}$. Note that $\check{\mathbf{A}}_{gc} \subseteq \mathbf{A}_{gc}$ since the set of graphs with strictly positive edge costs forms a subset of the graphs $\check{G}$.

The next theorem is a counterpart of Theorems 3.2.1 and 2.9.3 (cf. Theorem 11 in [18, p. 528]).

Theorem 3.2.5 *Let $\check{\mathbf{I}} = \{(\check{G}, s, \Gamma, h)\}$. Assume that a DP strategy B is in $\check{\mathbf{A}}_{gc}$. Then B explores, in every instance $\check{I} \in \check{\mathbf{I}}$, all the nodes in $\mathbf{N}^M_{g+h}$, i.e., all the nodes surely expanded by A^*.*

Proof: Let $\check{I} = (\check{G}, s, \Gamma, h)$ be a problem instance in $\check{\mathbf{I}}$. Let a node n in $\check{G}$ be in $\mathbf{N}^M_{g+h}$ and not explored by B. Obviously $C^* \leq M$.

Let us create a new graph $\check{G}'$, as in Figure 3.1, by adding to $\check{G}$ a goal node t with $h(t) = 0$ and an edge from n to t with a possibly negative cost $c(n,t) = C^* - g_{P_{s-n}}(n) - \epsilon$, where $C^* > \epsilon > 0$ and $g_{P_{s-n}}(n)$ is the cost of a path P_{s-n} in $\check{G}$ found by A^*. The graph $\check{G}'$ has an extended path P^*_{s-t} with cost $0 < C(P^*_{s-t}) = g_{P_{s-n}}(n) + c(n,t) < C^*$. Moreover, all the infinite paths in $\check{G}'$ are unbounded since the same holds in $\check{G}$. Let $\check{I}' = (\check{G}', s, \Gamma \cup t, h)$ be a new problem instance.

Algorithm A^* searching $\check{G}'$ will find t since $C(P^*_{s-t}) < C^* \leq M$ and, thus, finds the new optimal path P^*_{s-t}. Algorithm B, if given $\check{I}'$ as input, will search $\check{I}'$ in exactly the same way as it would have searched $\check{I}$ since B is deterministic; the only way B can reveal any difference between $\check{I}'$ and $\check{I}$ is by exploring n. Since it does not, it will not find the solution path P^*_{s-t}. This contradicts its property of being in $\check{\mathbf{A}}_{gc}$. $\square$

From now on, let us again study locally finite graphs G with strictly positive edge costs. Let a DPBF strategy B have an evaluation function $f_B(n) = f_B(n; P_{s-n}, I)$ that is a function of all the nodes, edges, edge costs and heuristic values of the nodes *along the path P_{s-n} found by B leading to n.*

The following theorem is a counterpart of Theorem 2.9.4 (cf. Lemma 6 and Theorem 12 (a) in [18, pp. 529-530]). The proof is a modifications of the proofs of Lemma 6 and Theorem 12 in [18].

Theorem 3.2.6 *Let $\mathbf{I} = \{(G, s, \Gamma, h)\}$. Let a DPBF strategy $B \in \mathbf{A}_{ad}$ use an evaluation function f_B such that for every problem instance $I \in \mathbf{I}$, f_B satisfies*

$$f_B(n) = f_B(n; P_{s-n}, I), \quad \forall n \in G \text{ of } I. \tag{3.13}$$

Assume that f_B also satisfies $\forall m_1 \in P_1, m_2 \in P_2$ in G_{AD} of any $I_{AD} \in \mathbf{I}_{AD}$:

$$h(m_1) > h(m_2) \Rightarrow f_B(m_1; P_1, I_{AD}(\cdot, h)) > f_B(m_2; P_2, I_{AD}(\cdot, h)) \tag{3.14}$$

independent of the paths P_1 and P_2 leading to m_1 or m_2. Then B explores, in every instance $I \in \mathbf{I}$, all the nodes in $\mathbf{N}^M_{g+h}$, i.e., all the nodes surely expanded by A^*.

Proof: Let $I = (G, s, \Gamma, h) \in \mathbf{I}$ be some problem instance. Assume $n \in G$ is in $\mathbf{N}^M_{g+h}$ but not explored by B. It follows there exists a path P_{s-n} on G such that for every $n' \in P_{s-n}: \; g_{P_{s-n}}(n') + h(n') < M$ by Theorem 2.8.6.

Define G' to contain *only* a path P_{s-n} with two additional goal nodes t_1 and t_2. The cost of an edge (s, t_2) is $c(s, t_2) = M + \epsilon > M$ $(\epsilon > 0)$. The cost of an edge (n, t_1) is $c(n, t_1) = M - g_{P_{s-n}}(n) > 0$. The goals t_1 and t_2 are assigned $h' = 0$. All the other nodes n' (along P_{s-n}) are assigned h' values such that $g_{P_{s-n}}(n') + h'(n') < M$. In particular, set $h'(n'') > 0$, where $n'' \neq t_2$ is the immediate neighbor of s. That latter can be done since $c(s, n'') < M$. It follows that the new heuristic h' is admissible on G'. Let $I' = (G', s, \{t_1, t_2\}, h')$ be a new problem instance.

The optimal path in G' is $P^*_{s-t_1}$ with cost M since $c(s, t_2) = M + \epsilon > M$. A^* will expand t_1 and not t_2 in G', since $f(n') \leq M$ for all $n' \in P_{s-t_1}$ and $f(t_2) > M$ and, thus, will find the optimal solution path $P^*_{s-t_1}$. However, since $h'(t_2) = 0 < h'(n'')$ implying $f_B(t_2; \cdot, h'(t_2)) < f_B(n''; \cdot, h'(n''))$, B halts on the suboptimal path P_{s-t_2} if I' is given as input to B. This contradicts its property of being in $\mathbf{A}_{ad}$. $\square$

The above theorem states that admissible any greedy or depth-first type DPBF algorithm (in $\mathbf{A}_{ad}$) will explore all the nodes surely expanded by A^* if both guided by a possibly nonadmissible h, cf. Theorem 3.1.2. In Theorem 3.1.2, the heuristic was assumed to satisfy the triangle inequality, and the evaluation function of B was assumed to depend on the *whole* search history of B solving a current problem instance. The above theorem, on the other hand, assumes that the evaluation function of B, for a node, depends *only* on the current path to that node.

The next theorem is a generalization of Theorem 2.9.4 (cf. Lemma 6 and Theorem 12 (a) in [18, pp. 529-530]) in situations, where the heuristic can be nonadmissible.

Theorem 3.2.7 *Let $\mathbf{I} = \{(G, s, \Gamma, h)\}$. Let a BF* algorithm $B \in \mathbf{A}_{ad}$ use an evaluation function f_B such that for every problem instance $I \in \mathbf{I}$, f_B satisfies*

$$f_B(n) = f_B(n; P_{s-n}, I), \ \ \forall n \in G \text{ of } I. \tag{3.15}$$

Moreover, assume that for every problem instance $I_{AD} \in \mathbf{I}_{AD}$, f_B satisfies

$$f_B(\gamma; P_{s-\gamma}, I_{AD}) = C(P_{s-\gamma}) \ \ \forall \gamma \in \Gamma, \tag{3.16}$$

where $C(P_{s-\gamma})$ is the cost of the solution path $P_{s-\gamma}$ in G_{AD} of I_{AD}. Then B explores, in every instance $I \in \mathbf{I}$, all the nodes in $\mathbf{N}^M_{g+h}$, where M is the minmax value related to the evaluation function of A^*.

Proof: (i): Assume $M_B > M$, where M_B is the minmax value related to the evaluation function of B. Let $I = (G, s, \Gamma, h) \in \mathbf{I}$ be some problem instance. Assume $n \in G$ is in $\mathbf{N}^M_{g+h}$ but not explored by B; i.e., there exists a path P_{s-n} such that for every $n' \in P_{s-n} : \ g_{P_{s-n}}(n') + h(n') < M$, and for some $n' \in P_{s-n} : \ f_B(n'; P_{s-n'}, I) \geq M_B$ by Theorem 2.8.6.

Define G' to contain *only* a path P_{s-n} with two additional goal nodes t_1 and t_2. The cost of an edge (s, t_2) is $c(s, t_2) = M_B - \epsilon$, where $0 < \epsilon < M_B - M$ ($M_B > M$ by the assumption of (i)). The cost of an edge (n, t_1) is $c(n, t_1) = M - g_{P_{s-n}}(n) > 0$. The goals t_1 and t_2 are assigned $h' = 0$. All the other nodes n' (along P_{s-n}) retain their old h values ($h'(n') = h(n')$). It follows that the new heuristic h' is admissible on G'. Let $I' = (G', s, \{t_1, t_2\}, h')$ be a new problem instance.

The optimal path in G' is $P^*_{s-t_1}$ with cost M since $c(s, t_2) = M_B - \epsilon > M$. A^* will expand t_1 and not t_2 in G', since $f(n') \leq M = C(P^*_{s-t_1})$ for all $n' \in P^*_{s-t_1}$ and $f(t_2) > M$ and, thus, will find the optimal solution path $P^*_{s-t_1}$. However, since $f_B(t_2; P_{s-t_2}, I') = C(P_{s-t_2}) < M_B$, B halts on the suboptimal path P_{s-t_2} if I' is given as input to B. This contradicts its property of being in $\mathbf{A}_{ad}$.

(ii): Assume $M_B \leq M$. Let us define a constant $K > 0$ and a BF* algorithm $\bar{B}$ as follows. Given K, define $\bar{B}$ to be identical to B except

$$f_{\bar{B}}(m; P_{s-m}, I) = f_B(m; P_{s-m}, I) + K; \quad \forall m \in G, m \notin \Gamma \text{ of any } I \in \mathbf{I} \tag{3.17}$$

so that both $\bar{B}$ and B explore nodes in the same order in solving any $I \in \mathbf{I}$. Hence $\bar{B}$ will find an optimal solution path whenever B does. Then $\bar{B} \in \mathbf{A}_{ad}$ since $B \in \mathbf{A}_{ad}$.

Next assume that $\bar{B}$ does not explore n in G'. From (i), it follows that there exists a node $n'' \in P_{s-n}$: $f_B(n''; P_{s-n''}, I) \geq M_B$. Let the constant K be large enough such that $f_{\bar{B}}(n''; P_{s-n''}, I) = f_B(n''; P_{s-n''}, I) + K > M$, and denote $M_{\bar{B}} = f_B(n''; P_{s-n''}, I) + K$. Thus $M_{\bar{B}} > M$.

Let G'' be the same as G', except that the cost of the edge (s, t_2) is $c(s, t_2) = M_{\bar{B}} - \epsilon$, where $0 < \epsilon < M_{\bar{B}} - M$. Let $I'' = (G'', s, \{t_1, t_2\}, h'')$. The heuristic $h''(m) = h'(m)\ \forall m$ is admissible in I'' since h' is admissible in I'.

The rest of the proof is identical to (i), except that I', B and M_B in (i) are replaced by I'', $\bar{B}$ and $M_{\bar{B}}$, respectively. □

Finally, let us present a proposition that we will use in Chapter 5, cf. Theorems 2.6.1 and 2.6.2:

Proposition 3.2.8 *Let h_1 and h_2 be possibly nonadmissible heuristics such that $h_1(n) \geq h_2(n)$ for all nodes n. Assume that $A^*(h_1)$ and $A^*(h_2)$ have the same minmax value M. Then $A^*(h_2)$ will expand all the nodes surely expanded by $A^*(h_1)$.*

Proof: $\{n \mid g(n) + h_1(n) < M\} \subseteq \{m \mid g(m) + h_2(m) < M\}$ since $h_1(n) \geq h_2(m)$. □

Notice, that above we did not assume anything about the costs of the paths that the algorithms find. Interesting comparisons arise if, in the above proposition, $M = C^*$ or, more strictly, $M = C^*$ and h_2 is admissible whereas h_1 is not.

3.3 CONCLUSIONS

In this chapter, we compared the computational work of several path planning algorithm classes to that of A^* in situations, where the heuristic function guiding the search process is nonadmissible. The computational work is measured as the number of the nodes that the algorithms are guaranteed to explore. Dechter and Pearl [18] give similar results, when the heuristic is admissible.

Based on the study of this chapter, it seems hard to generalize theorems, like the ones in [18] constrained on admissible heuristics, to situations where heuristics can be nonadmissible. The price to pay for the nonadmissibility appears in various additional assumptions in the theorems, see Theorems 3.1.2, 3.2.1, and 3.2.3.

The exception is Theorem 3.2.7 stating that A^* is optimal over certain admissible BF* algorithms. It is a generalization of the corresponding results in [18].

If we allow the search graph to contain also negative edge costs, then Theorem 3.2.5, stating that A^* is optimal over globally compatible DP strategies, is a generalization of the corresponding theorem in [18]. The negative edge costs has to be defined in such a way that the cost of every infinite path in the graph is unbounded, which guarantees that A^* finds a solution path if it exists in the graph.

We also showed that A^* is optimal over some greedy path finding algorithms in Theorem 3.2.6. In greedy algorithms, the most promising node next to be explored is chosen from the set of nodes with the minimum heuristic value.

4 A DYNAMICALLY IMPROVING HEURISTIC FOR A^*

In this chapter, we present an extension to A^* called A^*_A. The evaluation function of A^*_A at some time instant τ during the search is a function of the search tree of A^*_A at τ. Here we present a revised version of [2].

A^*_A is a method of improving an initial admissible heuristic function by using information gathered from the search process so far. We show that A^*_A, using the improved heuristic, is optimal over A^*, that is, A^* will expand all the nodes surely expanded by A^*_A (Definition 2.9.1). Moreover, when A^*_A expands a node n for the first time, it has found an optimal path to n, i.e., $g(n) = g^*(n)$. This means that re-expansions of nodes are unnecessary.

During search, A^*_A estimates successive lower bounds $F_k,\ k = 1, 2, 3, \ldots K$ for the cost C^* of an optimal path. For this, A^*_A utilizes information obtained from an approximation of the original problem. It is a simplification of the original problem generated by ignoring some of its constraints and including the search tree formed by A^*_A before τ. Node expansions in the approximating problem are assumed to be less costly than those in the original problem. The approximation is used as a "guide map" to improve the initial heuristic function during search.

Let the domain of problem instances on which A^* is admissible, $\mathbf{I}_{AD}$ in Section 2.9, be denoted here by $\mathbf{I}$:

$$\mathbf{I} = \{(G, \{c(n, m)\}, s, \Gamma, h) \mid h(n) \leq h^*(n)\ \forall n \in G\}, \tag{4.1}$$

G is a locally finite graph $G = (V, E)$, where V and E are sets of nodes and edges, respectively. A set $\{c(n, m)\}$ contains strictly positive costs of the edges between neighboring nodes n and m in G. A start node is s and Γ is a set of goal nodes. An admissible heuristic h, assigned to each node in G, is a lower bound of h^*.

Definition 4.0.1 *Let a problem instance I be in $\mathbf{I}$. A relaxed model of I is:*

$$\hat{I} = (\hat{G}, \{\hat{c}(n, m)\}, s, \Gamma, \hat{h}(n) = \hat{k}(n, \Gamma)\ \forall n \in \hat{G}). \tag{4.2}$$

The edge cost between neighboring nodes in $\hat{G}$ is $0 < \hat{c}(n, m) \leq c(m, n)$ of I. The value $\hat{k}(n, \Gamma)$ denotes the cost of the cheapest path in $\hat{G}$ from n to the set of goal nodes Γ. Moreover, if a node n is in G of I, then n is in $\hat{G}$, cf. Definition 2.10.1.

The heuristic $\hat{h}$ can be obtained, for example, by finding optimal paths from nodes in $\hat{G}$ to goal nodes Γ. However, relaxed models must be easily solvable compared with their original counterparts to reduce the overall computation time, see [30] for more details. So, $\hat{h}$ is not assumed to be obtained by a blind search in $\hat{G}$. The heuristic $\hat{h}$ is monotone (and consistent) if it is used as a heuristic in G of the original problem I by Theorem 2.10.1 ($\hat{h}(n) = \hat{k}(n, \Gamma)$)).

4.1 AN EXAMPLE

Figure 4.1 illustrates a search problem I in a two dimensional grid. The grid is formed by the horizontal and vertical hairlines and the thicker black lines. A node in the grid is a crossing point of two lines. A start node is on the left and a goal node is on the right. There are two rectangular obstacles on the grid. Let the relaxed model, or problem, $\hat{I}$ be obtained from I by allowing nodes and edges to lie also inside the obstacles in Figure 4.1. Let the edge costs of the relaxed and the original problem be one.

Let A^* find an optimal path from the start node to the goal of I using the Manhattan distance heuristic that equals to the $\hat{h}$ of the relaxed problem $\hat{I}$.[1] The cost of the optimal path is $C^* = 15$. Nodes marked with squares are the ones that A^* surely expands, i.e., nodes n for which $f(n) < 15$, see Theorem 2.4.2.

By looking at Figure 4.1, it is easy to construct another simple algorithm, say B, that also finds an optimal path. Nodes marked with black circles are the ones that B expands. Nodes marked with white circles are open nodes that B would choose to expand next if it had not found the goal. A search tree of B consists of the white and the black circled nodes connected by the thicker black lines. Algorithm B uses the Manhattan distance heuristic too. B can be described by a rule: *First expand the start node and then always expand the rightmost open node in Figure 4.1, say m, for which $f(m) \leq C^* = 15$.*

From Figure 4.1, we notice that the nodes surely expanded by B form a subset of the nodes surely expanded by A^*: the former ones are the black nodes p for which $f(p) < 15$ and the latter ones are the squared nodes.

How can an algorithm like B be sure that it really has found an optimal path and not expanded any nodes m for which $f(m) > C^*$? An answer is that B has to know C^* or the lower bound F for it ($F \leq C^*$) in advance.

In the following, we will construct a problem that approximates the original problem in a certain way. A new algorithm uses the approximating problem as a "guide map" to the unexplored search space and estimates $F_i \leq C^*$ ($i = 1, 2, \ldots$) during search. Then it uses these lower bounds to improve an admissible, static, heuristic. The algorithm has methods like the above B as tie-breaking rules.

4.2 BASIC IDEAS FORMALIZED

Here, we discuss how to find the lower bounds for C^* and how to improve a given admissible heuristic using them. We will define two subroutines A^*_F and A^*_X. They are A^* algorithms. A^*_F uses the improved heuristic and A^*_X estimates the lower bounds by searching in an approximating model of the original problem.

[1] The Manhattan distance is obtained by a simple calculation — not by searching a cheapest path from n to the goal in the grid.

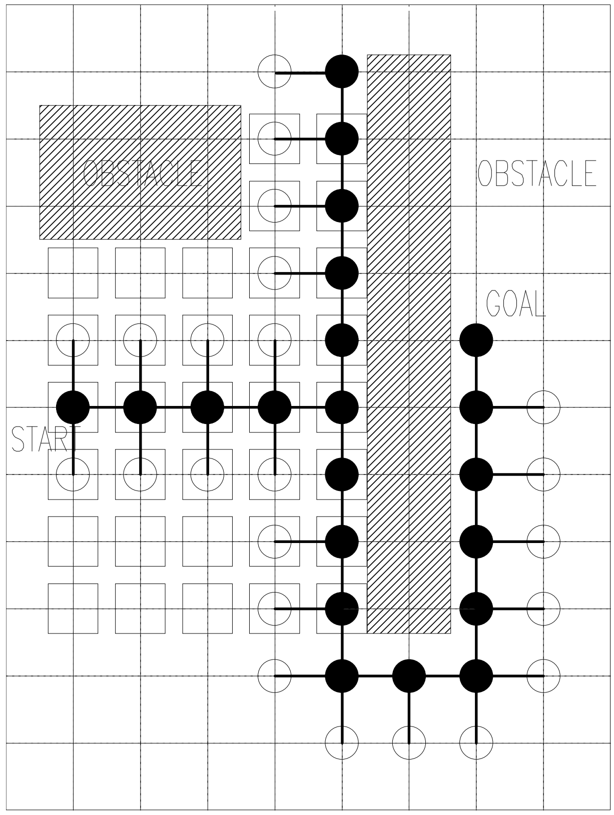

Figure 4.1: A Search in a 2-D grid [2].

An Improved Heuristic: A_F^*

Suppose that A^* searches G of the problem instance $I \in \mathbf{I}$ and is guided by a consistent heuristic $\hat{h}$ of $\hat{I}$ in Definition 4.0.1. Let $g(n)$ be the cost of a path from the start node to a node n in G found by A^*. Furthermore, assume that we know a lower bound F for the optimal path cost C^* in G ($F \leq C^*$). Then we can define a new heuristic h_F:

$$\begin{aligned} h_1(n) &= F - g(n), \\ h_F(n) &= \max\,\{\hat{h}(n),\ h_1(n)\}. \end{aligned} \tag{4.3}$$

Proposition 4.2.1 *h_F is monotone and $h_F(n) \geq \hat{h}(n)\ \forall n$.*

Proof: Let the path costs to a node n and to its successor n' in G, both found by A^*, be $g(n') = c(n, n') + g(n)$. Then $h_1(n) = F - g(n) = F - g(n') + c(n, n') = c(n, n') + h_1(n')$. Thus h_1 is monotone and consistent (and admissible), see Section 2.7. Since h_F is a maximum of two monotone heuristics h_1 and $\hat{h}$, h_F is itself monotone (consistent) and $h_F(n) \geq \hat{h}(n)\ \forall n$. $\square$

Note that the heuristic value $h_F(n)$ in Proposition 4.2.1 is dependent on the search "history" of A^* in G. The values of $h_F(n)$ can increase if F increases during the search.

If we know that $0 < F \leq C^*$, then we can replace the "old" f -values of each node m in G for which $f(m) = g(m) + \hat{h}(m) < F$ with $f(m) =$

$g(m) + h_F(m) = F$. It means that A^*, now guided by h_F, can expand all such nodes m in any order depending on its tie-breaking rule. This would not, of course, be possible if $\hat{h}$ were used instead of h_F.

Denote by A^*_F the A^* algorithm guided by the heuristic h_F. The core of A^*_F is a tie-breaking rule specifically constructed for the problem at hand. The algorithm B above was a naive example of such a rule for a simple 2-D grid search problem. There are some minor differences between the program structures of A^*_F and A^*. We will define them later.

Lower Bounds for C^*: A^*_X

Assume that an A^* algorithm is expanding nodes in G of the problem instance $I \in \mathbf{I}$.[2] Let us arbitrarily stop the search process at a "time" τ before a goal has been found, memorize the search tree at τ, and call it T_τ. T_τ contains the nodes in G, in OPEN and CLOSED, and the directed arcs, pointers, between them as described in the pseudocode of A^* in Section 2.1.

Definition 4.2.1 *Let a search tree $\tilde{T}_\tau$ be identical to T_τ of the above A^* at τ in G, except that all the directed arcs in T_τ are reversed.*

Let us construct a new problem instance, $\tilde{I}(\tau)$, based on I and $\hat{I}$. A search graph $\tilde{G}_\tau$ of $\tilde{I}(\tau)$ is composed of $\tilde{T}_\tau$ with additional nodes and edges in $\hat{G}$ of $\hat{I}$.

Definition 4.2.2 *Let there be a problem instance $I \in \mathbf{I}$ and its relaxed model $\hat{I}$. An approximating model of I is:*

$$\tilde{I}(\tau) = (\tilde{G}_\tau, \{\tilde{c}(n, m)\}, s, \Gamma, \tilde{h}(n) = \hat{h}(n)\ \forall n \in \tilde{G}_\tau) \tag{4.4}$$

The nodes in $\tilde{G}_\tau$ are the same ones that are in $\hat{G}$ of $\hat{I}$. There is an edge between nodes n and m in $\tilde{G}_\tau$ if there is one between n and m in $\hat{G}$ with the exception that all the directed arcs in $\tilde{T}_\tau$ are the only edges between the neighboring nodes in $\tilde{T}_\tau$.

The edge cost between neighboring nodes in $\tilde{G}_\tau$ is $\tilde{c}(n, m) = \hat{c}(n, m)$ of $\hat{I}$ with two exceptions. (1) if n and m are in $\tilde{T}_\tau$, then $\tilde{c}(n, m) = c(n, m)$ of I. (2) If the above A^* cannot generate m as a successor to n in $\hat{G}$, then $\tilde{c}(n, m) = \infty$.

Figure 4.2 shows an example of $\tilde{I}(\tau)$ although the directions of the arcs in $\tilde{T}_\tau$ are not marked. $\hat{G}$ and G are the same as in Figure 4.1. $\tilde{G}_\tau$ is the same as $\hat{G}$ in Figure 4.1 except some "missing" hairlines in Figure 4.2. These are the edges between the expanded, black, nodes in $\tilde{T}_\tau$ and the nodes in $\hat{G}$ that are inside the rightmost obstacle. This is because the above A^* has tried to generate the latter nodes but not succeeded and has assigned the cost of ∞ to each edge in $\tilde{T}_\tau$ coming to these nodes.

Assume that we know a lower bound $F = 11 < C^* = 15$. Then the above A^*, searching G of I, can use the heuristic $h_F(n) = \max(\hat{h}(n),\ 11 - g(n))$,

[2] From now on, we will say "nodes in G of I" for short.

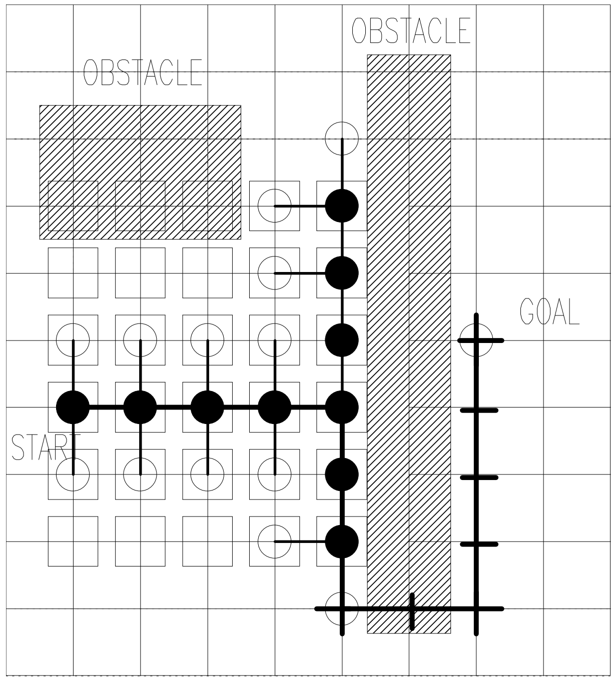

Figure 4.2: A 2-D search stopped at time τ (10 expanded nodes) [2].

where $\hat{h}$ is the Manhattan distance heuristic based on $\hat{I}$. Hence, the A^* in this example is actually the above mentioned A^*_F.

Let A^*_F use the same tie-breaking rule as the algorithm B in the previous example: *First expand the start node and then always expand the rightmost open node n for which $f(n) \leq 11$.* A search tree $\tilde{T}_\tau$ of A^*_F with ten expanded nodes is composed of the ten black nodes, in CLOSED, and the white circled nodes, in OPEN, with the corresponding edges in Figure 4.2.[3]

Let the A^* with the search tree T_τ solve a problem instance $I \in \mathbf{I}$ by using the *admissible* heuristic h provided by I. Let another A^* algorithm, A^*_X, solve the approximating model $\tilde{I}(\tau)$ of I using the consistent heuristic $\tilde{h}(n) = \hat{h}(n)\ \forall n \in \tilde{G}_\tau$. When A^*_X has solved $\tilde{I}(\tau)$, it outputs $\tilde{C}^*$ that is the cost of an optimal solution path in $\tilde{G}_\tau$. The proposition below follows immediately from Definition 4.2.2 and the fact that the A^* has expanded at least the start node in G of I at every $\tau > 0$.

Proposition 4.2.2 *Every path that A^*_X has found in $\tilde{G}_\tau$ of $\tilde{I}(\tau)$ has at least one node n in CLOSED and its successor node along that path in OPEN, where OPEN and CLOSED are the node sets of the A^* with the search tree T_τ at τ ($\tau > 0$).*

Proposition 4.2.3 *Assume that the A^* with the search tree T_τ, expanding nodes in G of I, uses an admissible heuristic $h(n) \geq \tilde{h}(n)\ \forall n \in G$. Then any optimal path $\tilde{P}^*$, from the start node s to a goal node γ in $\tilde{G}_\tau$ of $\tilde{I}(\tau)$, found by A^*_X at τ ($\tau > 0$) has a cost $\tilde{C}^* \leq C^*$, where C^* is the cost of an optimal solution path in G.*

[3]At the moment, do not pay any attention to the squared nodes, and crosses of thick bolded lines.

Proof. Let n be the deepest node in $\tilde{T}_\tau$ expanded by the A^* at time τ ($\tau > 0$) along an optimal path $\tilde{P}^*$; no descendants of n are expanded by the A^*. The successor to n along $\tilde{P}^*$ is in OPEN of the A^*. Proposition 4.2.2 guarantees that such nodes can always be found along any $\tilde{P}^*$.

Let $\hat{k}(n, \gamma)$ be the cost of the cheapest path from n to a goal γ in $\hat{G}$ of $\hat{I}$. Then $\tilde{C}^* = \tilde{g}^*(n) + \hat{k}(n, \gamma)$, where $\tilde{g}^*(n)$ is the cost of the cheapest path from s to n in $\tilde{G}_\tau$ when A^*_X has expanded n. The cost $\tilde{g}(n) = \tilde{g}^*(n)$ since $\tilde{h} = \hat{h}$ is consistent by Theorem 2.7.3. Definition 4.2.2 implies that $\tilde{g}^*(n) \leq g(n)$, where $g(n)$ is the cost of the path from s to n in G found by the A^*. Definition 4.0.1 says that $\hat{k}(n, \gamma) = \hat{h}(n)$. Hence $\tilde{C}^* \leq g(n) + h(n) \leq C^*$ by Theorem 2.4.1 since $h(n) \geq \tilde{h}(n)$ is admissible. □

From now on, let A^*_F denote the above A^* searching G of I with the search tree T_τ. Moreover, let it be possible for A^*_F to use a heuristic $H_F(n) = \max\{h_F(n), h(n)\}\ \forall n$, where h is the *admissible*, not necessarily consistent, heuristic provided by the original problem instance I. The heuristic h *is not necessarily related to* $\hat{I}$ or $\tilde{I}(\tau)$, like $\hat{h}$ and h_F. This is because we do not want to restrict ourselves to utilizing only $\hat{I}$ and $\tilde{I}(\tau)$ in constructing heuristics for a given problem.

Let a lower bound $F_1 < C^*$ be known. A better lower bound F_2 ($C^* \geq F_2 > F_1$) can be found by using the subroutines A^*_F and A^*_X as follows. First, let A^*_F expand q_1 new nodes in G until the time τ_1.[4] Second, let A^*_X find optimal paths in $\tilde{G}_{\tau_1}$, into which the q_1 newly expanded nodes, their immediate successors and the corresponding edges are included. If $\tilde{C}^* > F_1$, then set $F_2 = \tilde{C}^*$, otherwise let A^*_F expand another set of new nodes, etc. This is what the following algorithm A^*_A does.

4.3 ALGORITHM A^*_A

The subroutine A^*_F can expand nodes n for which $g(n) > g^*(n)$ since its guiding heuristic H_F is admissible but not necessarily consistent, see lines (5.2)–(5.3) in the pseudocode of A^* in Section 2.1. To avoid this and reopenings of expanded nodes, we introduce the following definition and proposition.

Definition 4.3.1 *Let n_τ be a node that is in OPEN of A^*_F at $\tau > 0$ and expanded by A^*_X such that $g(n_\tau) = \tilde{g}^*(n_\tau)$.*[5] *Let $S(n_\tau)$ be the set of all the nodes n_τ after A^*_X has found some optimal solution paths in $\tilde{G}_\tau$.*

Let us again look at Figure 4.2. The black nodes are expanded by the subroutine A^*_F and are in $\tilde{T}$. The white circled nodes are in OPEN of A^*_F. Let the start node in Figure 4.2 be the origin and the crossings of the thin lines be points in the plane. The x -axis is the horizontal and the y -axis is the vertical line going via the start node. Let the distance between two neighboring crossings be one as before.

[4] We will explain the details of this later.

[5] The existence of such nodes will be proven in the following Proposition 4.3.1.

Now, assume that the partial paths found by A^*_F go along the thicker lines connecting the black nodes. The horizontal thicker line from the start node to the node whose coordinates in Figure 4.2 are (4,0) is common to the two pointer paths, one leading upwards and one leading downwards from the node (4,0).

Let A^*_X use the following tie-breaking rule:[6] *Among the nodes with the same minimum $\hat{f}$ value, first expand those nodes that are in CLOSED of A^*_F.* At first, A^*_X expands and places the white circled nodes (0,1), (1,1), (2,1), and (3,1) in $S(n_\tau)$. Then, it expands and places the white circled nodes (0,-1), (1,-1), (2,-1), and (3,-1) in $S(n_\tau)$. Finally, the white circled nodes (4,4) and (4,-3) are placed in $S(n_\tau)$. The reason for the nodes (4,4) and (4,-3) being in $S(n_\tau)$ is that the edge costs $\hat{c} = \tilde{c} = c$ in $\hat{G}$, $\tilde{G}$ and G. If $\hat{c} < c$ outside $\tilde{T}_\tau$, then they would not be in $S(n_\tau)$ if A^*_X found a shorter path to them than A^*_F, going via nodes that are not in $\tilde{T}_\tau$. Finally, the white circled nodes (3,-2), (3,2), and (3,3) are not in $S(n_\tau)$ since A^*_X has found a shorter path to them than A^*_F.

A^*_X differs from A^* in the sense that, if needed, A^*_X does not stop after finding the first solution path. The user may determine for how long A^*_X runs. We will return to this subject in the pseudocode. Anyway, A^*_X must run at least as long as there are one or more nodes in $S(n_\tau)$. This modification is needed in the following proposition.

Proposition 4.3.1 *Assume a problem instance $I \in \mathbf{I}$ and its approximating model $\tilde{I}(\tau)$. The set $S(n_\tau)$ is never empty unless OPEN of A^*_F is empty. Moreover, when A^*_F expands a node n_τ in $S(n_\tau)$ for the first time, then $g(n_\tau) = g^*(n_\tau)$.*

Proof: A^*_F is guided by an admissible heuristic. From Lemma 2.3.2, it follows that the OPEN of A^*_F, and hence $\tilde{T}_\tau$, always has nodes n for which $g(n) = g^*(n)$. After running long enough, A^*_X must have expanded all the nodes in $\tilde{T}_\tau$. Some of the partial paths to the nodes n found by A^*_X must contain nodes only in $\tilde{T}_\tau$, hence $g(n) = \tilde{g}^*(n)$ by Definition 4.2.2 (A^*_X uses a consistent heuristic). It follows that $S(n_\tau)$ contains nodes unless the OPEN of A^*_F is empty.

Since A^*_X has expanded n_τ in $S(n_\tau)$, $\tilde{g}(n_\tau) = \tilde{g}^*(n_\tau) \leq g^*(n_\tau)$. The latter inequality follows from Definition 4.2.2. But then Definition 4.3.1 implies that $g(n_\tau) = \tilde{g}^*(n_\tau) \leq g^*(n_\tau)$. It follows that $g(n_\tau) = g^*(n_\tau)$. Hence A^*_F finds an optimal path to n_τ when it expands n_τ for the first time. □

Proposition 4.3.2 *Assume a problem instance $I \in \mathbf{I}$ and its approximating model $\tilde{I}(\tau)$. Let it be possible for A^*_F to expand nodes only in $S(n_\tau)$ at any τ. Assume that a lower bound $F_1 < C^*$ is known. Let $\tilde{\mathbf{P}}^*$ denote the set of all the cheapest solution paths in $\tilde{G}_\tau$ at $\tau > 0$ and $\tilde{C}^*$ their cost. Let r be the deepest node in $S(n_\tau)$ along a path in $\tilde{\mathbf{P}}^*$.*

A sufficient condition to have $C^* \geq \tilde{C}^* > F_1$ is that the above nodes r along every path in $\tilde{\mathbf{P}}^*$ have $\tilde{f}(r) = \tilde{g}^*(r) + \tilde{k}(r, \gamma) = \tilde{C}^* > F_1$.

[6] This is not obligatory but may improve the overall performance.

Proof. Follows from Proposition 4.2.3: $C^* \geq \tilde{C}^* = \tilde{f}(r) > F_1$. □

Proposition 4.3.2 states that in order for A_X^* to get a better lower bound than F_1 for C^*, A_F^* does not necessarily have to expand all the nodes l for which $g(l) + H_F(l) = F_1$. A_F^* has to expand only the nodes such that their successors r in $S(n_\tau)$, and thus in OPEN of the A_F^*, have $f(r) \geq \tilde{f}(r) > F_1$. The number of the nodes, nro_l, that can be saved from expansion depends on the tie-breaking rule of A_F^*. However, Proposition 4.3.2 is not constructive: it does not give an actual tie-breaking rule for A_F^* guaranteeing that $nro_l > 0$. Every l must be expanded in the worst case.

The following example illustrates Proposition 4.3.2. Let $F_1 = 11$ and A_F^* have the same tie-breaking rule as algorithm B in Figure 4.2. $H_F = h_F$ for simplicity. In Figure 4.2, A_F^* has expanded the black nodes (at τ). Hence it has expanded only a subset of the nodes l for which $g(l) + H_F(l) = 11$ such that every r, in Proposition 4.3.2, has $\tilde{f}(r) > 11$. The nodes r are the upmost and the lowest white circled nodes in Figure 4.2. The nodes l are the squared ones that are not inside the left obstacle.

Now, suppose that A_X^* begins to search $\tilde{G}_\tau$ with the Manhattan distance heuristic, in Figure 4.2. A_X^* finds an optimal path of cost 13, see the thick bolded line connecting nodes marked with thick bolded crosses together with some of the black nodes. Similarly, A_X^* finds another optimal path of cost 13 that goes via the upmost white circled node. This is enough to set $F_2 = 13 > F_1$. Later, A_F^* can use F_2 as part of its heuristic H_F.

Next, we will prove that the following algorithm, A_A^*, is optimal over A^*. Let there be a problem instance $I \in \mathbf{I}$ and its approximating model $\tilde{I}(\tau)$. A_A^* has subroutines A_F^* and A_X^*. A_F^* searches the graph G of I, using the admissible heuristic H_F, and A_X^* searches $\tilde{G}_\tau$ of $\tilde{I}(\tau)$, using the consistent heuristic $\hat{h}$. The pseudocode of A_A^* is:

ALGORITHM A_A^*:

```
(1)  Put the start node s in OPEN and in S
(2)  F = f(s)
(3)  WHILE OPEN not empty DO
(4)     Run AF*(F, S, CLOSED, OPEN) for some time
(5)     Run AX*(CLOSED, OPEN, S, COST)
            until S is not empty or longer
(6)     IF COST > F THEN F = COST
(7)  END
(8)  Exit with failure
```

On line (4), “AF*(F, S, CLOSED, OPEN)” is A_F^*. F and S are input parameters whereas CLOSED and OPEN are both input and output parameters of A_F^*. S is the set $S(n_\tau)$ in Definition 4.3.1: S $\subseteq$ OPEN.

A^*_F is somewhat different from the "plain" A^*. First, line (4) means that A^*_F does not have to find an optimal path from the start node to a goal node after one procedure call. Second, one procedure call denotes a search of A^*_F that starts from the T_{τ_1} ($\tau_1 \geq 0$) with possibly new F and S parameters and stops at T_{τ_2} ($\tau_2 > \tau_1$). Third, A^*_F can only expand nodes in S.

On line (5), "AX*(CLOSED, OPEN, S, COST)" is A^*_X. CLOSED and OPEN are input parameters whereas S and COST are output parameters of A^*_X. COST $= \tilde{C}^*$, the cost of an optimal path found by A^*_X in $\tilde{G}_\tau$ at τ. Here it is assumed that A^*_X begins to search from the start node s at every procedure call.[7] Recall that A^*_X may have to find more than one solution path until S is not empty.

A^*_A ends successfully if A^*_F ends successfully, that is, if A^*_F removes a goal node from OPEN and places it in CLOSED, see the pseudocode of A^* in Section 2.1.

Theorem 4.3.3 *Assume a set of problem instances $\mathbf{I}_A \subseteq \mathbf{I}$ such that every $I_A \in \mathbf{I}_A$ has an approximating model $\tilde{I}_A(\tau)$. Then A^*_A is admissible and optimal over A^* relative to $\mathbf{I}_A$, where the admissible heuristic given as input to both algorithms is provided by I_A. Moreover, when A^*_A expands a node n (in G of I_A) in the set S for the first time, then it has found an optimal path to n.*

Proof. Let $I_A = (G, \{c(n,m)\}, s, \Gamma, h)$ be a problem instance in $\mathbf{I}_A$, where $h(n) \leq h^*(n)\ \forall n \in G$. Thus h is the admissible heuristic given as input to both A^* and A^*_A.

From Proposition 4.2.3, it follows that the value of F never exceeds C^* in the pseudocode of A^*_A. Hence A^*_F is guided by the admissible heuristic $H_F(n) = \max\{h_F(n), h(n)\}\ \forall n$ [8] and does not expand any node n for which $f(n) > C^*$, by Theorem 2.4.1. Proposition 4.3.1 states that the set S is never empty unless the OPEN of A^*_F is empty, on line (4). Hence A^*_A finds an optimal path whenever A^*_F does and A^*_F finds an optimal path if it exists. When A^*_A expands a node n in S for the first time (in subroutine A^*_F), then $g(n) = g^*(n)$ by Proposition 4.3.1.

A^*_A (A^*_F) uses the dynamically improving heuristic $H_F(n) \geq h(n)\ \forall n$, which implies that

$$\{n \mid g(n) + H_F(n) < C^*\} \subseteq \{m \mid g(m) + h(m) < C^*\}, \qquad (4.5)$$

where the above sets are the sets of nodes surely expanded by A^*_A and A^*, respectively. Hence A^*_A is optimal over A^* relative to $\mathbf{I}_A$ in the sense of Definition 2.9.1. □

A *necessary condition* for A^*_A to surely expand fewer nodes than A^* is the following. Let γ be the goal found by A^*_F. Then it must be that $F = C^*$ in the above pseudocode when A^*_F selects γ from OPEN. This is to guarantee that $h_1(n) = F - g(n) > h(n)\ \forall n$ in S in the heuristic H_F before the selection of γ. In other words, F must converge to C^* before A^*_F finds a goal. This requires that for many enough nodes m in the original problem and its

[7] We will later remove this restriction.

[8] h_F is the same as above Proposition 4.2.1.

approximating model: $k(m, \gamma) = \tilde{k}(m, \gamma)$. Let us call this kind of an approximating model *effective*.

Let again $F = 11$, A^*_F have the same tie-breaking rule as B in Figure 4.1, and $H_F(n) = h_F(n)\ \forall n$. Now, starting from the situation in Figure 4.2 and looking at the pseudocode of A^*_A, we can see how A^*_A finds the optimal path of cost 15 shown in Figure 4.1. It is also easy to "run" A^*_A "on paper" from the start node in Figure 4.1 by first setting F = 7 until F = 15 and the optimal path is found.

4.4 ON THE EFFECTIVENESS OF A^*_A

The effectiveness of A^*_A compared to that of A^* depends mainly on the subroutines A^*_F and A^*_X, on lines (4) and (5) in the pseudocode of A^*_A.

Subroutine A^*_X

A^*_X searches and expands nodes in the graph $\tilde{G}_\tau$ of $\tilde{I}(\tau)$. From the construction of $\tilde{I}(\tau)$, in Definition 4.2.2, it follows that A^*_X expands new nodes in $\hat{G}$ of $\hat{I}$ and utilizes the nodes in G of I, already expanded by A^*_F. The efficiency of A^*_X requires that node expansions in the relaxed model $\hat{I}$ must be less costly than node expansions in the original problem I.

In Figure 4.2, A^*_X has surely expanded the white squared nodes in order to find an optimal path of cost 13. A^* can solve the whole problem by surely expanding the white squared nodes in Figure 4.1. By comparing the two figures we see that A^*_X has expanded three nodes inside the left obstacle that A^* never expands.

An example where node expansions in the approximating model, more precisely in $\hat{G}$ of $\hat{I}$, are less costly than in the original problem is a *piano-mover's problem*, cf. also Chapter 6. Figure 4.1 illustrates a common way to discretize it. The problem consists of a solid object in space with obstacles. The aim is to find a shortest path for the object from a start position to an end position such that the object does not collide with the obstacles. For more details and related problems for robot manipulators, see [44] and [35].

In the piano-mover's problems the most time consuming operation is a *collision test*: geometric calculations to determine whether the object collides with the obstacles or not. This is particularly true if the geometries are complicated and the space is high dimensional (three or more). Then collision tests typically require over 90 per cent of the overall computational work. Now, when A^*_F expands a node for the first time, it does the collision test. On the other hand, A^*_X expands nodes without collision tests.

Other problems, suitable for A^*_A, may arise in systems where data for a node to be expanded must be searched from databases and/or transmitted to the place where the planning algorithm is located.

A^*_X does not have to find all the optimal paths in $\tilde{G}_\tau$ in order to get a better lower bound for C^*, cf. Proposition 4.3.2 that gives a sufficient condition for better lower bounds. Only one optimal path may sometimes be enough to get a better lower bound F and a nonempty set S for A^*_F, in the pseudocode of A_A^*.

It is possible to get a better F value than $\tilde{C}^*$, in the pseudocode. From Theorem 2.4.1 we know that A_F^* expands only nodes n for which $f(n) \leq C^*$. Let n represent the nodes expanded by A_A^* (A_F^*) for which the f value is maximum. Since $H_F(n) = \max\{h_F(n), h(n)\}$ $\forall n$ is admissible, we can take as F:

$$C^* \geq F = \max\{\tilde{C}^*, f(n)\}. \tag{4.6}$$

It is not necessary to begin the next search of A_X^* from the start node in the WHILE -loop of A_A^*. Instead, it can be started from earlier expanded nodes, in CLOSED of A_F^*, whose immediate successors are in OPEN of A_F^*. This is because A_F^* has already found optimal paths to all the expanded nodes by Proposition 4.3.1.

Assume that A_X^* utilizes the search tree T_τ of A_F^* in the next iteration of the WHILE -loop of A_A^*. If $\tilde{G}(\tau)$ is "close enough" to G such that their final search trees are almost equal, after a goal has been found in G, then A_A^* and A^* consume approximately the same amount of memory.

Subroutine A_F^*

How many nodes A_F^* actually expands depends on its tie-breaking rule. The tie-breaking rule affects the total number of node expansions in every set $J_F = \{n \mid f(n) \leq F \leq C^*\}$. This requires, however, that the approximating model is effective, recall the text below Theorem 4.3.3. In the "traditional" A^*, a tie-breaking rule affects the total number of node expansions only in the set $\{m \mid f(m) = C^*\}$.

The construction of an effective tie-breaking rule for A_F^* is a problem specific task. However, a simple rule candidate to begin with suggests to be greedy: *Among the nodes n_τ in $S(n_\tau)$ with the minimum $f(n_\tau)$ value, first expand the ones that are nearest to a goal, measured perhaps in some other way than using H_F*. Decision rules, like the above, may be based on information other than contained in the probem instance I and its approximating model $\tilde{I}$, e.g., context information concerning the application area of I, cf. C_I at the beginning of Chapter 3.

For example, the expansion rule of the algorithm B in the example of Figure 4.1 states: *Always expand the rightmost nodes l for which $f(l) \leq C^*$*. The rightmost nodes are closest to the goal measured horizontally, not by the Manhattan distance heuristic that the algorithm uses.

Of course, the above rules do not tell how many node expansions of A_F^* are enough to guarantee that A_X^* actually finds a better lower bound for C^*. Hence one cannot know *a priori* how many times A_F^* has to be called on line (4) in the WHILE -loop of A_A^* (Proposition 4.3.2).

More Ways of Utilizing A_X^* and $\tilde{I}$

A_X^* and $\tilde{I}$ can also be used in finding paths or partial parts from selected nodes of the plain A^* [46]. In this case, no lower bounds for C^* are searched but a more informed heuristic for the A^* may be obtained.

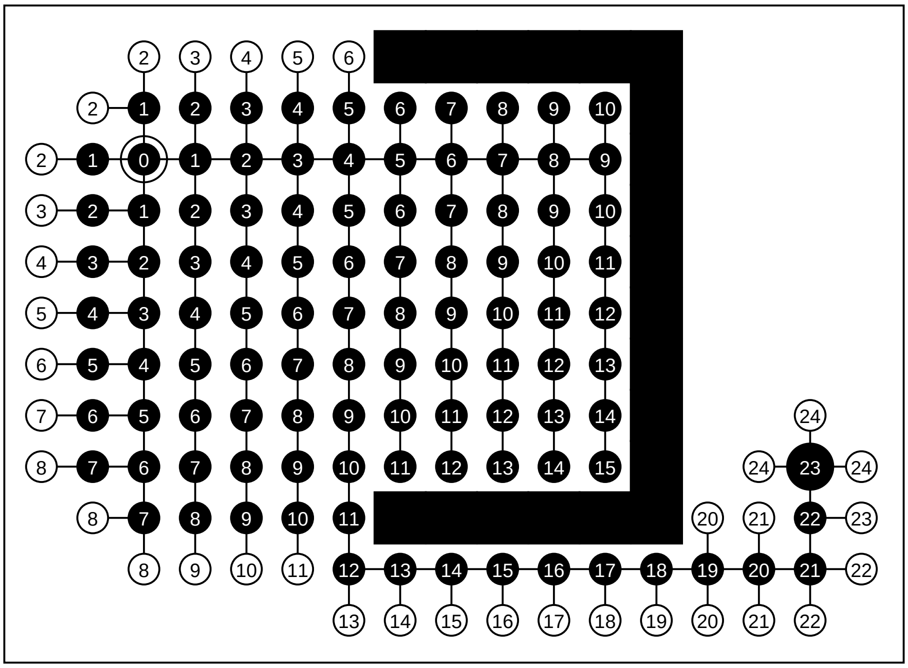

Figure 4.3: A 2-D grid with an obstacle: A^* [2].

4.5 EXAMPLES

In all the examples shown in Figures 4.3–4.6, graphs G, $\hat{G}$ and $\tilde{G}_\tau$ are based on the rectangular grid similarly as in Figure 4.1 and $c = \hat{c} = \tilde{c} = 1$. The heuristic, to which both A^*_A and A^* have access, is the Manhattan distance to the goal measured on the grid. Black and white circled dots, in the figures, are expanded nodes in CLOSED and nodes in OPEN, respectively. The values g are marked on the nodes. Start nodes are marked with bigger circles and are on the left. Goals are bigger black nodes on the right.

Figures 4.3 and 4.4 show a problem with a nonconvex black obstacle. In Figure 4.3 the search algorithm is A^*. In Figure 4.4 A^*_A is used instead. Figures 4.5 and 4.6 show a problem with three black obstacles. In Figure 4.5, the search algorithm is A^* whereas A^*_A is used in Figure 4.6.

A tie-breaking rule in all the examples is a greedy one: *Among the nodes in S with the minimum f value, first expand the ones that are located rightmost and lowest in the figures.* Here the context information, not explicitly present in the grid, is the direction of the goal: *rightmost and lowest nodes first.* In the current implementation, the set S contains only one node at a time obtained by finding one optimal path using A^*_X. However, even these simple rules make it possible for A^*_A to save many node expansions compared to A^*.

4.6 CONCLUSIONS

In this chapter, we presented an algorithm A^*_A that is optimal over A^*, if they both receive as input the same admissible heuristic function. We also showed that when A^*_A expands a node for the first time, it has found an optimal path to that node. It follows that no re-expansions of nodes are necessary, which is

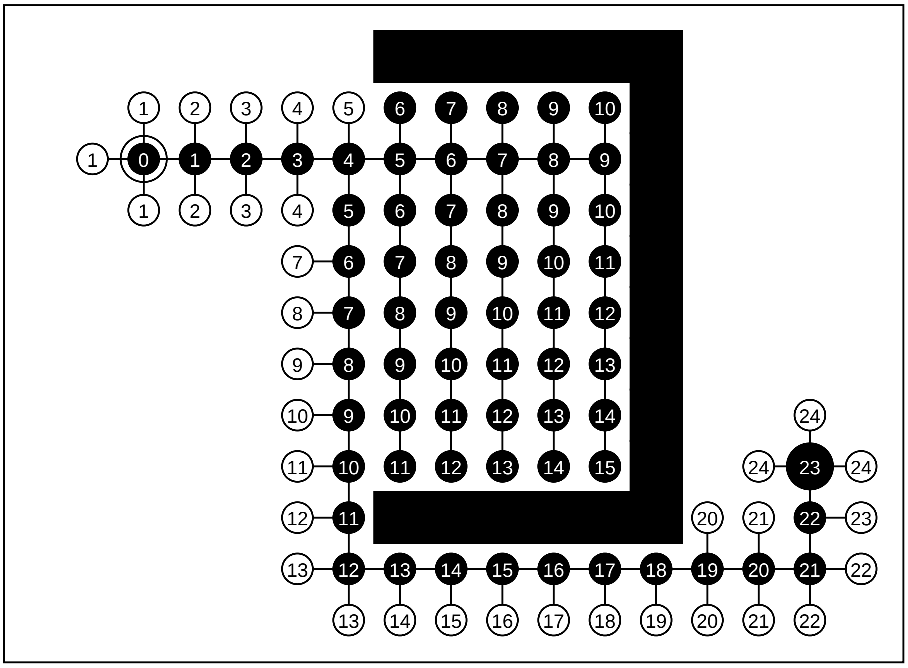

Figure 4.4: The 2-D grid with an obstacle: A^*_A [2].

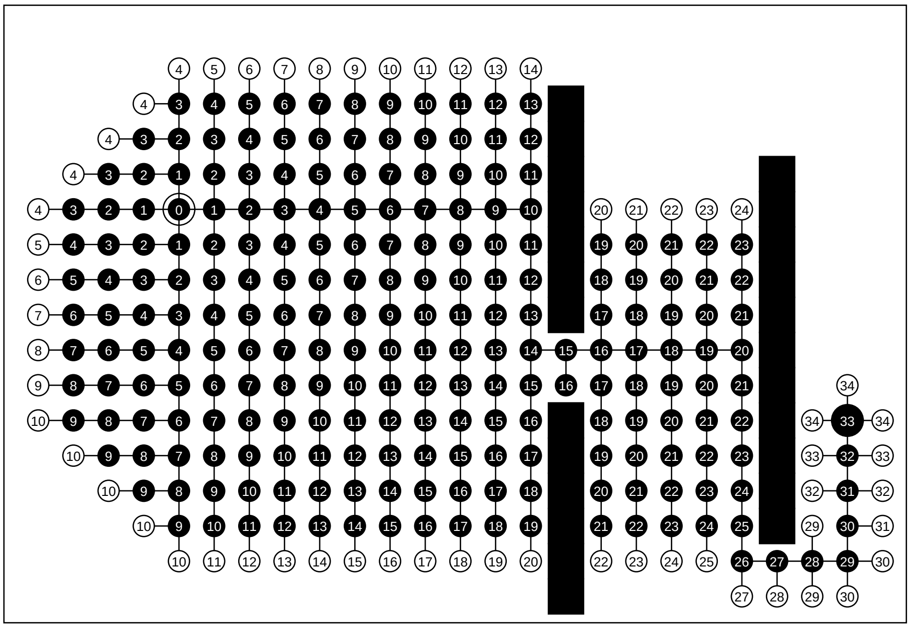

Figure 4.5: A 2-D grid with three obstacles: A^* [2].

an important property of effective search strategies.

A^*_A can utilize more information about the search history of the original problem than A^* does. By utilizing the information we constructed an effective approximating model, see Definition 4.2.2 and the text below the proof of Theorem 4.3.3. The approximating model is used as a "guide map" to improve the heuristic function during search.

Node expansions in the approximating model are required to be less costly than in the original problem. The latter assumption is satisfied, e.g., in sys-

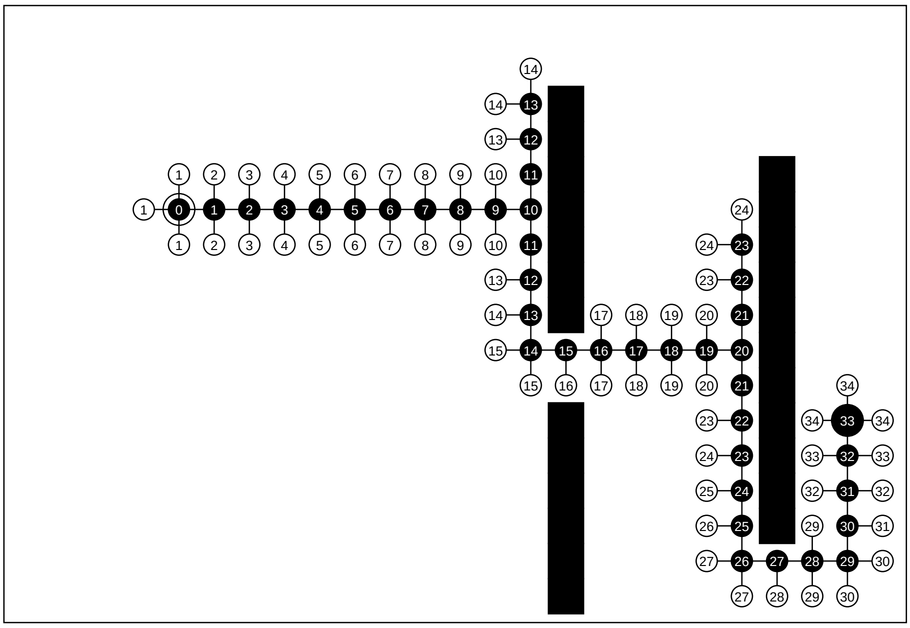

Figure 4.6: The 2-D grid with three obstacles: A_A^* [2].

tems where data for a node to be expanded must be searched from databases and/or transmitted to the place where the planning algorithm is located. Other examples are so called piano-mover's problems where database queries correspond to testing whether an object (a piano) collides with nearby obstacles or not.

If the construction of an effective approximating model is possible, then a problem specific tie-breaking rule of A_A^* reduces the total number of node expansions in the original problem on all "layers" of nodes n for which $f(n) =$ constant $\leq C^*$. This is because A_A^* uses the improved heuristic based on successive lower bounds for C^* by utilizing the effective approximating model. A tie-breaking rule of A^*, on the other hand, affects only expansions of nodes m for which $f(m) = C^*$.

In this way, a user has more freedom to adapt the program structure to a particular problem instance than only constructing admissible static heuristics.

Acknowledgments

The author wishes to thank Mr. Johannes Lehtinen for implementing the A_A^* algorithm and running the examples, and for valuable comments and discussions during the development process.

5 A^* AS A RESOURCE ALLOCATION POLICY

In many problems, A^* guided by an admissible heuristic is known to be slow. This is mainly because effective, or informative, admissible heuristics to prune the search tree are hard to obtain. Usually, more depth-first type or greedy algorithms can find solution paths much faster than A^*. However, then it is very difficult to know *a priori* which search methods are faster than others. This motivates to run a set of promising algorithms more or less at the same time in order to find a reasonably good solution path.

In this chapter, we examine how A^* can be used to allocate computing resources among several search algorithms solving the same path finding problem. The high-level A^* algorithm uses abstracted paths that are composed of sets of nodes expanded by the low-level algorithms during the search process. As an example of a resource allocation problem, we discuss and analyze the so called bidirectional search. More examples are presented in Chapter 6.

In some situations, A^* provides the optimal resource allocation policy in the sense of Definition 2.9.1 in Section 2.9. Roughly speaking, this means that the set of nodes expanded by all the search algorithms together is minimal. This chapter presents a revised version of [3].

At the end of this chapter, we prove that an A^* that searches a *tree* and is guided by an admissible heuristic is the optimal algorithm in the sense of Definition 2.9.1. (Theorem 5.4.1). A somewhat weaker result follows when the heuristic can be nonadmissible (Theorem 5.4.2). The above results, Theorems 3.2.6, and 3.2.7 in Chapter 3 can be used to generalize the optimality theorems in [3].

5.1 INTRODUCTION

Let $G = (N, E)$ be a locally finite graph, where N is a set of nodes [1] 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 is associated with every edge in E. A search problem refers to finding a path, a sequence of nodes and edges, from a start node to a goal node in 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, for example, in bidirectional search using the same graph but different start and goal nodes, see

[1] Here and in Chapter 6, we use the symbol N instead of V.

e.g. [54, 21, 59, 69, 16, 43, 29].

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

Let us formalize the above cases by using search graphs. 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 edges in a set $E_{k,j}$ connect 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.

Assume that the original path finding problem has a search graph G. We model the subproblems by subgraphs G_i of G: $G = G_1 \cup G_2 \cup G_3 \cup \cdots \cup G_r$. The intersection of G_i and G_j $(i \neq j)$ is not necessarily empty. In addition to a search graph, a path finding problem has also a start node s and a set of goal nodes Γ.

Every subgraph 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. In general, 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 subgraphs G_i.

If several algorithms solving the same problem use their own search graphs G_i whose intersection is empty, then every G_i itself represents the original problem. In this case, calling the G_i's 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 $G = G_1 = G_2$ 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 algorithms can also be instances of a single one, for example, Dijkstra's shortest path algorithm or A^* [59, 25, 29].

Usually we may be able to say which nodes belong to which subgraph without starting any search. An example of this is 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 Section 6 and [4].

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 also Section 6 and [4].

Resource Allocation

Let a search problem in 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, and that information is programmed in the search algorithms.

Assume that the subproblems G_i are not independently solvable or we have fewer 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. Let us call this strategy a *resource allocation policy* among the subgraphs G_i, or the subproblems. In this work, we assume that only one machine is available for all the G_i's.

A resource allocation policy is to first search for a solution in $\bigcup_{i=1}^{r_1} G_i,\ r_1 < r$. 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 some of the graphs G_i have 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 ($i = 1, 2, \ldots, r$) 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.[2] 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_k ($k = 1, 2, \ldots, r$) that has a minimum total weight W_k 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. Or 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.

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 starting to search any other G_i ($i \neq k$). 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_i's correspond to paths in the A^* algorithm. We will formalize this in the next section.

It will turn out that the new modified strategy is optimal over the breadth-first strategy. Moreover, in some cases the new strategy is the best one available in the sense of Definition 2.9.1 in Section 2.9. This is possible because it has the same structure as A^* and "inherits" its optimality properties.

In the following, we will examine situations where the new resource allo-

[2] In general, we do not require that complete solution paths exist in a single graph.

cation strategy is the best available, and how to calculate its heuristic H by using the heuristic h related to A^*.

5.2 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_i's is not necessarily empty. Let there be algorithms $A_1, A_2, \ldots, A_r$ expanding nodes in $G_1, G_2, \ldots, G_r$, respectively. Some or all the algorithms can be identical. For example the bidirectional search, in Section 5.1, has $r = 2$ and algorithms A_1 and A_2, where it is possible that $A_1 = A_2$.

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

The breadth-first strategy for allocating computing resources among the G_i's, in Section 5.1, is: At τ, choose the $G_j(\tau)$ for which $|N_j(\tau)|$ is minimum and expand one successor to a node in $N_j(\tau)$. If a goal node is first found in $G_k(\tau^*)$, then $|N_k(\tau^*)|$ is minimized among the values $|N_i(\tau^*)|$ $(i = 1, \ldots, r)$.[3]

Let $N_i(\tau^1)$ and $N_i(\tau^2)$ be the sets of the nodes that an algorithm A_i has expanded at time τ^1, and $\tau^2 > \tau^1$. Assume that $|N_i(\tau^1)| = l$. 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 new node and there is an edge between $N_i(l)$ and $N_i'(l) = N_i(l+1)$ for all $l = 1, 2, \ldots$. For example, if A_i expands the start node s in G_i at τ^1, then $N_i(1) = \{s\}$ is a node in the above sense. 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 in G_i, etc.

Definition 5.2.1 *Let a graph $G = \bigcup_{i=1}^{r} G_i$, and $N_i(l)$ be the set of l expanded nodes in $G_i \subseteq G$. Call $N_i(l)$ a node. $N_i'(l) = N_i(l+1)$ is the only successor to $N_i(l)$ if and only if $|N_i(l+1)| = |N_i(l)| + 1 = l + 1$ for all $l \geq 1$.*

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

A sequence of the nodes and the edges, starting from $N(0)$, form an abstract, or a meta, path and is denoted by MP_i $(i = 1, \ldots, r)$. The cost of MP_i is the sum of its edge costs.

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 is equivalent to minimizing $|N_i(\cdot)|$. For example, if a goal is found

[3]Here, the weights of the nodes are assumed to be one, and the goal is not considered expanded.

first in G_k containing l^* expanded nodes, then the cost of the optimal meta path MP_k^* is l^*. Hence the above breadth-first search strategy among the G_i's can be interpreted as one finding a cheapest meta path MP_i starting from $N(0)$.

Let a graph $MG = \bigcup_{i=1}^{r} MP_i$. Actually, MG is a tree with a root $N(0)$. Let us define a domain of problem instances, cf. Section 2.9:

$$\mathbf{MI}_{AD} = \{(MG, s, \Gamma, H) \mid H(N_i) \leq H^*(N_i)\, \forall N_i \in MG\}. \tag{5.1}$$

From now on, let N_i represent $N_i(l)$ for all l whenever the meaning is clear. 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 still to be expanded in G_i, after l, before a solution path in G is found. Similarly, $H^*(N_i(l))$ is the minimum number of nodes to be expanded in G_i before a goal is found in G. If $H(N_i) \leq H^*(N_i)$ for all N_i, then H is admissible by Definition 2.3.2 in Section 2.3.

The breadth-first strategy for finding a cheapest meta path MP_i in MG has $H(N_i) = 0 \leq H^*(N_i)$ for all N_i, and it can be implemented as an A^* algorithm. Hence it is admissible on the domain $\mathbf{MI}_{AD}$ of problem instances by Theorem 2.3.3. However, if we can estimate an admissible $H(N_i) \geq 0$ for all N_i, then an A^* guided by it is also admissible on $\mathbf{MI}_{AD}$ by Theorem 2.3.3. Then Theorem 2.6.2 states that the A^* is optimal over, or largely dominates, the breadth-first strategy relative to $\mathbf{MI}_{AD}$.

Theorem 5.2.1 *Let A^* be the resource allocation policy for node expansions among the subgraphs G_i ($i = 1, \ldots, r$), or among the algorithms A_i. Assume that A^* uses the nodes and the meta paths in Definition 5.2.1 and is guided by a consistent heuristic H on MG of $\mathbf{MI}_{AD}$.*

Then A^* is admissible on $\mathbf{MI}_{AD}$ and optimal over any DPBF resource allocation policy (using H) also admissible on $\mathbf{MI}_{AD}$ that has the same tie-breaking rule as A^*, in the sense of Definition 2.9.1.[4]

Proof. The question of the tie-breaking rule will be discussed later, in the context of Theorem 5.3.2. It is necessary *only* if expanding a node in N_i can produce successors that are in N_j, $j \neq i$. The rest of the proof follows directly from Theorem 2.9.1. □

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 will expand in $\bigcup_{i=1}^{r} G_i$ is minimum in order to find a solution path. Then Theorem 5.2.1 states that the A^* defined above is the best resource allocation method among DPBF strategies that return an optimal (meta)path, provided that H is consistent, and if expanding a node in N_i cannot produce successors that are in N_j, $j \neq i$. Otherwise, the above holds relative to the common tie-breaking rule of the resource allocation methods.

Theorem 5.2.1 can be generalized to hold also when the above heuristic H is admissible and not consistent because the search graph is a tree. We

[4] DPBF strategies were characterized at the beginning of Chapter 3.

will prove this later in Theorem 5.4.1. Weaker versions of Theorem 5.2.1: Theorem 5.4.2, Theorem 3.2.6, and Theorem 3.2.7, may be considered if the above heuristic H is nonadmissible.

5.3 USING THE ORIGINAL HEURISTIC IN G

Let a search graph $G = \bigcup_{i=1}^{r} G_i$ and algorithms A_i be as before. Assume that the computational work of expanding a node in G 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 m to be the father of n when n was found for the first time. Let us arbitrarily assign $w(s) = a > 0$ to the start node s.

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 nodes, then let $h(n_i) = 0$ or the heuristic is estimated in some other way. Usually it may be hard to tell in advance whether a graph G_i has a goal node or not. However, we may be able to construct a heuristic $h_{\lhd}$ that only satisfies the triangle inequality but does not necessarily vanish at the possible goal nodes in G_i, see Section 3.1 for more details.

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 5.2.1. From now on, let us call the A^* used in resource allocation A^*_{MG}: It searches the tree MG of $\mathbf{MI}_{AD}$. The edge cost between a node N_i and its successor N_i' in MG is $C(N_i, N_i') = c(n, n') = w(n')$, $\forall n' \in N_i \; \forall i$.

Paths with Nodes in a Single Subgraph

If every G_i contains a complete solution path, then the subproblems represented by the G_i's can be solved independent of each other. The intersection of G_j and G_k ($j \neq k$), however, may or may not be empty. The bidirectional search, in Section 5.1, 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. If the intersection of G_j and G_k ($j \neq k$) is not empty, then let us define that algorithm A_j does not react in any way to the fact that some of the nodes in G_j can also be in G_k.

Theorem 5.3.1 *Let all the subgraphs G_i have the start and all the goal nodes. Suppose that A^*_{MG} uses the nodes and the meta paths in Definition 5.2.1. Assume that every path candidate P_i is in 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 G_i, then A^*_{MG} using an evaluation function*

$$F(N_i) = G(N_i) + H(N_i) = \sum_{j=1}^{K_i} w(n_i^j) + \min_{n_i^j \in N_i} \{h(n_i^j)\}, \tag{5.2}$$

where K_i nodes have been expanded in G_i, is admissible on $\mathbf{MI}_{AD}$. Moreover, A^*_{MG} is optimal over any DP resource allocation policy (using h) also admissible on $\mathbf{MI}_{AD}$ in the sense of Definition 2.9.1.

Proof. Let A^*_{MG} 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 $H(N'_i) = h(n'_i) \geq h(n_i) - c(n_i, n'_i)$ by the monotonicity of h. Hence $H(N'_i) \geq H(N_i) - C(N_i, N'_i)$ since $H(N_i) = \min\{h(m_i) \mid m_i \text{ expanded in } N_i\}$ and $C(N_i, N'_i) = c(n_i, n'_i)$. Thus H is monotone, consistent and admissible.

Hence A^*_{MG} is admissible in $\mathbf{MI}_{AD}$ and optimal over any admissible algorithm in $\mathbf{MI}_{AD}$ by Theorem 2.9.1. □

Theorem 5.3.1 also holds when the heuristic h is admissible because MG is a tree. We will prove this later in Theorem 5.4.1. Weaker versions of Theorem 5.3.1: Theorem 5.4.2, Theorem 3.2.6, and Theorem 3.2.7, may be considered if the heuristic h is nonadmissible.

Theorem 5.3.1 can be generalized in situations where a subgraph G_i contains fewer than all the goal nodes in G. Let $h(n_i)$ be calculated by using only the goals in G_i. Assume that a solution path has been found in another subgraph $G_k, k \neq i$. Then $H(N_i)$, based on $h(n_i)$, may easily be nonadmissible in cases, where the minimum distance from the expanded nodes to a goal in G_i is much longer than the one in G_k has been. But then, if the H values used in the search in G_k have been admissible, then Proposition 3.2.8 holds. In this situation, the otherwise nonadmissible H is actually better than an admissible heuristic (see the text below Proposition 3.2.8). Moreover, Corollary 3.2.2 states that in this case A^*_{MG} is also the optimal strategy among DP strategies in a set $\mathbf{A}_{gc}$. The strategies in $\mathbf{A}_{gc}$ are assumed to find an optimal (meta)path whenever A^*_{MG} does, see Section 2.9.

Theorem 5.3.1 does not require that the algorithms A_i, expanding nodes in the G_i's, must themselves use the monotone or admissible heuristic h. Only the resource allocation algorithm A^*_{MG} utilizes it. The A_i's can be any path searching algorithms, not necessarily admissible ones. Recall that there can also be a single algorithm for expanding all the nodes in the subgraphs G_i.

As an example, let us discuss how A^*_{MG} can be used as the bidirectional search in G, introduced in Section 5.1. The search proceeds as follows. First, A_1 and A_2 expand their start 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 A_1 now expands one of the successors to s_1, say s'_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 placed in CLOSED. Next, A^*_{MG} selects a node from OPEN for which the F value is minimum, etc.

The bidirectional (A^*_{MG}) search can have at least two stopping rules. (i) A^*_{MG} stops when A_1 (A_2) chooses γ_1 (γ_2) for expansion. (ii) A^*_{MG} stops if one of the algorithms, A_1 or A_2, tries to expand a node already expanded by the other.

Assume that we can calculate admissible heuristics h_1 and h_2 for all nodes in the sets N_1 and N_2, respectively, by using the start node of A_2 as a goal to nodes in N_1 and vice versa. Let A^*_{MG} use the above stopping rule (i) and assume that A_1 (A_2) does not react in any way to the nodes expanded and generated by A_2 (A_1). In this case, A^*_{MG} is the optimal resource allocation policy between A_1 and A_2 by the generalized version of Theorem 5.3.1. Moreover, if h_1 and h_2 are monotone and $A_1 = A_2 = A^*$, then the bidirectional search finds an optimal path to any node in G it expands. This follows from Theorem 2.7.3. Also, the complete solution path found in G (for example from s_1 to γ_1) is optimal. However, if the stopping rule (ii) is used, then the complete solution path may no more be optimal, for more delicate stopping rules guaranteeing the optimality of the complete path, see e.g. [54, 59, 43, 25, 29].

Paths with Nodes in Many Subgraphs

Let us remove an assumption of Theorem 5.3.1: "every path candidate P_i is in G_i". Hence solution paths can contain nodes in several subgraphs G_i $(i = 1, 2, \ldots, r)$. Subgraphs G_j and G_k $(j \neq k)$ can have common nodes. It is not necessary that the start node and a goal node are in every G_i. Moreover, 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 in which set N_j $(j = 1, \ldots, r)$ to place the successors. Let us first extend Definition 5.2.1.

Definition 5.3.1 *Let the sets N_i in Definition 5.2.1 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 type k if n is in G_k; this is denoted by n_k. It is possible that the type of a 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 which set N_k a node n' is placed in may depend on the type of its predecessor n. Hence the decision can be made 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*.

In the case of the bidirectional search, a node can have two types: It is in both of the (sub)graphs G_1 and G_2 provided $G = G_1 = G_2$. The types of the node are thus explicitly defined.

As another example, let us discuss the search graph defined in a d-dimensional rectangular grid mentioned in Section 5.1, see Section 6 for more details ([4]). The nodes in the grid, G, are vectors $n \in \mathcal{Z}^d$ whose components are integers. For example, the neighboring nodes in G are $n = (n_1, n_2, \ldots, n_d)$ and $n' = (n_1, n_2 + 1, \ldots, n_d)$. $G = G_1 \cup G_2 \cup G_4 \cup G_8 \cup \cdots \cup G_{\max}$ defines a hierarchy of grids as follows. The nodes in G_i $(i = 1, 2, 4, \ldots, \max)$ are vectors $\{n = (n_1, n_2, \ldots, n_d) \mid gcd(n_1, n_2, \ldots, n_d) = i\}$, 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_i's such that the expansion of a node in G_i can produce successors that are in G_j $(i \neq j)$.

An example of implicit problem division is also presented in Section 6 ([4]). Denote now by G^i the graph G_i in the graph hierarchy in 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' in N_i and sets i as the type of n'. If $k < j$ then n' is placed in 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 contains 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 successors n'_k in N_k where the type $k = 1, 2, \ldots, r$ can be different from i. The successors are now open nodes in N_k. The evaluation function $F(N_k)$ for every N_k is calculated as in Theorem 5.3.1.

The pseudocode for algorithm 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 in OPEN.
(2)  Place m in N(m.type) and calculate F(N(m.type)).
(3)  Select N(i) 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 choose an open node n in N(i).
(7)  IF n is a goal THEN exit successfully ELSE expand n.
(8)  IF N(i) has no more open nodes
     THEN remove N(i) from OPEN and place it in CLOSED.
(9)  FOR all the successors n' to n
(10)     IF N(n'.type) does not already exist
(11)     THEN Create and place N(n'.type) in OPEN.
(12)     IF N(n'.type) is in CLOSED THEN reopen it.
(13)     Place n' in N(n'.type) if it is not already
            an open node there, and calculate F(N(n'.type)).
(14) END FOR
(15) Go to line (3)
```

In the above pseudocode, there are no computational steps such as on lines (5.2)–(5.3) in the pseudocode of A^* in Section 2.1 despite that A^*_{MG} is an instance of A^*. This is mainly because the graph MG that A^*_{MG} searches is a tree. The second reason is the definition of the weight w at the beginning of Section 5.3: $w(n)$ is based on the father of n when n was found for the first time.

The meta paths are not explicitly formed in the above pseudocode, 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.

Node selections and expansions in the sets N_i (on lines (1) and (6)), and successor generations (on line (7)), can be done by algorithms A_i if wanted. There can also be only a single algorithm for expanding all the nodes in G. Furthermore, line (13) can be replaced by

```
(13')   Place n' in N(n'.type) if it is not already an open node
            in any N(i), and calculate F(N(n'.type))
```

Then a single node is open only in one N_i. In general, a node can have many types.

Now, let us say something about the optimality properties of A^*_{MG} in this situation. We have to assume that any algorithm comparable to A^*_{MG} must have the *same tie-breaking rule* as A^*_{MG}. The reason is the following.

A counterexample: Let an $A^*_{MG}(a)$ have a tie-breaking rule a. Assume that the $A^*_{MG}(a)$ finds a goal node in G_k in a set $N_k(l)$, that is, after l node expansions in G_k. Now, let another $A^*_{MG}(b)$ have a tie-breaking rule b, different from a. Let $A^*_{MG}(a)$ and $A^*_{MG}(b)$ behave in the same way until the set $N_k(l-1)$ is reached. Assume that there are several sets N_i $(i = 1, 2, \ldots, q)$ for which the F value is the same as $F(N_k(l-1))$. Then $A^*_{MG}(b)$ may select one of them, say $N_j \neq N_k$, such that expanding a node in N_j generates new open nodes into $N_k(l-1)$. It can happen that the algorithm expanding nodes in G_k first expands those new open nodes before the goal node in G_k is chosen. Hence $A^*_{MG}(b)$ can miss the optimal meta path since more than l nodes may be expanded in N_k before the goal is found. It follows that the optimality among A^*_{MG} algorithms depends on the tie-breaking rule. This is true even if the heuristic H guiding both algorithms is admissible.

The situation in the above counterexample reveals one more thing. Suppose that algorithm A_k expands nodes in G_k and $A_k \neq A_j$. Then, in the counterexample, A_k was assumed to be able to expand nodes n that *it has not generated before*; A_j has generated the n's. Hence, the algorithms expanding nodes in graphs G_i $(i = 1, 2, \ldots, q)$ in this section are more general than the algorithms discussed elsewhere in this work, cf. Constraint 4 at the beginning of Chapter 3.[5] This is unless we define that every A_i is a subroutine of some "global" node expansion algorithm.

Theorem 5.3.2 *Assume that A^*_{MG} uses the nodes and the meta paths in Definitions 5.2.1 and 5.3.1. Assume that every G_i does not necessarily contain the start node and any goal node. Let it be possible for path candidates in G to have nodes in different subgraphs G_i $(i = 1, 2, \ldots, r)$. Let the cost $C(N_i, N'_i)$, the heuristic $H(N_i)$ and $F(N_i)$ be the same as in Theorem 5.3.1. Let $h(n)$ be admissible on every node n in G.*

Moreover, assume that $h(n'_j) \geq H(N_j)$, where n'_j is the immediate successor to n_i, $i \neq j$, when A^*_{MG} places n'_j in N_j.

[5] The algorithms can also be such that they explore nodes instead of expanding them, see Constraint 3 in Chapter 3.

Then A^*_{MG} is admissible on $\mathbf{MI}_{AD}$ relative to its tie-breaking rule. Moreover, any DPBF resource allocation policy (using h) that is admissible on $\mathbf{MI}_{AD}$ with the same tie-breaking rule as A^*_{MG} will explore, in any instance $I \in \mathbf{MI}_{AD}$, all the nodes surely expanded by A^*_{MG}.

Proof. Let A^*_{MG} expand a node $n_i \in N_i$ and place its successor n'_j as an open node in N_j. If $i = j$ then Theorem 5.3.1, or actually, its generalization holds since h is admissible. Assume that $i \neq j$. If $h(n'_j) \geq H(N_j) = \min\{h(m_j) \mid m_j \text{ expanded in } N_j\}$, then n'_j does not change $H(N_j)$ after it is expanded. Hence H remains admissible.

Since any other algorithm admissible on $\mathbf{MI}_{AD}$ was assumed to have the same tie-breaking rule as A^*_{MG}, the above counterexample is impossible. The following Theorem 5.4.1 implies the rest. □

The assumption $h(n'_j) \geq H(N_j)$ is restrictive: no node that is placed in N_j whose predecessor is in $N_i, i \neq j$, can alter the heuristic $H(N_j)$. However, if this is not the case, then the heuristic $H(N_j)$ may become nonadmissible and, again, Theorem 5.4.2, Theorem 3.2.6, and Theorem 3.2.7 may be considered. However, if H has been admissible along the optimal meta path found, then A^*_{MG} is the optimal strategy among the DPBF strategies in the set $\mathbf{A}_{gc}$ by Corollary 3.2.2.

Usually, we can check whether the above assumption ($h(n'_j) \geq H(N_j)$) is true or not only after the generation of n'_j when A^*_{MG} has determined its type j. One solution to the problem is to let the algorithms A_i avoid node expansions that violate the admissibility assumption as long as it is possible. However, at some point we may have to violate the assumption if a solution path cannot be found in any other way.

Consider again the bidirectional A^*_{MG} search discussed below Theorem 5.3.1. Let A^*_{MG} now use the stopping rule (ii): A^*_{MG} stops if A_1 (or A_2) tries to expand a node already expanded by A_2 (or A_1).

Let n_1 and n_2 be nodes expanded by A_1 and A_2, respectively. Assume that we can calculate $h_1(n_1) = \min_{n_2 \in N_2}\{k'(n_1, n_2)\}$ for all the nodes n_1 expanded by A_1 at all times during the search, where the cost $k'(n_1, n_2)$ is a lower bound for the cost of an optimal path from n_1 to n_2. Moreover, let $h_2(n_2)$ be calculated similarly as above. Then A^*_{MG} using the stopping rule (ii) is the optimal resource allocation policy between A_1 and A_2 among admissible DPBF strategies by Theorem 5.3.2 relative to its tie-breaking rule.

The calculation of the above heuristics may be too slow in practise. If we avoid the above calculations, then H my easily become nonadmissible when A^*_{MG} uses the stopping rule (ii). Then, however, the A^*_{MG} is still optimal in a smaller set of algorithms by Theorem 3.2.6 or Theorem 3.2.7.

5.4 OPTIMALITY IN TREES

It is possible to generalize Theorems 5.2.1 and 5.3.1, and to complete the proof of Theorem 5.3.2 in situations where the heuristics H in the theorems are not necessarily monotone but are admissible. The reason is that the search graph MG of $\mathbf{MI}_{AD}$ is a tree. The following theorem formalizes this. The set of algorithms $\mathbf{A}_{ad}$ are the ones that always return an optimal solution when the heuristic is admissible, cf. Chapter 3. Let the set of problem instances be

$$\mathbf{I}^T_{AD} = \{(T, s, \Gamma, h) \mid h \leq h^* \text{ on } (T, \Gamma)\}, \tag{5.3}$$

where T is a directed tree: The root of T is the start node s and the edges from any node n in T are directed from n to $\mathrm{succ}(n)$ (successor nodes: see Section 2.1). Clearly $\mathbf{I}^T_{AD} \subseteq \mathbf{I}_{AD}$, defined in Section 2.9.

Theorem 5.4.1 *(cf. Theorem 2.9.1). Any DP strategy in $\mathbf{A}_{ad}$ (admissible on $\mathbf{I}_{AD}$) will explore, in every instance $I \in \mathbf{I}^T_{AD}$, all nodes surely expanded by A^*.*

Proof: Let $I = (T, s, \Gamma, h)$ be a problem instance in $\mathbf{I}^T_{AD}$. Assume that n is surely expanded by A^* and not explored by B. Therefore, there exists a path P_{s-n} such that

$$g^*(n') + h(n') < C^* \quad \forall n' \in P_{s-n}, \tag{5.4}$$

where $g^*(n') = g(n')$ since T is a tree. Let B be in $\mathbf{A}_{ad}$, namely, halting with cost C^* in I. We now create a new tree T', similarly as in Figure 2.1, by adding to T a goal node t with $h(t) = 0$ and an edge from n to t with a nonnegative cost $c(n, t)$:

$$c(n, t) = C^* - g^*(n) - \epsilon > 0, \tag{5.5}$$

where $0 < \epsilon < \min_{n'' \in P_{s-n}}\{C^* - g^*(n'') - h(n'')\}$. Let a new instance be $I' = (T', s, \Gamma \cup \{t\}, h)$. This construction creates a new solution path P^*_t with a cost

$$C(P^*_t) = g^*(n) + c(n, t) = g^*(n) + C^* - g^*(n) - \epsilon = C^* - \epsilon. \tag{5.6}$$

The heuristic $h(m)$ is admissible for any m not in P^*_{s-n} since we kept the h values of all the nodes in T unchanged and T is a directed tree. It remains to verify the admissibility of h for every node n' in P^*_{s-n}:

$$\begin{aligned} h(n') &= C^* - g^*(n') - (C^* - g^*(n') - h(n')) \leq C^* - g^*(n') - \epsilon \\ &= k(n', n) + C^* - g^*(n) - \epsilon = k(n', n) + c(n, t) = k(n', t) \end{aligned} \tag{5.7}$$

Thus, the new instance I' is in $\mathbf{I}^T_{AD}$.

In searching T', algorithm A^* will find the new optimal path P^*_t with cost $C(P^*_t) < C^*$ because

$$f(t) = g^*(n) + c(n, t) + h(t) = C^* - \epsilon \tag{5.8}$$

and, so, t is reachable from s by a strictly C^*-bounded path, which ensues its selection. Algorithm B, being deterministic, must behave exactly in the same way as in solving I, if I' is given as input to B: It avoids exploring n in T' and halts with cost C^*, which is higher than that found by A^*. This contradicts the assumption that B is in $\mathbf{A}_{ad}$. $\square$

We can study algorithms in problems, where heuristics are nonadmissible by using Theorems 3.2.6 and 3.2.7 or by using the following theorem. In Theorems 3.2.6 and 3.2.7, A^* was compared to algorithms whose evaluation function for a node depends *only* on the current path to that node. The following theorem holds in more general situations.

Theorem 5.4.2 *Let $\mathbf{I}^T = \{(T, s, \Gamma, h)\}$, where h is a possible nonadmissible heuristic. Let $\Delta = \max_{m \in T}\{h(m) - k(m, \Gamma)\}$, where $k(m, \Gamma)$ denotes the cost of the shortest path from m to the set of goal nodes Γ. Then any DPBF strategy $B \in \mathbf{A}_{ad}$ will explore, in every instance $I \in \mathbf{I}^T$, all the nodes in $\mathbf{N}^M_{g+h} \cap \{n \mid h(n) \geq \Delta\}$ provided that $\forall m_1, m_2 \in G$* of *$I \in \mathbf{I}^T$:*

$$h(m_1) > h(m_2) \Rightarrow f_B(m_1; H, I(\cdot, h)) > f_B(m_2; H', I(\cdot, h)) \tag{5.9}$$

independent of the current search histories H and H' concerning I. (cf. Theorem 3.1.2).

Proof: Let $I = (T, s, \Gamma, h)$ be a problem instance in $\mathbf{I}^T$. If $\Delta \leq 0$, then $I \in \mathbf{I}^T_{AD}$, and by Theorem 5.4.1 we are done. Assume that $\Delta > 0$.

Let A^* expand a node n in $\mathbf{N}^M_{g+h} \cap \{n \mid h(n) \geq \Delta\}$ of I. Moreover, assume that B does not explore n. Let

$$0 < \epsilon < M - g^*(n) - h(n), \tag{5.10}$$

where M is the minmax value of A^* in T. Let us create a new tree T_1, which is the same as T except two modifications. First, the heuristic values of the nodes are changed:

$$h_1(m) = \max\{0, h(m) - \Delta\} \tag{5.11}$$

for all the nodes in T_1 except n, for which $h_1(n) = h(n) - \Delta + \epsilon$. The second modification is that the costs of the edges between n and all its successors n' are $c_1(n, n') = c(n, n') + \epsilon$, where $c(n, n')$ denotes the corresponding edge cost in T. Denote a new problem instance by $I_1 = (T_1, s, \Gamma, h_1)$. It follows that $I_1 \in \mathbf{I}^T_{AD}$.

Since A^* expanded n in T: $f(n) = g^*(n) + h(n) < M$ by Theorem 2.8.6. From the construction of h_1, it follows that the minmax value M_1 for A^* in T_1 satisfies $M_1 \geq M - \Delta$. A^* will expand n also in T_1 since $f_1(n) = g^*(n) + h_1(n) < M - \Delta$ by the definition of ϵ. Since B does not explore n in T, is deterministic and f_B satisfies the assumption of the theorem, B does not explore n in T_1 either if I_1 is given as input to B, cf. the proof of Theorem 3.1.2.

Now, $I_1 \in \mathbf{I}^T_{AD}$ and we can utilize the proof of Theorem 5.4.1 directly to create an instance $I' \in \mathbf{I}^T_{AD}$, where B will not find the optimal solution path (when I' is given as input to B) whereas A^* will. This contradicts the assumption that $B \in \mathbf{A}_{ad}$. $\square$

Here, if Δ is close to zero, then the set $\mathbf{N}^M_{g+h} \cap \{n \mid h(n) \geq \Delta\}$ may be close to $\mathbf{N}^M_{g+h}$, cf. Theorem 3.1.2.

A final, technical, comment concerns the proofs of Theorems 5.2.1, 5.3.1, and 5.3.2. It is the same independent of which theorem (Theorems 2.9.1, 5.4.1, 5.4.2, 3.2.6, or 3.2.7) is used in the proofs.

In all the proofs a new graph G' is created from G by attaching to a node n in G an edge and a new goal node t, or new goals t_1 and t_2. Now, n and t (t_1 and t_2) are *sets of nodes* N_i ($i = 1, \ldots, r$) and every N_i has only one successor in G, according to Definition 5.2.1. Formally, G' is constructed from G as follows.

Let the set corresponding to n, in the above proofs, be N^n and the graph (tree) corresponding to N^n be G^n. First, delete all the children of N^n. Second, place p_c new open nodes in N^n, where the new goal t (t_1 or t_2) is one of them. Construct a path P starting from n and going via the $p_c - 1$ nodes to t (t_1 or t_2) by adding p_c edges between them, and set a cost of every edge as $c(n,t)/p_c$. Next, construct a meta path of Definitions 5.2.1 and 5.3.1 starting from N^n with p_c node sets: Every node set in the meta path is $N_p^n = N^n \cup \{n_p\}$, where the set $\{n_p\}$ contains all the nodes in P that has been explored when N_p^n is formed. Moreover, let the nodes n_p be such that the algorithm exploring nodes in N^n explores them directly after n before finding the goal t (t_1 or t_2). In other words, t (t_1 or t_2) is the first goal found in the search process and no additional nodes, except the newly created ones and n, are explored before t. The cost between any two succeeding nodes in the meta path is thus $C(N_p^{n'}, N_p^{n''}) = c(n,t)/p_c$, where $c(n,t)$ is the cost of the new (meta)path in G' used in the proofs. In such a way, the cost of the newly included meta path is the required one, in the proofs.

5.5 CONCLUSIONS

Suppose that we have divided a 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 solution paths can have nodes in several subgraphs. In other words, the subproblems are not required to be solvable independent of each other. Every subproblem can have a different algorithm for expanding its nodes.

In this chapter, we showed that A^* can be used as a resource allocation policy for node expansions among the subgraphs. We used bidirectional search as an example of a resource allocation problem between two path finding algorithms.

In some cases, (Meta)A^* is the optimal resource allocation policy among admissible DPBF strategies.[6] This requires the possibility of underestimating, during search, the number of the nodes still to be expanded at least in that subgraph, where a goal is found, see Corollary 3.2.2 in Chapter 3, Theorems 5.2.1, 5.3.1, 5.3.2, and Theorem 5.4.1.

In Theorem 5.4.1 we showed that A^* guided by an admissible heuristic is the optimal algorithm, when the search graph a directed tree, cf. Theorem 2.9.1.

If every path candidate can be only in a single subgraph and every subgraph contains the start node and all the goal nodes, then one method of estimating the heuristic guiding the resource allocation process is to use an admissible heuristic h assigned to the nodes in the original graph. The admissibility of h guarantees the existence of an optimal resource allocation strategy (Meta A^*), see Theorem 5.3.1 and the text below it.

In general, the situation is more complicated. Additional assumptions are needed for the existence of an optimal resource allocation policy (Meta

[6] DPBF strategies were defined at the beginning of Chapter 3.

A^*). It is not always possible to know *a priori* whether these constraints are satisfied or not. Moreover, the optimality depends on the tie-breaking rule of the resource allocation strategy. See Theorem 5.3.2 and the text below it.

If the heuristic guiding the resource allocation process has been nonadmissible along the found (meta)path, then Meta A^* *is still* an optimal resource allocation strategy among certain admissible DPBF and BF* strategies, see Theorem 3.2.6 and Theorem 3.2.7 in Chapter 3.

6 HIERARCHICAL A^* BASED ROBOT PATH PLANNING — A CASE STUDY

When an automated production line is operating, it is often very expensive to stop it, for example, to re-program robots to deal with new products or product variants. A more economical way is to generate new movements for the robots off-line, using a simulator, and then download the programs into the robot controllers. Minor modifications to the programs, e.g., fine tuning the robot wrist and gripper movements may still be necessary but the time the production line needs to be stopped is small compared to the manual teaching of the robots.

Point-to-point path planning in known environments refers to finding a path from an initial robot configuration to a desired goal configuration such that the robot does not collide with itself or obstacles around it, see [44, 35]. This problem occurs in industry, e.g., in spot welding, riveting, and pick and place tasks, see Figure 6.1.

Usually it is impossible to know in advance how coarsely robot movements can be discretized in order to find a collision free path in the presence of obstacles, i.e., a sequence of discrete robot positions, points, that can be connected by straight collision free line segments. A solution to the problem is to introduce a new method of constructing hierarchical path planning algorithms. It is based on the Meta A^* structure discussed in Chapter 5.

We test four hierarchical path planning algorithms, two of which are based on the Meta A^* algorithm, using five simulated robot workcells. The simulations suggest that the Meta A^* based planners, on average, find paths faster and consume less memory than the two comparable algorithms. This chapter presents a revised version of [4].

6.1 INTRODUCTION

Figure 6.2 illustrates a path planning example of a 2-joint robot manipulator. The left picture shows the robot's *work space*. There the robot is shown at several different positions with some gray obstacles. "S" is a start position and "E" is a goal position. The positions between "S" and "E" are snapshots along a collision free path. The right picture shows a *configuration space* C of the 2-joint robot manipulator. A point $q \in C$ is called a *configuration* and it can be represented by the 2-dimensional vector whose components are the robot's joint angle values.

Subset of C consisting of all the configurations where the robot has no contact or does not intersect the obstacles is called *free space* and denoted by C_{free}, see white areas in the right picture. The complement of C_{free} consists of obstacles and is denoted by C_{obst}, see black areas. To determine whether a configuration q is in C_{free} or in C_{obst}, in a robot simulator, geometric calculations are needed. This is called *collision testing*. A collision free path from "S" to "E" is a continuous sequence of the configurations in C_{free}, see the thin black line.

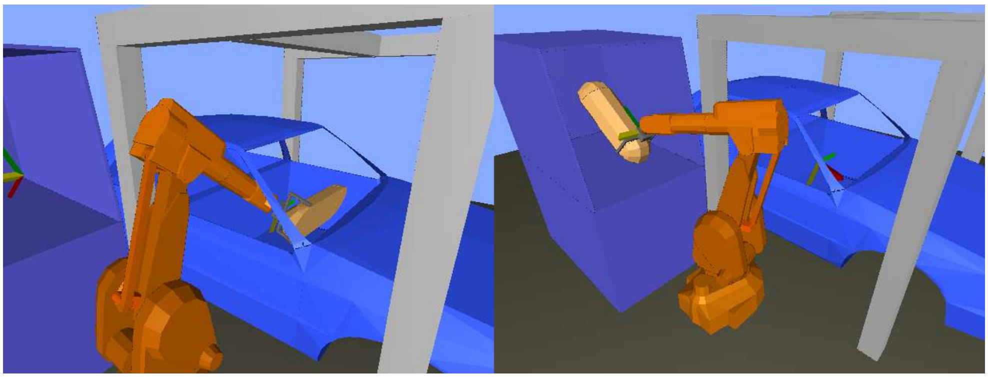

Figure 6.1: An IRB 2000 industrial robot assembling a dashboard into a car body.

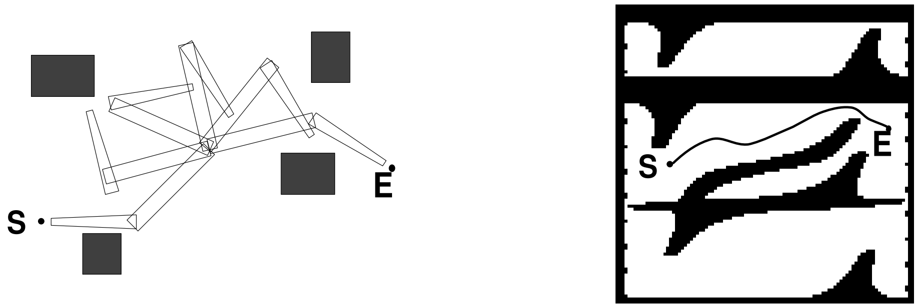

Figure 6.2: A 2-joint robot manipulator in its work space with obstacles (left picture). A configuration space and a collision free path (right picture).

Robot path planning has received much attention over the years and many different approaches have been presented, see for example [44, 35]. They can be roughly categorized into *cell decomposition*, *potential field*, and *roadmap* or *skeleton* methods. Most of these methods search a collision free path in the discretized configuration space. A detailed survey of path planning problems and algorithms can be found in [35].

Cell decomposition approaches are based on decomposing the set of free configurations into simple non-overlapping regions called cells, see e.g. [26, 9, 23, 35, 44]. If the cells are in the work space then one has to model so called swept volumes, sets of free configurations, around robot manipulators [44]. In the configuration space, the cells are often rectangular regions. They usually form a tree structure. If a cell is not free then it is subdivided into some smaller cells by creating new leaf nodes in the tree, etc. After the division process is completed the leaf nodes refer to the biggest cells in free space at a given resolution. The adjacency of cells is represented in a connectivity graph that is then searched for a collision free path. These approaches use sophisticated algorithms for finding adjacent free cells. Furthermore, to check whether a cell is free requires distance calculations between robots and obstacles [23, 35].

Roadmap or skeleton approaches attempt to map a set of feasible motions onto a network of one-dimensional lines, called roadmap, skeleton, or subgoal network, see e.g. [14, 39, 44, 35]. One node in the roadmap graph or the subgoal network corresponds to one robot configuration in free space. The nodes are generated by "intelligent" sampling in the search space. This can be done either by preprocessing the search space or during path planning. Local planning refers to finding a collision free path from one node or subgoal to another. Searching a collision-free path on the whole subgoal network (graph) is called global planning. In these approaches, local planners usually do not know how difficult it is to find a path from one subgoal to another. Therefore the algorithms have to decide when to stop searching for one subgoal and to choose another instead. Moreover, allocation of computational resources between global and local planning has to be carefully implemented.

In potential field methods, path candidates evolve towards a goal guided by a potential function. The paths proceed in directions of negative gradient of the potential function, see e.g. [15, 71, 36, 63, 44, 35]. The potential function is zero at the goal whereas on the boundaries of the space and obstacles it has a positive value, e.g. one. In general, potential functions are solutions of a Laplace partial differential equation with the above boundary conditions. It can be shown that such potential functions have their only minima at the goals. Hence after the potential function has been found, path planning is a fairly trivial task. The problem here is, however, that obstacles can be geometrically complicated. It follows that potential functions can be calculated only numerically. Solving the corresponding Laplace equation numerically is possible only in low dimensional spaces (2 or 3) using coarse resolutions. This is because calculations need all the points in the discretized space. A common solution is that potential functions are most often estimated either by using spherical approximations of obstacles or with the help of a few nearest obstacles. Then they have simple algebraic forms composed of two terms: an attractive force guiding paths towards goals and a repulsive force pushing them away from obstacle surfaces. A drawback is that the approximated potential functions usually have other, local, minima except the one at the goal. Hence sophisticated algorithms are needed to enable path candidates to escape from those local minima.

We have not compared our results with the above methods. This is mainly because there are very many variants of them presented in the literature. We have briefly mentioned here only the basic ideas behind those methods.

Related Work

In this chapter, the search space is a d-dimensional grid, where d is the dimension of the configuration space C, i.e., the number of joints or *degrees-of-freedom* (DOF) of a robot. A configuration $q \in C$ is represented as a d-dimensional vector $(q_1, q_2, \ldots, q_d)$. The configuration space is discretized so that each q_i has m_i distinct values: $q_i^1, q_i^2, \ldots, q_i^{m_i}$. Hence C is a graph of $m_1 \times m_2 \times \cdots \times m_d$ vectors. The graph has edges that connect each node q to its neighbors. For example, a neighbor of $(q_1, q_2^1, \ldots, q_d)$ is $(q_1, q_2^2, \ldots, q_d)$. A collision free path (continuous sequence of the configurations in C_{free}) is represented by a sequence of nodes and edges in free space C_{free}.

The theme in this chapter is hierarchical A^* (HA) based path planning. Path candidates generated by A^* from the start node to nodes n can be seen as "rubber bands". Their elasticity and potential energy is controlled by the function f. A^* tries to "stretch" these rubber bands by exploring successor nodes to n that are in free space. For deepest collision free nodes m found so far, the heuristic $h(m)$ resembles the approximated potential function discussed above. The $g(m)$ value penalizes the length of the path from the start to m: Generating longer paths consumes more energy. However, A^* explores only a subset of the search graph to find a collision free path unlike the numerical solution of the corresponding Laplace equation. Despite this A^* always finds a shortest or a cheapest collision free path if one exists (as long as the heuristic h is accurate enough). Methods that use approximated potential functions with local minima can miss existing paths.

Warren [73] presents a very simple HA based search method for robot path planning. First, a coarsely discretized configuration space is searched, using big step sizes, to find a collision free path (the above numbers $m_1, m_2, \ldots, m_d$ are small positive integers). If such a path is not found, then a graph with finer discretizations is searched, using smaller step sizes ($m_1, m_2, \ldots, m_d$ are increased), etc. If a collision free path exists on a coarse resolution graph, then it will be found very quickly. However, if collision-free paths exist only on very fine resolution graphs then the search process often does unnecessary work. For example, there may be a narrow passage somewhere between the start and the goal configurations. To go through the passage the search process has to use small step sizes. After that the search continues on the fine resolution graph even when there is enough room for bigger step sizes again.

Another HA based path planner in [6] searches both coarser and finer resolution graphs at the same time and changes step sizes adaptively during the search. The search process always tries to increase step sizes whenever there is enough free space around. When collisions occur the planner decreases step sizes near obstacles surfaces, see Figure 6.4 in Section 6.4. The algorithm needs the coarsest and the finest allowed graph resolutions as parameters. If the coarsest resolution is set as the same as the finest resolution then this method equals A^* with the same resolution. If the finest resolution is set much smaller than necessary to find a path, then this algorithm can do much unnecessary work.

We have developed two new HA based path planning algorithms from the planner in [6]. In the new algorithms, nodes in the search graph are distributed into several subgraphs — before or during the search depending on the particular algorithm. In general, the search process chooses first *the most promising subgraph* and only after that it starts to find collision free nodes (configurations) in the chosen subgraph. Subgraphs can form, for example, a hierarchy of graphs from coarser to finer discretizations of the search space. In this way we can control node explorations more freely or adaptively than in the above two HA based algorithms in [73, 6]. In the HA planner [6], adaptivity is achieved only when a single node is being explored, whereas our new algorithms "see" also sets of nodes in the subgraphs. The new algorithms use A^* in a novel way, which we call Meta A^*, for more details see Section 5.

A general strategy in all our algorithms is that we prefer searching coarse resolution graphs by exploring coarse resolution nodes whenever possible.

Hierarchical cell decomposition methods are somewhat related to the path planners tested here. However, the latter methods find collision free paths that are lines in C_{free} whereas the cell decomposition methods find volumes, or voxels, in C_{free}. A robot path can proceed in many ways through these voxels. However, the organization of the hierarchical search in the tested algorithms can also be applied to cell decomposition representations of search spaces. Then a node corresponds to a cell and the connectivity graph is constructed by the hierarchical search.

The tested methods differ from roadmap or subgoal networks although they all generate paths that are one dimensional lines. The roadmap methods generally do not know whether a path exists or not between two subgoals in the beginning. Local planners find this out. Here we do not have any subgoals nor "intelligent" local planning in this sense. We simply find a collision free path along one coordinate axis from one configuration to a neighboring configuration in C. When a configuration q is included in the search tree, then we already know that there exists a collision free path from the start node to q. However, it is straightforward to apply our HA methods to roadmap methods as local planners: to find collision free paths between any two subgoals chosen by a global planner.

First, we discuss how search graphs of the path planning problems are hierarchically divided into subgraphs. Second, we describe the idea and give a pseudocode for the new path planning algorithms that we compare to each other empirically. Third, we describe path planning tasks and present simulation results. Finally, we discuss the results and make conclusions.

6.2 SUBDIVIDING A PATH PLANNING PROBLEM

Here we define search graphs for hierarchical path planning and discuss how to subdivide a path planning problem.

Let d be the degrees-of-freedom of a moving object. If the object is flying freely in a 3 dimensional space, then its DOF is six — three linear directions and rotations around three linearly independent axes. In case of robot manipulators, the DOF is the number of the robot's joints — linear or rotary.

Every discretized collision free configuration q of an object, see Section 6.1, is mapped onto a corresponding node n in the search graph. The nodes in the search graph are vectors $n \in \mathcal{Z}^d$, whose components are integers. The graph has edges that connect neighboring configurations. For example, neighboring configurations $q = (q_1, q_2, \ldots, q_d)$ and $q' = (q_1, q_2 + stepsize, \ldots, q_d)$ are mapped onto $n = (n_1, n_2, \ldots, n_d)$ and $n' = (n_1, n_2 + 1, \ldots, n_d)$, respectively. In general, we define that two nodes are neighbors if they differ from each other only in one component by exactly one. Hence each node has $2 \cdot d$ neighbors. Costs assigned to edges between any two neighbors are one. We call this a *basic graph* G^1 and its resolution is one.

Let a graph of the finest resolution be G^1. A graph in the second level of resolution, G^2, has nodes $n = (n_1, n_2, \ldots, n_d) \in Z^d$ whose components

$n_1, n_2, \ldots, n_d \in Z$ are divisible by 2. Nodes in the next level graph, G^4, are those whose components are divisible by 4, and so forth. In general, the nodes in the graph G^i are $\{n = (n_1, n_2, \ldots, n_d) \mid gcd(n_1, n_2, \ldots, n_d) = i\}$, where $gcd(\cdot)$ denotes the greatest common divisor of the arguments. We call the numbers $i = 1, 2, 4, \ldots, 2^k, \ldots, 2^{\max}(= rmax)$ the *resolutions of the graphs*.

There are two kinds of edges in the graph hierarchy: edges that connect neighboring nodes in a single G^i and edges that connect nodes in graphs G^i and G^j, $i \neq j$. In the former case, we define the neighboring nodes, both in G^i, similarly as in the basic graph G^1. The latter case depends on how an algorithm expands nodes. In any case, before n' can be accepted as a successor, collision tests (Section 6.1), are performed to be sure that there are no obstacles along a path from n to n'. If n' is accepted then an edge with an appropriate cost is created between n and n'.

In general, when a node n in G^i is expanded, an edge is created from n to each n' where n' is in G^k for $k \geq i$, unless there is an object near n. If n is near an object surface, then n' can also be in G^j where $j < i$. Again, n and n' differ from each other by one component only. The detailed successor generation process used in three of the four algorithms in this chapter is presented in the Appendix.

Figure 6.3 illustrates some nodes and edges in graphs whose resolutions are $1, 2, 4$ and $8 = rmax$, see the nodes labeled by white circles, triangles, white rectangles and black rectangles, respectively. Thin lines show some of the edges. A cost between neighboring nodes n and n' is $c(n, n') = rmax/dist(n, n')$, where $dist(n, n') = ||n - n'|||_1$, *the Manhattan distance*. The distance is measured along the basic graph G^1 with unit edge costs. The costs are shown near the edges. We have used these costs in the HA planner in [6].

Let a search graph G be composed of the graphs in the above hierarchy: $G = G^1 \cup G^2 \cup G^4 \cup \cdots \cup G^{rmax}$. We can think of the graphs G^i as forming a subdivision of the search problem, where each G^i is a search graph of one subproblem. In general, we define a subdivision of G as $G = G_1 \cup G_2 \cup G_3 \cup \cdots \cup G_r$ as in Chapter 5. Thus, in the above example, $G_1 = G^1, G_2 = G^2, G_3 = G^4, \ldots, G_{k+1} = G^{2^k}, \ldots, G_r = G^{rmax}$. We will describe another subdivision of G later in connection with the tested algorithms. Subproblems with search graphs G_j are not required to be solvable independent of each other.

There are many ways to choose in which order the nodes in the above graph hierarchy are expanded. One way to solve the problem 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 $\bigcup_{i=1}^{r_2} G_i,\ r_1 < r_2 \leq r$, etc. This strategy can be called *greedy* or a *depth-first search* among the graphs G_i. If the original graph G has an infinite number of nodes, then it may happen that this method fails to find a solution path even if it exists.

Another way is to search all the graphs $\bigcup_{i=1}^{r} G_i$ more or less at the same time. We assign to each node $n_{i,k}$ in G_i a weight $w_{i,k}$. A weight can measure the computational work of expanding a node or the *importance* of a node. In general, we give more computational resources to expansions of

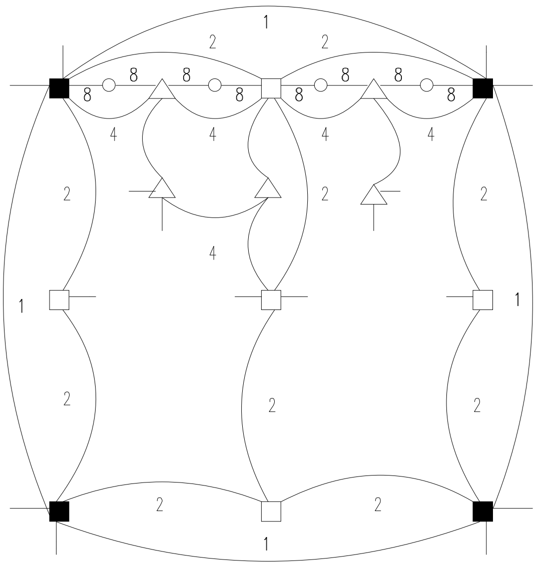

Figure 6.3: Hierarchy of graphs at resolutions 1,2,4 and 8 $= rmax$. Nodes labeled by different symbols, such as rectangles, triangles, and circles, belong to graphs with different resolutions.

important nodes. Here, nodes in coarser resolution graphs are more important than nodes in finer resolution graphs. This is mainly because there are significantly fewer of the former nodes than the latter. Thus there is a better possibility to explore most of the coarser resolution graphs.

An algorithm can minimize the numbers or *total weights* $W_i = \sum_{k=1}^{K_i} w_{i,k}$ of the expanded nodes in every G_i in order to find a solution path on G. K_i is the number of the expanded nodes in G_i at a given time. This is done by first choosing a graph G_j that has a minimum total weight W_j so far. After this, one of the nodes in G_j is expanded, W_j is updated and another G_i with the minimum W_i is chosen, and so forth. This strategy corresponds to a *breadth-first search* among the graphs G_i.

In this chapter, node expansions in a single graph G_i are done by the A^* algorithm. In the following, this should not be mixed with the Meta A^* algorithm that first chooses a graph G_i before any node expansions in G_i can happen, like in the above breadth-first strategy. Actually Meta A^* has more or less the same structure as A^*, cf. also Section 5.

6.3 THE META A^* ALGORITHM

Let us modify the above breadth-first strategy among the graphs G_i. If previous node expansions in G_i have generated paths that “seem to make regular progress towards a goal”, then let us explore more nodes in G_i before starting to search any other G_j, $j \neq i$. One way to implement this strategy is to estimate the remaining cost of the nodes still to be expanded in every $G_i, i = 1, 2, \ldots, r$, before a solution is found. The estimate, say H_i, works here similarly as a heuristic h in the A^* algorithm.

Assume that $n_{i,k}$ is a node in G_i that has a weight or a cost $w_{i,k} = c(m, n_{i,k})$, the cost of an edge between the father m of $n_{i,k}$ and $n_{i,k}$. After $n_{i,k}$ is expanded the W_i value of G_i is updated: $W_i = W_i + w_{i,k}$. Let $h(n')$ be an estimate of the cost of the cheapest path from a successor node n' of $n_{i,k}$ to a goal. The value $h(n')$ may be less than $h(n_{i,k})$. This indicates that a path from the start node via $n_{i,k}$ to n' has made progress towards the goal. Now we can construct the estimate H_i, concerning the graph G_i, by using the $h(n')$ values: $H_i = \min\{c(n_{i,k}, n') + h(n')\}$ if n' is not yet expanded in G_i. Finally, all the graphs $G_i, i = 1, 2, \ldots, r$, are ranked according to increasing $F_i = W_i + H_i$. This implements the above modified strategy.

In the new strategy, called Meta A^*, a graph G_j with the minimum value of F_j is first selected, then one of its nodes is expanded and F_j is updated. Then another G_i is selected and so forth. This differs from the earlier breadth-first strategy only by the estimate H.

If the H_i's underestimate the remaining cost of the nodes still to be expanded in G_i for all i, then Meta A^* is a better algorithm than the breadth-first strategy since the latter has $H = 0$. Actually, then Meta A^* is be the best strategy available, using the H_i's, for minimizing the W_i's of the expanded nodes among the graphs $G_i, i = 1, 2, \ldots, r$. The best strategy means, roughly speaking, that the sum of the weights of the expanded nodes in the whole search space $G = \bigcup_{i=1}^{r} G_i$ is minimized. This is possible because Meta A^* has the same structure as A^* and "inherits" its optimality properties. For more details, see Section 5.

From now on, we will apply the terminology presented in Section 5. For example, let the start node $s = n_{i,1}$ be in G_i. Expanding s generates successor nodes that are in graphs G_m, where m can be any number in $\{1, \ldots, r\}$. The expanded $n_{i,1}$ is placed on $N_i(1)$. $N_i(1) = \{n_{i,1}\}$ is now a node used by Meta A^*. Then assume that n_m, a successor to s, is expanded and its successors generated as above. If n_m is in G_m where $m \neq i$ then n_m is placed on $N_m(1)$: $N_m(1) = \{n_{m,1}\}$. On the other hand, if $m = i$ then $N_i(2) = \{n_{i,1}, n_{i,2}\}$ is a successor to a set $N_i(1)$, and so forth.

We write N_i instead of $N_i(l)$ for every l whenever the meaning is clear. In principle, Meta A^* maintains OPEN and CLOSED sets similarly as A^*. OPEN and CLOSED of Meta A^* are *sets of sets*:

$$\text{OPEN} = \{N_i^o \mid n \in N_i^o \text{ if } n \text{ is open in } G_i\}.$$

In the above definition, n is open in G_i if it is not yet expanded and its father has been expanded. In A^*-terminology, this means that n has been generated in the expansion process of its father. Similarly, a CLOSED set of the Meta A^* is:

$$\text{CLOSED} = \{N_i^c \mid n \in N_i^c \text{ if } n \text{ is expanded in } G_i\}.$$

Meta A^* always examines the nodes of the "most promising" set N_i^o based on the evaluation function $F(N_i^o) = G(N_i^c) + H(N_i^o)$. $G(N_i^c) = W_i$: the sum of the weights, or the total weight, of the nodes expanded so far in G_i. $H(N_i^o)$ is an (under)estimate of the remaining cost $H^*(N_i^o)$ of the nodes still to be expanded in G_i before a solution path on G is found.

After Meta A^* has chosen a particular N_i^o then a node $n \in N_i^o$ is chosen for expansion based on a local evaluation function $f_i(n)$. In general, every G_i can have its own private search algorithm A_i (using f_i) that will generate successors to n. In this chapter, there is only a single algorithm expanding nodes in G, namely A^*. Meta A^* then removes n from N_i^o and places it in $N_i^c(l+1)$, a successor set to $N_i^c(l)$. The successor nodes to n, depending in which G_j they are, are placed in N_i^o that is now a successor to $N_i^c(l+1)$, or in another $N_j^o,\ j \neq i$.

There are many possible evaluation functions $F(N_i^o)$ for Meta A^* using different weights $w_{i,k}$ for the nodes $n_{i,k} \in G_i$. They all yield different algorithms that usually can be compared to each other only empirically.

The following pseudocode describes Meta A^*. It is a variant of the pseudocode in Section 5, now applied to robot path planning. OPEN refers to the set OPEN of the Meta A^*, G(i) is G_i, and N(i) is $N_i^o \cup N_i^c$. F(N(i)) = G(N(i)) + H(N(i)) for each element (or set) N(i). G(N(i)) is the sum of the weights of the expanded nodes in N(i). H(N(i)) is an estimate of the remaining cost of the nodes still to be expanded in N(i). $f(n) = g(n) + h(n)$ is the value of an evaluation function for a single node n. Before Meta A^* starts, at least one set N(i) has to contain an open node, e.g., a start node s, such that OPEN is not empty.

THE META A^* ALOGORITHM

```
 1: Choose N(i) containing open nodes from OPEN
    for which F(N(i)) is minimum;
 2: IF no N(i) is found on line (1) THEN Exit with failure;
 3: ELSE Choose an open node n from N(i)
          for which f(n) is minimum;
 4: END IF
 5: IF n is a goal THEN Exit successfully;
 6: Expand n, and generate successors n' to n,
          by Rules 1-3 in Appendix;
 7: FOR all the successors n' DO
 8:     IF n' is not already in any set N(i)
 9:     THEN
10:        Determine in which G(j) n' is;
11:        IF N(j) does not yet exist
12:        THEN Create an empty N(j) and place it in OPEN;
13:        Place n' in N(j) and calculate f(n') and F(N(j));
14:     END IF
15: END FOR
16: Go to line 1
```

Note that no CLOSED set is implemented. If N(i) ($= N_i^o \cup N_i^c$) does not have any nodes that can be expanded, then line (1) behaves as if N(i) were closed. Furthermore, the meta paths composed of the N(i)'s are not explicitly created since the only successor of N(i) is N(i) ∪ n', where n' is the new open node in G(i), see line (13).

In the pseudocode, a node can be in one set N(i) only, see line (8). This means that the intersection of the subgraphs G_i is empty. This is not necessary though. Line (10) could be moved above line (8) and the latter replaced by writing: "(8') IF (n' is in G(j)) AND (n' is not already in N(j))". This would result, of course, in a different algorithm from the above.

On line (6), a node n for which $f(n)$ is minimum is expanded. The detailed expansion method is presented in the Appendix.

6.4 TESTED ALGORITHMS

Let us define four hierarchical path planning algorithms tested in the following simulations. Two of the algorithms are new ones based on the pseudocode of Meta A^*.

Hierarchical A^* (HAW)

The first tested algorithm is HAW resembling the one in [73], see also Section 6.1. The idea is that A^* starts searching first the coarsest resolution graph G^{rmax} in the graph hierarchy in Section 6.2. If it does not find a collision free path on G^{rmax}, then it starts searching also the finer resolution graph $G^{rmax} \cup G^{rmax/2}$, etc. HAW generates successors to a chosen node along each coordinate axis. Hence if the dimension of the search space is d then an expanded node has $2d$ neighbors. The heuristic value $h(n)$ assigned to each node n is the Manhattan distance between n and the goal. Functions g and h are measured using the costs of the finest resolution graph currently being searched.

This corresponds to the greedy strategy in Section 6.2. At best, the method is very fast since it has to explore only a union of a few coarse resolution graphs. HAW, being an A^* algorithm, is resolution complete: it finds a collision free path if one exists on the finest resolution graph.

Hierarchical Path Planner (HAPP)

The second algorithm, HAPP, is originally presented in [6]. It searches several different resolution graphs in the graph hierarchy of Section 6.2 at the same time. The search algorithm itself is A^*. The Appendix shows how HAPP generates successors for a given node and presents detailed pseudocodes (Rules 1–3). The idea is that HAPP tries to generate nodes that are in the coarser resolution graphs than their predecessors. The exception is when a father node is near an obstacle surface. Then finer resolution successors are allowed, too. All this happens within the limits of the finest and the coarsest resolutions given to the algorithm.

Figure 6.4 illustrates a 2-dimensional planning process of HAPP. A collision free path is the bolded line. Tiny black rectangles connected by thin lines form a search tree needed to find the path. In Figure 6.4, the planner uses four resolutions in the search: 1, 2, 4 and 8, see the thin lines of four different lengths. For comparison, Figure 6.5 illustrates the same path planning task solved by A^* searching the basic graph of resolution 1. The collision free

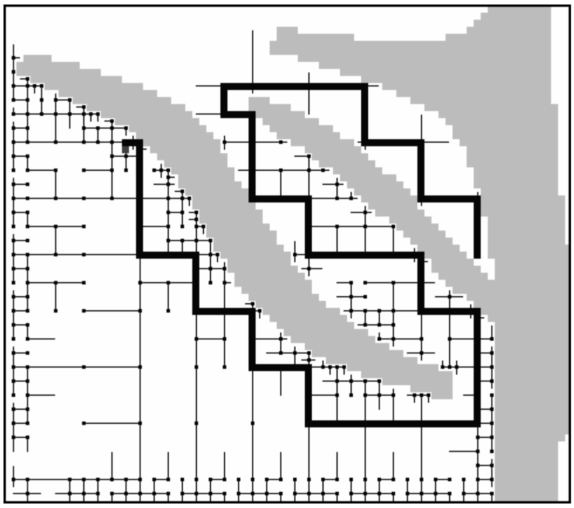

Figure 6.4: A 2-D search on the graphs with resolutions 1,2,4 and 8 by HAPP.

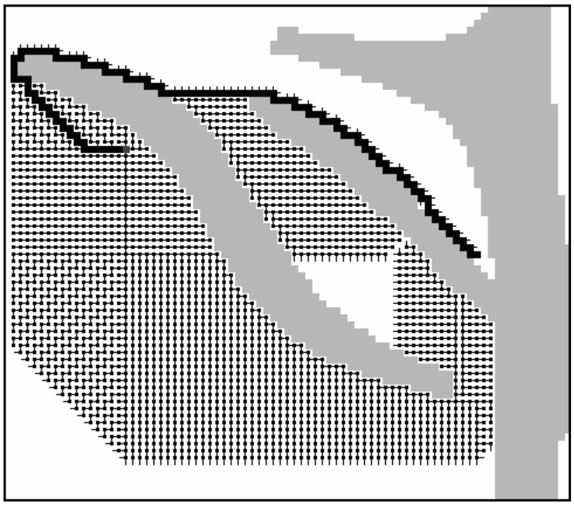

Figure 6.5: A 2-D search on the basic graph with resolution 1 by A^*.

paths in Figures 6.4 and 6.5 are different. This is because the A^* algorithms search different graphs.

Planners based on Meta A^* (Algorithm 1 and 2)

Let us assume that the search graph G is composed of a union of subgraphs G_i, as in Section 6.2: $G = \bigcup_{i=1}^{r} G_i$. Let us also use the graph hierarchy in Section 6.2: $G^j, j = 1, 2, 4, \ldots, 2^k, \ldots, 2^{\max}(= rmax)$. In the next two algorithms, we make two subdivisions G_i of G by using the graphs G^j. Both the algorithms are instances of Meta A^* and generate successors to a selected node by Rules 1–3 in the Appendix, similarly as HAPP.

We assign a weight $w(n') = c(n, n')$ to each n' in G_i:
$w(n') = rmax/dist(n, n')$, where n is a father of n' and $dist(n, n') = ||n - n'||_1$, as in Section 6.2. Nodes in finer resolution graphs are more costly than most of the nodes in coarse resolution graphs. By doing this we favor

expansions of nodes with coarse resolutions. This corresponds to the edge costs used in HAPP, too.

The Meta A^* based algorithms differ, however, from the greedy methods HAW and HAPP. Here we can control the expansions of nodes with different resolutions more adaptively by the "Meta A^*-mechanism" (the pseudocode of Meta A^*), whereas in HAW and HAPP this is achieved only by the node expansion mechanism itself and the edge costs.

Algorithm 1: Here every $G_i = G^j$ in the graph hierarchy ($i = 1, 2, 3, \ldots, r$; $j = 1, 2, 4, \ldots, rmax$). In other words, nodes in a set N_i of Meta A^* are in a graph whose resolution is $j = 2^{i-1}$. Hence Algorithm 1 tries to find a meta path (Section 6.3) that minimizes the sum of the weights of the expanded nodes with equal resolutions j (min $\leq j \leq$ max).

Algorithm 2: Here $G_i \neq G^j$. Let Meta A^* choose a set N_i and then pick up an open node $n \in N_i$ whose resolution is k, i.e., n is in G^k. In general $k \geq 2^{i-1}$. Rules 1–3, in the Appendix, generate successors n' to n. If n' is in G^l, where $l \geq k$ then n' is inserted in N_i. Alternatively, if n' is in G^m, where $m < k$ then n' is inserted as an open node in N_m.

We can imagine that every N_i contains nodes of resolution 2^{i-1}, from which trees of partial paths start. Nodes in a particular tree have resolutions $\geq 2^{i-1}$ and every child node has at least the same resolution as its father. In this way, Algorithm 2 tries to find a meta path that minimizes the sum of the weights of the expanded nodes in the forest whose trees start from nodes with equal resolutions. The weights w make this strategy favor trees that start from coarse resolution nodes.

We could at once construct variants of Algorithms 1 and 2. First, the weight of a node could be its resolution. Second, in Algorithm 2, we could minimize the *(weighted) number* of the trees in the forest instead of something relating to their nodes. These variants are not, however, tested here.

6.5 EXPERIMENTAL RESULTS

Test Cases

There are five path planning tasks for the four tested algorithms.

Figure 6.6 shows a cell layout adopted from [35] although the actual dimensions disagree. There is a 5 DOF Adept robot with an L-shaped object attached to its gripper. The left picture shows the start configuration and the right picture the goal. The robot has to bend its wrist to get the L-shaped object out of the wicket. Then it has to lift its linear axis and to rotate its first and second link to avoid the middle block. The collision free path goes between the robot and the middle block. This is the first task: Task 1.

Figure 6.7 shows a cell of Task 2. There is a 6 DOF IRB 2000 robot with a stick attached to its gripper and two arches. Task 2 has the start in the left picture and the goal in the right picture. The robot has to rotate all its joints to carry the stick under the first arch. Then it has has to bend its wrist to get the stick through the second arch towards the goal.

Figure 6.8 shows a cell of Task 3. It has been adopted from [34] although the actual dimensions disagree. The IRB 2000 has a cube attached to its gripper. Task 3 has the start in the left and the goal in the right picture.

Figure 6.9 shows a cell of Task 4. There is a car body and the IRB 2000 robot with a fictitious dashboard in its gripper. This task mimics a dashboard assembly process although here it is reversed: The start is in the left picture and the goal in the right picture.

Figure 6.10 shows a table and three chairs (Task 5). One chair can move on the floor, i.e., it has 3 DOFs. The moving chair is the leftmost one in the left picture, in its start position. The goal is on the opposite side of the table, under it, shown in the right picture. A collision free path goes between the table and the rightmost chair.

Simulations

In Tables 6.1–6.6, we recorded the total number of the expanded nodes (EXP) and path planning times (TIME) for all the four algorithms (ALG): Hierarchical A^* (HAW), Hierarchical Path Planner (HAPP), and the Meta A^* based Algorithms 1 and 2. The results of HAW are on a separate Table 6.6. The planning times are CPU times in seconds on a 1000 Mhz AMD Athlon computer with 256 megabytes of main memory.

Collision testing was done using a software package called RAPID (Rapid and Accurate Polygon Interference Detection) [27]. All the collision tests were calculated at the finest allowed graph resolutions independent of the resolutions that the algorithms used during their search. Collision testing took about 90 per cent of the computational work of the algorithms.

In Task 1, the joint movements of the 5 DOF Adept robot are discretized into $300 \times 200 \times 200 \times 200 \times 100$ different configurations. In Task 2, the 6 DOF IRB 2000 has a discretization $1000 \times 500 \times 300 \times 200 \times 50 \times 200$. In Task 3, the discretization is $1000 \times 500 \times 300 \times 200 \times 200 \times 200$. In Tasks 4 and 5, the discretizations are $200 \times 150 \times 150 \times 100 \times 100 \times 100$ and $300 \times 300 \times 300$, respectively.

In the path planning Tasks 1–3, the discretizations correspond to the real robot's maximum wrist movements of 1–3 cm, and 5 cm in Task 4. Then the resolution and step size is one. The coarsest allowed resolution and step size 128 is chosen such that empty configuration spaces of the robots contain only a few nodes.

Each path planning Task 1–5 has three test runs. In the first runs, the algorithms can only utilize resolutions at least 4, 32, 32, 8 and 16, respectively, see the first columns labeled "MIN" in Tables 6.1–6.5. This corresponds to the maximum resolutions of the graphs at which collision free paths exist. The subsequent runs have smaller MIN -values. Hence in the second and the third runs, the algorithms can search finer resolution graphs than is necessary, the information that is not known a priori. For example, MIN = 1 denotes the finest allowed resolution. Note that HAW always finds paths on the maximum resolution graphs that contain solution paths since it is a greedy method, see Table 6.6.

ALG	MIN	EXP	TIME	MIN	EXP	TIME	MIN	EXP	TIME
1	4	3755	31	2	4658	34	1	5295	36
2	4	5176	44	2	5711	45	1	5720	45
HAPP	4	4672	35	2	5422	36	1	4778	35

Table 6.1: Task 1

ALG	MIN	EXP	TIME	MIN	EXP	TIME	MIN	EXP	TIME
1	32	7342	602	8	11297	744	1	12461	779
2	32	14451	1230	8	19484	1500	1	20717	1550
HAPP	32	42000	2970	8	86579	4650	1	86519	4660

Table 6.2: Task 2

ALG	MIN	EXP	TIME	MIN	EXP	TIME	MIN	EXP	TIME
1	32	11647	1090	8	20651	1550	1	23196	1650
2	32	10592	1290	8	18892	1570	1	18963	1840
HAPP	32	12246	1130	8	24162	1800	1	24267	1810

Table 6.3: Task 3

ALG	MIN	EXP	TIME	MIN	EXP	TIME	MIN	EXP	TIME
1	8	29977	1720	2	45947	2200	1	48585	2270
2	8	58706	3480	2	85762	4160	1	84868	4170
HAPP	8	41685	2230	2	$> 2 \cdot 10^5$	> 7200	1	$> 2 \cdot 10^5$	> 7200

Table 6.4: Task 4

ALG	MIN	EXP	TIME	MIN	EXP	TIME	MIN	EXP	TIME
1	16	2672	241	8	3859	284	1	4869	301
2	16	2667	287	8	3664	328	1	4842	348
HAPP	16	2861	236	8	7450	412	1	6791	379

Table 6.5: Task 5

TASK	MIN	EXP.	TIME
1	4	$> 10^6$	> 7200
2	32	20756	1430
3	32	20921	1890
4	8	32947	1750
5	16	1733	134

Table 6.6: HAW

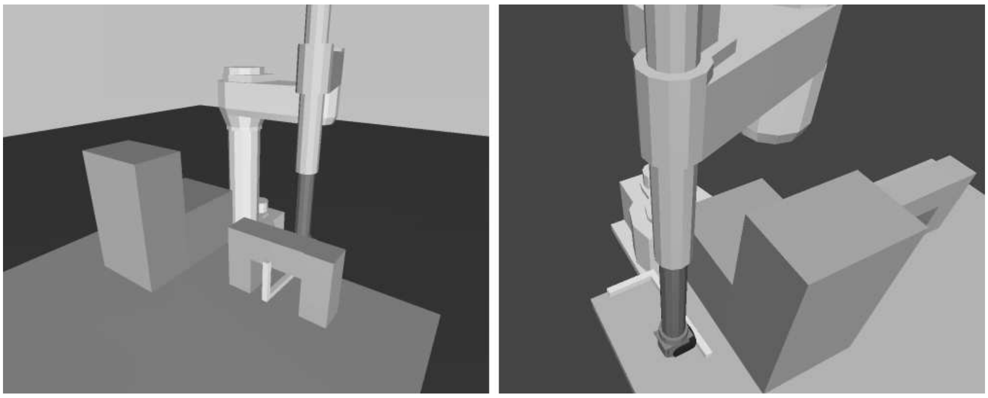

Figure 6.6: Task 1: A 5 DOF Adept robot with an L-shaped object attached to its gripper.

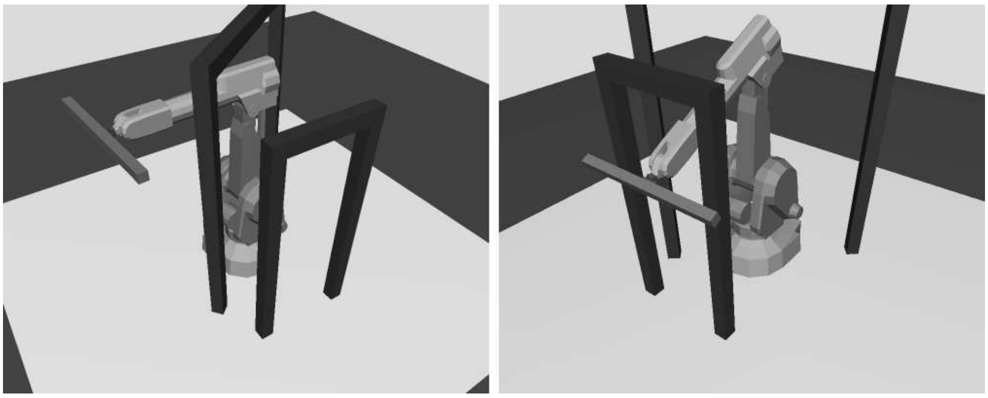

Figure 6.7: Task 2: A 6 DOF IRB 2000 robot with a stick.

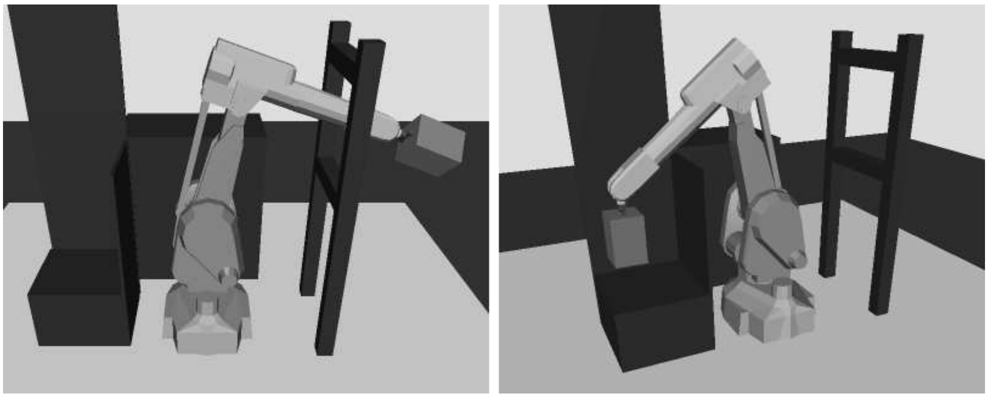

Figure 6.8: Task 3: A 6 DOF IRB 2000 robot with a cube.

6.6 DISCUSSION

In Tables 6.1–6.6, the number of the expanded nodes "EXP" measures the planning times as well as the amount of memory consumed by the algorithms. "TIME" is the CPU time of our computer, in seconds and it correlates with EXP. The planning times of the Meta A^* based Algorithms 1 and

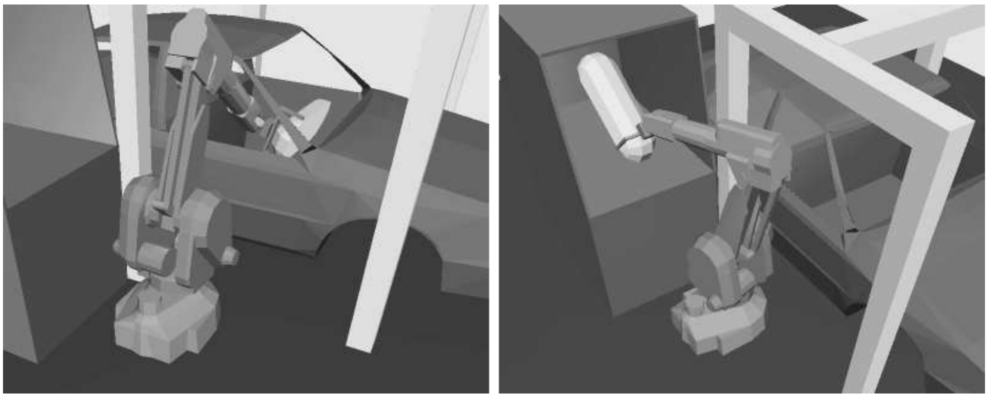

Figure 6.9: Task 4: An IRB 2000 robot assembling dashboard into a car body.

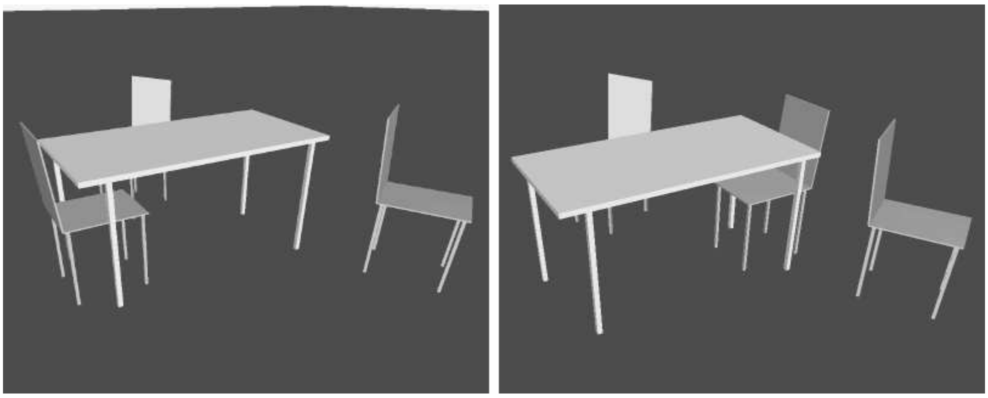

Figure 6.10: Task 5: A 3 DOF chair with a table and other chairs.

2 varied from about 30 seconds to 40 minutes. The exception is Algorithm 2 in Task 4: 69 minutes (Table 6.4).

Based on the CPU times Algorithm 1, on average, found paths fastest in three of the five tasks, Tasks 1–3 (Tables 6.1–6.3, and 6.6). However, in Task 3, the differences between all the four tested algorithms were small. Also, in Task 1, the hierarchical planner HAPP was almost as good as Algorithm 1. In Task 4, the greedy method HAW was almost as fast as Algorithm 1 at best. Algorithm 1, however, was faster than Algorithm 2 and HAPP in Task 4. HAW was the fastest in Task 5 whereas it performed very poorly in Task 1. In Task 5, Algorithm 1 slightly outperformed Algorithm 2 and HAPP. At best, Algorithm 1 was the fastest one in Tasks 1–4.

Observations based on the number of the expanded nodes, EXP, correlate with the above CPU -results. However in Tasks 3 and 5, Algorithm 2 expanded slightly fewer nodes than Algorithm 1. HAPP performed very poorly in Tasks 2 and 4.

We studied also the behavior of the algorithms in situations where they searched finer resolution graphs than was actually needed in order to find collision free paths, see the second and the third columns labeled MIN in Tables 6.1–6.5. This data suggests that the Meta A^* based algorithms 1 and 2 are more robust than HAPP: Algorithms 1 and 2 did not unnecessarily expand many more nodes when minimum allowed search resolutions were finer than needed. The maximum numbers of the expanded nodes were less than two times the numbers of the expanded nodes obtained by using the coarsest possible MIN values, see all the columns labeled MIN. This was not always the case with HAPP, see Tables 6.2, 6.4, and 6.5.

Algorithms 1, 2, and HAPP can also be run in a greedy mode, similarly as HAW: First search coarser resolution graphs (set MIN high) and if a solution is not found then search finer resolution graphs (set MIN lower), etc. In Tables 6.1–6.5, the leftmost runs roughly correspond to this: MIN $= 4, 32, 32, 8, 16$, respectively. These MIN values represent the coarsest resolution graphs on which solution paths exist. Here the Meta A^* based Algorithms 1 and 2 were faster than HAW in Tasks 1–3. Moreover, Algorithm 1 was faster or equally fast as HAW in four of the five tasks (Tasks 1–4) and expanded fewer nodes than HAPP in all the five tasks.

These observations suggest that the Meta A^* mechanism indeed had a positive effect on the running times and the memory consumption of the algorithms — not only the adaptive node expansion method itself, presented in Appendix.

The simulation runs done here were one directional searches. However, all the tested algorithms can be used in bidirectional searches as well. In a bidirectional search, there are two algorithms: The start and the goal of the first algorithm are the goal and the start of another, respectively. The Meta A^* mechanism can also be adopted to combine the two search processes in bidirectional searches, see Section 5.

6.7 CONCLUSIONS

In this chapter, we presented and tested four hierarchical path planning algorithms using five simulated robot workcells. Two of the algorithms are based on a novel application of A^*, Meta A^*. Although the results may lack statistical significance, we feel that conclusions can be drawn.

In the algorithms, we do not have to guess right the coarsest resolutions of the graphs on which solution paths exist, i.e., the minimum robot movement discretizations. If we discretize the robot movements slightly too finely, then the Meta A^* based path planners do not unnecessarily expand very many additional fine resolution nodes. In general, guessing of the smallest discretizations would be needed especially in situation where search graphs (grids) are infinite.

If search graphs are finite then path planning can also be done by reducing the minimum allowed search resolution step by step starting from a coarse one. In this case, the Meta A^* based Algorithm 1 was the fastest method in all the test cases including a robot.

The simulation data suggests that Meta A^* is a useful mechanism that, in addition to the adaptive node expansion method in Appendix, reduces path planning times and memory consumption in hierarchical path planning. The Meta A^* mechanism can also be adopted to combine two search processes in bidirectional search applications.

The path planning times of the Meta A^* based Algorithm 1 varied from about 30 seconds (Task 1) to 40 minutes (Task 4). Hence planning times may grow too much compared to the manual teaching of robots if degrees of freedom of tasks are increased (beyond 5 or 6 for example). Then, however, one may consider to "plug" our algorithms into roadmap methods as local planners: to find collision free paths between any two subgoals chosen by a global planner.

Acknowledgments

The author wishes to thank Mr. Johannes Lehtinen for implementing the HAPP algorithm, Mr. Jaakko Väyrynen for implementing the Meta A^* code and running the simulations, and Mr. Tuomo Hyyryläinen for implementing some of the test cells. Moreover, the author is grateful to the above persons for valuable comments and discussions during the development of the algorithms. The author also thanks the anonymous referees of [4] for their comments.

APPENDIX

Here we describe the node expansion process of the hierarchical search. We present here a more detailed version of the pseudocodes in [6]. Now OPEN an CLOSED refer to the sets used by A^*, not to those sets that Meta A^* uses in the pseudocode in Section 6.3. The A^* algorithm with the following three rules expand all the nodes in the graph $G = G_i \cup G_2 \cup \cdots \cup G_r$. The goal is always set at the origin. At first, let us introduce some terminology.

Let us denote by $n.m$ the resolution of a node n. Recall that the resolution of $n = (n_1, \ldots, n_d)$ is the greatest common divisor $(1, 2, 4, \ldots, rmax)$ of the components $n_1, n_2, \ldots, n_d$. Next, assume that n has a resolution $n.m$. Then denote by $n.j$ a *j-value* of a node. It is the number of the components $n_1, \ldots, n_d$ that are divisible by $2 \cdot n.m$. Intuitively, the j-value measures how close a node of resolution $n.m$ is to nodes of the next coarser resolution $2 \cdot n.m$.

A node can be *a surface node*, *a corner node* or *a free-space node*. A node n is a "surface node" if the distance between n and an obstacle surface along any coordinate axis is less or equal to $n.m$, the resolution of n. A node n is a "corner node" if the distances between n and obstacle surfaces are greater than $n.m$, the father n'' of n is a surface node, and a so called *corner condition* is satisfied: *The distance from n'' to an obstacle surface is less or equal to $n''.m$ along a direction orthogonal to the one from n'' to n.* For example, imagine a two dimensional search space to visualize a corner node. Finally, n is a "free-space node" if it is not a surface node and not a corner node.

The three rules below generate all the successors n' to n. Input data to the rules are n, OPEN and CLOSED sets of A^* together with max and min that are the maximum and the minimum allowed resolutions of nodes, respectively. The rules maintain OPEN and CLOSEDsets.

Let $\pm e_k$ $(k = 1, \ldots, d)$ denote unit vectors: $\pm e_k = (0, \ldots, \pm 1, \ldots, 0)$, where the index of the number ± 1 is k. Hence there are $2d$ such vectors. A function DIST(n, e_k) returns a distance between n and the closest obstacle surface along e_k. The distance is measured up to $n.m + 1$. Operators '$<$', '$<=$', '$==$' and '$>$' denote 'less than', 'less or equal to', 'equal to' and 'greater than', respectively.

```
RULE-1(min, max, OPEN, CLOSED):

 0: Remove n, for which f(n) is minimum, from OPEN
    and put it in CLOSED;
 1: FOR (All e_k, except the one from n to its father) DO
 2:     d = DIST(n, e_k);
        % n is "on the surface" of an obstacle
 3:     IF (d <= min) THEN
 4:        FOR (All e_l orthogonal to e_k) DO
               % a successor candidate
 5:            n' = n + min*e_l; Determine n'.m;
 6:            IF (n' is in free space, C_free)
               % place n' on OPEN
 7:            THEN RULE-3(n, n', OPEN, CLOSED);
 8:        END FOR
 9:     ELSE % min < d <= n.m
           % check the "size" of the free space
10:        i = min; found = 0;
11:        WHILE ((i < n.m) AND (found == 0)) DO
12:            IF (i < d <= 2*i) THEN
13:               found = 1;
                  % a successor candidate
14:               n' = n + i*e_k; Determine n'.m;
                  % place n' on OPEN
15:               RULE-3(n, n', OPEN, CLOSED);
16:            END IF
17:            i = 2*i;
18:        END WHILE
19:     END IF
20: END FOR
    % Determine the type of the node n,
    % n can already be a "corner node" (by RULE-3)
21: IF (d > n.m along every e_k on line 2)
22: THEN
23:     IF (n is not "corner node")
24:     THEN Mark n as "free-space node";
25: ELSE Mark n as "surface node";
26: END IF
    % Some successors to surface nodes already generated
    % The rest of the successor generation process
27: RULE-2(n, max, OPEN, CLOSED). (End of RULE-1)
```

RULE-1 generates some successor candidates n' only to "surface nodes" n, on lines (5) and (14). Actually, we mark the father node n either as a "surface" or as a "free-space node" later, on lines (21)–(26). The decision whether n'

is a “corner node” is done in RULE-3. Some of the successors are generated only along the directions from n to nearby obstacle surfaces, see lines (11)–(18). If n happens to be very near to an obstacle surface, then successors are generated along every orthogonal direction to the surface, see lines (3)–(8). The latter is done to ensure that all the nodes on the surfaces of big obstacles will be expanded if needed. The successor candidates are inserted in OPEN if they are not already there by RULE-3 on lines (7) and (15).

RULE-2 generates successor n' to all kinds of nodes. The resolutions of the successors are greater or equal to the resolution of their father n. The ‘EX-OR’ operator below denotes ‘exclusive-or’.

```
RULE-2(n, max, OPEN, CLOSED):

28: FOR (all e_k, except the one from n to its father) DO
29:     n’ = n + n.m*e_k;
30:     Determine n’.m, n’.j and n.j;
31:     IF ((n is "surface node") AND (DIST(n, e_k) > n.m))
           EX-OR
32:        (n is  "corner node")
           EX-OR
33:        ((n is "free-space node") AND (n.m < max) AND
            (n’.j > n.j))
           EX-OR
34:        ((n is "free-space node") AND (n.m == max))
        % place n’ on OPEN
35:     THEN RULE-3(n, n’, OPEN, CLOSED);
36: END FOR (End of RULE-2)
```

If a father node n is a “surface node” or a “corner node”, then RULE-2 generates new successors candidates along every direction where there is enough free space, see lines (29), (31) and (32). If n is a “free-space node”, then successor candidates are generated only along directions that lead towards higher resolution nodes (increasing j-values), see lines (29) and (33). However, the exception is when $n.m$ = max. Then successor candidates are generated along every direction, see line (34). The distance between n and its successor n' is always $n.m$. If min = max, then RULE-1 and RULE-2 together generate the same successor candidates as the “basic” A^* searching the graph with resolution min.

```
RULE-3(n, n’, OPEN, CLOSED):

37: Calculate f(n’) = g(n’) + h(n’) similarly as A* does;
38: IF (DIST(n’, e_k) > n’.m along every e_k) AND
       (n is "surface node")                  AND
       (n’ satisfies "corner condition")
39: THEN Mark n’ as "corner node";
40: IF (n’ is not already in OPEN or CLOSED)
41: THEN Place n’ in OPEN;
42: IF (n’ is already in OPEN or CLOSED)
```

```
43: THEN
44:   Direct the pointer from n' along the path yielding
      the lowest g(n');
45:   IF (n' required pointer adjustment)
46:   THEN % n' has now a new father, say n''
47:      Check whether n' has to be marked as "corner node",
            similarly as on lines 38-39,
            by using the new father n'';
48:      IF (n' was found in CLOSED) THEN Reopen n';
49:   END IF
50: END IF (End of RULE-3)
```

On line (38), if n does not yet have a type, then check whether n is a "surface node", similarly as on lines (21–26). RULE-3 does the same operations as A^* does before inserting a new node n in OPEN, on lines (5.1)–(5.3) in the pseudocode of A^* in Section 2.1. In addition, RULE-3 marks a node n' as a "corner node" if necessary, see lines (38), (39) and (47). We described the "corner condition", on line (38), in the text above RULE-1.

The above pseudocode could be implemented in a more effective way. The presentation here tries to emphasize clarity.

Variants of the above successor generation rules can be constructed easily. The following suggestions are some examples. More successors could be generated to "surface nodes" on line (31), e.g., along distances less than $n.m$ from n. Fewer successors could be generated to "free-space nodes" on line (33), e.g., generate only those successors, with increasing j-values, whose distance to a goal is less than the distance between their father and the goal. Successors to "free-space nodes" n could also be generated along every direction when $n.m$ is less than max, see line (33). The latter would simplify the code since there is no longer a need to distinguish between "corner" and "free-space" nodes.

Furthermore, lines (3)–(8) could be deleted and/or fewer successors could be generated to "corner nodes" on line (32) in order to decrease the number of the nodes on and near to obstacle surfaces. A possible general strategy is the following. Expand all the above successors to the best nodes in OPEN for a while. Then select a subset of the nodes in OPEN and forget all the others. This subset forms a new reduced OPEN, etc. The selection of the subset can be based on the f values or some other, adaptive, criteria. We can decide how the size of the reduced OPEN increases, e.g., it can be constant, it can grow linearly or according to a small polynomial relative to search time. Then, of course, the possibility of missing an existing path increases. We can also apply the pseudocode of the X_β algorithm in Section 2.12.4.

7 A^*-BASED POWER-AWARE ROUTING ALGORITHMS IN WIRELESS NETWORKS

In this chapter, we describe three A^*-based algorithms for power-aware routing of messages in large communication networks where future message sequences are not known. We seek to maximize the average lifetime of the network. For achieving this goal the algorithms use different optimization criteria. The new algorithms are simpler and their running times are shorter than those of the widely quoted *max-min* zP_{min} algorithm [48]. In addition they are not as sensitive to parameter settings as *max-min* zP_{min}. We show empirically that one of the algorithms produces longer average lifetimes than *max-min* zP_{min}. The other two algorithms perform similarly as *max-min* zP_{min}. This chapter presents an enlarged version of [5].

7.1 INTRODUCTION AND RELATED WORK

The rapid development of low-power electronics has made it possible to create wireless networks of hundreds or even thousands of devices of low computation, communication and battery power. The networks can be used, e.g., as distributed sensors to monitor large geographical areas or as grids of computation, etc. In these applications, great care is required in the utilization of power since every message sent and computation performed drains the battery.

The problem of minimizing power consumption of wireless networks has received significant attention [68, 64, 11, 33, 47, 48, 74, 12, 8, 20, 37, 1, 24]. Much of the power consumption of the devices is used for transmitting and receiving messages [48, 67]. We concentrate on optimizing the power consumption during communication. Optimizing the idle mode of the devices is discussed, e.g., in [13, 67].

Several metrics have been introduced to optimize power-aware routing of messages, see e.g. [68]. Minimal energy consumption was used in [64]. Other metrics include minimizing the variance in the devices' power levels, minimizing the energy cost per packet and minimizing the maximum energy cost. However, these metrics can lead to suboptimal solutions: Many devices can maintain high power while some devices, critical nodes, may have consumed almost all their energy. The network may then become disconnected because of the critical nodes.

The lifetime of the network has been defined as the time for the first node or device to die due to low energy level [11, 33, 68], or as the time for a certain percentage of nodes to die [74]. Blough and Santi [8] define the network lifetime by using the number of the alive nodes and the size of the largest connected component in the network. For sensor networks, Blough and Santi take also into account the percentage of the region monitored by the sensors. Li, Aslam, and Rus [48] define network lifetime as the earliest time when a message in a sequence of messages cannot be sent due to nodes with low residual energy. Kar et al. [37] maximize the throughput of the network, that is, minimize the number of messages that cannot be routed for a sequence of message routing requests.

Li, Aslam, and Rus [48] develop an online approximate power aware routing algorithm *max-min* zP_{min} to maximize the lifetime, which involves choosing between a minimal power consumption path and a path that maximizes the minimum residual power in the network. In its basic version, *max-min* zP_{min} needs to know all the locations and the energy levels of the nodes in the network. The authors later develop a distributed version of their algorithm, see [1]. These algorithms operate *online*: Message sequences in the future are not assumed to be known. Later we shall refer to the *max-min* zP_{min} algorithm simply as the Li-Aslam-Rus, or LAR algorithm.

Here, we maximize the *average lifetime of a network* that is an average of the lifetimes of the network after routing online many random message sequences. The lifetime of the network is the same as in [48]. We develop three new A^* based power-aware routing algorithms. They have the same input as the LAR method and are online algorithms too but are simpler than LAR. Their running times are asymptotically shorter than that of LAR. We compare the average lifetimes of the new algorithms with LAR. This is done by simulating message transmissions on two dimensional grids containing between 200 and 1600 nodes, or devices (Li, Aslam, and Rus [48] used networks containing only 40 nodes at maximum in their simulations of LAR). Two of the algorithms and LAR produced similar average lifetimes and the third one was between 10 and 21 per cent better than LAR.

7.2 THE POWER-AWARE ROUTING PROBLEM

Most of the power consumption in a wireless network can be divided into two parts: (1) the idle mode and (2) the transmit/receive mode. The idle mode corresponds to a baseline power consumption when a device is in a "standby" state. Optimizing the idle mode is also important but here we focus on optimizing the transmit/receive mode. When a message is routed through the network, all the nodes except the source and destination receive the message and then immediately relay it.

Let a wireless network be represented by a weighted graph $G(V, E)$. The vertices, V, correspond to devices (computers or sensors, etc.) in the network. Every vertex has a weight that measures the device's energy level. The edges, E, connect pairs of devices that are within communication range. Each edge weight is the energy cost of sending a unit message between two devices. All messages are assumed to be unit messages. Suppose a device i needs power e to transmit a message to a device j that is distance d_{ij} away. This power consumption is usually approximated as $e \propto d_{ij}^c$ where the exponent c ($2 \leq c \leq 4$) models the decay of the radio signal in the intervening medium [11, 33, 48].

The batteries of the devices have finite energy. We try to route messages in such a way that the batteries would have enough energy for the network to be connected as long as possible. This is measured by *the average lifetime of the network*. To estimate the average we route messages one after another, online, between randomly selected nodes in the network and record the number of successfully transmitted messages until the time when the next message request cannot be routed. We repeat this process one thousand times with the same algorithm, network and initial energy levels of nodes.

7.3 THE STUDIED ROUTING ALGORITHMS

Another Implementation of LAR

We use the notation of Li, Aslam, and Rus [48]. Let $P(v_i)$ be the initial power level of node v_i in the graph G and $P_t(v_i)$ the power of v_i at time t. Let e_{ij} be the weight of the edge v_iv_j. Define $u_{tij} = (P_t(v_i) - e_{ij})/P(v_i)$ as the residual power fraction after sending a message from v_i to v_j.

The LAR algorithm (*max-min* zP_{min}) [48] maximizes the minimal residual power fraction of the graph under the constraint that the power consumption of any message must be below a value zP_{min}. Here P_{min} is the minimum power consumption of the paths found so far and $z \geq 1$ is a parameter. LAR is an iterative method.

The LAR algorithm first finds a path with the least power consumption P_{min} on the initial network graph. Then the algorithm iteratively searches minimum power consumption paths on a series of graphs. During every iteration, the minimal residual power fraction $u_{\min}$ is found on the recent optimal path. Then all the edges whose u_{tij} is less than $u_{\min}$ are removed from the previous graph to obtain a new graph for the next iteration. The iteration process stops if the power consumption of the newly found path is greater than zP_{min}.

To find paths with the least power consumption in each iteration LAR uses Dijkstra's shortest path algorithm. Here we replace Dijkstra's algorithm with the A^* algorithm guided by a monotone (and admissible) heuristic h. Dijkstra's algorithm can be seen as an A^* with $h(n) = 0 \ \forall n$. In this way, we can improve the running times of LAR. The running times of A^* with a monotone h and Dijkstra's algorithm can be directly compared with each other. The larger the h is the shorter the running time of A^* is compared to that of Dijkstra's algorithm (ignoring the computations of h).

In the present application, $h(v) = ||\gamma - v||$ is a distance function between a node v and the goal node γ, where v and γ are 2-dimensional Euclidean vectors. The heuristic $h(v)$ underestimates the true message sending cost from v to γ, since we assumed the power consumption model in Section 7.2: $e = kd^c = k||\gamma - v||^c$ for some positive constant k ($c > 1$) and, without loss of generality, assume here that $k, ||\gamma - v|| \geq 1$. The heuristic h is also monotone since all distance functions by definition satisfy the triangle inequality. When A^* using a monotone heuristic finds a path to n for the first time, it has already found the shortest path to n and does not have to visit n any more.

The New Algorithms

We consider three new A^*-based routing algorithms with the following type of evaluation function, for a node v in G:

$$f(v) = g(v) + (1 - \lambda)h(v) = \lambda g_1(v) + (1 - \lambda)g_2(v) + (1 - \lambda)h(v). \quad (7.1)$$

The cost function g is a linear combination of two functions g_1 and g_2. Constant $\lambda \in [0, 1)$ is a parameter. The algorithms have different g_1 functions whereas g_2 and h remain the same. Function g_2 measures the power consumption for sending one message from the source node s to v, that is the sum of the weights e_{ij} along the route from s to v. The functions $h(v) = ||\gamma - v||$ and $(1 - \lambda)h$ are monotone and admissible heuristics since they are distance functions between 2-D vectors.

The A^* algorithms route messages on a graph G' that is identical to G except that the cost of the edge between two neighboring nodes v_i and v_j in G' is:

$$\begin{aligned} c_{G'}(v_i, v_j) = g(v_j) - g(v_i) &= \lambda(g_1(v_j) - g_1(v_i)) + (1 - \lambda)(g_2(v_j) - g_2(v_i)) \\ &= \lambda(g_1(v_j) - g_1(v_i)) + (1 - \lambda)e_{ij}. \quad (7.2) \end{aligned}$$

Let $|V|$ and $|E|$ be the number of nodes and edges in G. The algorithms below execute A^* only once to find the final route (if it exists) for one message. Their running time is $O(|E| + |V| \log |V|)$, the same as Dijkstra's algorithm, since the heuristic is monotone. The LAR method, on the other hand, calls Dijkstra's algorithm $O(\log |E|)$ times and has a running time of $O(\log |E| \cdot (|E| + |V| \log |V|))$ [48].

Algorithm 1

Let $r_{tij} = P_t(v_i) - e_{ij}$ be the residual power after sending a message from v_i to v_j. The LAR algorithm routes a message by maximizing the minimal residual power fraction $r_{tij}/P(v_i)$ in the graph while keeping the power consumption for one message below the limit zP_{min}. Algorithm 1 mimics this behavior by penalizing both low minimal residual powers along message paths and high power consumption of messages.

Let Max be the maximum of the $P(v_i)$'s. The evaluation function of Algorithm 1, for a node v_j, is:

$$f(v_j) = \lambda(Max - \min(r_{t0j})) + (1 - \lambda)g_2(v_j) + (1 - \lambda)h(v_j), \quad (7.3)$$

where $\min(r_{t0j})$ is the minimum residual power along the message path from the source node $s = v_0$ to v_j. The parameter λ defines the relative weights of the two penalizing functions.

Algorithm 2

The evaluation function of Algorithm 2 is:

$$f(v_j) = \lambda(1/ \min(r_{t0j})) + (1 - \lambda)g_2(v_j) + (1 - \lambda)h(v_j). \quad (7.4)$$

Algorithm 2 resembles Algorithm 1 by penalizing both low minimal residual powers r_{t0j} along paths and high power consumption of messages. The first penalizing function is, however, different from the one of Algorithm 1.

Algorithm 3

The evaluation function of Algorithm 3 is:

$$f(v_l) = \lambda \sum (Max - r_{tij}) + (1-\lambda)g_2(v_l) + h(v_l). \qquad (7.5)$$

The summation in the first term is taken over all the edges, message transmissions between succeeding nodes, along the path from s to v_l. Algorithms 1 and 2 focus only on the minimal residual power along the path whereas Algorithm 3 routes a message in such a way that the sum of the residual powers along the path is high and the total power consumption of the message is low.

Here the heuristic can be h instead of $(1-\lambda)h$ which means shorter running times. The cost between a node v_i and its successor v_j in the graph G' is

$$\begin{aligned} c_{G'}(v_i, v_j) &= \lambda(g_1(v_j) - g_1(v_i)) + (1-\lambda)e_{ij} = \lambda(Max - r_{tij}) + (1-\lambda)e_{ij} \\ &= \lambda(Max - P_t(v_i) + e_{ij}) + (1-\lambda)e_{ij} \geq e_{ij}, \qquad (7.6) \end{aligned}$$

since $Max - P_t(v_i) \geq 0$. The heuristic h being a distance function is admissible and monotone on the graph G, that is, $h(v_i) \leq e_{ij} + h(v_j)$. But then h is also admissible and monotone on G' since $c_{G'}(v_i, v_j) \geq e_{ij}$.

We could have modified Algorithm 3 by changing the $Max - r_{tij}$ term, in the definition of its $f(v_l)$, to $1/r_{tij}$ similarly as in Algorithm 2 but this was not done here anymore. In Algorithms 1, 2, and 3, A^* can be replaced by Dijkstra's algorithm, if wanted, at the expense of longer running times.

On Distributed Implementations

Li, Aslam, and Rus [48] use a zone based approach for routing in large networks. In this approach, information is aggregated in zones and the message is routed from one zone to another. The information required by Algorithms 1, 2, and 3 as well as LAR is identical, except the heuristic h that can at least be estimated between the zones. Therefore one can use the same zone based approach to route with Algorithms 1, 2, and 3. Moreover in [1], the same authors develop a distributed version of LAR based on a distributed version of Dijkstra's algorithm. Again, similarly modified versions of our algorithms are possible.

In [24, 20], nodes estimate the power levels and location information of all the nearby nodes by local broadcasts. This can be done repeatedly or an on-demand fashion. Here the A^* could be replaced by its variants, IDA^* and RTA^* [40, 41, 49], that use only local information and the paths are sought only when needed, in reactive manner. However, here we do not discuss distributed implementation issues any further.

7.4 PERFORMANCE EVALUATION

In all the experiments, the network is a 2-D grid: every node has four neighbors and can send messages only to these neighbors. In this way we are able to study grids with hundreds or even thousands of nodes. Moreover, according to the power consumption model in Section 7.2: $e \propto d_{ij}^c$ $(2 \leq c \leq 4)$, sending a message via relay nodes that are near each other consumes less energy than sending a message straight from a start node to a distant goal node. Also the grid resembles the situation when distributed routing is done: A node mostly records data of nearby nodes.

In every simulation run, one of the tested algorithms routes a sequence of message routing requests online between randomly chosen pairs of nodes. The source and target nodes for each routing request are selected uniformly at random from the grid. Transmitting messages consumes energies of the nodes along message paths. A simulation run stops when the first message cannot be routed, that is, all possible path candidates contain at least one node with zero energy. The number of successfully transmitted messages are recorded.

All the simulation results are averages over 1000–20000 single simulation runs. The average estimates the average lifetime of the grid. The simulations in Figures 7.1–7.8 use square grids with $K \times K = 20 \times 20$ nodes and every node has an initial power level of $P = 10$ units. The cost of an edge between neighboring nodes v_i and v_j is $e_{ij} = 1$.

Figure 7.1 shows average lifetimes produced by the LAR algorithm [48]. LAR maximizes the minimal residual power fraction of the graph while keeping the power consumption of any message under zP_{min}, where $z \geq 1$ is a parameter. The average lifetimes depend on the parameter z in Figure 7.1. The maximum is obtained when $z \approx 1.3$. When messages are allowed to consume more power with larger values of z, the average lifetimes decreases. Li, Aslam, and Rus [48] report also similar behavior of LAR while using different network graphs.

We have replaced Dijkstra's algorithm in LAR with A^*. The heuristic h guiding the search of A^* is the Manhattan distance between a node $n = [n_1, n_2]$ and the destination $t = [t_1, t_2]$: $h(n) = |t_1 - n_1| + |t_2 - n_2|$. The simulation programs were implemented in Matlab. The CPU running times of the A^* version were about 60 per cent of those of the Dijkstra version. The CPU times were average values of 2000 simulation runs on grids $K = 20$, $P = 10$, and 20. These simulations were run on an SGI Origin 2000 multiprocessor system with R12000 300 MHz processors (RISC architecture).

Figure 7.2 shows average lifetimes produced by Algorithms 1, 2, and 3. In these A^* algorithms, the cost of a message route is $\lambda g_1 + (1 - \lambda) g_2$ where $\lambda \in [0, 1)$ is a parameter, see Section 7.3. The measurements marked with asterisks, plus signs and circles correspond to Algorithm 1, 2, and 3, respectively. At $\lambda = 0$, all the A^* algorithms minimize the power of sending one message. This strategy gives the lowest average lifetimes in Figure 7.2. It is thus insufficient. Li, Aslam, and Rus [48] already demonstrated that to obtain a good routing algorithm, namely LAR, it is not sufficient to optimize with

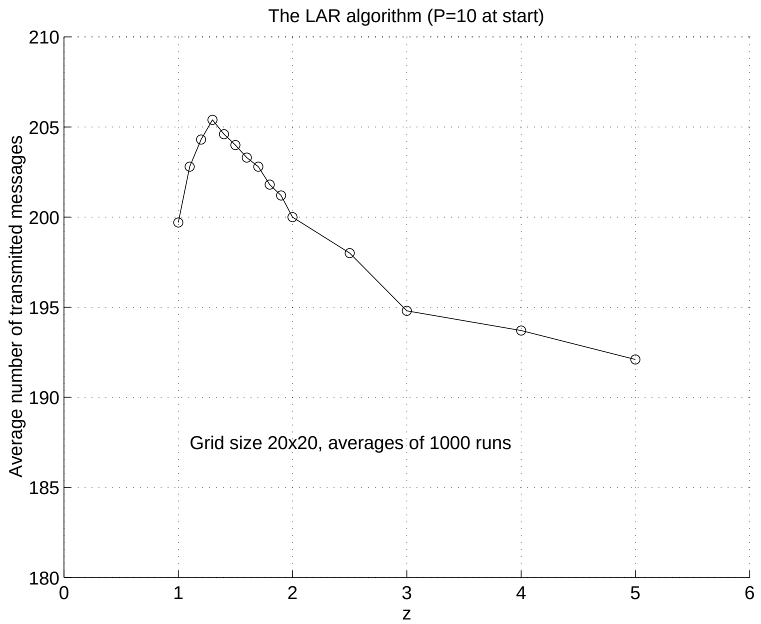

Figure 7.1: The LAR algorithm on a 20×20 grid ($P = 10$). The average number of transmitted messages vs. parameter z. Averages over 1000 simulation runs.

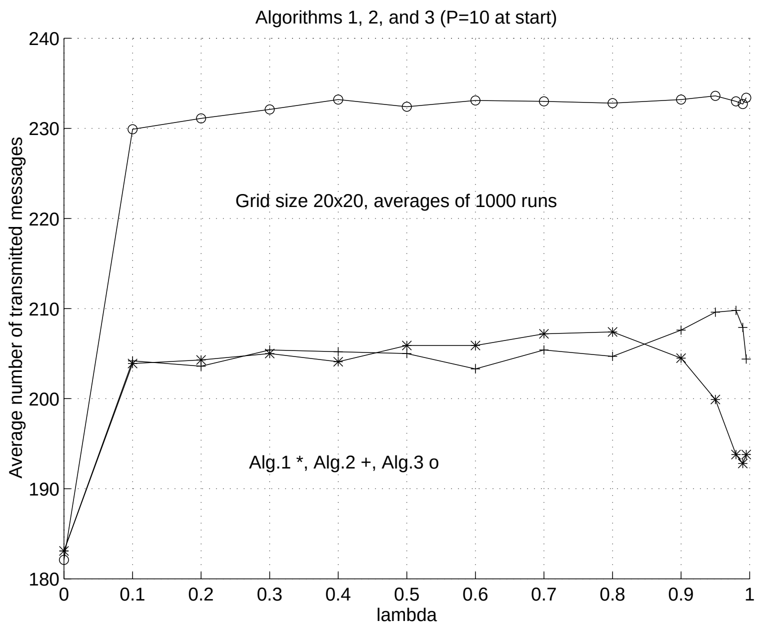

Figure 7.2: Algorithms 1, 2, and 3 on a 20×20 grid ($P = 10$). The average number of transmitted messages vs. parameter λ. Averages over 1000 simulation runs.

respect to only one criterion. Our solution to the problem is to use a linear combination of the two criteria g_1 and g_2. The advantage of doing this is visible in Figure 7.2 where $\lambda > 0$: The average lifetimes are much higher than those at $\lambda = 0$. Figure 7.2 also shows that Algorithms 1, 2, and 3 are not very sensitive to the parameter λ, where $0.3 \leq \lambda \leq 0.8$. LAR is more sensitive to

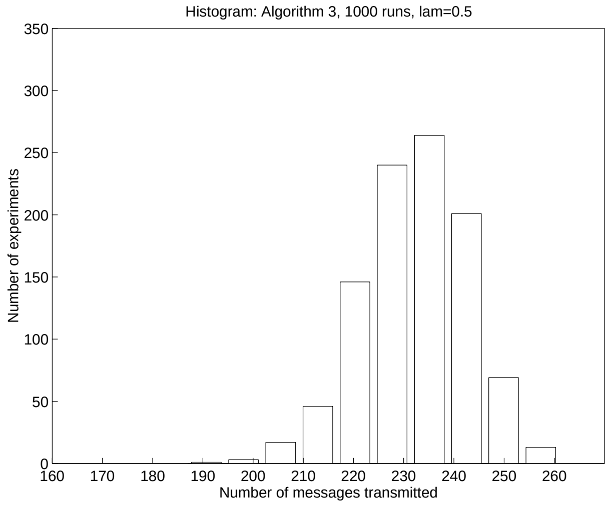

Figure 7.3: Histogram of 1000 runs of Algorithm 3 at $\lambda = 0.5$ on a 20×20 grid ($P = 10$). The number of experiments vs. the number of transmitted messages.

its parameter z than Algorithms 1, 2, and 3, cf. Figure 7.1.

The average lifetimes produced by Algorithms 1 and 2 are roughly the same as those produced by LAR when $z \approx 1.3$. It seems that converting the constraint "power consumption of a message $\leq zP_{min}$" in LAR to a "soft constraint" or a penalty function $(1-\lambda)g_2$ in Algorithms 1 and 2 provides close to optimal performance of LAR while removing its sensitivity to the parameter z. Algorithm 3 ($\lambda > 0$) produces about 11 per cent longer average lifetimes than the other tested algorithms. Figures 7.3 and 7.4 plot histograms of the 1000 simulation runs of Algorithm 3 ($\lambda = 0.5$) and LAR ($z = 1.3$), respectively. The grid size is the same (20×20) and $P = 10$. The histograms clearly confirm that Algorithm 3, on average, can route more messages than LAR at its best. This suggests that using a more global criterion as a penalty function g_1 is better than only focusing on the minimal residual power along routes as Algorithms 1 and 2 do, see Section 7.3.

We also studied the algorithms when 10, 20, and 30 per cent of the nodes in the grid have randomly been removed. In every simulation run, a different random set of nodes is missing.

Figures 7.5 and 7.6 show average lifetimes produced by the LAR method and Algorithms 1, 2, and 3 on 20×20 grids ($P = 10$) from which 10 per cent of the nodes have been randomly removed. Here the situation is quite similar to the one in Figures 7.1 and 7.2. Algorithm 3 gives the longest average lifetimes. The average numbers of transmitted messages in Figures 7.5 and 7.5 are about 60 per cent of those in grids with no missing nodes, in Figures 7.1 and 7.2.

Figures 7.7 and 7.8 present results on 20×20 grids ($P = 10$) from which 20 per cent of the nodes have been randomly removed. Here all the tested algorithms perform more or less equally well. Perhaps Algorithm 3 is still better than the other ones, on average. The numbers of transmitted messages are now less than 30 per cent of those in full grids, in Figures 7.1–7.2.

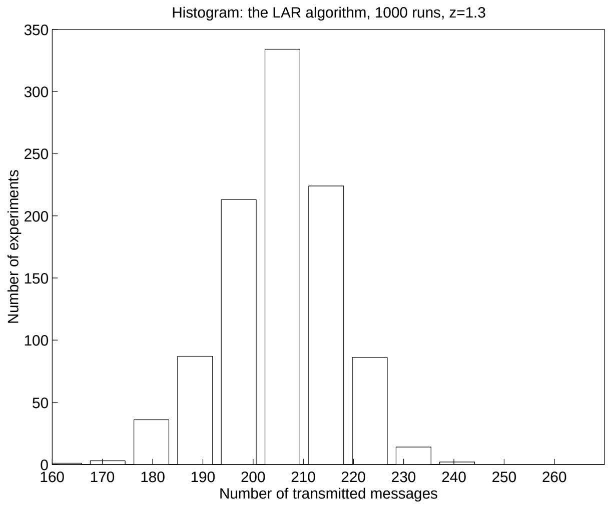

Figure 7.4: Histogram of 1000 runs of the LAR algorithm at $z = 1.3$ on a 20×20 grid ($P = 10$). The number of experiments vs. the number of transmitted messages

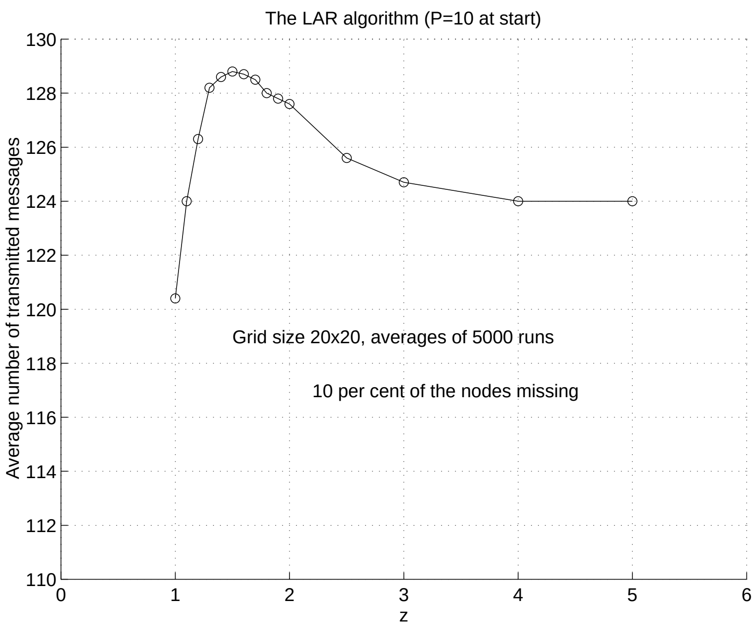

Figure 7.5: The LAR algorithm on 20×20 grids ($P = 10$) from which 10 per cent of the nodes are randomly removed. Averages over 5000 simulation runs.

Finally, we removed randomly 30 per cent of the nodes in the 20×20 ($P = 10$) grids (figures not shown here). No algorithm was significantly better than the others in this situation. The average numbers of transmitted messages over 10000 simulation runs were approximately 11.5 in the same parameter setting as above. This is only about 5 per cent of those in full grids. It seems now that the grids no longer provided a significant number of alternative routes among which to optimize.

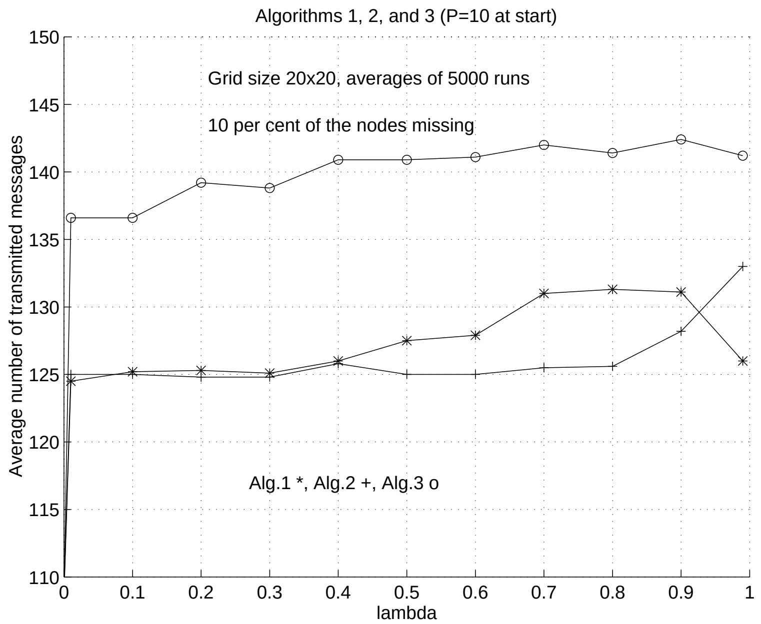

Figure 7.6: Algorithms 1, 2, and 3 on 20×20 grids ($P = 10$) from which 10 per cent of the nodes are randomly removed. Averages over 5000 simulation runs.

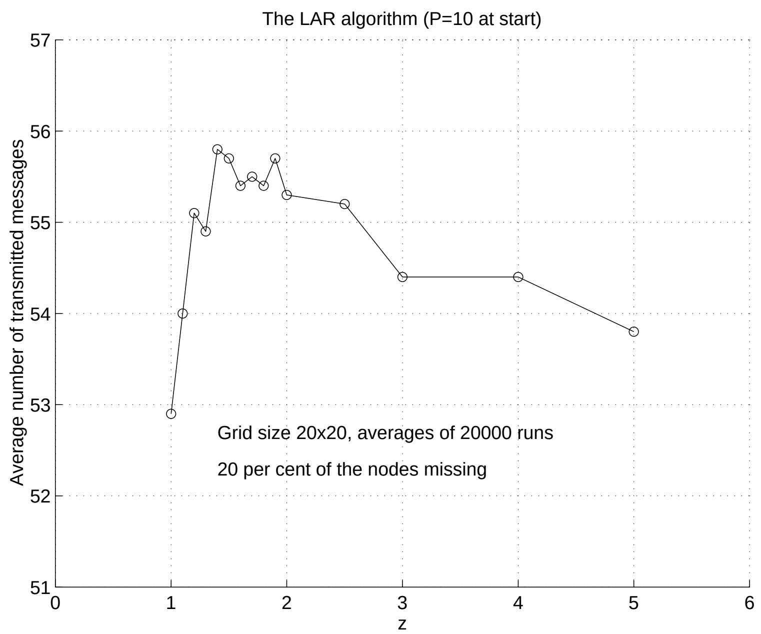

Figure 7.7: The LAR algorithm on 20×20 grids ($P = 10$) from which 20 per cent of the nodes are randomly removed. Averages over 20000 simulation runs.

In Figure 7.9, there are 10×10 grids with different initial power levels of nodes $P = 5, 10, 15, 20, 25, 30$. The measurements labeled by circles belong to Algorithm 3. The other graphs belong to the other three algorithms. Figure 7.9 shows that average lifetimes grow linearly with initial power levels of nodes. Algorithm 3 produces about 10 per cent longer average lifetimes than

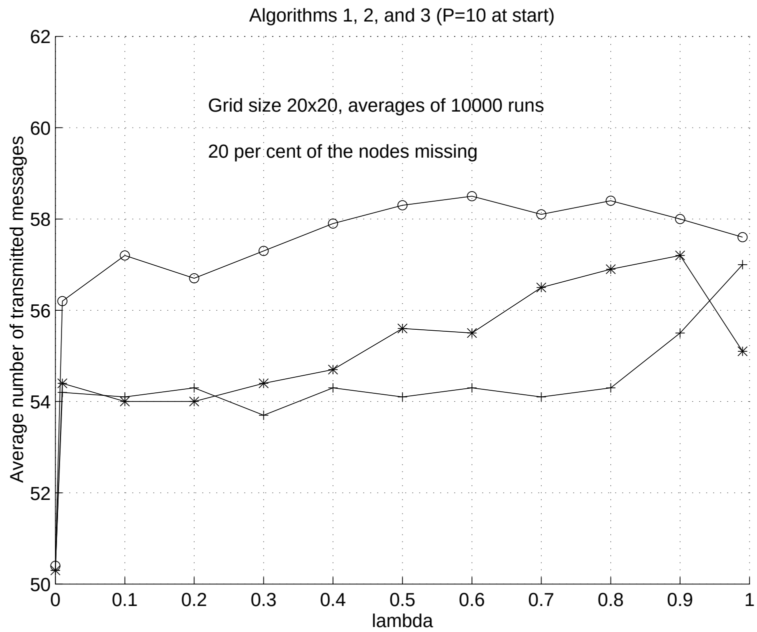

Figure 7.8: Algorithms 1, 2, and 3 on 20×20 grids ($P = 10$) from which 20 per cent of the nodes are randomly removed. Averages over 10000 simulation runs.

the other algorithms when $P \geq 20$.

In Figure 7.10, there are grids of different sizes, 10×10, 20×20, 30×30, and 40×40 with initial power levels of nodes $P = 10$. Again, the measurements labeled by circles belong to Algorithm 3 and the other graphs to the other three algorithms. Figure 7.10 shows that average lifetimes grow linearly with the square root of the grid size. Algorithm 3 produces about 10, 17, and 21 per cent longer average lifetimes than the other algorithms on 20×20, 30×30, and 40×40 grids, respectively. In Figures 7.9–7.10, Algorithms 1, 2, and 3 have $\lambda = 0.5$ and LAR has $z = 1.3$.

7.5 CONCLUSION

We have described three new A^* -based power-aware routing algorithms with different optimization criteria. We compared the algorithms with the *max-min* zP_{min} (LAR) method [48]. The algorithms require locations and energy levels for all the nodes in the network at all times. No knowledge of the future message sequence is assumed.

LAR runs Dijkstra's shortest path algorithm several times during its iterations. Our algorithms route a message by running the A^* algorithm only once, which means a shorter asymptotic running time compared to LAR. We also implemented LAR by replacing Dijkstra's algorithm with A^*. The CPU running times of the A^* version were about 60 per cent of those of the Dijkstra version.

In the simulation runs, we recorded the average number of transmitted

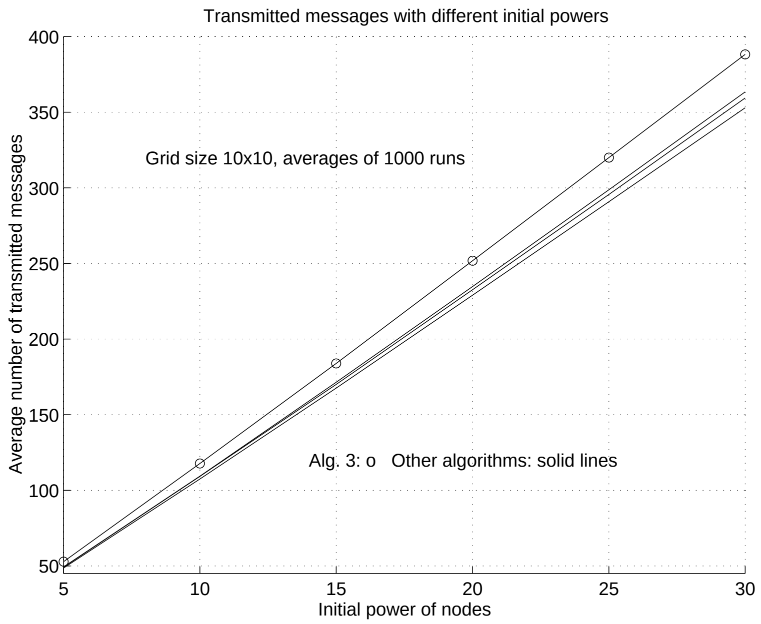

Figure 7.9: All the tested algorithms on 10×10 grids. The average number of transmitted messages vs. initial power levels of nodes.

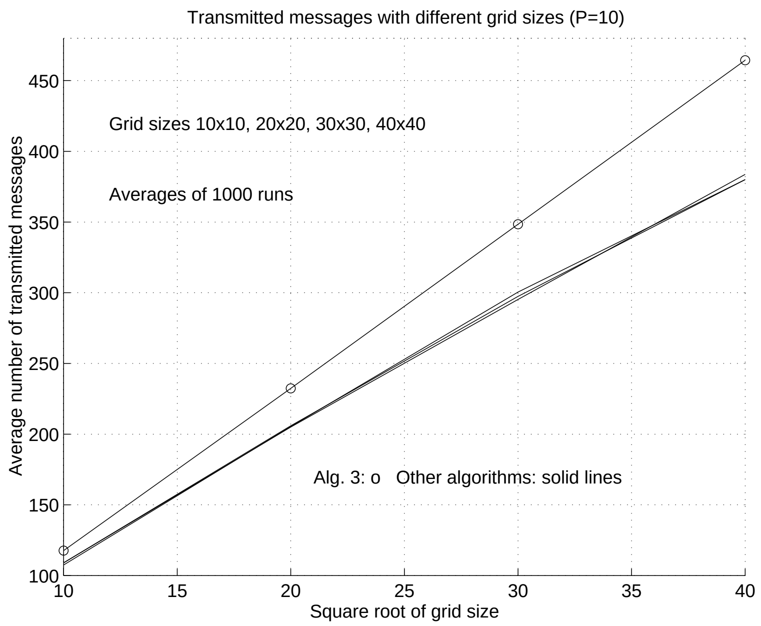

Figure 7.10: All the tested algorithms on grids with an initial power level of $P = 10$. The average number of transmitted messages vs. square root of grid size.

messages for each algorithm. This was done by simulating 1000–20000 message routing requests for each parameter setting between randomly selected nodes in 2-D grids. The grid sizes ranged from 100 to 1600 nodes, or devices. The results show that the new algorithms are not as sensitive to parameter settings as LAR. One of the new algorithms, Algorithm 3, routed on average more messages than LAR at its best: about 10, 17, and 21 per cent more on grids with 20 × 20, 30 × 30, and 40 × 40 nodes, respectively. When 10 per cent of the nodes in the 20 × 20 grids were randomly removed, Algorithm 3 was still the best one. After removing more nodes, Algorithm 3 was as good as LAR at its best.

Li, Aslam, and Rus [48] report that LAR achieves over 80 per cent of the optimal off-line algorithm (where future message sequences are known) for most problem instances they studied. If the latter and the data shown in Figure 7.1–7.10 are generalizable, then Algorithm 3 will achieve about 90 per cent of the optimal off-line algorithm for most of the time.

8 CONCLUSIONS

Path planning problems arise in many fields of technology, such as production planning, energy-aware message routing in large communication networks, resource allocation, and vehicles navigating systems.

This work includes new theoretical results, extensions, and applications on the A^* path finding algorithm.

First, we compared the computational work of several path planning algorithm classes to that of A^* in situations, where the heuristic function guiding the search process is *nonadmissible*.

We showed in Theorem 3.2.7 that A^* is optimal over certain admissible BF* algorithms independent of the admissibility of the heuristic. An admissible BF* algorithm is guaranteed to find an optimal path whenever A^* does if they both are guided by the same admissible heuristic. Theorem 3.2.7 is a generalization of the corresponding results in [18].

We showed in Theorem 3.2.5 that A^* is optimal over globally compatible DP strategies, independent of the admissibility of the heuristic. However, Theorem 3.2.5 needs an assumption that the search graph can also have negative edge costs. A globally compatible DP strategy is guaranteed to find an optimal path whenever A^* does if they both are guided by the same heuristic, possibly a nonadmissible one. Dechter and Pearl [18] proved a similar theorem constrained on admissible heuristics and graphs with strictly positive edge costs.

We also showed that A^* is optimal over some greedy path finding algorithms in Theorem 3.2.6. In greedy algorithms of this work, the most promising node next to be explored is chosen from the set of nodes with the minimum heuristic value.

Otherwise, it seems hard to generalize theorems like the ones in [18] that need the assumption of admissible heuristic. The price to pay for the nonadmissibility appears in various additional constraints in the other theorems in Chapter 3.

Second, we presented an algorithm A^*_A that improves a given static admissible heuristic dynamically, during search. We showed that A^*_A guided by the new dynamic heuristic is optimal over A^* guided by the static heuristic alone. We also showed that when A^*_A expands a node for the first time, it has found an optimal path to the node. It follows that no reexpansions of nodes are necessary, which is an important property considering the efficiency of the search.

A^*_A can utilize more information about the search history of the original problem than A^* does. The new information, called an approximating model, is used as a "guide map" to improve the heuristic function during search. We assume that searching the approximating problem is less costly than searching the original problem. If the construction of an effective approximating model is possible, then a problem specific tie-breaking rule of A^*_A reduces the total number of node expansions in the original problem on all "layers" of nodes n for which $f(n) = \text{constant} \leq C^*$. A tie-breaking rule of A^*, on the other hand, affects only expansions of nodes m for which $f(m) = C^*$. In this way, a user has more freedom to adapt the program structure to a particular problem instance than only constructing admissible static heuristics.

Third, we examined how A^* can be used as a resource allocation policy among several path planning algorithms solving the same task or its subtasks. We used bidirectional search as an example of a resource allocation problem between two path finding algorithms.

In some cases, (Meta)A^* is the optimal resource allocation policy among certain DP strategies by Theorem 5.4.1. This requires the possibility of underestimating, during the search, the number of the nodes still to be expanded at least in that subgraph, where a goal is found; in other words, the heuristic has to be admissible in that subgraph, see also Corollary 3.2.2. However, if the path planning algorithms exchange information, then the optimality of (Meta)A^* is relative to its tie-breaking rule.

If the heuristic guiding the resource allocation process has been nonadmissible, then (Meta)A^* is still an optimal resource allocation strategy among certain admissible DPBF and BF* strategies, see Theorems 3.2.6 and 3.2.7.

Fourth, we presented and tested four hierarchical robot path planning algorithms using five simulated robot workcells. Two of the algorithms are based on the new resource allocation scheme Meta A^*. In the algorithms, we do not have to guess right the coarsest resolutions of the graphs on which solution paths exist, i.e., the minimum robot movement discretizations. If we discretize the robot movements slightly too finely, then the Meta A^* based path planners do not unnecessarily expand very many additional fine resolution nodes. The Meta A^* based path planning can also be done by reducing the minimum allowed search resolution step by step starting from a coarse one.

The simulation data suggests that Meta A^* is a useful mechanism that reduces path planning times and memory consumption in hierarchical path planning. The algorithms can be “plugged into” roadmap methods as local planners: to find collision free paths between any two subgoals chosen by a global planner.

Fifth, we described three A^*-based algorithms for power-aware routing of messages in large communication networks where future message sequences are not known. We seek to maximize the average lifetime of the network. The lifetime of the network is the number of the messages with random start and goal nodes that can be sent via intermediate nodes within their energy capacities. For achieving this goal the algorithms use different optimization criteria.

The new algorithms are simpler and their running times are shorter than those of the widely quoted *max-min* zP_{min} algorithm. In addition, they are not as sensitive to parameter settings as *max-min* zP_{min}. We showed empirically that one of the algorithms produces longer average lifetimes than *max-min* zP_{min}.

BIBLIOGRAPHY

[1] Javed Aslam, Qun Li, and Daniela Rus. Three power-aware routing algorithms for sensor networks. *ACM Wireless Communications and Mobile Computing*, 3:187–208, 2003.

[2] Antti Autere. A dynamically improving heuristic for A^*. In *Proc. of the Int. ICSC Congress on Intelligent Systems and Applications (ISA'2000)*, Wollongong, NSW Australia, December, 12–15 2000. ICSC Academic Press. paper nro: 1514–114.

[3] Antti Autere. A^* as an optimal resource allocation policy in path finding problems. In *Proc. of the 14th International FLAIRS Conference*, pages 139–144, Key West, Florida, USA, May, 21–23 2001. AAAI Press.

[4] Antti Autere. Hierarchical A^* based path planning — a case study. *Knowledge Based Systems*, 15(1–2):53–66, 2002.

[5] Antti Autere. New online power-aware algorithms in wireless networks. In *Proc. of the 12th Int. Conference on Software, Telecommunications and Computer Networks (SoftCOM 2004)*, pages 439–443, Split–Dubrovnik, Croatia, Venice, Italy, October, 11–13 2004. University of Split.

[6] Antti Autere and Johannes Lehtinen. Robot motion planning by A^* search on a modified discretized configuration space. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'97)*, pages 1208–1213, Grenoble, France, September, 8–12 1997.

[7] A. Bagchi and A. Mahanti. Search algorithms under different kinds of heuristics - a comparative study. *J. ACM*, 30(1):1–21, 1983.

[8] Douglas M. Blough and Paolo Santi. Investigating upper bounds on network lifetime extension for cell-based energy conservation techniques in stationary ad hoc networks. In *Proc. of the 8th Annual International Conference on Mobile Computing and Networking (MOBICOM 2002)*, Atlanta, Georgia, September 2002.

[9] R. A. Brooks and T. Lozano-Peres. A subdivision algorithm in configuration space for findpath with rotation. In *Proc. of the 8th Int. Joint Conf. Artificial Intelligence*, pages 799–806, Karlsruhe, Germany, 1983.

[10] P.P. Chakrabarti, S. Ghose, A. Acharya, and S.C. de Sakar. Heuristic search in restricted memory. *Artificial Intelligence*, 41:197–221, 1989.

[11] Jae-Hwan Chang and Leandros Tassiulas. Energy conserving routing in wireless ad-hoc networks. In *Proc. of IEEE INFOCOM 2000*, pages 22–31, Tel Aviv, Israel, March 2000.

[12] Mario Čagalj, Jean-Pierre Hubaux, and Christian Enz. Minimum-energy broadcast in all-wireless networks: NP-completeness and distribution issues. In *Proc. of the 8th Annual International Conference on*

Mobile Computing and Networking (MOBICOM 2002), pages 172–182, Atlanta, Georgia, September 2002.

[13] Benjie Chen, Kyle Jamieson, Hari Balakrishnan, and Robert Morris. Span: An energy-efficient coordination algorithm for topology maintenance in ad-hoc wireless networks. In *Proc. of the 7th Annual International Conference on Mobile Computing and Networking (MOBICOM 2001)*, pages 85–96, Rome, Italy, July 2001.

[14] Bernhard Clavina. Solving findpath by combination of goal-directed and randomized search. In *IEEE International Conference on Robotics and Automation*, pages 1718–1723, Cincinnati, Ohio, May 13–18 1990.

[15] C. I. Connolly, J. B. Burns, and R. Weiss. Path planning using laplace's equation. In *IEEE International Conference on Robotics and Automation*, pages 2102–2106, May 13–18 1990.

[16] D. de Champeaux. Bidirectional heuristic search again. *J. ACM*, 30(1):22–32, 1983.

[17] Rina Dechter and Judea Pearl. The optimality of A^* revisited. In *Proc. of the 3rd AAAI National Conference on AI*, pages 95–99, Washington D.C., USA, 1983.

[18] Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality of A^*. *J. ACM*, 32(3):505–536, 1985.

[19] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematic*, 1:269–271, 1959.

[20] S. Doshi, S. Bhandare, and T. Brown. An on-demand minimum energy routing protocol for a wireless ad hoc networks. *ACM Mobile Computing and Communications Review*, 6(3):50–66, 2002.

[21] D. Dreyfus. An appraisal of some shortest path algorithms. *Operations Research*, 17:395–412, 1969.

[22] S. E. Dreyfus and A. M. Law. *The Art and Theory of Dynamic Programming*. Academic Press, Orlando, 1977.

[23] Brad Paden et al. Path planning using a jacobian-based freespace generation algorithm. In *IEEE International Conference on Robotics and Automation*, pages 1732–1737, Washington, 1989.

[24] P. Bergamo et al. Distributed power control for energy efficient routing in ad hoc networks. *Wireless Networks*, 10:29–42, 2004.

[25] T. Ikeda et al. A fast algorithm for finding better routes by AI search techniques. In *Proc. Vehicle Navigation and Information Systems Conference*, pages 291–296, IEEE, 1994.

[26] Tomas Lozano-Peres et al. Spatial planning: A configuration space approach. *IEEE Trans. Computers C-32(2):108-120*, 1983.

[27] S. Gottschalk et.al. Obbtree: A hierarchical structure for rapid interference detection. In *Computer Graphics Proceedings, SIGGRAPH 96*, August 4–9 1996.

[28] Henri Farreny. Completeness and admissibility for general heuristic search algorithms - a theoretical study: Basic concepts and proofs. *Journal of Heuristics*, 5:353–376, 1999.

[29] Andrew V. Goldberg and Chris Harrelson. Computing the shortest path: A^* search meets graph theory. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 156–165, Vancouver, Canada, January, 23–25 2005.

[30] Othar Hansson, Andrew Mayer, and Marco Valtorta. A new result on the complexity of heuristic estimates for the A^* algorithm. *Artificial Intelligence*, 55:129–143, 1992.

[31] L. R. Harris. The heuristic search under conditions of error. *Artificial Intelligence*, 5(3):217–234, 1974.

[32] 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.

[33] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient routing protocols for wireless microsensor networks. In *Proc. of the 33rd Hawaii International Conference on System Sciences (HICSS 2000)*, pages 1–10, Maui, Hawaii, January 2000.

[34] Th. Horsch, F. Schwarz, and H. Tolle. Motion planning with many degrees of freedom - random reflections at c-space obstacles. In *IEEE International Conference on Robotics and Automation*, pages 3318–3323, San Diego, California, May 8–13 1994.

[35] Y. K. Hwang and N. Ahuja. Gross motion planning - a survey. *ACM Computing Surveys*, 24(3):219–291, 1992.

[36] F. Janabi-Sharifi and D. Vinke. Integration of the artificial potential field approach with simulated annealing for robot path planning. In *IEEE International Symposium on Intelligent Control*, pages 536–541, Chicago, Illinois, August 1993.

[37] K. Kar, M. Kodialam, T. V. Lakshnam, and L. Tassiulas. Routing for network capacity maximization in energy-constrained ad-hoc networks. In *Proc. of IEEE INFOCOM 2003*, San Francisco, CA, March-April 2003.

[38] M. Karp and J. Pearl. Searching for an optimal path in a tree with random costs. *Artificial Intelligence*, 21(1):99–116, 1983.

[39] Lydia Kavraki and Jean-Claude Latombe. Randomized preprosessing of configuration space for fast path planning. In *IEEE International Conference on Robotics and Automation*, pages 2138–2145, San Diego, California, May 8–13 1994.

[40] Richard E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

[41] Richard E. Korf. Real-time heuristic search. *Artificial Intelligence*, 42:189–211, 1990.

[42] Richard E. Korf. Linear-space best-first search. *Artificial Intelligence*, 62:41–78, 1993.

[43] J. Kwa. BS^*: An admissible bidirectional staged heuristic search algorithm. *Artificial Intelligence*, 38(1):95–109, 1989.

[44] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1993.

[45] E.L. Lawler and D.E. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719, 1966.

[46] Johannes Lehtinen. Private communications.

[47] L. Li, V. Bahl, Y. Wang, and R. Wattenhofer. Distributed topology control for power efficient operation in multihop wireless ad hoc networks. In *Proc. of IEEE INFOCOM 2001*, April 2001.

[48] Qun Li, Javed Aslam, and Daniela Rus. Online power-aware routing in wireless ad-hoc networks. In *Proc. of the 7th Annual International Conference on Mobile Computing and Networking (MOBICOM 2001)*, pages 97–107, Rome, Italy, July 2001.

[49] J. Liu, F. Zhao, and D. Petrovic. Information-directed routing in ad hoc sensor networks. In *Proc. 2nd ACM Int. Conf. on Wireless Sensor Networks and Applications*, pages 88–97, San Diego, CA, September 19 2003.

[50] Laszlo Mérõ. A heuristic search algorithm with modifiable estimate. *Artificial Intelligence*, 23:13–27, 1984.

[51] A. Mahanti and K. Ray. Heuristic search in networks with modifiable estimate. In *Proc. of the 15th ACM Annual Conference on Computer Science*, pages 166–173, St. Louis, Missouri, USA, February 17–19 1987.

[52] Alberto Martelli. On the complexity of admissible search algorithms. *Artificial Intelligence*, 8:1–13, 1977.

[53] Jack Mostow and Armand E. Prieditis. Discovering admissible heuristics by abstracting and optimizing: A transformational approach. In *Proc. of the 11th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 701–707, Detroit, USA, 1989.

[54] T.A.J. Nicholson. Finding the shortest route between two points in a network. *Computer J.*, 9:275–280, 1966.

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

[56] J. Pearl and J. Kim. Studies in semi-admissible heuristics. *IEEE Trans. on Pattern Analysis and Machine Intelligence PAMI-4*, 4(4):392–399, 1982.

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

[58] I. Pohl. First results on the effect of error in heuristic search. *Machine Intelligence*, 5:219–236, 1969.

[59] I. Pohl. Bi-directional search. *Machine Intelligence*, 6:127–140, 1971.

[60] I. Pohl. The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving. In *Proc. of the 3rd International Joint Conference on Artificial Intelligence (IJCAI)*, Stanford, Calif., USA, August 20–23 1973. William Kaufman, Inc.

[61] Armand Prieditis and Robert Davis. Quantitatively relating abstractness to the accuracy of admissible heuristics. *Artificial Intelligence*, 74:165–175, 1995.

[62] Armand E. Prieditis. *Discovering Effective Admissible Heuristics by Abstraction and Speedup: a Transformational Approach.* PhD thesis, Rutgers University, New Brunswick, NJ, 1990.

[63] E. Ralli and G. Hirzinger. Fast path planning for robot manipulators using numerical potential fields in the configuration space. In *IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS'94)*, pages 494–501, Munich, Germany, September 12–16 1994.

[64] Volkan Rodoplu and Teresa H. Meng. Minimum energy mobile wireless networks. In *Proc. of the 1998 IEEE International Conference on Communications (ICC'98)*, pages 1633–1639, Atlanta, Georgia, June 1998.

[65] S. Russel. Efficient memory-bounded search methods. In *Proc. of the 10th European Conference on Artficial Intelligence (ECAI)*, pages 1–5, Vienna, Austria, 1992.

[66] A. K. Sen and A. Baghi. Fast recursive formulations for best-first search that allow controlled use of memory. In *Proc. of the 11th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 297–302, Detroit, USA, 1989.

[67] Eugene Shih, Paramvir Bahl, and Michael J. Sinclair. Wake on wireless: An event driven energy saving strategy for battery operated devices. In *Proc. of the 8th Annual International Conference on Mobile Computing and Networking (MOBICOM 2002)*, Atlanta, Georgia, September 2002.

[68] S. Singh, M. Woo, and C.S. Raghavendra. Power-aware routing in mobile ad-hoc networks. In *Proc. of the 4th Annual International Conference on Mobile Computing and Networking (MOBICOM 1998)*, pages 181–190, Dallas, Texas, October 1998.

[69] L. Sint and D. de Champeaux. An improved bidirectional heuristic search algorithm. *J. ACM*, 24(2):177–191, 1977.

[70] A. Stentz. The focused D^* algorithm for real-time replanning. In *Proc. of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1652–1659, Montreal, Canada, 1995.

[71] L. Tarassenko and A. Blake. Analogue computation of collision-free paths. In *IEEE International Conference on Robotics and Automation*, Sacramento, California, April 9–11 1991.

[72] Marco Valtorta. A result on the computational complexity of heuristic estimates for the A^* algorithm. *Information Sciences*, 34:47–59, 1984.

[73] Charles W. Warren. Fast path planning using modified A^* method. In *IEEE International Conference on Robotics and Automation*, pages 662–667, Atlanta, Georgia, May, 2–6 1993.

[74] Y. Xu, J. Heidemann, and D. Estrin. Geography-informed energy conservation for ad-hoc routing. In *Proc. of the 7th Annual International Conference on Mobile Computing and Networking (MOBICOM 2001)*, pages 70–84, Rome, Italy, July 2001.

HELSINKI UNIVERSITY OF TECHNOLOGY LABORATORY FOR THEORETICAL COMPUTER SCIENCE
RESEARCH REPORTS

HUT-TCS-A85	Emilia Oikarinen Testing the Equivalence of Disjunctive Logic Programs. December 2003.
HUT-TCS-A86	Tommi Syrjänen Logic Programming with Cardinality Constraints. December 2003.
HUT-TCS-A87	Harri Haanpää, Patric R. J. Östergård Sets in Abelian Groups with Distinct Sums of Pairs. February 2004.
HUT-TCS-A88	Harri Haanpää Minimum Sum and Difference Covers of Abelian Groups. February 2004.
HUT-TCS-A89	Harri Haanpää Constructing Certain Combinatorial Structures by Computational Methods. February 2004.
HUT-TCS-A90	Matti Järvisalo Proof Complexity of Cut-Based Tableaux for Boolean Circuit Satisfiability Checking. March 2004.
HUT-TCS-A91	Mikko Särelä Measuring the Effects of Mobility on Reactive Ad Hoc Routing Protocols. May 2004.
HUT-TCS-A92	Timo Latvala, Armin Biere, Keijo Heljanko, Tommi Junttila Simple Bounded LTL Model Checking. July 2004.
HUT-TCS-A93	Tuomo Pyhälä Specification-Based Test Selection in Formal Conformance Testing. August 2004.
HUT-TCS-A94	Petteri Kaski Algorithms for Classification of Combinatorial Objects. June 2005.
HUT-TCS-A95	Timo Latvala Automata-Theoretic and Bounded Model Checking for Linear Temporal Logic. August 2005.
HUT-TCS-A96	Heikki Tauriainen A Note on the Worst-Case Memory Requirements of Generalized Nested Depth-First Search. September 2005.
HUT-TCS-A97	Toni Jussila On Bounded Model Checking of Asynchronous Systems. October 2005.
HUT-TCS-A98	Antti Autere Extensions and Applications of the A^* Algorithm. November 2005.

ISBN 951-22-7974-9
ISSN 1457-7615