State Space Abstraction in Artificial Intelligence and Operations Research

Robert Holte
Computing Science Department
University of Alberta
Edmonton, Canada T6G 2E8
(rholte@ualberta.ca)

Gaojian Fan
Computing Science Department
University of Alberta
Edmonton, Canada T6G 2E8
(gaojian@ualberta.ca)

Abstract

In this paper we compare the abstraction methods used for state space search and planning in Artificial Intelligence with the state space relaxation methods used in Operations Research for various optimization problems such as the Travelling Salesman problem (TSP). Although developed independently, these methods are based on exactly the same general idea: lower bounds on distances in a given state space can be derived by computing exact distances in a "simplified" state space. Our aim is to describe these methods so that the two communities understand what each other has done and can begin to work together.

Introduction

In this paper we compare the abstraction methods used for state space search and planning in Artificial Intelligence (A.I.) with the state space relaxation methods used in Operations Research (O.R.) for various optimization problems such as the Travelling Salesman problem (TSP). We find that the two communities have independently invented exactly the same general idea: lower bounds on distances in a given state space can be derived by computing exact distances in a "simplified" state space. The intimate connection between the ideas of these two communities has been noted once before, by Hernádvölgyi (2003; 2004).

This idea was introduced to the O.R. community by Christofides, Mingozzi, and Toth (1981). Even a cursory survey of the O.R. literature reveals that this technique has continued to be developed up to the present day and is the key technology for state-of-the-art performance in a variety of O.R. problems (Abdul-Razaq and Potts 1988; Baldacci, Mingozzi, and Roberti 2012; Boland, Dethridge, and Dumitrescu 2006; Frantzeskakis and Watson-Gandy 1989; Gouveia, Paias, and Voß 2011; Righini and Salani 2008; 2009; Roberti and Mingozzi 2014).

In A.I., a specific form of this idea, which Pearl (1984) later called "state space relaxation", was first developed in the Milan Polytechnic Artificial Intelligence Project (Guida and Somalvico 1979) and independently invented by John Gaschnig (1979). The general idea of state space abstraction is best credited to Mostow and Prieditis (1989). The first major successes in using abstraction to achieve state-of-the-art performance in A.I. are due to Culberson and Schaeffer (1996) and Korf (1997).

State Space Search and Abstraction in A.I.

In optimal state space planning or search, one is given a weighted directed graph (the state space), a start state, $start$, and a goal state (or goal condition), $goal$, and the aim is to find an optimal (least-cost) path from $start$ to $goal$.[1] A variety of different algorithms to solve this problem have been developed, with A* (Hart, Nilsson, and Raphael 1968) and IDA* (Korf 1985) being the most widely used. The order in which states are expanded[2] by these algorithm is based on the function $f(s) = g(s) + h(s)$, where $g(s)$ is the lowest cost of all the paths from $start$ to s seen so far, and $h(s)$, the heuristic function, is an estimate of the distance (cost of the cheapest path) from s to $goal$, denoted $d(s, goal)$. If $h(s)$ is a lower bound on $d(s, goal)$, for all s, the heuristic function is said to be "admissible". A* and IDA* are guaranteed to return an optimal path from $start$ to $goal$ if h is admissible.

An abstraction of a state space $\mathcal{S}$ is any mapping ψ to any state space $\mathcal{T}$ such that the distance between a pair of states in $\mathcal{S}$ is never smaller than the distance between the corresponding states in $\mathcal{T}$. Stated formally, $\psi : \mathcal{S} \rightarrow \mathcal{T}$ is an abstraction if and only if $d_{\mathcal{T}}(\psi(s_1), \psi(s_2)) \leq d_{\mathcal{S}}(s_1, s_2)$ for all states $s_1, s_2 \in \mathcal{S}$. Here $d_{\mathcal{S}}$ and $d_{\mathcal{T}}$ refer to distances measured in $\mathcal{S}$ and $\mathcal{T}$ respectively. If $\psi : \mathcal{S} \rightarrow \mathcal{T}$ is an abstraction, the function $h_\psi(s) = d_{\mathcal{T}}(\psi(s), \psi(goal))$ is an admissible heuristic for state space search in $\mathcal{S}$.

Using abstraction to define heuristics for state space search in this way is only practical if $\psi(s)$ and $d_{\mathcal{T}}(\psi(s), \psi(goal))$ can both be quickly calculated for every state $s \in \mathcal{S}$ and every possible $goal$. We will discuss each of these in turn.

Computing ψ Efficiently

If the state space $\mathcal{S}$ is given "explicitly", i.e. as a set of nodes and a set of edges enumerated individually, then $\mathcal{T}$ can also be represented explicitly[3] and so can ψ, as a set of

[1] The nodes in the graph are called states. The weight of an edge in the graph is the cost of including the edge in a path. Costs are assumed to be non-negative.

[2] to "expand" state s is to generate all its successors, i.e. all those states that can be reached directly, in a single step, from s.

[3] unless $\mathcal{T}$ is much larger than $\mathcal{S}$, which sometimes happens. It also sometimes happens that states outside $\psi(\mathcal{S})$ are needed to compute distances.

edges from states in $\mathcal{S}$ to the corresponding states in $\mathcal{T}$. In this case the calculation of $\psi(s)$ is very efficient: one just follows the ψ-edge from s to the corresponding state in $\mathcal{T}$.

Normally, however, $\mathcal{S}$ is far too large to be given explicitly. Instead, it is defined "implicitly" in a formalism specifically designed for defining state spaces, such as PDDL (Ghallab et al. 1998), SAS$^+$ (Bäckström 1992), or PSVN (Holte, Arneson, and Burch 2014). These formalisms differ in many details, but the following is representative of how they define a state space. A state is represented by a set of state variables whose values are drawn from a finite set of possible values, the domain Dom. Transitions between states (the edges in the state space) are defined by a set of operators, each of which has a precondition, an effect, and a cost. Operator op's precondition defines the set of states, $pre(op)$, to which it can be applied. Operator op's effect is a function, which we denote $op(s)$, mapping a state $s \in pre(op)$ to a state. For the sake of illustration, we will assume that the precondition of an operator is a (possibly empty) set of tests of the form $V_i = v_i$ where V_i is state variable i and $v_i \in Dom$, and the effect of an operator is a set of assignments of the form $V_i \leftarrow v_i$ for all the state variables that are changed by the operator.

As an example of a state space definition in this formalism, consider the 5-disk Towers of Hanoi puzzle. In this puzzle, there are three pegs and five disks of different sizes, and at any point in time each disk is on one of the pegs. The disks on each peg form a stack so that only the top disk on a peg can be moved. The top disk on peg_i can be moved to another peg, peg_j, only if peg_j has no disks on it or the top disk on peg_j is larger than the disk being moved. There are many ways of representing this puzzle's state space in the formalism described in the previous paragraph. The simplest is to have five state variables $(V_1, \ldots, V_5)$, with V_i indicating the peg on which disk i currently sits (disk 1 is the smallest disk, disk 5 is the largest). The domain for these variables is the set of pegs, $Dom = \{peg_1, peg_2, peg_3\}$. An operator moves a specific disk from one specific peg to another. There are therefore a total of 30 operators in this representation (5 choices for the disk, and 6 choices for the two pegs involved in the move). The precondition of the operator that moves disk 4 from peg_2 to peg_3 tests that disk 4 is on peg_2 and that the smaller disks are on peg_1. In the representation we're using in this paper, this would be expressed as

$$V_1 = peg_1, V_2 = peg_1, V_3 = peg_1, V_4 = peg_2$$

The only effect of this operator is to change the peg on which disk 4 sits, which is expressed in our representation as

$$V_4 \leftarrow peg_3$$

When a state space is given implicitly, an abstraction ψ must operate on the state space definition, not on the underlying state space (graph). Given a definition, $D_{\mathcal{S}}$ of state space $\mathcal{S}$, $\psi(D_{\mathcal{S}})$ produces a state space definition $D_{\mathcal{T}}$ for some state space $\mathcal{T}$. Note the ψ operates on both the states of $\mathcal{S}$ and on the operators of $D_{\mathcal{S}}$, in the form in which they are represented in the chosen formalism.

In order for ψ, defined on state space definitions, to satisfy the formal requirements of being an abstraction ($d_{\mathcal{T}}(\psi(s_1), \psi(s_2)) \leq d_{\mathcal{S}}(s_1, s_2)$ for all states $s_1, s_2 \in \mathcal{S}$), it is sufficient[4] to show that ψ has the following properties (here op_ψ is the abstract operator produced by applying ψ to operator op):

(i) $s \in pre(op) \Longrightarrow \psi(s) \in pre(op_\psi)$
(ii) $s \in pre(op) \Longrightarrow \psi(op(s)) = op_\psi(\psi(s))$
(iii) $cost(op_\psi) \leq cost(op)$

The first two properties guarantee that every edge in $\mathcal{S}$ has a corresponding edge in $\mathcal{T}$ and the third property guarantees that the cost of the corresponding edge in $\mathcal{T}$ is no larger than the cost of the edge in $\mathcal{S}$.

In order to be able to calculate $\psi(s)$ efficiently, for any state s, its calculation must be efficient when expressed in terms of the state variables used to represent states. Likewise the preconditions and effects of the abstract operators (op_ψ) must be succinctly representable in the chosen formalism and efficient to calculate.

One example of such an abstraction is "projection", which merely ignores some of the state variables. This is trivial to apply to states and to preconditions and effects in the form described above. In the Towers of Hanoi representation above, projecting out (ignoring) variable V_i has the intuitive meaning "ignore disk i". For example, if V_1 and V_2 were both projected out, the operator shown above would have only two preconditions ($V_3 = peg_1, V_4 = peg_2$) and would allow disk 4 to be moved from peg_2 to peg_3 regardless of the locations of disks 1 and 2. If V_4 were projected out, this operator would have no effects (it would be the identity operator).

Another example of an abstraction technique that is easily applied to state space definitions is domain abstraction, which is based on a mapping ϕ from the domain Dom to a smaller domain Dom'. In this case, $\psi(s)$ is calculated by replacing each value v in the vector representing state s by $\phi(v)$. Likewise, ψ is applied to operator preconditions and effects by replacing all occurrences of value v in them by $\phi(v)$. In the Towers of Hanoi representation above, domain abstraction would make some of the pegs indistinguishable from one another. For example, suppose $Dom' = \{peg_a, peg_b\}$ and ϕ maps peg_1 and peg_2 to peg_a and maps peg_3 to peg_b. Knowing that a disk is on peg_a indiciates that it is either on peg_1 or peg_2 but we can't tell which. The precondition of the operator above would become

$$V_1 = peg_a, V_2 = peg_a, V_3 = peg_a, V_4 = peg_a$$

and its effect

$$V_4 \leftarrow peg_b.$$

This allows disk 4 to be moved to peg_3 (= peg_b) as long as it and the disks smaller than it are not currently on peg_3.

As a final example, the historically first kind of state space abstraction (Gaschnig 1979; Guida and Somalvico 1979) was called state space relaxation by Pearl (1984). A state space relaxation, in this sense, is a mapping that eliminates one or more of the tests in the precondition of one or more of the operators but otherwise acts as the identity function

[4]These conditions are sufficient but not necessary. For example, delete relaxation is an abstraction but it does not always satisfy property (ii).

($\psi(s) = s$ and operators' effects are not changed by ψ). For example, if the precondition $V_4 = peg_2$ was removed from the operator above, disk 4 could be moved to peg_3 no matter which peg it was on as long as the smaller disks were on peg_1. Heuristics defined by state space relaxation were shown to be ineffective by Valtorta (1984).

The most recent forms of state space abstraction are delete relaxation (Hoffmann and Nebel 2001), Cartesian abstraction (Seipp and Helmert 2014), and merge-and-shrink (Helmert et al. 2014; Fan, Müller, and Holte 2014). There are also other, powerful methods for defining heuristic functions that do not neatly fit this particular definition of state space abstraction but are, in the same spirit, based on computing some sort of distance-like measure in a space derived from a given state space definition $D_{\mathcal{S}}$. Most notable among these are h^m (Haslum and Geffner 2000) and operator counting methods (Pommerening et al. 2014).

Computing $d_{\mathcal{T}}(\psi(s), \psi(goal))$ Efficiently

The most straightforward approach to computing $d_{\mathcal{T}}(\psi(s), \psi(goal))$ is to search in $\mathcal{T}$ for an optimal path from $\psi(s)$ to $\psi(goal)$ on demand, i.e. when $h(s)$ is needed in the search from $start$ to $goal$ in $\mathcal{S}$. This is, in fact, how heuristics based on delete relaxation are computed.[5]

In many situations, however, a much more efficient calculation is possible. For example, if the abstract space $\mathcal{T}$ is sufficiently small, it is feasible to compute, and store in a data structure, distances to $\psi(goal)$ for all states in $\mathcal{T}$ from which $\psi(goal)$ can be reached. This is done as a preprocessing step before search in $\mathcal{S}$ begins. During search in $\mathcal{S}$ the calculation of $h(s)$ is extremely efficient, it is simply a lookup of the distance associated with $\psi(s)$ in the data structure. This is how pattern databases (Culberson and Schaeffer 1996; Ball and Holte 2008) and merge-and-shrink (Helmert et al. 2014; Fan, Müller, and Holte 2014) are implemented.

When $d_{\mathcal{T}}(\psi(s), \psi(goal))$ is being computed by searching in $\mathcal{T}$ on demand, two methods have been proposed to make this search faster. The first is to use hierarchical heuristic search (Holte et al. 1996; Holte, Grajkowski, and Tanner 2005; Leighton, Ruml, and Holte 2011). The idea here is to speed up the search in $\mathcal{T}$ by using heuristic search (e.g. A* or IDA*) instead of a "blind" search algorithm like breadth-first search or Dijkstra's algorithm.

The heuristic used for searching in $\mathcal{T}$ is created by abstracting $\mathcal{T}$ to create a more abstract space $\mathcal{T}'$ and using exact distances in $\mathcal{T}'$ as estimates of the distances in $\mathcal{T}$. The search to compute distances in $\mathcal{T}'$ can either be blind or heuristic, with the heuristic being defined by abstracting $\mathcal{T}'$ to create a yet more abstract space $\mathcal{T}''$, and so on. In this way a hierarchy of abstract spaces is created with $\mathcal{S}$ at the bottom, then $\mathcal{T}$, then $\mathcal{T}'$, then $\mathcal{T}''$ etc.

Another method for speeding up on-demand search in $\mathcal{T}$ is to cache information from one search in $\mathcal{T}$ and use it to speed up subsequent searches in $\mathcal{T}$. For example, when $h(start)$ is calculated by finding an optimal path in $\mathcal{T}$ from $\psi(start)$ to $\psi(goal)$, optimal distances to $\psi(goal)$ are known for all the abstract states on this path, not just for $\psi(start)$. If sufficient memory is available, it makes sense to store these so that if they are needed in the future they can just be looked up rather than being recalculated. Holte et al. (1996) found it necessary to use such caching techniques in order for hierarchical heuristic search based on A* to outperform blind search in $\mathcal{S}$.

State Space Search and Relaxation in O.R.

In this section we describe the state space search and relaxation methods introduced to the O.R. community by Christofides, Mingozzi, and Toth (1981), who we will refer to as CMT, and compare these methods to the A.I. search and abstraction methods described in the previous section.

The set of "routing" problems addressed by CMT seems, on the face of it, to be much broader than, and include as a special case, the shortest path problem to which the A.I. techniques described in the previous section have been applied. CMT defines a routing problem as follows (a "route" in their terminology is what we have called a path). Given a weighted, directed graph G "find a route, or set of routes, in the graph G, so that certain constraints are satisfied and the total cost of the routes is minimized." This includes, for example, the travelling salesman problem (TSP), which is not a shortest path problem in the given graph G.

However, CMT is quick to point out that "every routing problem is essentially a shortest path problem on some underlying graph with additional constraints" and that the dynamic programming formulation of a routing problem defines the state space in which a shortest path is to be sought and also the start state and goal condition. For example, in the natural dynamic programming formulation of the TSP, a state summarizes the information about a partial tour using two components (state variables): (a) the set S of vertices in G ("cities") that have been visited so far, and (b) the last city visited. The start state, $start$, is $\langle\{\}, c_0\rangle$, where c_0 is the city in which the tour starts and ends, and, if there are $n+1$ cities in G ($c_0, \ldots, c_n$), the goal state, $goal$, is $\langle\{c_0, , \ldots, c_n\}, c_0\rangle$.[6] For every edge $e_{xy} = (c_x, c_y)$ in G there is an operator, $go\text{-}from\text{-}c_x\text{-}to\text{-}c_y$, which can be applied to state $\langle S, c\rangle$ as long as $c = c_x$ and $c_y \notin S$. The cost of this operator is the weight of e_{xy} in G and its result, when applied to state $\langle S, c\rangle$, is the state $\langle S \cup \{c_y\}, c_y\rangle$. The graph G therefore serves as an implicit representation of this state space, in exactly the sense defined in the previous section and, indeed, could be directly encoded in the standard A.I. formalisms for defining state spaces.

The state space relaxations developed by CMT are relaxations of this state space, not the graph G, and are directly aimed at producing a lower bound, $h(s)$, on the cost of reaching $goal$, in this state space, from state s, in order to speed up search for a shortest path from $start$ to $goal$ in this state space. They are therefore directly analogous to the state space abstraction methods discussed in the previous section.

The general definition CMT gives of a state space relaxation (Section II.A. in (Christofides, Mingozzi, and Toth

[5] Actual implementations may be more complex than this simple description for efficiency's sake.

[6] The exact formulation in CMT is slightly different in certain unimportant details.

1981)) is virtually identical to properties (i)-(iii) given in the previous section, and their discussion of what is required for a state space relaxation to be useful in practice touches on issues very similar to those in the previous section.

Much of the remainder of CMT's paper is devoted to examples of state space relaxations specifically for the TSP and two other routing problems, the Travelling Salesman Problem with Time Windows, and the Vehicle Routing Problem. In all these problems, the representation of a state involves the set, S, of cities that have been visited so far. CMT observes that the state space is exponentially larger than the given graph of cities, G, because S can be any subset of the cities. Their relaxations are different ways of mapping S's 2^n possible values onto a smaller set of possible values.

One of their methods for the TSP, for example, they call q-path relaxation. In this method, an integer $q_i \geq 1$ is associated with each city c_i, and the set S is mapped to the quantity $\phi(S) = \sum_{c_i \in S} q_i$. The choice of the q_i determines the size of the relaxed state space. For example, if $q_i = 1$ for all i then $\phi(S) = |S|$, the cardinality of S, and the relaxed state space contains n^2 states. At the other extreme, if $q_i = 2^i$ then each different S gets mapped to a unique integer and the relaxed state space is isomorphic to the original state space.

Given a maximum value $\hat{Q}$ for the sum of all the q_i, CMT proposes searching for a set of q_i values that maximizes $h(start)$, the distance from $start$ to $goal$ in the relaxed state space. They refer to this process as "state-space modification". In A.I. terminology, this is searching through the space of possible abstractions of a particular type to find the "best" abstraction of that type (for a particular definition of "best"). For example, Hernádvölgyi searched in the space of possible domain abstractions of Rubik's Cube for the largest pattern database that would fit in memory ((Hernádvölgyi 2001); for additional details see Section 4.5 of (Hernádvölgyi 2004)), and Haslum et al. (2007) searched among sets of projection abstractions for the set whose "canonical heuristic function" was estimated to minimize search effort. For the TSP, CMT's state-space modification search consisted of adjusting the current value of q_i up or down depending on the occurrences of city c_i in the optimal path from $start$ to $goal$ in the relaxed state space defined by the current q_i values. Roughly speaking, q_i would be decreased if city c_i did not occur in the path, and would be increased if it occurred more than once.[7] The whole process was then repeated with the new set of q_i values. This is not guaranteed to improve the value of $h(start)$ but in the experiment reported by CMT it increased $h(start)$ from being 90% of optimal (1536 compared to the true optimal solution cost of 1704) to being 99.8% of optimal (1700). A similar search for good q_i has been used for a job scheduling problem (Abdul-Razaq and Potts 1988).

A modern variation on this method is "decremental" state space relaxation (Boland, Dethridge, and Dumitrescu 2006; Righini and Salani 2008; 2009). The set of cities is divided into two groups, critical and non-critical. The non-critical cities have q_i values associated with them. In the relaxed state space, a state records precisely which subset of the critical cities has been visited en route to the state and $\sum q_i$ for the non-critical cities. There is an iterative process in which additional cities are moved from the non-critical group to the critical group. It starts with all cities in the non-critical group. This is ordinary q-path relaxation, as described above. If the sequence of cities in the optimal path from $start$ to $goal$ in this relaxed state space satisfies all the constraints of the original problem, then it is an optimal solution to the original problem. If a constraint is violated, one or more of the non-critical cities are added to the critical group, with the aim of preventing that constraint violation from possibly occurring in the new relaxed state space. For example, in the TSP, every city is supposed to be visited once and only once. Some or all of the cities that are not visited at all, or that are visited more than once, would be added to the critical group. This process continues until an optimal solution is found that violates none of the constraints. This must eventually happen, because once all the cities are critical, the relaxed state space is equal to the original state space. It is hoped, however, that it will happen when there are few critical cities, since that implies the relaxed state space is small. A similar idea has recently been introduced in the A.I. community under the name Counterexample Guided Abstraction Refinement (Seipp and Helmert 2013; 2014).[8]

Any of the alternatives for computing distances in the relaxed state space discussed in the previous section could be applied, but the O.R. community tends to favour on-demand calculation because it allows different relaxed state spaces to be used for different states.[9] For example, consider the two states of the TSP $s_1 = \langle\{c_1, c_2, c_5\}, c_5\rangle$ and $s_2 = \langle\{c_3, c_4, c_5\}, c_5\rangle$. The relaxation $\phi(S) = |S|$ maps both states to $\langle 3, c_5\rangle$, so if the same state space relaxation is used throughout the state space, they will have the same heuristic value. However, the cities on the path to c_5 in these two states are entirely distinct and it is often advantageous to take that into account when defining the relaxed state spaces in which $h(s_1)$ and $h(s_2)$ are calculated.

Conclusions

The O.R. community based on the seminal work of Christofides, Mingozzi, and Toth (1981) and the A.I. community using state space abstraction to define heuristic functions for state space planning/search clearly have much of value to exchange with one another. The routing problems studied in O.R., when formulated as dynamic programs, are shortest path problems in state spaces that have special properties compared to the generic states spaces studied by A.I., but the general notion of state space relaxation/abstraction is identical in both cases, and often provides the key technology for obtaining state-of-the-art performance. The O.R. routing problems would be challenging tests of the generic

[7] personal communication, Aristide Mingozzi, Oct. 9, 2014.

[8] Yang et al. (2007) test for the feasibility of an additive combination of heuristic values, but do not modify the abstractions when infeasibility is detected.

[9] personal communication, Aristide Mingozzi, Oct. 9, 2014.

A.I. abstraction methods, and the idea of state-space modification in response to constraint violations is deserving of close study by the A.I. community.

Acknowledgements

Aristide Mingozzi was very helpful in clarifying details about state space relaxation in the O.R. sense (Christofides, Mingozzi, and Toth 1981). Financial support for this research was in part provided by Canada's Natural Science and Engineering Research Council (NSERC).

References

Abdul-Razaq, T. S., and Potts, C. N. 1988. Dynamic programming state-space relaxation for single-machine scheduling. *Journal of the Operational Research Society* 39(2):141–152.

Bäckström, C. 1992. Equivalence and tractability results for SAS$^+$ planning. In *Proceedings of the 3rd International Conference on Principles on Knowledge Representation and Reasoning*, 126–137.

Baldacci, R.; Mingozzi, A.; and Roberti, R. 2012. New state-space relaxations for solving the traveling salesman problem with time windows. *INFORMS Journal on Computing* 24(3):356–371.

Ball, M., and Holte, R. C. 2008. The compression power of symbolic pattern databases. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS)*, 2–11.

Boland, N.; Dethridge, J.; and Dumitrescu, I. 2006. Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Operations Research Letters* 34(1):58 – 68.

Christofides, N.; Mingozzi, A.; and Toth, P. 1981. State-space relaxation procedures for the computation of bounds to routing problems. *Networks* 11(2):145–164.

Culberson, J., and Schaeffer, J. 1996. Searching with pattern databases. In *Proceedings of the 11th Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, volume 1081 of *Lecture Notes in Computer Science*, 402–416. Springer.

Fan, G.; Müller, M.; and Holte, R. 2014. Non-linear merging strategies for merge-and-shrink based on variable interactions. In *Proceedings of the Seventh Annual Symposium on Combinatorial Search (SoCS)*.

Frantzeskakis, M., and Watson-Gandy, C. D. T. 1989. The use of state-space relaxation for the dynamic facility location problem. *Annals of Operations Research* 18.

Gaschnig, J. 1979. A problem similarity approach to devising heuristics: First results. In *Proceedings of the 6th International Joint Conference on Artificial Intelligence*, 301–307.

Ghallab, M.; Isi, C. K.; Penberthy, S.; Smith, D. E.; Sun, Y.; and Weld, D. 1998. PDDL - the planning domain definition language. Technical report, CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

Gouveia, L.; Paias, A.; and Voß, S. 2011. Models for a traveling purchaser problem with additional side-constraints. *Computers & OR* 38(2):550–558.

Guida, G., and Somalvico, M. 1979. A method for computing heuristics in problem solving. *Information Sciences* 19(3):251–259.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Systems Science and Cybernetics* 4(2):100–107.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*, 140–149.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence*, 1007–1012.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *J. ACM* 61(3):16:1–63.

Hernádvölgyi, I. T. 2001. Searching for macro operators with automatically generated heuristics. In *Advances in Artificial Intelligence, 14th Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, 194–203.

Hernádvölgyi, I. 2004. *Automatically Generated Lower Bounds for Search.* Ph.D. Dissertation, University of Ottawa.

Herndvlgyi, I. T. 2003. Solving the sequential ordering problem with automatically generated lower bounds. In *Selected Papers of the International Conference on Operations Research (OR 2003)*, 355–362. Springer Verlag.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *J. Artif. Intell. Res. (JAIR)* 14:253–302.

Holte, R. C.; Arneson, B.; and Burch, N. 2014. PSVN Manual (June 20, 2014). Technical Report TR14-03, Computing Science Department, University of Alberta.

Holte, R. C.; Perez, M. B.; Zimmer, R. M.; and MacDonald, A. J. 1996. Hierarchical A*: Searching abstraction hierarchies efficiently. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI)*, 530–535.

Holte, R. C.; Grajkowski, J.; and Tanner, B. 2005. Hierarchical heuristic search revisited. In *Abstraction, Reformulation and Approximation, 6th International Symposium (SARA)*, 121–133.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence* 27(1):97–109.

Korf, R. 1997. Finding optimal solutions to Rubik's Cube using pattern databases. In *Proceedings of the 14th AAAI Conference on Artificial Intelligence*, 700–705.

Leighton, M. J.; Ruml, W.; and Holte, R. C. 2011. Faster optimal and suboptimal hierarchical search. In *Proceedings of the Fourth Annual Symposium on Combinatorial Search (SoCS)*.

Mostow, J., and Prieditis, A. 1989. Discovering admissible heuristics by abstracting and optimizing: A transformational approach. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, 701–707.

Pearl, J. 1984. *Heuristics – Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2014. Lp-based heuristics for cost-optimal planning. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS)*.

Righini, G., and Salani, M. 2008. New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks* 51(3):155–170.

Righini, G., and Salani, M. 2009. Decremental state space relaxation strategies and initialization heuristics for solving the orienteering problem with time windows with dynamic programming. *Computers & OR* 36(4):1191–1203.

Roberti, R., and Mingozzi, A. 2014. Dynamic ng-path relaxation for the delivery man problem. *Transportation Science* 48(3):413–424.

Seipp, J., and Helmert, M. 2013. Counterexample-guided cartesian abstraction refinement. In *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS)*.

Seipp, J., and Helmert, M. 2014. Diverse and additive cartesian abstraction heuristics. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS)*.

Valtorta, M. 1984. A result on the computational complexity of heuristic estimates for the A* algorithm. *Information Sciences* 34(1):47–59.

Yang, F.; Culberson, J. C.; and Holte, R. 2007. Using infeasibility to improve abstraction-based heuristics. In *7th International Symposium on Abstraction, Reformulation, and Approximation (SARA)*, 413–414.