# 3

# Mechanical Generation of Admissible Heuristics

Robert Holte, Jonathan Schaeffer, and Ariel Felner

## 1 Introduction

This chapter takes its title from Section 4.2 of Judea Pearl's landmark book *Heuristics* [Pearl 1984], and explores how the vision outlined there has unfolded in the quarter-century since its appearance. As the book's title suggests, it is an in-depth summary of classical artificial intelligence (AI) heuristic search, a subject to which Pearl and his colleagues contributed substantially in the early 1980s.

The purpose of heuristic search is to find a least-cost path in a state space from a given start state to a goal state. In principle, such problems can be solved by classical shortest path algorithms, such as Dijkstra's algorithm [Dijkstra 1959], but in practice the state spaces of interest in AI are far too large to be solved in this way. One of the seminal insights in AI was recognizing that even extremely large search problems can be solved quickly if the search algorithm is provided with additional information in the form of a heuristic function $h(s)$ that estimates the distance from any given state $s$ to the nearest goal state [Doran and Michie 1966; Hart, Nilsson, and Raphael 1968]. A heuristic function $h(s)$ is said to be *admissible* if, for every state $s$, $h(s)$ is a lower bound on the true cost of reaching the nearest goal from state $s$. Admissibility is desirable because it guarantees the optimality of the solution found by the most widely-used heuristic search algorithms.

Most of the chapters in *Heuristics* contain mathematically rigorous definitions and analysis. In contrast, Chapter 4 offers a conceptual account of where heuristic functions come from, and a vision of how one might create algorithms for automatically generating effective heuristics from a problem description. An early version of the chapter had been published previously in the widely circulated *AI Magazine* [Pearl 1983].

Chapter 4's key idea is that distances in the given state space can be estimated by computing exact distances in a "simplified" version of the state space. There are many different ways a state space can be simplified. Pearl focused almost exclusively on *relaxation*, which is done by weakening or eliminating one or more of the conditions that restrict how one is allowed to move from one state to another. For example, to estimate the driving distance between two cities, one can ignore the constraint that driving must be done on roads. In this relaxed version of the problem, the distance between two cities is simply the straight-line distance. It is

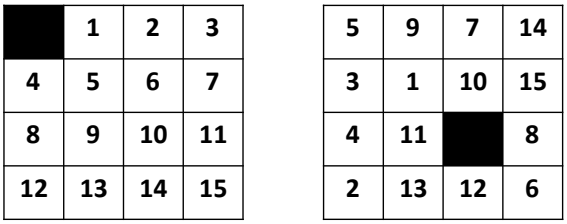


Figure 1. 15-puzzle

easy to see, in general, that distances in a relaxed space cannot exceed distances in the given state space, and therefore the heuristic functions defined in this way are guaranteed to be admissible. An alternate way of looking at this is to view the elimination of conditions as equivalent to adding new edges to the search graph. Therefore, optimal solutions to the relaxed graph (with the additional edges) must be a lower bound on the solution to the original problem.

As a second example of relaxation, consider the 15-puzzle shown in Figure 1, which consists of a set of tiles numbered 1-15 placed in a $4 \times 4$ grid, leaving one square in the grid unoccupied (called the "blank" and shown as a black square). The only moves that are permitted are to slide a tile that is adjacent to the blank into the blank position, effectively exchanging the tile with the blank. For example, four moves are possible in the right-hand side of Figure 1: tile 10 can be moved down, tile 11 can be moved right, tile 8 can be moved left, and tile 12 can be moved up. To solve the puzzle is to find a sequence of moves that transforms a given scrambled state (right side of Figure 1) into a goal state (such as the one on the left). One possible relaxation of the 15-puzzle state space can be defined by removing the restriction that a tile must be adjacent to the blank to be moveable. In this relaxation any tile can move from its current position to any adjacent position at any time, regardless of whether the adjacent position is occupied or not. The number of moves required to solve this relaxed version (called the Manhattan Distance) is clearly less than or equal to the number of moves required to solve the 15-puzzle itself. Note that in this case the relaxed state space has many more states than the original 15-puzzle (many tiles can now occupy a single location) but it is easier to solve, at least for humans (tiles move entirely independently of one another).

Pearl observes that in AI a state space is almost always defined implicitly by a set of operators that describe a successor relation between states. Each operator has a precondition defining the states to which it can be applied and a postcondition describing how the operator changes the values of the variables used to describe a state. This implies that relaxing a state space description by eliminating one or more preconditions is a simple syntactic operation, and the set of all possible relaxations of a state space description (by eliminating combinations of preconditions) is well-defined and, in fact, easy to enumerate. Hence it is entirely feasible for a mechanical

system to generate heuristic functions and, indeed, to search through the space of heuristic functions defined by eliminating preconditions in all possible ways.

The mechanical search through a space of heuristic functions has as its goal, in Pearl's view, a heuristic function with two properties. First, the heuristic function should return values that are as close to the true distances as possible (Chapter 6 in *Heuristics* justifies this). Second, the heuristic function must be efficiently computable, otherwise the reduction in search effort that the heuristic function produces might be outweighed by the increase in computation time caused by the calculation of the heuristic function. Pearl saw the second requirement as the more difficult to detect automatically and proposed that mechanically-recognizable forms of decomposability of the relaxed state space would be the key to mechanically generating efficiently-computable heuristic functions. Pearl recognized that the search for a good heuristic function might itself be quite time-consuming, but argued that this cost was justified because it could be amortized over an arbitrarily large number of problem instances that could all be solved much more efficiently using the same heuristic function.

The preceding paragraphs summarize Pearl's vision for how effective heuristics might be generated automatically from a state space description. The remainder of our chapter contains a brief look at the research efforts directed towards realizing Pearl's vision. We conclude that Pearl correctly anticipated a fundamental breakthrough in heuristic search in the general terms he set out in Chapter 4 of *Heuristics* although not in all of its specifics. Our discussion is informal and the ideas presented and their references are illustrative, not exhaustive.

## 2 The Vision Emerges

The idea of using a solution in a simplified state space to guide the search for a solution in the given state space dates to the early days of AI [Minsky 1963] and was first implemented and shown to be effective in the ABSTRIPS system [Sacerdoti 1974]. However, these early methods did not use the cost of the solution in the simplified space as a heuristic function; they used the solution itself as a skeleton which was to be *refined* into a solution in the given state space by inserting additional operators.

The idea of using distances in a simplified space as heuristic estimates of distances in the given state space came later. It did not originate with Judea Pearl (in fact, he credits Stan Rosenschein for drawing the idea to his attention). However, by devoting a chapter of his otherwise technical book to the speculative idea that admissible heuristic functions could be created automatically, he became an important early promoter of it.

The idea was first developed in the Milan Polytechnic Artificial Intelligence Project in the period 1973-1979. In a series of papers (*e.g.* [Sangiovanni-Vincentelli and Somalvico 1973; Guida and Somalvico 1979]) the Milan group developed the core elements of Pearl's vision. They proposed defining a heuristic function as the exact distance in a relaxed state space and proved that such heuristic func-

tions would be both admissible and *consistent*.[1] To make the computation of such heuristic functions efficient the Milan group envisaged a hierarchy of relaxed spaces, with search at one level being guided by a heuristic function defined by distances in the level above. The Milan group also foresaw the possibility of algorithms for searching through the space of possible simplified state spaces, although the first detailed articulation of this idea, albeit in a somewhat different context, was by Richard Korf [1980].

John Gaschnig [1979] picked up on the Milan work. He made the key observation that if a heuristic function is calculated by searching in a relaxed space, the total time required to solve the problem using the heuristic function could exceed the time required to solve the problem directly with breadth-first search (*i.e.* without using the heuristic function). This was formally proven shortly afterwards by Marco Valtorta [1981, 1984]. This observation led to a focus on the efficiency with which distances in the simplified space could be computed. The favorite approach to doing this (as exemplified in *Heuristics*) was to search for simplified spaces that could be decomposed.

## 3 The Vision Becomes a Reality

Directly inspired by Pearl's vision, Jack Mostow and Armand Prieditis set themselves the task of automating what had hitherto been paper-and-pencil speculation. The result was their ABSOLVER system [Mostow and Prieditis 1989; Prieditis 1993], which fully vindicated Pearl's enthusiasm for the idea of mechanically generating effective, admissible heuristics.

The input to ABSOLVER was a state space description in the standard STRIPS notation [Fikes and Nilsson 1971]. ABSOLVER had a library containing two types of transformations, each of which would take as input a STRIPS representation of a state space and produce as output one or more other STRIPS representations. The first type of transformation were *abstracting* transformations. Their purpose was to create a simplification (or "abstraction") of the given state space. One of these was *drop precondition*, exactly as Pearl had proposed. Their other abstracting transformations were a type of simplification that Pearl had not anticipated—they were *homomorphisms*, which are many-to-one mappings of states in the given space to states in the abstract space. Homomorphic state space abstractions for the purpose of defining heuristic functions were first described by Dennis Kibler in an unpublished report [1982], but their importance was not appreciated until ABSOLVER and the parallel work done by Keki Irani and Suk Yoo [1988].

An example of a homomorphic abstraction of the 15-puzzle is shown in Figure 2. Here tiles 9-15 and the blank are just as in the original puzzle (Figure 1) but tiles 1-8 have had their numbers erased so that they are not distinguishable from each other. Hence for any particular placement of tiles 9-15 and the blank, all the different ways

[1] Heuristic function $h(s)$ is consistent if, for any two states $s_1$ and $s_2$, $h(s_1) \leq dist(s_1, s_2) + h(s_2)$, where $dist(s_1, s_2)$ is the distance from $s_1$ to $s_2$.

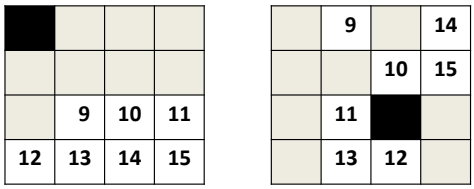


Figure 2. Homomorphic abstraction of the 15-puzzle

of permuting tiles 1-8 among the remaining positions produce 15-puzzle states that map to the same abstract state, even though they would all be distinct states in the original state space. For example, the abstract state in the left part of Figure 2 is the abstraction of the goal state in the original 15-puzzle (left part of Figure 1), but it is also the abstraction of all the non-goal states in the original puzzle in which tiles 9-15 and the blank are in their goal positions but some or all of tiles 1-8 are not. Using this abstraction, the distance from the 15-puzzle state in the right part of Figure 1 to the 15-puzzle goal state would be estimated by calculating the true distance, in the abstract space, from the abstract state in the right part of Figure 2 to the state in the left part of Figure 2.

In addition to abstracting transformations, ABSOLVER's library contained "optimizing" transformations, which would create an equivalent description of a given STRIPS representation in which search could be completed more quickly. This included the "factor" transformation that would, if possible, decompose the state space into independent subproblems, one of the methods Pearl had suggested.

ABSOLVER was applied to thirteen state spaces and found effective heuristic functions in six of them. Five of the functions it discovered were novel, including a simple, effective heuristic for Rubik's Cube that had been overlooked by experts:

> after extensive study, Korf was unable to find a single good heuristic evaluation function for Rubik's Cube [Korf 1985]. He concluded that "if there does exist a heuristic, its form is probably quite complex."
>
> ([Mostow and Prieditis 1989], page 701)

## 4 Dawn of the Modern Era

Despite ABSOLVER's success, it did not launch the modern era of abstraction-based heuristic functions. That would not happen until 1994, when Joe Culberson and Jonathan Schaeffer's work on *pattern databases* (PDBs) first appeared [Culberson and Schaeffer 1994]. They used homomorphic abstractions of the kind illustrated in Figure 2 and, as explained above, defined the heuristic function, $h(s)$, of state $s$ to be the actual distance in the abstract space between the abstract state corresponding to $s$ and the abstract goal. The key idea behind PDBs is to store the heuristic function as a lookup table so that its calculation during a search is extremely fast.

To do this, it is necessary to precompute all the distances to the goal state in the abstract space. This is typically done by a backwards breadth-first search starting at the abstract goal state. Each abstract state reached in this way is associated with a specific storage location in the PDB, and the state's distance from the abstract goal is stored in this location as the value in the PDB.

Precomputing abstract distances to create a lookup-table heuristic function was actually one of the optimizing transformations in ABSOLVER, but Culberson and Schaeffer had independently come up with the idea. Unlike the ABSOLVER work, they validated it by producing a two orders of magnitude reduction in the search effort (measured in nodes expanded) needed to solve instances of the 15-puzzle, as compared to the then state-of-the-art search algorithms using an enhanced Manhattan Distance heuristic. To achieve this they used two PDBs totaling almost one gigabyte of memory, a very large amount in 1994 when the experiments were performed [Culberson and Schaeffer 1994]. The paper's referees were sharply critical of the exorbitant memory usage, rejecting the paper three times before it finally was accepted [Culberson and Schaeffer 1996].

Such impressive results on the 15-puzzle could not go unnoticed. The fundamental importance of PDBs was established beyond doubt in 1997 when Richard Korf used PDBs to enable standard heuristic search techniques to find optimal solutions to instances of Rubik's Cube for the first time [Korf 1997].

Since then, PDBs have been used to build effective heuristic functions in numerous applications, including various combinatorial puzzles [Felner, Korf, and Hanan 2004; Felner, Korf, Meshulam, and Holte 2007; Korf and Felner 2002], multiple sequence alignment [McNaughton, Lu, Schaeffer, and Szafron 2002; Zhou and Hansen 2004], pathfinding [Anderson, Holte, and Schaeffer 2007], model checking [Edelkamp 2007], planning [Edelkamp 2001; Edelkamp 2002; Haslum, Botea, Helmert, Bonet, and Koenig 2007], and vertex cover [Felner, Korf, and Hanan 2004].

## 5 Current Status

The use of abstraction to create heuristic functions has profoundly advanced the fields of planning and heuristic search. But the current state of the art is not entirely as Pearl envisaged. Although he recognized that there were other types of state space abstraction, Pearl emphasized relaxation. In this detail, he was too narrowly focused. Researchers have largely abandoned relaxation in favor of homomorphic abstractions, of which many types have been developed and shown useful for defining heuristic functions, such as domain abstraction [Hernádvölgyi and Holte 2000], $h$-abstraction [Haslum and Geffner 2000], projection [Edelkamp 2001], constrained abstraction [Haslum, Bonet, and Geffner 2005], and synchronized products [Helmert, Haslum, and Hoffmann 2007].

Pearl argued for the automatic creation of effective heuristic functions by searching through a space of abstractions. There has been some research in this direction [Prieditis 1993; Hernádvölgyi 2003; Edelkamp 2007; Haslum, Botea, Helmert,

Bonet, and Koenig 2007; Helmert, Haslum, and Hoffmann 2007], but more is needed. However, important progress has been made on the subproblem of evaluating the effectiveness of a heuristic function, with the development of a generic, practical method for accurately predicting how many nodes IDA* (a standard heuristic search algorithm) will expand for any given heuristic function [Korf and Reid 1998; Korf, Reid, and Edelkamp 2001; Zahavi, Felner, Burch, and Holte 2008].

Finally, Pearl anticipated that efficiency in calculating the heuristic function would be achieved by finding abstract state spaces that were decomposable in some way. This has not come to pass, although there is now a general theory of when it is admissible to add the values returned by two or more different abstractions [Yang, Culberson, Holte, Zahavi, and Felner 2008]. Instead, the efficiency of the heuristic calculation has been achieved either by precomputing the heuristic function's values and storing them in a lookup table, as PDBs do, or by creating a hierarchy of abstractions and using distances at one level as a heuristic function to guide the calculation of distances at the level below [Holte, Perez, Zimmer, and MacDonald 1996; Holte, Grajkowski, and Tanner 2005], as anticipated by the Milan group.

## 6 Conclusion

Judea Pearl has received numerous accolades for his prodigious research and its impact. Amidst this impressive body of work are his often-overlooked contributions to the idea of the automatic discovery of heuristic functions. Even though *Heuristics* is over 25 years old (ancient by Computing Science standards), Pearl's ideas still resonate today.

**Acknowledgments:** The authors gratefully acknowledge the support they have received over the years for research in this area from Canada's Natural Sciences and Engineering Research Council (NSERC), Alberta's Informatics Circle of Research Excellence (iCORE), and the Israeli Science Foundation (ISF).

## References

Anderson, K., R. Holte, and J. Schaeffer (2007). Partial pattern databases. In *Symposium on Abstraction, Reformulation and Approximation*, pp. 20–34. Springer-Verlag LNAI #4612.

Culberson, J. and J. Schaeffer (1994). Efficiently searching the 15-puzzle. Technical Report 94-08, Department of Computing Science, University of Alberta.

Culberson, J. and J. Schaeffer (1996). Searching with pattern databases. In G. McCalla (Ed.), *AI'96: Advances in Artificial Intelligence*, pp. 402–416. Springer-Verlag LNAI #1081.

Dijkstra, E. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik 1*, 269–271.

Doran, J. and D. Michie (1966). Experiments with the graph traverser program. In *Proceedings of the Royal Society A*, Volume 294, pp. 235–259.

Edelkamp, S. (2001). Planning with pattern databases. In *European Conference on Planning*, pp. 13–24.

Edelkamp, S. (2002). Symbolic pattern databases in heuristic search planning. In *Artificial Intelligence Planning and Scheduling (AIPS)*, pp. 274–283.

Edelkamp, S. (2007). Automated creation of pattern database search heuristics. In *Model Checking and Artificial Intelligence*, pp. 35–50. Springer-Verlag LNAI #4428.

Felner, A., R. Korf, and S. Hanan (2004). Additive pattern database heuristics. *Journal of Artificial Intelligence Research (JAIR) 22*, 279–318.

Felner, A., R. Korf, R. Meshulam, and R. Holte (2007). Compressed pattern databases. *Journal of Artificial Intelligence Research (JAIR) 30*, 213–247.

Fikes, R. and N. Nilsson (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence 2*(3/4), 189–208.

Gaschnig, J. (1979). A problem similarity approach to devising heuristics: First results. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 301–307.

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

Hart, P., N. Nilsson, and B. Raphael (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics SCC-4(2)*, 100–107.

Haslum, P., B. Bonet, and H. Geffner (2005). New admissible heuristics for domain-independent planning. In *National Conference on Artificial Intelligence (AAAI)*, pp. 1163–1168.

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

Haslum, P. and H. Geffner (2000). Admissible heuristics for optimal planning. In *Artificial Intelligence Planning and Scheduling (AIPS)*, pp. 140–149.

Helmert, M., P. Haslum, and J. Hoffmann (2007). Flexible abstraction heuristics for optimal sequential planning. In *Automated Planning and Scheduling*, pp. 176–183.

Hernádvölgyi, I. (2003). Solving the sequential ordering problem with automatically generated lower bounds. In *Operations Research 2003 (Heidelberg, Germany)*, pp. 355–362.

Hernádvölgyi, I. and R. Holte (2000). Experiments with automatically created memory-based heuristics. In *Symposium on Abstraction, Reformulation and Approximation*, pp. 281–290. Springer-Verlag LNAI #1864.

Holte, R., J. Grajkowski, and B. Tanner (2005). Hierarchical heuristic search revisited. In *Symposium on Abstraction, Reformulation and Approximation*, pp. 121–133. Springer-Verlag LNAI #3607.

Holte, R., M. Perez, R. Zimmer, and A. MacDonald (1996). Hierarchical A*: Searching abstraction hierarchies efficiently. In *National Conference on Artificial Intelligence (AAAI)*, pp. 530–535.

Irani, K. and S. Yoo (1988). A methodology for solving problems: Problem modeling and heuristic generation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 10*(5), 676–686.

Kibler, D. (1982). Natural generation of admissible heuristics. Technical Report TR-188, University of California at Irvine.

Korf, R. (1980). Towards a model of representation changes. *Artificial Intelligence 14*(1), 41–78.

Korf, R. (1985). *Learning to solve problems by searching for macro-operators*. Marshfield, MA, USA: Pitman Publishing, Inc.

Korf, R. (1997). Finding optimal solutions to Rubik's Cube using pattern databases. In *National Conference on Artificial Intelligence (AAAI)*, pp. 700–705.

Korf, R. and A. Felner (2002). Disjoint pattern database heuristics. *Artificial Intelligence 134*(1-2), 9–22.

Korf, R. and M. Reid (1998). Complexity analysis of admissible heuristic search. In *National Conference on Artificial Intelligence (AAAI)*, pp. 305–310.

Korf, R., M. Reid, and S. Edelkamp (2001). Time complexity of iterative-deepening-A*. *Artificial Intelligence 129*(1-2), 199–218.

McNaughton, M., P. Lu, J. Schaeffer, and D. Szafron (2002). Memory efficient A* heuristics for multiple sequence alignment. In *National Conference on Artificial Intelligence (AAAI)*, pp. 737–743.

Minsky, M. (1963). Steps toward artificial intelligence. In E. Feigenbaum and J. Feldman (Eds.), *Computers and Thought*, pp. 406–452. McGraw-Hill.

Mostow, J. and A. Prieditis (1989). Discovering admissible heuristics by abstracting and optimizing: A transformational approach. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 701–707.

Pearl, J. (1983). On the discovery and generation of certain heuristics. *AI Magazine 4*(1), 23–33.

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

Prieditis, A. (1993). Machine discovery of effective admissible heuristics. *Machine Learning 12*, 117–141.

Sacerdoti, E. (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence 5*(2), 115–135.

Sangiovanni-Vincentelli, A. and M. Somalvico (1973). Theoretical aspects of state space approach to problem solving. In *International Congress on Cybernetics*, Namur, Belgium.

Valtorta, M. (1981). *A Result on the Computational Complexity of Heuristic Estimates for the A* Algorithm*. Ph.D. thesis, Department of Computer Science, Duke University.

Valtorta, M. (1984). A result on the computational complexity of heuristic estimates for the A* algorithm. *Information Sciences 55*, 47–59.

Yang, F., J. Culberson, R. Holte, U. Zahavi, and A. Felner (2008). A general theory of additive state space abstractions. *Journal of Artificial Intelligence Research (JAIR) 32*, 631–662.

Zahavi, U., A. Felner, N. Burch, and R. Holte (2008). Predicting the performance of IDA* with conditional probabilities. In *National Conference on Artificial Intelligence (AAAI)*, pp. 381–386.

Zhou, R. and E. Hansen (2004). Space-efficient memory-based heuristics. In *National Conference on Artificial Intelligence (AAAI)*, pp. 677–682.