Unifying Single-Agent and Two-Player Search

Jonathan Schaeffer[1] and Aske Plaat[2]

[1] Computing Science Dept.
University of Alberta
Edmonton, Alberta
Canada T6G 2H1
jonathan@cs.ualberta.ca

[2] Aske Plaat
Computer Science Dept.
Vrije Universiteit
Amsterdam
The Netherlands
aske@xs4all.nl

Abstract. The seminal works of Nilsson and Pearl in the 1970's and early 1980's provide a formal basis for splitting the field of heuristic search into two subfields: single- and two-agent search. The subfields are studied in relative isolation from each other; each having its own distinct character. Despite the separation, a close inspection of the research shows that the two areas have actually been converging. This paper argues that the single/two-agent distinction is not the essence of heuristic search anymore. The state space is characterized by a number of key properties that are defined by the application; single- versus two-agent is just one of many. Both subfields have developed many search enhancements; they are shown to be surprisingly similar and general. Given their importance for creating high performance search applications, it is these enhancements that form the essence of our field. Focusing on their generality emphasizes the opportunity for reuse of the enhancements, allows the field of heuristic search to be redefined as a single unified field, and points the way towards a modern theory of search based on the taxonomy proposed here.

1 Introduction

Heuristic search is one of the oldest fields in artificial intelligence. Nilsson and Pearl [20, 21] wrote the classic introductions to the field. In these works (and others) search algorithms are typically classified by the kind of problem space they explore. Two classes of problem spaces are identified: state spaces and problem reduction spaces. Many problems can be conveniently represented as a state space; these are typically problems that seek a path from the root to the goal state. Other problems are a more natural fit for problem reduction spaces, typically problems whose solution is a strategy. Sometimes both representations are viable options. Problem reduction spaces are AND/OR graphs; AO* is the best-known framework for creating search algorithms for this class of problems [2, 20]. State spaces are OR graphs; the A* algorithm can optimally solve this class of problems [10]. Note that a state space (OR graph) is technically just a special case of a problem reduction space (AND/OR graph).

Since their inception, the notions of OR graphs and AND/OR graphs have found widespread use in artificial intelligence and operations research. Both areas have active research communities which continue to evolve and refine new search algorithms and enhancements. Of the two representations, the state space representation has proven to be the more popular. It appears that many real-world problem solving tasks can be modeled naturally as OR graphs. Well-known examples include the shortest path problems, sliding-tile puzzles, and NP-complete problems.

One application domain that fits the AND/OR graph model better is two-agent (two-player) games such as chess. In these games, one player chooses moves to maximize a payoff function (the chance to win) while the opponent chooses moves to minimize it. Thus, the AND/OR graphs become MIN/MAX graphs, and the algorithms to search these spaces are known as minimax algorithms. Curiously, it appears that two-player games are the *only* applications for which AND/OR algorithms have found widespread use. To contrast A*-like OR graph algorithms with two-player minimax algorithms, they are often referred to as single-agent (or one-player) search algorithms.

With the advent of Nilsson's AND/OR framework, two-agent search has been given a firm place within the larger field of heuristic search. Since AND/OR graphs subsume OR graphs, there is a satisfying conceptual unification of the two subfields. However, the impact of this unified view on the practice of research into heuristic search methods has been minor. The two subfields have continued to develop in parallel, with little interaction between them. One reason for the lack of coherence between the two communities could be the difference in objectives: a case can be made that winning chess tournaments requires a different mind set than optimizing industrial problems to increase revenue.

This article has the following contributions: to our understanding of heuristic search:

- Single-agent and two-agent search algorithms both traverse search graphs. The difference between the two algorithms is not in the graph, but in the semantics imposed by the application. Much of the research done in single- and two-agent search does not depend on the search algorithm, but on the search space properties.
- Nilsson's [20] and Pearl's [21] dichotomy—the OR versus AND/OR choice—is misleading. Heuristic search consists of identifying properties of the search space and implementing a number of search techniques that make effective use of these properties. There are many such properties, and the choice of backup rule (minimaxing in two-agent search; minimization in single-agent search) is but one. The implication of Nilsson's and Pearl's model is that the choice of backup rule is in some way fundamental; it is not. This paper argues

for viewing heuristic search as the process in which properties of a search space are specified. Once that has been done, the relevant search techniques (basic algorithm and enhancements) follow naturally.
- Over the years researchers have uncovered an impressive array of search enhancements that can have a dramatic effect on search efficiency. The typical scenario is that the idea is developed in one of the domains and possibly later reinvented in the other. In this paper we list search space properties under which many search enhancements are applicable, showing that the distinction between single- and two-agent search is not essential. By merging the work done in these two areas, the commonalities and differences can be identified. This can be used to construct a generic search framework for designing high performance search algorithms.

The message of this article is that single- and two-agent search can and should be considered as a single undivided field. It can, because the essence of search is *enhancements*, not algorithms as is usually thought. It should, because researchers can benefit by taking advantage of work done in a related field, without reinventing the technology, if they would only realize its applicability. Given all the similarities between the two areas, one has to ask the question: why is it so important to make a distinction based on the backup rule?

This article is organized as follows: Section 2 discusses the importance of search enhancements. Section 3 gives a taxonomy of properties of the search space, which are matched up with the applicable search techniques in Section 4. Section 5 draws some conclusions. The article is restricted to classical search; algorithms such as simulated annealing and hill climbing are outside our scope.

2 Algorithms vs Enhancements

Most introductory texts on artificial intelligence start off explaining heuristic search by differentiating between different search strategies, such as depth-first, breadth-first, and best-first. Single-agent search is introduced, perhaps illustrated with the 15-Puzzle. Another section is then devoted to two-player search algorithms. The minimax principle is explained, often followed by alpha-beta pruning. The focus in these texts is on explaining the basic search algorithms and possibly their fundamental differences (the backup rule and the decision as to which node to expand next). And that is where most AI books stop their technical discussion.

In contrast, in real-world AI applications, it is the next step—the search enhancements—that is the topic of interest, not so much the basic algorithm. The algorithm decision is usually easily made. The choice of algorithm enhancements can have a dramatic effect on the efficiency of the search. Although it goes too far to say that the underlying algorithm is of no importance at all, it is fair to say that most research and development effort for new search methods and applications is spent with the enhancements.

Some of the enhancements are based on application-specific properties; others work over a wide range of applications. Examples of application-dependent enhancements include the Manhattan distance for the sliding-tile puzzle, and first searching moves that capture a piece before considering non-capture moves in chess. Examples of application-independent enhancements are iterative deepening [24] and cycle detection [9, 26].

The performance gap between search algorithms with and without enhancements can be large. For example, something as simple as removing repeated states from the search can lead to large reductions in the search tree (e.g. [26] using IDA* in sliding-tile puzzles; [23] using alpha-beta in chess). Combinations of enhancements can lead to reductions of several orders of magnitude.

In the traditional view, new applications are carefully analyzed until an appropriate algorithm and collection of algorithm enhancements is found that satisfies the user's expectations. In this view, each problem has its own unique algorithmic solution; a rather segmented view. In reality, most search enhancements are small variations of general ideas. Their applicability depends on the properties of the search space, and the single/two-agent property is but a minor distinction that effects very few enhancements. It is the search enhancements that tie single/two-agent search together, achieving the unity that Nilsson's and Pearl's models strived for, albeit of a different kind.

3 Modeling Search

Our thesis is that most search enhancements are independent of the single/two-agent distinction. This section identifies key properties of a search application that dictate the applicability of the search enhancements. The next section illustrates this point with some representative enhancements.

Search program design consists of two parts. First, the problem solver must specify the properties of the state space. Second, based on this information, an appropriate implementation is chosen. Defining the properties of the state space includes not only the domain-specific constraints (graph and solution definition), but also constraints imposed by the problem solver (resources, search objectives, and domain knowledge).

- Graph Definition: The problem definition allows one to construct a graph, where nodes represent states, and edges are state transition operators. This is typically just a translation of the transition rules to a more formal (graph) language. It provides the syntax of the state space.
- Solution Definition: Goal nodes are defined and given their correct value. A rule for combining the values of a node's successors to determine the value of the parent node is provided (such as minimization, or minimaxing). This adds semantics to the state space graph.
- Resource Constraints: Identify execution constraints that the search algorithm must conform to.
- Search Objectives: The problem solver defines the goal of the search: an optimal or satisficing answer. This is usually influenced by the resource constraints.

– Domain Knowledge: Non-goal nodes may be assigned a heuristic value (such as a lower bound estimator or an evaluation score). The properties of the evaluation function fundamentally influence the effectiveness of many search enhancements, typically causing many iterations of the design-and-test cycle.

Once these properties are specified, the problem solver can design the application program. This is a three step process.

1. Search Algorithm: The single/two-agent distinction is usually unambiguous, and the algorithm selection is often trivial (although, for example, there exists a large number of inventive, lesser-known alternatives, including [4, 6, 18]).
2. Search Enhancements: The literature contains a host of search enhancements to exploit specific properties of the search space. The right combination can dramatically improve the efficiency of the basic algorithm.
3. Implementation Choices: Given a search enhancement, the best implementation is likely to be dependent on the application and the choice of heuristics. These considerations are outside the scope of this paper.

Typically the choice of basic algorithm (single/two-agent) is easily made based on the problem definition. For most applications, the majority of the design effort involves judiciously fine tuning the set of algorithm enhancements [11, 12].

The applicability of search algorithm enhancements is determined by the five categories of properties of the state space. Figure 1 summarizes the interaction between the state space properties (x axis) and step 2 of the algorithm design process—the enhancements (the y axis). A sampling of enhancements are illustrated in the figure. The table shows how the search enhancements match up with the properties. An "x" means that the state space property affects the effectiveness of the search enhancement. A "v" means that the search enhancement (favorably) affects a certain property of the search space. For example, the "v"s on the row for time constraints indicate that most search enhancements make the search go faster. Star "*" entries mean that a search enhancement was specifically invented to attack a property.

The five categories of search properties have been subdivided into individual properties. The following provides a brief description of these properties.

3.1 Graph Definition

The problem specification, the rules of the application, implicitly define a graph. Following the terminology of [19] a problem space consists of states and transition functions to go from one state to another. For example, in chess a state would be a board description (piece locations, castling rights, etc.). The transition function specifies the rules by which pieces move. In the traveling salesperson problem (TSP), a state can be a tour along all cities, or perhaps an incomplete tour. The transition function adds or replaces a city.

The graph is treated as merely a formal representation of the problem, as yet devoid of meaning. It has not yet been decided what concepts like "payoff

state sp. properties \ enhancements	cycle/trans	enumeration	move ordering	iter. deepening	sol. database	selectivity
Graph Definition						
out degree of a node			> 1			
in degree of a node	> 1					
presence of cycles	x					
graph size		x				
Solution Definition						
solution density		x			x	
solution depth					x	
solution backup rule		x			x	
Resources						
space	x	x		*	x	
time	v	v	v	*/x	v	v
Search Objectives						
optimization	v	v	v	v	v	
satisficing	v		v	v	v	v
Domain Knowledge						
heur eval quality			x	x	*	x
heur eval granularity				x		
heur parent/child value			x	x		x
heur parent/child state			x			
next move to expand	*	v	*	v		*

Fig. 1. Search Properties vs Enhancements

function" and "backup rule" mean. The problem graph is purely a syntactical description of the problem space. Semantics are added later.

The graph has a number of interesting properties that can be exploited to improve the efficiency of the search. Of interest are the **in degree** and **out degree** (branching factor) of nodes, the **size** of the graph, and whether the graph contains **cycles**. These properties are self-explanatory.

3.2 Solution Definition

In this part of the problem solving process *meaning* is attached to some of the states. If the graph definition provides us with a syntactic description of the problem, then the solution definition associates semantics to the graph. The meaning, or value, of certain states in the graph is defined by the application rules. For example, in chess all checkmate states have a known value. In the TSP, a tour that visits all cities and ends in the original one is a possible solution. The objective of the search is to find these goal or solution states, and to report back how they can be reached. Solutions are a subset of the search space, and this space can be defined by the solution density, solution depth, and the backup rule for solution states.

Solution Density. The distribution of solution states determines how hard searching for them will be. When there are many solution states it will be easier to find one, although determining whether it is a least cost solution (or some other optimality constraint) may be harder.

Solution Depth. An important element of how solution states are distributed in the search space is the *depth* at which they occur (the root of the graph is at depth 0). Search enhancements may take advantage of a particular distribution. For example, breadth-first search may be advantageous when there is a high variability in the depth to solution.

Solution Backup Rule. The problem description defines how solution values should be propagated back to the root. Two-agent games use a minimax rule; optimization problems use minimization or maximization.

3.3 Resource Constraints

Resource constraints (**space** and **time**) play a critical role in determining which enhancements are feasible.

3.4 Search Objective

One of the most important decisions to be taken is the objective of the search. This decision is influenced by the size of the problem graph, solution density and depth, and resource constraints. It is closely related to the classical choice: optimize or satisfice [19]. The choice of search objective defines a global stop condition.

Optimization. Optimization involves finding the best (optimal) value for the search problem. Given a problem graph, the properties that determine whether optimization is feasible are solution density and depth.

Satisficing. Sometimes optimization is too expensive and one needs real-time or anytime algorithms. In this case, a payoff, or evaluation function, is applied to a set of states that lie closer to the root of the graph. The evaluation function is a heuristic approximation of the true value of the state. The search progresses, trying to find the best approximation to the true solution, subject to the available resources.

3.5 Domain Knowledge

The **heuristic evaluation** function encodes application-dependent domain knowledge about the search. Typically, it is the most important component of a search application. Unfortunately, it has to be redeveloped anew for each problem domain. Since the heuristic function is application dependent, most of its internals cannot be discussed in a general way. The external characteristics, however, can.

There are many different types of information that can be returned by a heuristic evaluation. Some examples include: lower/upper bound estimates on the distance to solution, point estimates on the quality of a state, ranges of values, and probability distributions.

The most important aspect of the heuristic evaluation function is the difference between the heuristic value h and the true value for a state. In general, the better the quality of h, the more efficient the search. Related to the **quality of the heuristics** are **parent/child correlation of state** (how much the state changes by a state transition), **parent/child correlation of value** (how similar the value is between a parent and child node), and the **granularity** [27] of the heuristic function (the coarseness of the values; finer granularity generally implies more search effort).

The search algorithm together with heuristic information is used to decide on the **next node to expand** in the search. For some applications, the decision may be mechanical, such as depth-first, breadth-first or best-first, but heuristic information can be instrumental in ordering nodes from most- to least-likely to succeed.

4 Search Enhancements

This section classifies various search enhancements used. The enhancements have been grouped into classes, of which a few of the more interesting ones are discussed (the ones illustrated in Figure 1). For each class, a representative technique is given and its applicability to single- and two-agent search is discussed. The material is intended to be an illustrative sample (because of space constraints), not exhaustive. Since in most cases the preconditions necessary for using an enhancement are not tied to any fundamental property of an application, the search enhancements presented are applicable to a wide class of applications.

4.1 State Space Techniques

These techniques depend only on the application definition and are therefore independent of the algorithm selected.

Path Transposition and Cycle Detection

Precondition: In-degree is > 1. Two search paths can lead to the same state. *Idea:* Repeated states encountered in the search need only be searched once. Search efficiency can (potentially) be improved dramatically by removing these redundant states. *Advantages:* Reduces the search tree size. *Disadvantages:* Increases the cost per node and/or storage required. *Techniques:* Two-agent: the typical technique is to store positions in a hash table to allow for rapid determination if a state has been previously seen [9]. Single-agent: in addition to hash tables [17], finite state machines have been used to detect cycles [26].

4.2 State- and Solution-Space Interaction

These enhancements depend on the state space graph and on the definition of the solution space.

State Space Enumeration
Precondition: Size of the state space graph and/or solution search tree be "small." *Idea:* If the state space is small enough, then the optimal answer can be computed. For some applications, traversal of the entire state space may not be necessary; one need only traverse the solution tree, ignoring parts of the state space that can logically be proven irrelevant. *Advantages:* Optimal answer for some/all nodes in the state space. *Disadvantages:* May require large amounts of time and space to traverse the state space and save the results. *Techniques:* Several games and puzzles with large state spaces have been solved by enumeration, including Nine Men's Morris [7], Qubic, Go Moku [1], and the 8-Puzzle [22] and 12-Puzzle.

4.3 Successor Ordering Techniques

The order in which the successors of an interior node are visited may effect the efficiency of the search. For example, in the alpha-beta algorithm, searching the best move first achieves the maximal number of cutoffs. In single-agent search, searching the best move first allows one to find the solution sooner. These enhancements depend on one property of the application: whether the order of considering branches influences when a cutoff occurs.

There are many techniques for doing this in the literature including **previous best move ordering** [25] and the **history heuristic** [23]. Both ideas have been tried in single- and two-agent applications (although the benefits in optimization seem to be necessarily small [17]).

4.4 Repeatedly Visiting States

One of the major search results to come out of the work on computer chess was that repeatedly visiting a state, although seemingly wasteful, may actually prove to be beneficial. The effectiveness of this enhancement depends ultimately on the heuristic evaluation function, although it works for a large class of applications.
Iterative Deepening
Precondition: Information from a shallow search satisfying condition d must provide some useful information for a deeper search satisfying $d + \Delta$. *Idea:* Search down a path until a condition d is met. After the entire tree has been searched with condition d, and no solution has been found, repeat a deeper search to satisfy condition $d + \Delta$. *Advantages:* For two-agent search, the main advantages are move ordering and time management for real-time search. For single-agent the benefit is reduced space overhead. *Disadvantages:* Repeated visitations cost time. The value of the information gathered must outweigh the cost of collecting it. *Techniques:* In many two-agent applications, the search iterates on the search depth. Move ordering is critical to the efficiency of alpha-beta search. By storing the best moves of each searched node, in each iteration the move ordering of another level of the search tree is improved [24, 25]. In single-agent search, iterative deepening is used to refine the upper (lower) bound on the value

being minimized (maximized). It is primarily used because it reduces the space requirements of the application [14].

4.5 Off-line Computations

It is becoming increasingly possible to precompute and store large amounts of interesting data about the search space that can be used dynamically at runtime.
Solution Databases
Precondition: One must be able to identify goal nodes in the search (trivial). *Idea:* The databases define a perimeter around the goal nodes. In effect, the database increases the set of goal nodes. *Advantages:* The search can stop when it reaches the database perimeter. *Disadvantages:* The databases may be costly to compute. Furthermore, the memory hierarchy makes random access to tables increasingly costly as their size grows. *Techniques:* In two-agent search, solution (or endgame) databases have been built for a number of games, in some cases resulting in dramatic improvements in the search efficiency and in the quality of search result. In single-agent applications solution databases have been tried in the 15-Puzzle. An on-line version of this idea exists, dynamically building the databases at runtime (**bi-directional** [13] or **perimeter** search [16]).

4.6 Search Effort Distribution

The simplest search approach is to allocate equal effort (search depth) to all children of the root. Often there is application-dependent knowledge that allows the search to make a more-informed distribution of effort. Promising states can be allocated more effort, while less promising states would receive less. (Essentially, this enhancement can be regarded as a generalization of successor ordering.) In satisficing single-agent search this idea is used to concentrate the search effort on promising branches. For optimizing single-agent search, it is of limited value since even if an extended search, for example, finds a solution, all possible non-extended nodes must still be checked for a better solution. It is also beneficial for real-time single-agent search such as RTA* [15] and other anytime algorithms. In two-agent search it is used in forward pruning or selective search. Popular ideas used in two-agent search include **singular extensions** [3], the **null move heuristic** [8], and **ProbCut** [5].

5 Conclusion

For decades researchers in the fields of single- and two-agent heuristic search have developed enhancements to the basic graph traversal algorithms. Historically the fields have developed these enhancements separately. Nilsson and Pearl popularized the AND/OR framework, which provided a unified formal basis, but also stressed the difference between OR and AND/OR algorithms. The fields continued their relatively separate development.

This paper advances the view that the essence of heuristic search is not searching either single- or two-agent graphs, but which search enhancements one uses. First, the single/two-agent property is but one of the many properties of the search space that play a role in the design process of a high performance heuristic search application. Second, the single/two-agent distinction is not the dominant factor in the design and implementation of a high-performance search application—search enhancements are. Third, most search enhancements are quite general; they can be used for many different applications, regardless of whether they are single- or two-agent.

The benefit of recognizing the crucial role played by search techniques is immediate: application developers will have a larger suite of search enhancements at their disposal; ideas first conceived of in two-agent search will not have to be rediscovered later independently for single-agent search, and vice versa. In an implementation the best combination of techniques depends on the expected search benefits versus the programming efforts, not on the single- or two-agent algorithm.

For twenty years, most of the research community has (explicitly and implicitly) treated single- and two-agent search as two different topics. Now it is time to take stock and recognize the pivotal role that search enhancements have come to play: the algorithm distinction is minor, and most research and implementation efforts are directed towards the enhancements. *All* the properties of the search space—not just the single/two-agent distinction—play their role in determining the effectiveness of that what heuristic search is all about: enhancing the basic search algorithms to achieve high performance.

6 Acknowledgments

This research was funded by the Natural Sciences and Engineering Research Council of Canada.

References

1. V. Allis. *Searching for Solutions in Games and Artificial Intelligence*. PhD thesis, University of Limburg, 1994.
2. S. Amarel. An approach to heuristic problem-solving and theorem proving in the propositional calculus. In J. Hart and S. Takasu, editors, *Systems and Computer Science*, 1967.
3. T. Anantharaman, M. Campbell, and F. Hsu. Singular extensions: Adding selectivity to brute-force searching. *Artificial Intelligence*, 43(1):99–109, 1990.
4. H. Berliner. The B* tree search algorithm: A best-first proof procedure. *Artificial Intelligence*, 12:23–40, 1979.
5. M. Buro. ProbCut: A powerful selective extension of the $\alpha\beta$ algorithm. *Journal of the International Computer Chess Association*, 18(2):71–81, 1995.
6. P. Chakrabarti. Algorithms for searching explicit AND/OR graphs and their applications to problem reduction search. *Artificial Intelligence*, 65(2):329–345, January 1994.

7. R. Gasser. *Efficiently harnessing computational resources for exhaustive search*. PhD thesis, ETH Zürich, 1995.
8. G. Goetsch and M. Campbell. Experiments with the null move heuristic. In *AAAI Spring Symposium*, pages 14–18, 1988.
9. R. Greenblatt, D. Eastlake, and S. Crocker. The Greenblatt chess program. In *Fall Joint Computer Conference*, volume 31, pages 801–810, 1967.
10. P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science and Cybernetics*, SSC-4(2):100–107, July 1968.
11. A. Junghanns. *New Developments in Single-Agent Search*. PhD thesis, University of Alberta, 1999.
12. A. Junghanns and J. Schaeffer. Domain-dependent single-agent search enhancements. In *IJCAI-99*, pages 570–575, 1999.
13. H. Kaindl, G. Kainz, A. Leeb, and H. Smetana. How to use limited memory in heuristic search. In *IJCAI-95*, pages 236–242, Montreal, 1995.
14. R. Korf. Iterative deepening: An optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.
15. R. Korf. Real-time heuristic search. *Artificial Intelligence*, 42:189–211, 1990.
16. G. Manzini. BIDA*: An improved perimeter search algorithm. *Artificial Intelligence*, 75:347–360, 1995.
17. T. Marsland and A. Reinefeld. Enhanced iterative-deepening search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(7):701–710, July 1994.
18. D. McAllester. Conspiracy numbers for min-max searching. *Artificial Intelligence*, 35:287–310, 1988.
19. A. Newell and H. Simon. *Human Problem Solving*. Prentice-Hall, 1972.
20. N. Nilsson. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill, 1971.
21. J. Pearl. *Heuristics*. Addison-Wesley, 1984.
22. A. Reinefeld. Complete solution of the eight-puzzle and the benefit of node ordering in IDA*. In *IJCAI-93*, pages 248–253, Chambery, France, 1993.
23. J. Schaeffer. The history heuristic and alpha-beta search enhancements in practice. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(1):1203–1212, 1989.
24. J. Scott. A chess-playing program. In *Machine Intelligence 4*, pages 255–265, 1969.
25. D. Slate and L. Atkin. Chess 4.5 — the Northwestern University chess program. In P.W. Frey, editor, *Chess Skill in Man and Machine*, pages 82–118, New York, 1977. Springer-Verlag.
26. L. Taylor and R. Korf. Pruning duplicate nodes in depth-first search. In *AAAI-93*, pages 756–761, 1993.
27. W. Zhang and R. Korf. Performance of linear-space search algorithms. *Artificial Intelligence*, 79(2):241–292, 1996.