All the Needles in a Haystack: Can Exhaustive Search Overcome Combinatorial Chaos?

Jürg Nievergelt, Ralph Gasser, Fabian M&aumlser, Christoph Wirth

Abstract

For half a century since computers came into existence, the goal of finding elegant and efficient algorithms to solve ``simple'' (well-defined and well-structured) problems has dominated algorithm design. Over the same time period, both processing and storage capacity of computers have increased roughly by a factor of 10^6. The next few decades may well give us a similar rate of growth in raw computing power, due to various factors such as continuing miniaturization, parallel and distributed computing. If a quantitative change of orders of magnitude leads to qualitative changes, where will the latter take place? Many problems exhibit no detectable regular structure to be exploited, they appear ``chaotic'', and do not yield to efficient algorithms. Exhaustive search of large state spaces appears to be the only viable approach. We survey techniques for exhaustive search, typical combinatorial problems that have been solved, and present one case study in detail.

Table of Contents

1. The Scope of Search Techniques

The concept of search plays an ambivalent role in computer science. On one hand, it is one of the fundamental concepts on which computer science is built, perhaps as fundamental as computability or complexity: any problem whatsoever can be seen as a search for ``the right answer''. On the other hand, despite its conceptually clear algorithmic nature, the problem domain ``search'' has not developed as rich a theory as, for example, algorithm complexity. When discussing search, a dozen unrelated techniques come to mind, ranging from binary search to fashionable trends such as neural nets or genetic algorithms.

The field of artificial intelligence has, at times, placed search at the center of its conceptual edifice. For example in the dichotomy of ``search'' versus ``knowledge'' as complementary or competing techniques to achieve ``intelligent'' behavior. Although the distinction between ``search'' and ``knowledge'' has an intuitive appeal (``where is it?'' as opposed to ``there it is!''), it is difficult to pin down. At the implementation level, it is often reduced to a distinction between a data structure (the

``knowledge'' of how data is organized, e.g. a binary search tree) and its access algorithms, e.g. the operation ``find''. The efficiency trade-off between knowledge and search turns merely into issues of preprocessing time and query time.

A discussion of search in general, independently of any specific problem, leads to four main concepts:

1. **State space, or search space.** The given problem is formalized by defining an appropriate set of ``states'' that represent possible configurations we may encounter along our way from the initial problem statement to a solution. In interesting search problems, this set has a rich structure, and hence is called a ``space''. The structure typically includes an adjacency relation to express the constraint that in a single step, we can only go from the current state to an adjacent state. Thus the state space is often modeled as a graph with directed or undirected edges. Various functions may be defined on the state space or on the set of edges, such as exact or approximate values to be optimized, and exact or estimated costs of going from one state to another.
2. **Goal, or search criterion.** This can vary widely with respect to both quantity and quality: from a single desired state to an exhaustive enumeration of all the states in the space; and from accepting only exact matches, or globally optimum states, to approximate matches, local optima, or heuristic values with unknown error bounds.
3. **Search algorithm**, a framework for specifying a class of possible traversals of the state space. Examples: greedy, gradient, or hill-climbing techniques, where each step is taken with the sole criterion of an immediate gain, regardless of past history or future consequences; or backtrack, a systematic enumeration of all states that meet given constraints. The direction of search can vary: from a given initial state to goal states, or, if the latter are known but paths leading to them are unknown, from the goal states back towards the initial state. Bidirectional search [Pohl 71] and other combinations are of interest.
4. **Data structures** used to organize the chosen search, such as: remember what part of the state space has been visited, find one's way to yet unexplored regions, store the collected information. A stack for depth-first search (DFS) and a queue for breadth-first search (BFS) are the most prominent examples, but combinations of these two and other structures, such as union-find, frequently occur.

The first two concepts, state space and goal, lend themselves to a rough but meaningful characterization of most search algorithms into the four categories described below, depending on whether:

- the state space has a regular structure to be exploited, such as a total order, or not, i.e. its only known structures are irregular, ``random'',
- the search goal is exact or approximate.

Regular structure, exact goal. Typical table-look-up techniques in a totally ordered space, such as binary search or interpolation search. Iterative techniques for finding zeros or minima of functions with special properties, such as Newton iteration or Fibonacci search for the minimum of a unimodal function. When the state space is a Cartesian product of totally ordered domains, such as d-dimensional Euclidean space, fast search techniques still apply in many cases.

Regular structure, approximate goal. Includes approximate pattern-matching algorithms, typically in 1 dimension (strings) or 2 (pictures).

Irregular structure, exact goal. The domain of exhaustive search techniques, such as sieves, backtrack, DFS, BFS, or reverse search for systematic enumeration of a state space, or retrograde analysis of of combinatorial puzzles and games.

Irregular structure, approximate goal. The domain of heuristic search. Includes greedy, gradient or hill-climbing techniques to find a local optimum. Along with enhancements to escape local minima to continue the search for better ones, without ever knowing whether a global optimum has been found. Includes simulated annealing, genetic algorithms, neural nets, tabu search.

This paper is about the class of search problems we call ``state space of irregular structure, exact goal". Almost by definition, these problems can only be approached by techniques called ``exhaustive search". Exhaustive search algorithms visit the entire state space in the worst case, and a significant fraction on the average. For any state not visited (``pruned"), a rigorous argument guarantees that visiting it would not change the result. This class includes dynamic programming, branch-and-bound, and alpha-beta evaluation of game trees - search algorithms that make extensive use of bounds to prune the search space.

At the risk of oversimplification we venture some comments about the relative importance of each of the four aspects mentioned above from a programmer's point of view.

1. Designing an effective state space is the first and foremost step! The main point to remember is that, in constructing a state space, the designer has the greatest opportunity to bring problem-specific knowledge to bear on the efficiency of the search techniques. This is the domain of conceptual clarity, common sense, and mathematics. Representations that exploit symmetry to avoid redundancy; heuristics that detect good candidates early without wasting time analysing poor candidates; theorems that preclude solutions from lying in certain subspaces, or assert lower or upper bounds on the value or cost of solutions, can turn a search problem from infeasible to trivial.
2. The goal is rarely under control of the programmer - it is given by the problem statement. It may be useful, however, to gather experience by first relaxing the goal, so as to reach a partial objective soon.
3. The choice of search algorithm, i.e. the class of traversals of the space that are considered, is often predetermined by characteristics of the problem. Available choices have clear consequences. Among forward searches, for example, DFS requires less temporary storage for its stack than BFS does for its queue, hence BFS may be impractical for a large problem. As another example, a backward search such as retrograde analysis is feasible only if all the goal states can be listed efficiently, i.e., an enumeration problem is solved first. Thus, it cannot be argued that one exhaustive search technique dominates others across the board. This situation is in contrast to heuristic search, where each of several fashionable paradigms (simulated annealing, neural nets, genetic algorithms, etc.) can often be used interchangeably, and each has its proponents who proclaim it a superior general-purpose search technique.
4. Data structures are the key to program optimization in the small. A good implementation often saves a constant factor but is hardly decisive.

2. Emerging Achievements of Exhaustive Search

Exhaustive search is truly a creation of the computer era. Although the history of mathematics records amazing feats of paper-and-pencil computation, as a human activity, exhaustive search is boring, error-prone, exhausting, and never gets very far anyway. As a cautionary note, if any is needed, Ludolph van Ceulen died of exhaustion in 1610 after using regular polygons of 2^62 sides to obtain 35 decimal digits of Pi - they are engraved on his tombstone.

With the advent of computers, ``experimental mathematics" became practical: the systematic search for specific instances of mathematical objects with desired properties, perhaps to disprove a

conjecture or to formulate new conjectures based on empirical observation. Number theory provides a fertile ground for the team ``computation + conjecture'', and Derrick Lehmer was a pioneer in using search algorithms such as sieves or backtrack in pursuit of theorems whose proof requires a massive amount of computation [Lehmer 53 64]. We make no attempt to survey the many results obtained thanks to computer-based mathematics, but merely recall a few as entry points into the pertinent literature:

- the continuing race for large primes, for example Mersenne primes of form 2^p-1
- the landmark proof of the ``Four-Color Theorem'' by Appell and Haken [Appell 77]
- recent work in Ramsey theory or cellular-automata [Horgan 93].

The applications of exhaustive search to which we refer in more detail is the esoteric field of games and puzzles. Because their rules and objectives are simple and rigorously defined, and progress can be quantified, games and puzzles have long been embraced by game theory [von Neumann 43], [Berlekamp 82] and artificial intelligence (AI) communities as ideal testing environments. Chess was long regarded as the ``drosophila of artificial intelligence'', and if four decades of experimentation with chess-playing machines has taught us anything at all, it must be the amazing power of massive search to amplify the effect of a meager amount of game-specific positional knowledge. It has become evident that brute-force search suffices to play chess at a level of skill exceeded by at most a few hundred humans.

We are not concerned with heuristic game-playing in this paper, we are interested in ``solving'' games and puzzles by exhaustive search of their state spaces. Games successfully solved include Qubic [Patashnik 80], Connect-4 [Allen 89], Go-Moku [Allis 94], and Merrils or Nine Men's Morris [Gasser 94], [Gasser 95]. [Allis 94] surveys the state of the art and lists all games known to have been solved. Exhaustive search has also been applied to puzzles, which can be considered to be one-player games. For the Rubik's cube there is an algorithm by Thistlethwaite which provably solves any position in at most 52 moves, while a recently introduced algorithm seemingly solves any position in at most 21 moves [Kociemba 92].

Sam Loyd's 15-puzzle (15 tiles sliding inside a 4 by 4 matrix) has so far resisted complete solution, although interesting mathematical results have been obtained. [Wilson 74] shows that the state space decomposes into two independent subspaces. When generalized to an $n \times n$ puzzle, finding optimal solution is NP-complete [Ratner 90], whereas non-optimal strategies are polynomial [Sucrow 92]. But as is often the case, such mathematical results for general n give little or no hint on how to approach a specific instance, say $n = 4$. Experience shows that improved lower bounds are the single most effective step towards efficient search, since they prune larger subspaces. Instead of the standard Manhattan distance bound, more informed bounds have been developed, e.g.: linear-conflict [Hansson 92], or fringe and corner databases [Culberson 94]. With further improved bounds, [Gasser 95] discovered positions requiring 80 moves to solve and proved that no position requires more than 87 moves to solve. This gap between upper and lower bound remains a challenge.

Returning to two-person games, there have been long-standing efforts to solve parts of games, typically endgames, where there is little hope of solving the entire game, now or perhaps ever. The motivation for this may come from attempts to improve the strength of heuristics game-playing programs. During their forward search, these use heuristic evaluation functions to assess the value of a position, and such functions are necessarily approximations of unknown quality. If exact values are available not only at the leaves, but already further up in the game tree thanks to a precomputed minimax evaluation, this has two desirable consequences. First, exact values have a beneficial effect on the quality of the ``minimax-mix'' of heuristic values. Second, longer endgames can be played perfectly, i.e. such as to guarantee an outcome of at least the game-theoretic value. We are aware of two endgame databases that were computed with this goal as primary motivation:

- *Checkers:* An *8 x 8* checkers project is headed by Jonathan Schaeffer [Schaeffer 92]. His group has computed all 7-piece positions and is working on 8 and 9 piece positions. Their databases contain about 1.5 * 10^11 positions. A workstation cluster (25-90 machines) and a BBN TC2000 allow an average of 425 million positions to be computed per day [Lake 94].
- *Awari:* Victor Allis first computed Awari endgame databases for use in his playing program Lithidion [Allis 94]. We have extended this work and now have all 5.5 * 10^8 positions with 22 or fewer stones at our disposal (T. Lincke).

Last but not least, the problem domain that spearheaded the drive to explore the limits of exhaustive search, and that gives us the clearest measure of progress, is chess endgames. [Stroehlein 70] started the race. For a summary of early results see [Herik 86]. Ken Thompson computed many four and five piece chess databases, which have been released on two compact disks [Thompson 91], [Thompson 92]. Lewis Stiller used a Connection Machine CM-2 to compute six-piece chess databases, each of which consists of up to 6'185'385'360 positions [Stiller 91]. Grandmaster John Nunn's books tellingly entitled ``Secrets of Rook Endings", ``Secrets of Pawnless Endings", and ``Secrets of Minor Piece Endings" [Nunn 92-95] is the first attempt to interpret such databases for human consumption.

Although a direct comparison of different enumeration problems is not easy, due to their state spaces of different structure and varying connectivity, it is striking that at present, the size of state spaces of various solved problems, including Merrils, is of the order of 10^10 positions. The empirical fact that the raw size of the state space is the dominant parameter that affects complexity might be explained by the observation that the structure of all these spaces shows no regularity - they appear to be random graphs. Without predictable regularity to exploit, all exhaustive searches look like random walks in a random graph. Thus, the most telling indicator of complexity is the size of the space, and its connectivity is second.

3. The Role of Exhaustive Search: Speculation

After this brief presentation of a small niche of computer science and the work of a few researchers who have been driven by curiosity more than by any other motivation, let us ask some basic questions: What can we learn from dozens of case studies of exhaustive search reported since computers became available a few decades ago? Why keep computers busy for days or months exhaustively traversing the state space of some puzzle whose solution is merely a matter of curiosity? What contribution, if any, can we expected from further experiments based on exhaustive search?

These questions are hard to answer. Brute-force solutions are not ``elegant" from a mathematical point of view. Their cryptical assertions such as ``the game of Merrils is a draw", reached after producing Gigabytes of data, cannot be checked without computer. Neither algorithms nor results can be presented in a ``user-friendly" manner. Attempts to extract rules from a data base of positions, in order to distinguish won, lost, and drawn positions, have not been very successful. If done automatically, e.g. by finding predicates and producing a decision tree, one obtains huge trees and predicates for which humans lack intuition [Gasser 95]. The answer that no simpler description exists fails to satisfy players. They prefer a simple, intuitive rule that allows exceptions to an unmotivated, impractical rule that claims perfection. Extracting rules from a database in such a way that players can understand it requires a huge effort by a top expert, as shown by Grandmaster Nunn's books [Nunn 92-95].

So why waste computer time to ``solve" games in a way that has little or no interest to game players? One answer is undoubtedly the same as for the question ``why climb Mount Everest?": to prove that it can be done. But we are intrigued by a second answer: exhaustive search in large state spaces is a

very general approach to combinatorial problems of any kind, and promises to be increasingly important. Why? For decades, computer scientists have focused attention on problems that admit efficient algorithms. Algorithms whose running time grows no faster than some polynomial of low degree in the size of the input data still dominate textbooks on algorithms. In contrast, problems that appear not to admit polynomial time algorithms, specifically, NP-complete or NP-hard problems, are often considered mere objects for a theorem, but dismissed as ``intractable'' from an algorithmic point of view.

But efficient algorithms can only be found for selected, relatively simple problems, whereas the world of applications is not simple. Most problems of practical importance, if they can be approached by computation at all, call for compute-intensive methods. Accordingly, the attitude towards ``intractable'' problems began to change. First, researchers relaxed the goal of exact or optimal solutions and sought efficient approximation algorithms. Second, one noticed that not all ``intractable problems'' are equally intractable. There appear to be many problems where the average, or typical, instance is relatively easy to solve, and only the worst-case instances are computationally very demanding. The venerable traveling salesman problem (TSP) may be an example. In tsplib [Reinelt 95], a collection of TSP benchmark problems that serves as a yardstick of progress, the currently largest provably optimal solution involves 7397 cities. This progress could hardly have been expected a decade ago, when the largest instance of a TSP known to have been solved had 318 cities [Lawler 85].

In view of the fact that exhaustive search occasionally does succeed in solving surprisingly large instances of NP-complete problems, it is time to reconsider identifying ``NP-complete'' with ``intractable''. But whereas computer scientists feel they know everything worth knowing about sorting and binary search, we know little about how to organize large state spaces for effective exhaustive search. Judging from the experience that the resources needed to complete a given exhaustive search can rarely be predicted, we are still in the process of learning the basics. And for sharpening our tools, the well-defined ``micro worlds'' of games and puzzles, with state spaces of any size we feel challenged to tackle, are well suited. Playful experiments provide moving targets that let us explore the shifting boundary of feasible brute-force problem-solving.

It is common knowledge that the raw computing power available has increased spectacularly over the past three or four decades - easily a factor of 10^6 with respect to both: processing speed and memory capacity. Whereas ``super computers'' around 1960 offered Kiloflops to Megaflops and memories of size 10 to 100 KiloBytes, today's top-of-the-line machines are in the range of Gigaflops to Teraflops and 10 to 100 GigaBytes. What is less well known, perhaps controversial but certainly arguable, is that there are reasons to expect similar rates of growth in raw computing power over the coming few decades. A number of independent factors can be listed in support of such a prediction. The physical limits of current technologies have not yet been reached - Moore's ``doubling law'' is still in force; new technologies, such as optical storage devices, are only in the beginning stages of their potential development; last but not least, parallel and distributed computing are bound to increase in popularity with the continuing price drop of hardware. As a portent of trends we mention that, at the 1995 World computer chess championship a program Star Socrates participated, running on an Intel Paragon with 1800 processors.

We hasten to emphasize the phrase ``raw computing power''! Whereas the asymptotic complexity analysis of efficient (polynomial-time) algorithms tells us with amazing accuracy what we can and cannot compute with given resources of time and memory, it is far from clear what additional problems can be solved by brute-force techniques with a millionfold increase of raw computing power. It is easy to demonstrate a problem where such a boost does not even suffice to go from a problem of size n to the same problem of size $n+1$. But whereas problems like evaluating Ackermann's function are contrived by theoreticians, we observe the same phenomenon in empirical

settings. To return to the example of computer chess: the 1800 processor monster Paragon came second to World Champion ``Fritz'' competing on a personal computer.

One expects a quantitative change of six orders of magnitude to lead to qualitative changes, but we don't know ahead of time where the latter will take place. This is particularly true for many problems such as combinatorial optimization that exhibit no detectable regular structure to be exploited. Their state spaces appear to be ``chaotic'', they allow no description briefer than an exhaustive enumeration of all states, and thus do not yield to efficient algorithms. Whatever progress has been made in this field owes a lot to tinkerers who apply great ingenuity to attack intriguing toy problems. Let us describe some of these.

4. State Spaces, their Size and Structure

Although one often talks about ``the state space of a problem'', this terminology is misleading. A problem does not come with ``its state space'' - rather, the problem solver must design a state space that models the problem adequately. A given problem may have many plausible state spaces that differ greatly in size, in structure, and ease of understanding.

In most cases, designing a state space, understanding its structure, and finding a good representation for it is by far the most important ingredient of an efficient search. Properties of the state space decide whether there exist invariants, bounds, symmetries that serve to ``prune'', i.e. avoid visiting, large parts of the state space. As a well-known illustration, consider the problem: is it possible to cover all the squares of an amputated chess board that is missing two diagonally opposite squares with dominoes, such that each domino covers two vertically or horizontally adjacent squares? A plausible state space might consists of all partial covers, with the initial state ``no dominoes'' and final states that leave no room to place any additional domino. But a simple parity argument suffices to answer ``no'', even if the problem is generalized to *n x n* chess boards, avoiding search altogether: Diagonally opposite corners are of the same color, thus the amputated chess board has an unequal number of white and black squares, and cannot possibly be covered by dominoes each of which covers one white and one black square.

As an example of how a well-designed state space and representation supports intuition and human problem solving, consider the following graphical presentation of an introductory problem discussed in many artificial intelligence textbooks, called ``cannibals and missionaries''. Three cannibals and three missionaries wish to cross a river in a two-person boat. The cannibals insist on never being left in a minority on either river bank, for fear of being converted by a majority of missionaries. A minority of zero cannibals is ok, of course - converting the empty set is no threat.

A few sentences should suffice to explain the space depicted below. A problem state consists primarily of the pair (CL, ML) of cannibals and missionaries on the left bank, where the team starts. Thus we consider a matrix 0 <= CL <= 3, 0 <= ML <= 3. The constraint ``CL = 0 or CL >= ML'' rules out three entries. But the same matrix can also be labeled with the complement (CR, MR) , and the constraint ``CR = 0 or CR >= MR'' rules out three more entries. The problem state contains an additional bit, namely, whether the boat is on the left or right shore. This bit flips at every move, so we take care of it by distinguishing odd moves and even moves. Odd moves start from the left bank and thus decrease (CL, ML) in one of 5 ways shown by arrows. Even moves start from the right bank and decrease (CR, MR) as shown by arrows in the opposite direction. Visual inspection quickly reveals that there is only one way to cross the barrier that separates the initial state from the goal state. Under the constraint of alternating between moves towards the upper left and moves back towards the lower right, the ``N-shaped'' pattern shown in bold arrows is unique. It is easy to extend these critical three steps to a solution consisting of 11 moves. This concise visual representation of the

state space supports intuition and saves a lot of trial-and-error.

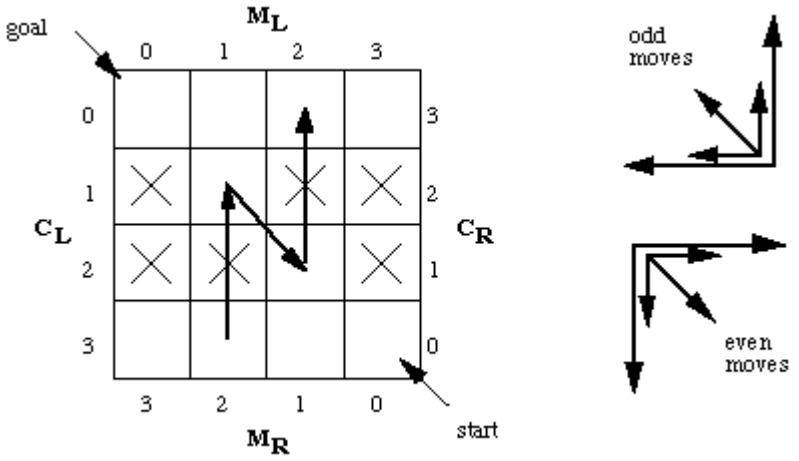

Figure 1: State space and operators of the ``Cannibals and missionaries" puzzle

The two examples of state spaces shown above are not typical. The first one is an extreme case where an invariant defined on the state space prunes the entire space, thus making search unnecessary. The second problem can be solved only by searching, but the state space is so tiny that even unsystematic search will soon find a solution. The examples do illustrate, however, the two basic steps in designing a state space. They become increasingly important the more challenging the problem: a rigorous, understandable definition of the state space, and the effectiveness of invariants in pruning the search.

As an example of a state space that taxes the state of the art, consider the game of Merrils (Nine Men's Morris, Muehle, Moulin) solved in [Gasser 95]. The board position below gives a hint of the complexity of the game. Two players White and Black alternate in first placing, then sliding, and ultimately jumping one stone of their own color. During the 18-ply (ply = a move by either player) opening, each player places his 9 stones on any unoccupied point among the 24 playable points of the board. During the midgame, each player slides one of his stones from its current location to an adjacent point. During the endgame, when at least one player has exactly three stones left, that player may jump, i.e. move any one of his stones to any unoccupied point. At all times, when a player ``closes a mill", i.e. gets three of his stones in a row along adjacent points, he may remove an opponent's stone. That player loses who first cannot move or has fewer than three stones left. For Merrils players: the position below is a mutual ``Zugzwang", i.e. the player to move loses against optimal play: White to move loses in 74 plys, Black in 60 plys.

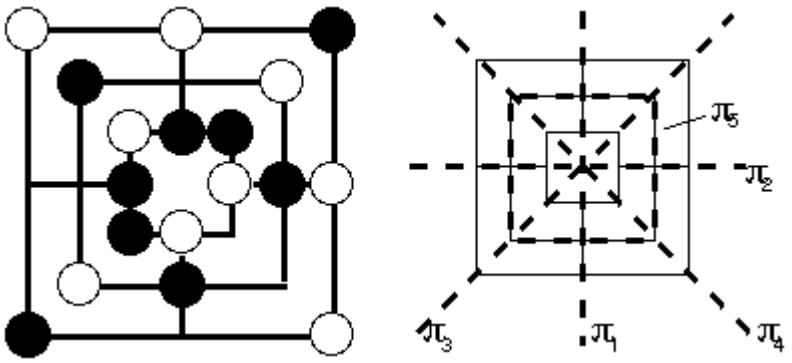

Figure 2: State space of Merrils with symmetries

The Merrils board possesses the symmetry axes shown, which generate a group consisting of 16 permutations. Using Polya's theory of counting to eliminate symmetric duplicates, we obtain a state space with a total of 7'673'759'269 states, as detailed below. The space is partitioned into 28 subspaces according to the number of stones on the board, ranging from the smallest subspace 3-3 to the largest subspace 8-7, with over 10^9 positions. Part of the structure of the space is shown in the diagram at right, which illustrates the information flow among the subspaces. For example, a position with 9 white and 8 black stones transfers, after capture of a stone, either into an 8-8 or a 9-7 position. Thus the value of a 9-8 position is deduced from values of 8-8 and 9-7 positions, as explained in section 5.

DB	States	DB	States
3-3	56'922	8-4	167'411'596
4-3	1'520'796	9-3	73'827'570
4-4	3'225'597	7-6	535'586'576
5-3	5'160'780	8-5	401'705'976
5-4	20'620'992	9-4	216'132'556
6-3	13'750'640	7-7	420'793'096
5-5	30'914'424	8-6	735'897'724
6-4	51'531'584	9-5	452'411'838
7-3	29'451'376	8-7	1'046'720'480
6-5	144'232'144	9-6	692'194'494
7-4	103'033'056	8-8	568'867'453
8-3	51'527'672	9-7	778'293'448
6-6	156'229'360	9-8	608'860'826
7-5	267'821'792	9-9	95'978'501

Total: 7'673'759'269 states

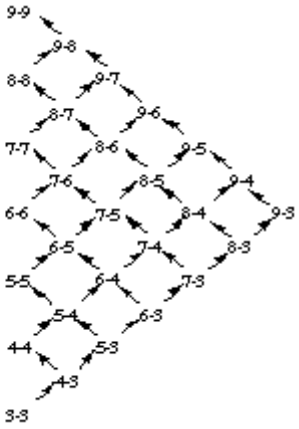

Figure 3: State space of Merrils: size and structure

5. Exhaustive Search Techniques

The literature on search uses an abundance of terminology to refer to the same few basic concepts and to techniques that differ only with respect to details. At the risk of oversimplification, let us single out a few fundamental ideas and their relation to each other, and point out modifications that capture a great variety of exhaustive search techniques.

The state space is a graph, directed or undirected, with possibly a host of other givens: initial states and goal states, functions defined on vertices (states) or edges, and constraints of the most varied kind. ``Exhaustive'' refers to the fact that we lack a priori information for excluding any subspace, hence the first requirement of an exhaustive search is a graph traversal algorithms capable of visiting all the states. Depth-first search (DFS) is the prime candidate. Its simple logic: ``keep going as long as you see anything new, and when that is not possible, back up as far as necessary and proceed in a new direction'', can be traced to the myth of Theseus traversing the Minotaur's labyrinth with the help of a thread held by Ariadne. This thread, called a stack in computer jargon, is obviously a key data structure for backing up. The myth does not tell us in detail sufficient to a programmer how Theseus kept track of the subspace already visited. It is instructive to investigate whether Theseus would have needed a chalk to guarantee visiting every cave in the labyrinth, and if not, what properties of the labyrinth and what algorithm guarantee an exhaustive traversal.

Whether or not a chalk is needed to mark states already visited is a major issue for the space complexity of the search. The chalk marks require memory proportional to the size of the graph, and that is typically much more than the other memory requirements. In realistic computations, the graph is rarely present in its entirety. Rather, its required parts are recomputed on demand, based on the usual assumption that given any state s, we can compute all its neighbors (adjacent states). But the

totality of the chalk marks would effectively require storing the entire graph. In contrast, Ariadne's thread or stack can be expected to require only a diminishing fraction of memory - ideally, storage space proportional to the logarithm of the number of states. An entry in the stack used in DFS is usually a small amount of data. Since consecutive states in the stack are neighbors in the graph and differ little, only the difference is stored, not the entire state. In game terminology, two consecutive positions differ by one move, and the former is obtained by playing the reverse move from the later position.

Backtrack is one important special case of DFS that does not require a chalk. Although the terminology of search is not standard, all the conventional examples of backtrack show a state space given explicitly as a tree. If the Minotaur's labyrinth is a tree, Theseus will see nodes visited earlier only along the path from his current position to the root. Since these are momentarily marked by the thread, they need no chalk mark. The only assumption Theseus needs is that all the edges that emanate from a node are ordered, from first to last: when he backs up along edge *i* he will next plunge deeper into the labyrinth along edge *i+1*, if it exists.

Problems may admit additional assumptions that define a subgraph T of the state space that is a spanning tree, or a spanning forest of trees. Since ``spanning'' means that T reaches every state, a backtrack search on T becomes a space-saving DFS of the entire state space. An interesting example is ``reverse search'' [Avis 92], which has been used to exhaustively enumerate all configurations in a variety of combinatorial problems, including: vertices of polyhedra, triangulations, spanning trees, topological orderings. The idea is to use greedy local optimization algorithms as can be found for most problems. A greedy algorithms g is a mapping g: S -> S from the state space into itself. The trajectories of g are free of cycles, so they form a forest. Each fix point of g, i.e. each sink of a g-trajectory, is the root of one tree. If it is possible to efficiently enumerate all these roots as an initialization step, then a backtrack search starting from each root becomes an exhaustive enumeration of the entire space. Backtracking can even dispense with a stack, because the function g always points to the predecessor of any node.

Whereas DFS, in the image of the labyrinth, is a policy for quickly penetrating as deeply as possible, its cautious partner breadth-first search (BFS) can be likened to a wave propagating through the labyrinth at equal speed in all directions. The memory cost of maintaining the wave front is significant, since all states in the front must be stored in their entirety. Combinations of DFS and BFS can be used to shape the wave front and to focus the search in a promising direction, an aspect often used in heuristic search.

The traversals above respect locality to various extents. DFS, while advancing and when backing up, always steps from one state to an adjacent state. BFS also moves to an adjacent state whenever it expands the wave front, and although states within the front need not be adjacent, they obey some locality in that they are concentric from the root. In contrast, other ``traversals'' sequence the states ``at random'', ignoring the graph structure of a space. There are two reasons for such a counter-intuitive approach.

First, there are situations where we expect little or no pruning of the space will occur at search time, and we are resigned to visit all states without exception, in arbitrary order. For many challenging problem classes, the effort required to solve one instance is comparable to solving all instances in the class. One suspects, for example, that determining the value of the initial position in chess requires knowing the value of very many chess positions - the one instance ``initial position'' is comparable in difficulty to the class ``all positions''. Second, every state has some internal representation, and these representations can be considered to be integers, or can be mapped to integers in an invertible way: given the integer, the state can be reconstructed. Such unique identifiers are called Goedel numbers in honor of the famous logician who used this technique to prove incompleteness theorems of

mathematical logic. When states are identified with integers, the easiest way to list them all is from smallest to largest, even if the numerical order of the Goedel numbers has no relation to the graph structure.

Such chaotic traversals of the space have become standard, under the name of retrograde analysis, for solving games and puzzles (recall examples in section 2). Pick any state of Merrils, for example, such as the one labeled ``Black to move loses" in the picture at left. Without any further information we can construct an arbitrary predecessor state by ``unplaying" a White move and labeling it ``White to move wins". Backing up a won position is somewhat more elaborate, as the picture at right shows. Thus, starting from known values at the leaves of a game tree, repeated application of such minmax operations eventually propagates correct values throughout the entire space.

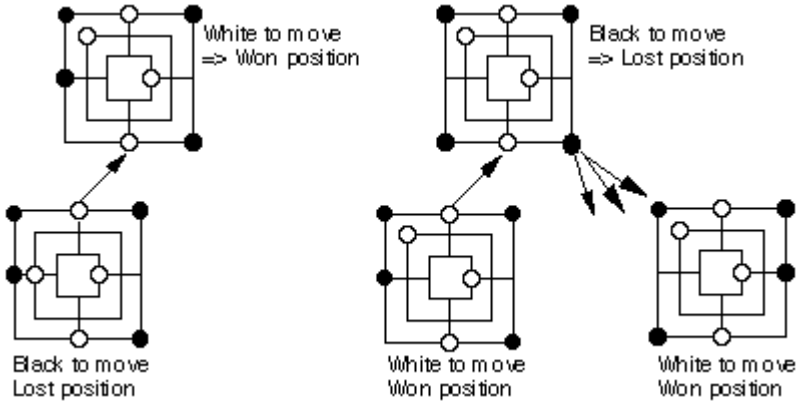

Figure 3: Backing up lost and won positions

Using retrograde analysis, the midgame and endgame state space of Merrils detailed in section 4, with 7.7 Billion positions, was calculated over a period of three years by a coalition of computers.

6. Case Study: Merrils and its Verification

Although the retrograde analysis mentioned above is the lion's share of a huge calculation, the case study of Merrils is not yet finished. The total state space of Merrils is roughly double this size, because the same board position represents two entirely different states depending on whether it occurs during the opening, while stones are being placed on the board, or during the mid- or endgame, while stones slide or jump.

It is interesting to compare the two state spaces: opening vs. midgame and endgame. The two spaces are superficially similar because each state is identified with a board position, most of which look identical in both spaces. The differences between the board positions that arise in the opening and the midgame are minor: the opening space contains states where a player has fewer than 3 stones, but lacks certain other states that can arise in the midgame. Thus, the two spaces are roughly of equal size, but their structure is entirely different. Whereas the midgame space contains many cycles, the opening space is a directed acyclic graph (dag) of depth exactly 18. If symmetries were ignored, the fanout would decreases from 24 at the root of this dag to about 7 at the leaves (not necessarily exactly 7 because of possible captures). Taking symmetries into account, the dag becomes significantly slimmer - for example, there are only four inequivalent first moves out of 24 possible.

The limited depth and fanout of the opening space suggests a forward search may be more efficient than a backward search. This is reinforced by the fact that the goal is not to determine the value of all

positions that could arise in the opening, but to prove the conjecture that Merrils is a draw under optimal play by both parties. Thus, the effective fanout of the dag is further decreased by best-first heuristics. By first investigating moves known or guessed to be good, one can often prove the hoped-for result without having to consider poor moves. Experienced Merrils players know that at most one or two mills will be closed during a well-played opening. Thus one aims to prove the draw by opening play limited to reaching three midgame databases only: 9-9, 9-8, and 8-8.

The forward search chosen replaces the dag by a tree many of whose nodes represent the same state. For reasons of program optimization whose intricate, lengthy justification we omit, two auxiliary data structures supported this search. A database of opening positions at depth 8 plies, and a hash table of positions at 14 plies. Thus, whenever the same 14th-ply-state is reached along different move sequences (transpositions), all but the first search finds the correct value already stored. This is illustrated in the following picture of a tree constricted at the 14th ply.

The draw was proven with two minmax evaluations using the well-known alpha-beta algorithm [Knuth 75]. One search proves that White, the first player to move, has at least a draw. The second search proves that Black has at least a draw. The effectiveness of alpha-beta is demonstrated by the fact that, of roughly 3.5 million positions at the 8 ply level, only 15'513 had to be evaluated to prove that White can hold the draw, and only 4'393 to prove that Black can hold the draw (the second player, Black, appears to have a small edge in Merrils). All others are pruned, i.e. can be eliminated as inferior without having been generated. This forward search took about 3 weeks of total run time on a Macintosh Quadra 800.

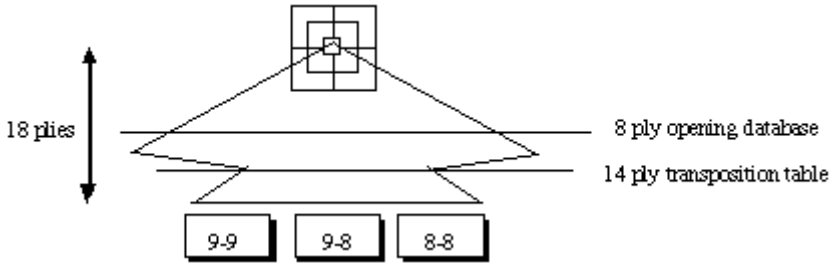

Figure 3: Analyzing the opening: a deep but narrow forward search

How credible is the cryptic result ``Merrils is a draw'', obtained after three years of computation on a variety of machines that produced 10 Giga-Bytes of data? We undertook a most thorough verification of this result. Not because the Merrils draw is vital, but because the scope of this question greatly exceeds this particular problem.

Computer-aided proofs are becoming more important in mathematics and related branches of science - for example, they were essential for the four-color theorem, Ramsey theory or cellular-automata [Horgan 93]. A common objection raised against computer proofs is that they cannot be understood, nor their correctness conclusively established by humans. Thus we took Merrils as a challenge to investigate the general issues and approaches to the problem of verifying computer proofs.

Computer-aided proofs typically lead to complex software packages, long run-times, and much data - three major sources of diverse errors. The latter fall into three categories:

1. *Software errors*: These may occur in the algorithms programmed for the specific problem, or in the system software and tools used (compilers, linkers, library programs). Although the latter errors are beyond the programmer's direct control, he must be vigilant, because software that handles everyday tasks reliably is often untested on files larger than usual.

2. *Hardware errors*: The much-debated recent bug in the Pentium microprocessor is a reminder that even arithmetic can't be relied on. Other hardware is even less reliable. It is not uncommon for a bit to be flipped on the disk, especially if the data is unused for long periods. This may be caused by cosmic radiation, electric or magnetic fields.
3. *Handling errors*: Long computations are seldom run without interruption. Thus, the user gets involved in many error-prone chores such as move data to different disks, compressing files or resuming computation on a different platform.

Hardware and handling errors are easiest to detect. Consistency checks at run-time or thereafter will uncover handling errors and non-recurring hardware glitches. Redoing the entire computation using independent system software and compiler on a different machine should find any errors in the hardware or system software. This leaves software errors stemming directly from the proof code. The best method of uncovering these, is to develop a second proof using a different algorithm. If no other algorithm is known, a third party can independently write code based on the same algorithm.

The Merrils databases were verified repeatedly with different algorithms and on different machines. Errors of all types mentioned above were detected and corrected until no further errors surfaced. A first series of verifications merely checked that the values stored with the positions are locally consistent. The game-theoretic value of any position p must be the min or max of the values of the successors of p. Although local consistency everywhere is a strong endorsement, it does not guarantee correctness, as the following example shows.

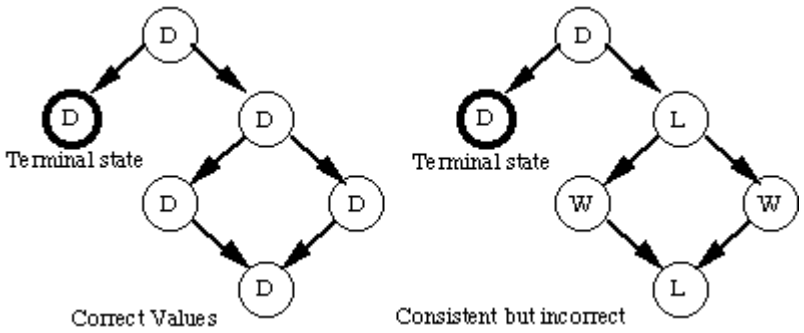

Figure 3: Locally consistent values in cycles are not necessarily correct

The diagram at left shows a subspace all of whose positions are drawn (D). If the 5 positions that contain a cycle of length 4 are alternatingly labeled loss (L) and win (W) for the player to move, as shown at right, everything is locally consistent. It is incorrect, however, because in Merrils, as in many games, endless repetition is declared a draw. This error, moreover, can propagate upwards and infect all predecessor positions.

Thus, a second series of verifications used as additional information the length of the shortest forced sequence that realizes the game-theoretic value of any position. When considering the position attributes: 0 = loss in 0 moves (terminal position), 1 = win in 1 move, 2 = loss in 2 moves, ..., 255 = draw (cycle), in addition to the game-theoretic value, the cycle with positions labeled L, D, L, D is no longer locally consistent and the error is detected.

The entire verification was run on an Intel Paragon parallel computer with 96 processors, at the rate of about 50'000 positions verified per second. The largest database (8-7, 1'072'591'519 positions) required 400 minutes.

7. Projects and Outlook

In an attempt to explore the possibilities and limits of exhaustive search, we pursue a number of other problems that span a variety of applications and techniques. The following snapshots of ongoing projects illustrate the potential breadth of exhaustive search (see also [Nievergelt 93]).

7.1. Using Heuristics to Trade Accuracy for Space and Time: Pawn endings in chess

An irritating feature of many exhaustive search computations is the fact that the program will search large subspaces where human insight could tell at a glance that nothing will be found. Such expert knowledge is frequently found in problem domains that admit natural visualizations, where humans' powerful visual pattern recognition ability let's us say ``this doesn't look right", even if we are unable to say exactly why. And if we can't rigorously formulate an invariant, and prove its validity, chances are intuition will mislead us once in a while.

Because of the irregular structure characteristic of state spaces for which we use exhaustive search, it is not easy to formulate pruning insight in such a way that a program can act on it. As a test field for experiments, we have chosen King and Pawn chess endgames. There is much expert knowledge about this domain, explained in excellent books, and there are databases of exact values to compare against when there are only a few Pawns on the board.

The peculiar difficulty of King (K) and Pawn (P) endgames comes from the fact that Pawns can be promoted to any other piece: Queen (Q), Rook (R), Bishop (B), Knight (N). Thus an exhaustive analysis of KP endgames with a total of p Ps potentially calls upon all endgames with 2 Ks and p pieces of the right color. But the vast majority of KP endgames are decided soon after (or even before) a P promotion, because the material balance typically changes drastically - one party has a Q, the other does not. Thus, storing all the support databases of other piece endgames is an extremely expensive overhead when compared to their rare use.

We are experimenting with simple and often safe heuristics of the type: In a KP endgame, if Black promotes a P to a Q, and within x plies can prevent White from promoting one of its Ps, Black wins. This particular heuristic has well-known exceptions, such as when a White P on the seventh rank ensures a draw. Thus it is supplemented by other heuristics that, jointly, capture elementary chess lore about KP endgames.

The reduction in the size of state spaces and compute time effected by heuristics that obviate the need for support databases is drastic indeed, as the following examples show. KPPKP stands for the state space of endgames where White has 2 Ps and Black has 1 P.

State spaces		Pawns only	
KK + 2 pieces:	350 M states	KPKP:	4.5 M states
KK + 3 pieces:	100 G states	KPPKP:	250 M states
KK + 4 pieces:	30 T states	KPPKPP:	3 G states

Comparisons of exact databases with simple heuristics for KPKP combined with a 2-4 ply search on P promotion have shown that the latter contain about 2.5% errors. This is merely one data point along the trade-off curve between accuracy on one hand, and space and time on the other. Improved heuristics, larger databases, more forward search at the fringe of P promotion are just some of the parameters to play with in an attempt to push exhaustive search beyond the boundaries wherein exact calculation is feasible.

7.2. Enumeration of Maximally Elastic Graphs (A. Marzetta)

We study weighted graphs that can be embedded in Euclidean space in such a way as to preserve an edge's weight as distance between its two endpoints. Such questions arise in a variety of layout problems. In automatic graph drawing, for example, vertices are to be placed in the plane so as to approximate desired pairwise distances. The analogous 3-d problem arises in the distance geometry approach to molecular modeling, where edge weights are approximate distance measurements. The concept of elastic embeddability [Nievergelt 95] is designed to deal with distances subject to error. Elastic graphs are related to generically rigid graphs known in structural engineering. In particular, maximally elastic graphs are the same as minimally rigid (isostatic) graphs [Tay 95]. Although graphs isostatic in the plane are well understood, the analogous problem in 3 or more dimensions has generated interesting conjectures that might be settled by a search for counter examples.

7.3. Primes in Intervals of Fixed Length (R. Gasser, J. Waldvogel)

Although the distribution of prime numbers has been studied empirically and theoretically for centuries, it remains an inexhaustible source of interesting conjectures to be attacked by search. Let *P(x)* denote the number of primes <= x. It follows from the prime number theorem: P(x) ~ x / ln(x), that the average density of primes decreases with increasing x. Specifically, let R(x) = max(P(y+x)) - P(y), where the maximum is over all y >= 0, denote the maximally possible number of primes in an interval (y, y+x] of length x > 0. Thus one might expect R(x) <= P(x). But although the average decreases, the peak densities do not - surprisingly, they even increase in places. Using sieve techniques, [Gasser, Waldvogel 95] prove, among other results, that for all x,10 < x < 1416, R(x) < P(x) as expected. But x = 3250 is the first known counter example where R(x) > P(x). Apparently, the densest clusters of primes in an interval of fixed length x occur starting at huge values of y.

7.4. Quo Vadis Exhaustive Search?

Brute-force techniques, as the name reveals, have never been considered elegant instruments in a computer scientist's toolbox. When used by novice programmers, the computer science community would automatically assume, often correctly, that an algorithms expert could obviously have designed a much more efficient program. But brute-force techniques have survived as a research niche for half century because a few top experts have always been intrigued by this unconventional approach to problem solving. And the complexity of problems solved has slowly but steadily grown in proportion to the power of available hardware.

The question is whether exhaustive search will remain a niche, used primarily for exotic topics such as games or number theory, or become a mainstream approach to a wide variety of applications. Instead of answering this question, let us close with one argument against and one in favor of an expanded role.

As stated in section 1, the effectiveness or ``intelligence'' of a search procedure is due mostly to problem-specific knowledge, not to differences between one or another general-purpose search procedure. But to the extent that a search uses properties specific to one problem to prune the state space, it becomes less of a brute-force technique by definition. From this point of view, exhaustive search applies to problems we do not yet understand well, and is forever being replaced by selective search as knowledge advances.

The argument for a more important role of brute-force techniques consist of two simple observations. We will never run out of problems whose structure we understand so poorly that exhaustive search is the only available approach. Second, computer science will continue to be technology-driven as it has been for decades; raw computing power will continue to increase, in particular with increased use of

distributed and parallel computation. Thus the issue will always be present whether or not newly available raw power can solve new problems.

In any case, brute-force techniques are part of the computer science culture. Basic techniques of exhaustive search belong in introductory courses on algorithms and data structures along with the more traditional polynomial-time algorithms.

References

[Allen 89]
J. D. Allen: A Note on the Computer Solution of Connect-Four, Heuristic Programming in Artificial Intelligence 1: the first computer Olympiad (eds. D.N.L. Levy and D.F. Beal), Ellis Horwood, Chichester, England. (1989) 134-135

[Allis 94]
L.V. Allis: Searching for Solutions in Games and Artificial Intelligence, Doctoral dissertation, University of Limburg, Maastricht, (1994).

[Appell 77]
K. Appell and W. Haken: The Solution of the Four-Color-Map Problem, Sci. American. (Oct. 1977) 108-121,

[Avis 92]
D. Avis and K. Fukuda: Reverse Search for Enumeration, Report, U. Tsukuba. To appear, Discrete Applied Math.

[Berlekamp 82]
E. Berlekamp, J.H. Conway and R.K. Guy: Winning Ways for your Mathematical Plays, Academic Press, London, England.

[Culberson 94]
J. Culberson and J. Schaeffer: Efficiently Searching the 15-Puzzle, internal report, University of Alberta, Edmonton, Canada.

[Gasser 90]
R. Gasser: Applying Retrograde Analysis to Nine Men's Morris, Heuristic Programming in Artificial Intelligence 2: the second computer Olympiad (eds. D.N.L. Levy and D.F. Beal), Ellis Horwood, Chichester, England. (1990) 161-173

[Gasser 91]
R. Gasser: Endgame Database Compression for Humans and Machines, Heuristic Programming in Artificial Intelligence 3: the third computer Olympiad (eds. H.J. van den Herik and L.V. Allis), Ellis Horwood, Chichester, England. (1990) 180-191

[Gasser 94]
R. Gasser, J. Nievergelt: Es ist entschieden: Das Muehlespiel ist unentschieden, Informatik Spektrum, 17, No.5, Okt 1994. 314-317

[Gasser 95]
R. Gasser: Harnessing Computational Resources for Efficient Exhaustive Search, Doctoral dissertation, ETH Zurich, 1995.

[Gasser, Waldvogel 95]
R. Gasser, J. Waldvogel: Primes in intervals of fixed length, in preparation.

[Hansson 92]
O. Hansson, A. Mayer and M. Yung: Criticizing Solutions to Relaxed Models Yields Powerful Admissible Heuristics, Information Sciences 63. (1992) 207-227

[Herik 86]
H.J. van den Herik and I.S. Herschberg: A Data Base on Data Bases, ICCA Journal 9(1), 29.

[Horgan 93]
J. Horgan: The Death of Proof, Scientific American. (1993) 74-82

[Knuth 75]
D. E. Knuth and R. W. Moore: An analysis of Alpha-Beta Pruning, Artificial Intelligence, Vol. 6. (1975) 293-326

[Kociemba 92]
H. Kociemba: Close to God's Algorithm, Cubism for Fun 28. (1992) 10-13

[Korf 85]
R.E. Korf: Depth-first Iterative Deepening: An Optimal Admissible Tree Search, Artificial Intelligence, Vol. 27. 97-109

[Lake 94]
R. Lake, J. Schaeffer and P. Lu: Solving Large Retrograde Analysis Problems Using a Network of Workstations, internal report, University of Alberta, Edmonton, Canada, 1994.

[Lawler 85]
E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys: The Traveling Salesman Problem -- A Guided Tour of Combinatorial Optimization, Wiley 1985.

[Lehmer 53]
D. H. Lehmer: The Sieve Problem for all-purpose Computers, Mathematical Tables and Aids to Computation, 7. (1953) 6-14

[Lehmer 64]
D. H. Lehmer: The Machine Tools of Combinatorics, ch.1, 5-31 in E. F. Beckenbach (ed.) Applied combinatorial mathematics, Wiley NY 1964.

[Levy 91]
D. Levy: Nine Men's Morris, Heuristic Programming in Artificial Intelligence 2: the second computer Olympiad (eds. D.N.L. Levy and D.F. Beal), Ellis Horwood, Chichester, England. (1991) 55-57

[Nievergelt 93]
J. Nievergelt: Experiments in Computational Heuristics, and their Lessons for Software and Knowledge Engineering, Advances in Computers, Vol 37 (M. Yovits, ed.), Academic Press. (1993) 167-205

[Nievergelt 95]
J. Nievergelt, N. Deo: Metric Graphs Elastically Embeddable in the Plane, to appear, Information Processing Letters 1995.

[Nunn 92-95]
J. Nunn: Secrets of Rook Endings 1992, Secrets of Pawnless Endings 1994, Secrets of Minor Piece Endings to appear 1995, Batsford Ltd. UK.

[Patashnik 80]
O. Patashnik: Qubic: $4 \times 4 \times 4$ Tic-Tac-Toe, Mathematics Magazine, Vol. 53 No.4. 202-216

[Pearl 84]
J. Pearl: Heuristics, Addison-Wesley.

[Pohl 71]
I. Pohl: Bi-Directional Search, Machine Intelligence Vol. 6 (eds. B. Metzler and D. Michie), American Elsevier, New York. 127-140

[Ratner 90]
D. Ratner and M. Warmuth: Finding a Shortest Solution for the ($N \times N$)-Extension of the 15-Puzzle is Intractable, J. Symbolic Computation, 10. 111-137.

[Reinelt 95]
G. Reinelt: ``TSPLIB", the Worldwide Web WWW, http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/TSPLIB95/TSPLIB.html

[Schaeffer 92]
J. Schaeffer, J. Culberson, N. Treloar, B. Knight, P. Lu and D. Szafron: A World Championship Caliber Checkers Program, Artificial Intelligence, Vol. 53. 273-289

[Stiller 91]

L. Stiller: Group Graphs and Computational Symmetry on Massively Parallel Architecture, The Journal of Supercomputing, 5. 99-117.

[Stroehlein 70]
T. Stroehlein: Untersuchungen ueber Kombinatorische Spiele, Doctoral thesis, Technische Hochschule Muenchen, Munich.

[Sucrow 92]
B. Sucrow: Algorithmische und Kombinatorische Untersuchungen zum Puzzle von Sam Loyd, Doctoral thesis, University of Essen.

[Tay 95]
S. Tay, J. Nievergelt: A Minmax Relationship between Embeddable and Rigid Graphs, in preparation.

[Thompson 86]
K. Thompson: Retrograde Analysis of Certain Endgames, ICCA Journal 9(3). 131-139.

[Thompson 91]
K. Thompson: Chess Endgames Vol. 1, ICCA Journal 14(1). 22

[Thompson 92]
K. Thompson: Chess Endgames Vol. 2, ICCA Journal 15(3). 149

[von Neumann 43]
J. von Neumann, O. Morgenstern: Theory of Games and Economic Behavior, Princeton Univ. Press 1953.

[Wells 71]
M. Wells: Elements of Combinatorial Computing, Pergamon Press, Oxford 1971.

[Wilson 74]
R.M. Wilson: Graph Puzzles, Homotopy, and the Alternating Group, J. Combinatorial Theory Series B 16. 86-96