In Proceedings AI Meets the Real World '98,
University of CT, Stamford, CT, Sept., 1998.

Can Search Play A Role in Practical Applications?

Matthew L. Ginsberg
Brian Drabble
David W. Etherington

CIRL
1269 University of Oregon
Eugene, OR 97403-1269
{ginsberg,ether,drabble}@cirl.uoregon.edu

Abstract

Received wisdom has it that search is ineffective in fielded AI systems; it is argued that realistic problem spaces are simply too large to allow effective search. We suggest that this is because search-based techniques fall prey to early mistakes, making fatal errors sufficiently early in the search that backtrack-based techniques are incapable of correcting them. We describe four general techniques that have been used in practical applications to circumvent this difficulty: limited discrepancy search, schedule packing, squeaky wheel optimization, and relevance-bounded learning. We explain how each technique avoids the early mistakes problem, and describe the impact of the technique on specific problems of practical interest.

1 Introduction

Fielded AI systems tend not to search. The reason is simple: the search spaces involved are astronomically large, and the developers of those systems have learned that adding search capabilities to an application generally does not enhance its performance.

Harvey and Ginsberg [16] show that the ineffectiveness of search-based techniques can be attributed to the fact that heuristics are imperfect, and existing search algorithms are often incapable of backtracking to the point at which a heuristic mistake was made. Harvey and Ginsberg refer to points at which the heuristic errs as *wrong turns.* These wrong turns are particularly devastating near the beginning of the search, since the system may never have time to recover from them. Unfortunately, such "early mistakes" are often likely near the root of the search tree because of the typical lack of information on which to base decisions early in the search. For the large search spaces associated with realistic problems, conventional systematic search algorithms are often incapable of backtracking to the point at which the crucial error occurred.

Instead of abandoning search in the face of this observation, it seems more appropriate to use a search technique that is designed to address the early mistakes problem. In this paper, we discuss four techniques that are designed specifically with this goal in mind. Each of the techniques has already been described elsewhere; our aim here is to present a brief summary of the methods, to cast them specifically in terms of circumventing the early mistakes problem, and then to present results showing the impact of the techniques on problems of practical interest.

The four techniques we will discuss are limited discrepancy search, schedule packing, squeaky wheel optimization, and relevance-bounded learning. We compare the performance of these different techniques on two real world problems: aircraft manufacturing and optical fiber cable assembly. The paper concludes with a summary of the techniques and their applicability.

2 Limited discrepancy search

2.1 Method

Consider the simple search space shown in Figure 1. The tree has been ordered so that the heuristically preferred choice is shown to the left at each point. Depth-first heuristic search will thus examine this space from left to right, visiting the nodes in the order shown in the figure itself. Suppose that node 9 is the unique goal node, so that the heuristic choice of expanding the left node first is wrong at the root, but otherwise correct. The goal is not found until the entire left-hand half of the space has been expanded.

In *limited discrepancy search*, or LDS, the space is examined not from left to right, but in an order that searches the nodes with the fewest "deviations" from the heuristic first [15, 16]. Given that the heuristic choice is always the left-hand one, node 1 in the figure has no heuristic deviations and will be examined first. Nodes 2, 3, 5 and 9 each have one

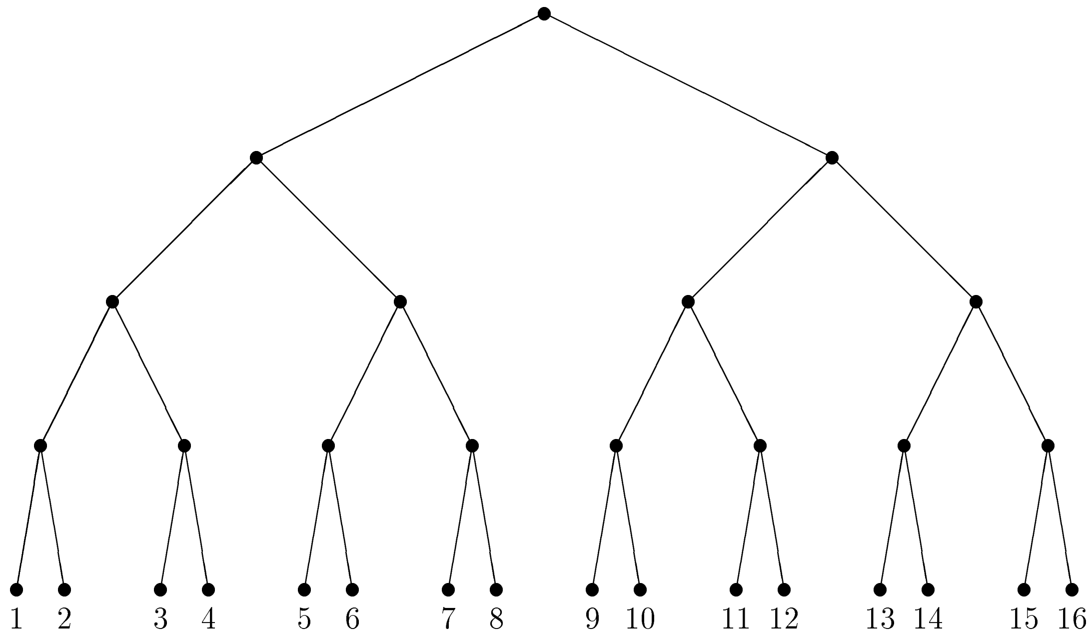


Figure 1: A simple search space

heuristic deviation (one choice of right-hand branch) and will be examined next. Nodes 4, 6, 7, 10, 11 and 13 have two such deviations and are examined next, and so on.

LDS is an iterative algorithm; each iteration repeats the expansions of the previous one.[1] Since the number of nodes expanded in iteration k is generally d times the number expanded in iteration $k-1$ (where d is the depth of the tree), the bulk of the computation is spent on the final iteration and earlier iterations do not significantly affect the run time.

2.2 Early mistakes

The way LDS avoids the early mistakes problem should be clear; in fact, it was during the development of LDS that early mistakes themselves were first recognized and the algorithm is a direct effort to avoid them. By forcing early reconsideration of heuristic choices made near the root of the search tree (or elsewhere), early mistakes can be avoided provided that the heuristic tends, overall, to suggest valid choices.

More recent developments have attempted to focus LDS even more clearly on early mistakes. In *weighted* LDS, or WLDS, the nodes are sorted not simply by counting the number of deviations from the heuristic, but in terms of the heuristic's estimate of the prior probability that the nodes are goal nodes [3]. This allows the algorithm to exploit information indicating how likely it is that the heuristic is correct at any given point. It is possible to show [3] that WLDS properly dominates earlier methods such as iterative sampling [20].

[1]If the depth of the tree is known, some of this repetition can be avoided [19].

2.3 Results

LDS has been applied to the problem of scheduling a variety of activities related to aircraft manufacture. This problem involves sequencing 575 tasks sharing 17 resources and can be found at http://www.NeoSoft.com/~benchmrx.

A 56-day schedule was produced by Mark Ringer (Honeywell) using a simple first fit, interval-based algorithm with no optimization. Agarwal (SAS) has produced a similar result. Both of these results are reported on the previously cited web site, although that site has not been updated recently and Ringer/Agarwal may have improved their times. Colin Bell (University of Iowa) was able to improve on these approaches on a smaller problem but has not posted results for the problem in its entirety. LDS produces schedules somewhat shorter than these, and was found to produce far better schedules still when used in conjunction with schedule packing, which is described in the next section.

3 Schedule packing

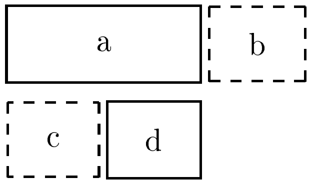
3.1 Method

Perhaps the simplest way of avoiding the early mistakes problem is to introduce stochastic elements into the search process itself, thereby ensuring that all decisions are reconsidered with reasonable frequency. In this section, we consider one stochastic search technique known as *schedule packing;* in the next section we consider a generalization known as *squeaky wheel optimization.*

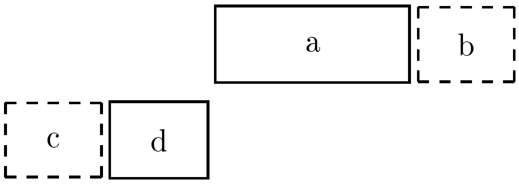
Schedule packing was originally discovered by Barry Fox [11] in the mid-1990's but was not published at that time. Fox subsequently introduced Crawford to a particular set of benchmark problems on which the method proved useful, and collaborated loosely with him on their solution. Schedule packing was rediscovered by Crawford during this time; he extended the method slightly and published it under the name "doubleback optimization" [6].

The basic idea is a simple one. Imagine that you have constructed a (not necessarily good) solution to some scheduling problem. It is then straightforward to construct a new, "packed" schedule by selecting an arbitrary ending time that is well past the actual duration of the schedule and then rescheduling each task, starting with the one that finished last in the original and working backwards. Each task is rescheduled to occur as *late* as possible, honoring only precedences and resource conflicts with those tasks that have already been rescheduled. If there are multiple tasks with the same finish time, one of them is selected at random. (This is one source of the stochasticity in the algorithm.) This repacking procedure effectively gives priority to tasks that have occurred late in the original schedule when building the next incarnation.

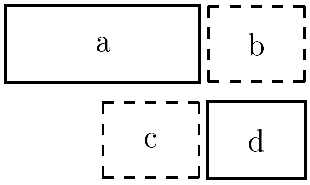
Having packed the schedule to the right, one can now "repack" it by shifting it to the left as far as possible, this time starting with the task with the earliest start time. Again, tasks that "stick out" are given priority in the hopes that they will find a better position in the resulting schedule. The resulting schedule is frequently shorter than the original. The process can be iterated (remembering the best schedule seen so far) until a sufficiently good schedule is obtained or time expires. Fox draws an analogy to forcing a box of cereal to settle by turning it over and then back again.



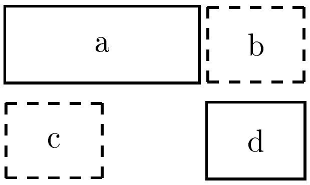
As an example, consider the scheduling problem shown above. The notation is intended to capture the fact that task a is required to precede task b and task c is required to precede task d. The outlines of the boxes indicate that tasks a and d share a resource (and therefore cannot be scheduled simultaneously), as do tasks b and c. If we break the resource contention between a and d by scheduling d first, we obtain the following:



To pack this schedule to the right, we begin by scheduling each job to start as late as possible. This produces:



We then reschedule each task to start as early as possible, scheduling the tasks in the order suggested above. Thus we begin by scheduling a, then c, then either b or d, and then the remaining task. This produces the fully packed schedule shown below.



3.2 Early mistakes

Schedule packing addresses the early mistakes problem in two ways. To understand the first, consider the original scheduling problem in the previous subsection. If we apply schedule packing to the original diagram, we have a choice: we can begin by scheduling task a or task c. The fact that we make this choice randomly gives the algorithm a stochastic element that helps ensure that all decisions are possibly revisited, independent of their location in the schedule.

Schedule packing also addresses the early mistakes problem directly. In the example, we made a mistake by scheduling task d before task a, even though task a has a following task and task d does not. The packing to the right allows task d to slide past task a, thus identifying the mistake and ensuring that the repacking to the left does not repeat it. In realistic applications, this can potentially move a large part of the schedule.

3.3 Results

On the aircraft manufacturing problem described earlier, a 39-day schedule produced by LDS in combination with schedule packing is the best schedule that has been produced without manual intervention. The best schedule known is a 38-day schedule produced by these two methods in conjunction with some manual focusing of the search. Fox has produced a 41-day schedule using only schedule packing. The combination of LDS as a seed schedule generator and the use of a schedule packing as a post-processing optimizer has proved to be a good match for problems of the complexity of aircraft assembly.

4 Squeaky wheel optimization

4.1 Method

Squeaky wheel optimization (SWO) is a general approach to optimization that was developed by Joslin and Clements [4, 17] through their attempts to better understand and to generalize schedule packing.

In schedule packing, a candidate schedule is first "analyzed" by shifting all of the tasks to the right; the effect of this analysis is to indirectly determine how much each task is contributing to the overall duration of the schedule. Based on this determination, the schedule is then reconstructed, giving priority to the tasks that appear to have the greatest contribution to the overall length.

SWO is a generalization of this technique. It begins by using simple techniques to determine a baseline measure of schedule quality; as an example, a straightforward analysis of precedence conditions can be used to establish a minimum duration for any valid schedule. SWO then takes a specific candidate schedule and examines it to ascertain the extent to which each task contributes to the difference in quality between the baseline schedule and the current one. The schedule is then reconstructed, increasing the priority given to tasks that contribute heavily to the quality difference. In more colorful terms, "The squeaky wheel gets the grease:" in the next iteration, the tasks with increased priority are dealt with earlier and thus get a better pick of the available resources. Swo differs from schedule packing in that the priorities of the various tasks change only slowly from one schedule generation to the next. This appears to allow the method to move a bit more steadily through the search space, gradually improving on the schedule being developed. As with many stochastic methods, SWO is periodically restarted with a completely fresh schedule so that it can escape from local extrema in the search space.

4.2 Early mistakes

Swo addresses the early mistake problem in the same ways that schedule packing does, by directly and stochastically revisiting problematic decisions. In the direct case, SWO tries to identify tasks leading to difficulties, on the grounds that such tasks may correspond to inopportune early scheduling decisions that need to be revisited. Also, multiple tasks may appear to contribute equally to the quality of an overall schedule, and SWO, like schedule packing, breaks ties randomly. The approach also inherits a stochastic element from the fact that it is periodically restarted with a new random seed schedule.

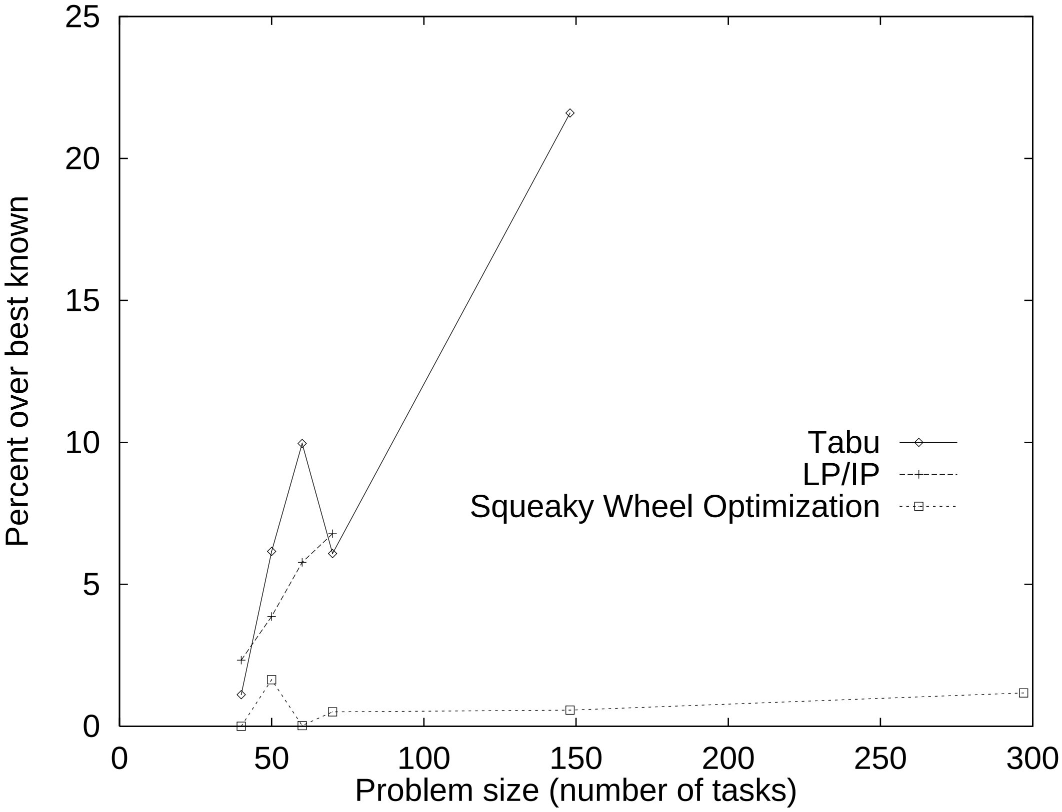


Figure 2: Performance of squeaky wheel optimization

4.3 Results

SWO has been tested on graph coloring problems [17] and on manufacturing problems arising in the construction of fiber-optic cables; we discuss only the latter application here.

Results are shown in Figure 2 comparing the performance of SWO and two well-known techniques from the operations research (OR) community, tabu search [14], and a linear/integer programming approach combining MINTO [21] and CPLEX [5]. The x-axis gives the number of cables being produced, and the y-axis compares the solution found using the given method to the best solution known.

Previous methods are incapable of finding legal schedules on problems involving more than 150 cables; even on smaller problems, SWO generally finds schedules of substantially higher quality. The best solutions known on the larger problems have been found by using SWO to generate candidate solutions and then refining these solutions using OR techniques.

5 Relevance-bounded learning

5.1 Method

The final approach we will discuss to the early mistake problem is known as *relevance-bounded learning* (RBL) [1, 2]. The basic idea is an extension of truth maintenance [7, 9, 10].

In a truth maintenance system, a reason (or "nogood") is recorded when one is forced to backtrack during the search process. The basic difficulty with this is that search techniques spend the bulk of their time backtracking and it is possible to record a nogood with every inference step. The result of this is that the memory used by the program grows linearly with the runtime, which is obviously impractical on realistic problems.

Ginsberg took a first step toward addressing this difficulty with the introduction of *dynamic backtracking* [12]. The suggestion there was that one only retain learned nogoods as long as they were relevant to the ongoing search, in that they continued to constrain the values of subsequent variables. For example, a nogood such as, "If task 32 takes place at time 46 or later, then task 56 cannot be scheduled after task 133," would remain relevant only as long as task 32 were indeed scheduled to take place at time 46 or later. With the addition of the relevance requirement, Ginsberg showed that truth maintenance techniques would take memory only polynomial in the size of the problem in question (not the run time).

Bayardo and Schrag generalized Ginsberg's ideas, pointing out that instead of requiring that nogoods be relevant, it was sufficient to bound the number of search decisions that would need to be changed for the rule to become relevant again. As an example, suppose that the antecedent in our previous rule was, "If task 32 takes place at time 46 or later, *and task 19 runs on assembly line 12 ...*" As in the previous paragraph, this rule will be relevant only as long as the antecedent is satisfied. But it will be one step "away" from relevance if *either* of the two conditions in the antecedent is satisfied. Bayardo and Schrag observed that the memory requirements of dynamic backtracking would remain polynomial if a relevance bound were established, as opposed to the original requirement that the learned rules actually be relevant at all times ("zero" steps away from relevance, as it were). The performance of the method improved substantially, as more information was learned and retained from early mistakes in the search.

Other authors have also attempted to address this difficulty, perhaps by restricting the length of the learned nogood as opposed to its relevance [8]. This approach tends not to work as well on practical problems because in many cases, the learned nogood applies only in a very specific context and the size of the antecedent needed to represent that context precludes its inclusion in the size-based scheme. On realistic problems, these large contexts may well be a consequence of choices that are forced by the overall constraints of the problem being solved.

5.2 Early mistakes

RBL addresses the early mistake problem directly. In many cases where a mistake is made early in the search, it is subsequently obvious what has gone wrong. RBL can record the nature of the problem, thereby allowing a direct backtrack to the mistake, skipping over subsequent decisions. RBL also ensures that the same mistake is not repeated subsequently, focusing the search into another area.

5.3 Results

Bayardo and Schrag report on the results of applying RBL to satisfiability problems arising in planning domains [18], circuit diagnosis (from the 1993 DIMACS Challenge, available at ftp://dimacs.rutgers.edu/pub/challenge/satisfiability), and planning, scheduling and circuit synthesis (from the 1996 DIMACS Challenge, http://www.cirl.uoregon.edu/crawford/beijing). On all of these problems, RBL's performance is comparable to that of the best general-purpose satisfiability tools. RBL typically outperforms other systematic methods.

The performance of RBL has not been compared to the performance of special-purpose scheduling or other tools. The basic reason for this is that it is difficult to apply general-purpose satisfiability engines such as RELSAT (Bayardo and Schrag's implementation of RBL) to large problems. Doing so requires translating the problem into a Boolean satisfiability format, typically increasing its size enormously. (As an example, a blocks-world planning problem involving a dozen or so blocks can involve hundreds of thousands of axioms after being converted to an instance of Boolean satisfiability.) If RBL and other satisfiability techniques are to be of practical importance, this difficulty will need to be overcome. Some very recent results of Ginsberg and Parkes [13] suggest that this problem can be met by working not with the ground version of a problem, but directly with the first-order axiomatization that produced it.

6 Conclusion

Search has been of little value on practical problems not because the method itself is inapplicable, but because existing implementations are incapable of correcting mistakes made early in the search. In this paper, we have suggested that it will be possible to address this difficulty using any one of four approaches. Two of these (LDS and RBL) are complete, systematic methods, while the other two (schedule packing and SWO) are stochastic.

Limited discrepancy search addresses the problem by rearranging the order in which the search space is explored. This reordering is correlated with the prior expectation that any particular node is a solution to the problem in question, and has the property that early decisions are always revisited before large portions of the space have been explored. LDS is a component of the best known solution method on realistic problems related to aircraft manufacture.

Schedule packing is a stochastic method that manipulates candidate solutions to scheduling problems in order to identify those that should have been given priority when the schedule was originally constructed. Assuming that early scheduling mistakes are reflected in the eventual schedule, these mistakes will be identified and can then be avoided when a new schedule is produced. Schedule packing, in conjunction with LDS, leads to the best known solutions on the problems described in the previous paragraph.

Squeaky wheel optimization is an extension and generalization of schedule packing that attempts to identify mistakes generally. Unlike schedule packing, it gradually increases the priority of the decisions associated with these mistakes in an attempt to slowly improve the quality of the solution being produced. On realistic problems from the domain of fiber optic cable construction, SWO is the only known method capable of producing viable solutions to large problems.

Relevance-bounded learning allows a problem-solving system to automatically learn nogoods that can then be used to correct early mistakes and to the avoid their recurrence. Because this technique can only be applied directly to problems that have been expressed as Boolean satisfiability problems, tests of its applicability have thus far been limited. As advances are made in our abilities to translate general problems into this form, RBL can be expected to have an ever increasing role in their solution.

Acknowledgments

This work was sponsored in part by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under agreement numbers F30602-95-1-0023 and F30602-97-1-0294. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views, findings, and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, AFRL, or the U.S. Government.

References

[1] R. J. Bayardo and D. P. Miranker. A complexity analysis of space-bounded learning algorithms for the constraint satisfaction problem. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 298–304, 1996.

[2] R. J. Bayardo and R. C. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 203–208, 1997.

[3] T. Bedrax-Weiss. *Weighted Limited Discrepancy Search.* PhD thesis, University of Oregon, Eugene, OR, 1998.

[4] D. Clements, J. Crawford, D. Joslin, G. Nemhauser, M. Puttlitz, and M. Savelsbergh. Heuristic optimization: A hybrid AI/OR approach. In *Workshop on Industrial Constraint-Directed Scheduling*, 1997.

[5] CPLEX Optimization, Inc. Using the CPLEX callable library and CPLEX mixed integer library, Version 4.0. Technical report, 1996.

[6] J. M. Crawford. An approach to resource constrained project scheduling. In *Artificial Intelligence and Manufacturing Research Workshop*, 1996.

[7] J. de Kleer. An assumption-based truth maintenance system. *Artificial Intelligence*, 28:127–162, 1986.

[8] R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41:273–312, 1990.

[9] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12:231–272, 1979.

[10] J. Doyle. The ins and outs of reason maintenance. In *Proceedings of the Eigth International Joint Conference on Artificial Intelligence*, pages 349–351, 1983.

[11] B. Fox. *An algorithm for scheduling imporvement by schedule shifting.* Technical Rpt 96.5.1, McDonnell Douglas Aerospace, Houston, TX, 1996.

[12] M. L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.

[13] M. L. Ginsberg and A. J. Parkes. Search, subsearch and QPROP. In *Fourth International Conference on Principles and Practice of Constraint Programming (PPCP-98)*, Pisa, Italy, 1998. Submitted.

[14] F. Glover and M. Laguna. *Tabu Search*. Kluwer, 1997.

[15] W. D. Harvey. *Nonsystematic Backtracking Search*. PhD thesis, Stanford University, Stanford, CA, 1995.

[16] W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 607–613, 1995.

[17] D. Joslin and D. Clements. Squeaky wheel optimization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 1998.

[18] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1996.

[19] R. E. Korf. Improved limited discrepancy search. Technical report, UCLA, Los Angeles, CA, 1995.

[20] P. Langley. Systematic and nonsystematic search strategies. In *Artificial Intelligence Planning Systems: Proceedings of the First International Conference*, pages 145–152. Morgan Kaufmann, 1992.

[21] G. Nemhauser, M. Savelsbergh, and G. Sigismondi. MINTO, A mixed integer optimizer. *Operations Research Letters*, 15:47–58, 1994.

[22] B. Selman, H. A. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In *Proceedings 1993 DIMACS Workshop on Maximum Clique, Graph Coloring, and Satisfiability*, 1993.