Evaluating Constraint Processing Algorithms [*]

Rina Dechter and **Daniel Frost**
Dept. of Information and Computer Science
University of California, Irvine, CA 92697-3425 U.S.A.
{dechter,frost}@ics.uci.edu

Abstract

Since for most artificial intelligence problems worst-case analysis does not necessarily reflect actual performance and since informative performance guarantees are not always available, empirical evaluation of algorithms is necessary. To do that we need to address the question of distributions, and benchmarks. Based on our study of CSP algorithms we propose the use of multiple types of benchmarks and multiple forms of presenting the results. The benchmarks should include: 1. Individual problem instances representing domains of interest, 2. Parameterized random problems, 3. Application-based parameterized random problems. Results should be presented using 1. Average and variances of the data, 2. frequency and distribution graphs, 3. scatter diagrams

The target is to identify a small number of algorithms (not one) that are dominating, namely proved superior on some class of problems. For dominating algorithms we wish to identify problem characteristics on which they are likely to be good.

Introduction

The study of algorithms for constraint satisfaction problems has often relied upon experimentation to compare the relative merits of different algorithms or heuristics. Experiments for the most part have been based on simple benchmark problems, such as the 8-Queens puzzle, and on randomly generated problem instances. In the 1990s, the experimental side of the field has blossomed, due to the increasing power of inexpensive computers and the identification of the "cross-over" phenomenon, which has enabled hard random problems to be generated easily.

Worst-case analysis in the area of Constraint Satisfaction has focussed largely on identifying and characterizing tractable classes of problems, that is, problems that have a structure which guarantee polynomial complexity. The primary classification of tractable classes is by a graph parameter called induced width (Dechter 1992; Arnborg 1985). It is known that if a constraint problem's graph has an induced width of size w^*, the problem can be solved in time and space exponentially in w^*, using a variable elimination algorithm like adaptive-consistency (Dechter & Pearl 1987). However, the space complexity of elimination algorithms render them useful only for a narrow class of problems having a very small induced width. In addition, the average complexity of elimination algorithms was observed to be very close to their worst-case performance (Dechter & Rish 1994; Dechter & Meiri 1994). On the other hand, simple backtracking search, although exponential in the worst case, can have good average performance.

As a result, and because backtracking requires only linear space, most practical algorithms for solving constraint problems are based on backtracking rather than on elimination. A partial ordering of some simple search algorithms, subject to certain variable ordering conditions, has been developed in (Kondrak & van Beek 1997). Nevertheless, the lack of effective worst-case analysis of backtrack search makes empirical evaluation mandatory.

To do that researchers have to face the question of problem distributions, and benchmarks. Based on our study of constraint satisfaction algorithms we advocate the use of multiple types of benchmarks and multiple ways of presenting the results. Benchmarks should include 1. individual instances that come from various applications. 2. Parameterized random problems 3. Application-based parameterized problems. Subsequently, the results should be presented using 1. Average and variances of measurements, 2. their frequency graphs, and 3. scatter diagrams displaying data from individual instances.

The goal is to identify a small number of algorithms that are dominating, namely observed to be superior on some class of problems. For these dominating algorithms we wish to identify problem characteristics on which they are likely to perform well.

Individual instances as benchmarks: The merit of this approach is that it is close, if not identical, to the underlying goal of all research: to solve real

[*]This work was partially supported by NSF grant IRI-9157636 and by Air Force Office of Scientific Research grant AFOSR 900136.

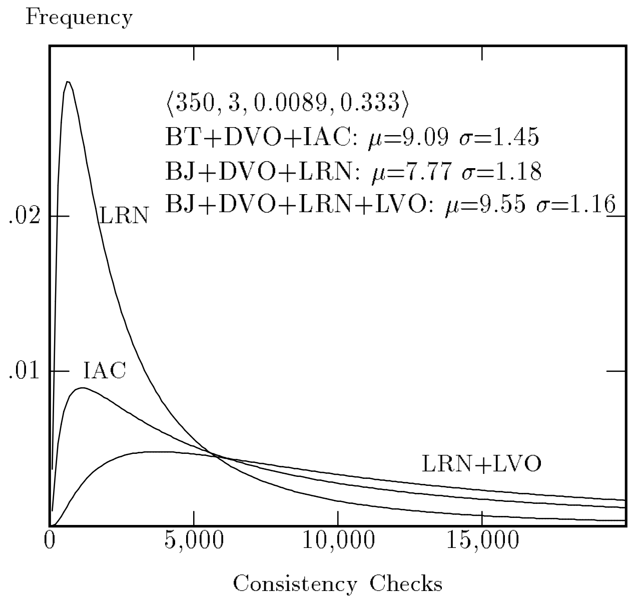

Figure 1: Lognormal curves based on unsolvable problems generated from parameters $\langle 350, 3, 0.0089, 0.333\rangle$. The graph is based on consistency checks.μ and σ parameters were estimated using the Maximum Likelihood Estimator

problems. If the benchmark problems are interesting, then the results of such a comparison are likely to be interesting. The drawback of using benchmarks of this type is that it is often impossible to extrapolate the results. Algorithm A may beat algorithm B on one benchmark and lose on another.

Random problems A contrasting technique for evaluating or comparing algorithms is to run the algorithm on artificial, synthetic, parameterized, randomly generated data. Since a practically unlimited supply of such random problems is easily generated, it is possible to run an algorithm on a large number of instances, thereby minimizing sampling error. Because the problem generator is controlled by several parameters, the experimenter can observe the possibly changing efficacy of the algorithm as one or more of the parameters change.

Application-based random problems The idea is to identify a problem domain (e.g., job shop scheduling) that can be used to define parameterized problems having a specific structure, and to generate instances by randomly generating values for the problem's parameters. This approach combines the virtues of the two approaches above: it focusses on problems that are related to real-life applications and it allows generating many instances for the purpose of statistical validity of the results.

In the following we present examples from our empirical evaluation with each of these three types of benchmarks. Most of our experiments were conducted with parameterized random binary problems. By varying the parameters of the random problem generator, we can observe how the relative strength of different algorithms is affected by the type of problems they are applied to. We also define a class of random problems that model scheduling problems in the electric power industry, and report the performance of several algorithms on those constraint satisfaction problems. To complement these random problems, we report on experiments with benchmark problems drawn from the study of circuits, and which have been used by other researchers. For details about this study see (Frost 1997).

We experimented with various new and old enhancements to a backtracking algorithms. For a recent survey on backtracking algorithms see (Dechter & Frost 1998). The algorithms are: *BT+DVO*, it augments backtracking with dynamic variable ordering; *IAC*, it interleaves arc-consistency during backtrack search; *BJ*, is conflict-directed backjumping; *LVO*, a particular value ordering heuristics we developed; *LRN*, a particular style of constraint learning during search that was proved to be the best among several schemes we tried. The various combinations of algorithms are explicitly noted. For example *BJ+DVO+LRN+LVO* is an algorithm that augments backjumping and dynamic variable ordering with value-ordering and learning.

These experiments show that the new algorithms and their hybrids can improve the performance of previous techniques by an order of magnitude on many instances.

Displaying the results

Due to the big variance in the computational effort required for each set of parameters it is clear the averages and variances are not sufficient to capture the whole picture. Clearly, a display of the distribution of the data would be preferable (see Figure 1).

Recently there have been indications (Rish & Frost 1997; Gomes, Selman, & Crato 1997) that the distribution of the search effort required to solve a set of CSPs created by a random generator can be succinctly approximated by well-known probability distributions such as the lognormal. This line of research may lead to more informative reporting of experimental results on random problems. To have confidence in the results it is useful to add some scatter diagrams for pairwise comparison of algorithms.

Experimenting with random binary problems

Most of our experiments were on randomly generated problems using 4 parameters: N, D, C, T: N being the number of variables, D is the number of values in each domain, T is the tightness of each constraint (the number of allowed pairs divided by the total number of pairs), and C: the number of constraints. For each

Parameters	Algorithm	CC	Nodes	CPU
$\langle 200, 3, 0.0592, 0.111\rangle$	BT+DVO	5,871,215	207,183	68.46
	BT+DVO+IAC	23,836,368	40,098	55.44
	BJ+DVO	5,365,467	188,726	69.28
	BJ+DVO+LVO	4,793,417	167,211	73.78
	BJ+DVO+LRN	5,731,244	186,582	63.39
	BJ+DVO+LRN+LVO	5,622,825	159,739	74.00
$\langle 300, 3, 0.0206, 0.222\rangle$	**BT+DVO+IAC**	141,606	632	0.65
	BJ+DVO	2,483,520	222,285	119.26
	BJ+DVO+LVO	1,623,455	131,593	86.76
	BJ+DVO+LRN	419,193	32,221	15.62
	BJ+DVO+LRN+LVO	392,606	25,771	13.98
$\langle 350, 3, 0.0089, 0.333\rangle$	**BT+DVO+IAC**	24,641	494	0.43
	BJ+DVO	1,238,479	182,328	140.81
	BJ+DVO+LVO	969,224	118,854	111.79
	BJ+DVO+LRN	3,727	464	0.46
	BJ+DVO+LRN+LVO	22,688	1,036	3.78

Table 1: Comparison of five algorithms on random CSPs with D=3. Each number is the mean of 2,000 solvable and unsolvable instances. The algorithm with the lowest mean CPU seconds in each group is in boldface.

Parameters	Algorithm	CC	Nodes	CPU
$\langle 60, 6, 0.4797, 0.111\rangle$	BT+DVO	24,503,115	412,494	59.72
	BT+DVO+IAC	104,319,923	65,432	130.54
	BJ+DVO	24,228,726	407,253	63.10
	BJ+DVO+LVO	23,904,430	401,131	66.47
	BJ+DVO+LRN	24,368,062	406,332	57.43
	BJ+DVO+LRN+LVO	24,103,544	405,899	65.75
$\langle 75, 6, 0.1744, 0.222\rangle$	BT+DVO	7,766,594	249,603	40.77
	BT+DVO+IAC	18,419,580	16,395	22.22
	BJ+DVO	7,530,726	241,124	42.67
	BJ+DVO+LVO	7,228,548	230,073	44.05
	BJ+DVO+LRN	7,856,321	230,367	42.13
	BJ+DVO+LRN+LVO	7,321,890	231,455	43.79
$\langle 100, 6, 0.0772, 0.333\rangle$	**BT+DVO+IAC**	4,718,685	4,625	5.67
	BJ+DVO	6,248,608	293,922	67.30
	BJ+DVO+LVO	6,581,314	305,121	76.49
	BJ+DVO+LRN	5,979,767	232,780	54.34
	BJ+DVO+LRN+LVO	6,034,538	235,509	61.22
$\langle 125, 6, 0.0395, 0.444\rangle$	**BT+DVO+IAC**	479,228	566	0.60
	BJ+DVO	3,526,619	238,584	66.17
	BJ+DVO+LVO	3,007,791	195,720	62.54
	BJ+DVO+LRN	2,050,232	108,482	28.80
	BJ+DVO+LRN+LVO	1,970,645	102,788	29.45
$\langle 150, 6, 0.0209, 0.555\rangle$	**BT+DVO+IAC**	32,537	111	0.06
	BJ+DVO	3,253,255	359,095	111.70
	BJ+DVO+LVO	1,328,189	124,415	47.08
	BJ+DVO+LRN	339,191	28,056	8.25
	BJ+DVO+LRN+LVO	601,454	25,769	12.87

Table 2: Comparison of five algorithms on random problems with D=6. Each number is the mean of 2,000 solvable and unsolvable instances. The algorithm with the lowest mean CPU seconds in each group is in boldface.

N, D and T we determined empirically the value C that correspond to the phase transition (Mitchell, Selman, & Levesque 1992) as determined by BJ+DVO.

In Table 1 and Table 2 we display the results of experimenting with particularly big and difficult problem instances, using a collection of algorithms that were observed to be superior over various other variants in our earlier experiments. The tables display average cpu seconds.

From these tables alone, the combination of BT+DVO+IAC seems to be superior, in most cases, by a big margin. The main competitor is BJ+DVO+LRN. Interestingly when we look at the distribution of the data in Figure 1. We see that although the average cpu times of LRN and IAC are very close, there were many more problems solved quickly by learning than by IAC, as reflected by consistency-checks in Figure 1. This illustrates the need of using the distributions as a more informative basis of comparison.

Experiments with DIMACS benchmark Problems

The Second Dimacs Implementation Challenge in 1993 (Johnson & Trick 1996) collected a set of satisfiability problem instances for the purpose of providing benchmarks for comparison of algorithms and heuristics. We compared our algorithms against six of the problems that were derived from circuit fault analysis. The problems are encoded as Boolean satisfiability problems in conjunctive normal form. Clauses contain from one to six variables.

Each of our six algorithms was applied to these benchmark problems. The results are displayed in Table 3 and Table 4. The tables show other CPU times on these problems reported in (Johnson & Trick 1996). Dubois *et al.* (Dubois *et al.* 1996) uses a complete algorithm based on the Davis-Putnam procedure; computer is a Sun SparcStation 10 model 40. Hampson and Kibler (Hampson & Kibler 1996) use a randomized hill climbing procedure; computer is a Sun SparcStation II. Jaumard *et al.* (Jaumard, Stan, & Desrosiers 1996) use a complete Davis-Putnam based algorithm with a tabu search heuristic; computer is a Sun SparcStation 10 model 30. Pretolani's H2R algorithm (Pretolani 1996) is based on the Davis-Putnam procedure and uses a pruning heuristic; computer is a Sun SparcStation 2. Resende and Feo (Resende & Feo 1996) present a greedy randomized adaptive search procedure called GRASP-A; the computer used was not reported. Spears (Spears 1996) uses a simulated annealing based algorithm; computer is a Sun SparcStation 10. Van Gelder and Tsuji (Van Gelder & Tsuji 1996) use a complete algorithm that combines search and resolution; computer is a Sun SparcStation 10 model 41.

Among our six algorithms, we observe that BT+DVO+IAC had the best CPU time on three problems, including one tie with BJ+DVO+LRN, and BJ+DVO+LRN was best on two, including the tie. BJ+DVO+LVO and BJ+LVO+LRN+LVO were each best one on problem. On the hardest problem, `ssa2670-141`, we cancelled BT+DVO after 24 CPU hours had passed without the algorithm completing. Overall, the two most effective algorithms that emerge are BT+DVO+IAC and BJ+DVO+LRN (with or without LVO).

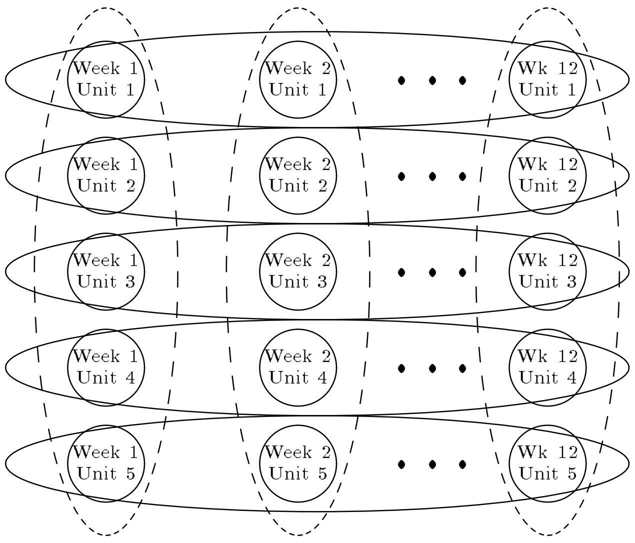

Figure 2: A diagrammatic representation of a maintenance scheduling constraint satisfaction problem. Each circle stands for a variable representing the status of one unit in one week. The dashed vertical ovals indicate constraints between all of the units in one week: meeting the minimum power demand and optimizing the cost per week. The horizontal ovals represent constraints on one unit over the entire period: scheduling an adequate period for maintenance.

Experimenting with Maintenance Scheduling Problems

The problem of scheduling off-line preventative maintenance of power generating units is of critical interest to the electric power industry. Computational approaches to maintenance scheduling have been intensively studied since the mid 1970's (Dopazo & Merrill 1975; Zurm & Quintana 1975).

In (Frost 1997) chapter 9, we formalize the maintenance scheduling problem as a constraint optimization problem and experiment with various algorithms for its solution.

Since the maintenance scheduling problem is an optimization problem we solve it iteratively as a sequence of constraint problem having a fixed cost bound for the cost function. The idea of this method is straightforward: we fix a certain cost for the cost function and try to see if there exists a satisfying solution for the

Problem	Algorithm	CC	Nodes	CPU
`ssa0432-003`	BT+DVO	51,190	901	0.73
435 variables	BT+DVO+IAC	73,817	512	0.70
1,027 clauses	BJ+DVO	48,811	865	0.81
unsatisfiable	BJ+DVO+LVO	69,100	823	0.92
	BJ+DVO+LRN	52,505	827	0.71
	BJ+DVO+LRN+LVO	59,091	816	0.78
	Dubois			1.40
	Jaumard			9.00
	Pretolani			0.83
	Van Gelder			0.55
	Wallace			499.30
`ssa2670-141`	BT+DVO			
1,359 variables	BT+DVO+IAC	535,875,109	1,279,009	1,943.51
3,321 clauses	BJ+DVO	173,446,699	7,117,071	2,791.01
unsatisfiable	BJ+DVO+LVO	41,073,083	1,858,408	803.81
	BJ+DVO+LRN	35,689,610	1,036,554	488.25
	BJ+DVO+LRN+LVO	31,854,918	843,099	449.76
	Dubois			2,674.40
	Van Gelder			164.58
`ssa7552-038`	BT+DVO	755,034	45,796	14.52
1,501 variables	BT+DVO+IAC	1,274,887	3,766	3.51
3,575 clauses	BJ+DVO	687,122	40,008	12.17
satisfiable	BJ+DVO+LVO	578,909	31,899	11.01
	BJ+DVO+LRN	439,755	22,884	5.50
	BJ+DVO+LRN+LVO	398,541	16,001	3.78
	Dubois			1.20
	Pretolani			3.67
	Hampson			152.2
	Resende			8.31
	Van Gelder			1.85

Table 3: Comparison of five algorithms on DIMACS problems. The names refer to authors who participated in the DIMACS challenge; references are given in the text. Numbers for our algorithms are all results from single instances. Some CPU times from other authors are averages over multiple randomized runs on the problem.

Problem	Algorithm	CC	Nodes	CPU
`ssa7552-158`	BT+DVO	1,009,736	51,756	19.98
1,363 variables	BT+DVO+IAC	1,863,152	17,938	8.25
3,034 clauses	BJ+DVO	845,991	25,611	10.72
satisfiable	BJ+DVO+LVO	445,172	8,122	4.55
	BJ+DVO+LRN	612,791	19,088	8.64
	BJ+DVO+LRN+LVO	467,890	12,876	7.07
	Dubois			0.80
	Hampson			82.50
	Jaumard			43.00
	Pretolani			2.28
	Resende			2.42
	Van Gelder			1.14
`ssa7552-159`	BT+DVO	883,614	39,110	14.45
1,363 variables	BT+DVO+IAC	1,378,253	4,167	3.20
3,032 clauses	BJ+DVO	674,091	20,093	6.07
satisfiable	BJ+DVO+LVO	691,654	18,987	6.98
	BJ+DVO+LRN	503,122	12,077	3.20
	BJ+DVO+LRN+LVO	563,871	12,890	3.67
	Dubois			0.90
	Hampson			82.30
	Jaumard			6.00
	Pretolani			2.68
	Resende			1.63
	Van Gelder			1.14
`ssa7552-160`	BT+DVO	712,009	35,877	12.92
1,391 variables	BT+DVO+IAC	1,265,887	4,098	3.02
3,126 clauses	BJ+DVO	687,833	22,088	6.28
satisfiable	BJ+DVO+LVO	792,615	23,766	7.14
	BJ+DVO+LRN	453,788	9,745	3.67
	BJ+DVO+LRN+LVO	495,166	10,687	4.50
	Dubois			0.90
	Hampson			86.00
	Jaumard			6.00
	Pretolani			2.80
	Resende			22.79
	Van Gelder			1.44

Table 4: Continuation of Table 3.

original set of constraints including the constraint of the cost function. If there is a solution we lower the cost bound and solve the problem again. We repeat the process untill we encounter an unsatisfiable problem. This approach reminds one of the method of solving planning problems using satisfiability (e.g., satplan), and more generally it resembles previous methods for solving optimization iteratively such as iterative deepening A*.

Iterative Learning

An interesting algorithm within this iterative scheme is learning. We used the learning algorithm, BJ+DVO+LRN, to solve the maintenance scheduling CSPs (MSCSPs), and the new constraints introduced by learning are retained for use in later iterations with tighter cost-bounds. We call this approach *iterative learning.* Retaining a memory of constraints is safe because as the cost-bound is lower the constraints become tighter. Any solution to an MSCSP with a certain cost-bound is also a valid solution to the same problem with a higher cost-bound. If the the cost-bound were both lowered and raised, as suggested by a binary search approach, then some learned constraints would have to be "forgotten" when the cost-bound was raised.

Our experiments demonstrated (not shown here for lack of space) that including the learned constraint improved significantly to effectiveness of this scheme. Also, using this set of experiment as an example to application-based random problems we observed (see Table 5) that on small problems (100 instances) BT+DVO+IAC is the best algorithm. However on larger problems it had the worst average time while "pure" BJ+DVO was best.

	Average		
Algorithm	CC	Nodes	CPU
100 smaller problems:			
BT+DVO+IAC	315,988	3,761	51.65
BJ+DVO	619,122	8,981	70.07
BJ+DVO+LVO	384,263	5,219	54.48
BJ+DVO+LRN	671,756	8,078	67.51
BJ+DVO+LRN+LVO	476,901	5,085	57.45
100 larger problems:			
BT+DVO+IAC	7,673,173	32,105	694.02
BJ+DVO	2,619,766	28,540	460.42
BJ+DVO+LVO	6,987,091	26,650	469.65
BJ+DVO+LRN	5,892,065	27,342	521.89
BJ+DVO+LRN+LVO	6,811,663	26,402	475.12

Table 5: Statistics for five algorithms applied to solvable randomly generated Maintenance Scheduling problems.

Conclusion

The empirical evaluation of constraint processing algorithms aims at finding dominating algorithms and classes of problems instances on which each algorithm is likely to perform best. Since empricial evaluation can cover only a small portion of the problem space, and since the statistics of a *realistic* problem space are not likely to be available, we must seek general guidelines that will lead to superior and more robust algorithms and heuristics. Our recent studies of constraint processing algorithms have identified several dominating algorithms and have provided initial guidelines for optimizing their utilization (Frost 1997).

References

Arnborg, S. 1985. Efficient algorithms for combinatorial problems on graphs with bounded decomposability - a survey. *BIT* 25:2–23.

Dechter, R., and Frost, D. 1998. Backtracking algorithms for constraint satisfaction problems– a tutorial survey. In *UCI technical report. Also on web page www.ics.uci.edu/~dechter.*

Dechter, R., and Meiri, I. 1994. Experimental evaluation of preprocessing algorithms for constraint satisfaction problems. *Artificial Intelligence* 68:211–241.

Dechter, R., and Pearl, J. 1987. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence* 34:1–38.

Dechter, R., and Rish, I. 1994. Directional resolution: The davis-putnam procedure, revisited. In *Principles of Knowledge Representation and Reasoning (KR-94)*, 134–145.

Dechter, R. 1992. Constraint networks. *Encyclopedia of Artificial Intelligence* 276–285.

Dopazo, J. F., and Merrill, H. M. 1975. Optimal Generator Maintenance Scheduling using Integer Programming. *IEEE Trans. on Power Apparatus and Systems* PAS-94(5):1537–1545.

Dubois, O.; Andre, P.; Boufkhad, Y.; and Carlier, J. 1996. SAT versus UNSAT. In Johnson, D. S., and Trick, M. A., eds., *Cliques, Coloring, and Satifiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science.* American Mathematical Society.

Frost, D. H. 1997. Algorithms and heuristics for constraint satisfaction problems. Technical report, Phd thesis, Information and Computer Science, University of California, Irvine, California.

Gomes, C.; Selman, B.; and Crato, N. 1997. Heavy-Tailed Distributions in Combinatorial Search. In Smolka, G., ed., *Principles and Practice of Constraint Programming – CP97*, 121–135.

Hampson, S., and Kibler, D. 1996. Large Plateaus and Plateau Search in Boolean Satisfiability Problems: When to Give Up Searching and Start Again. In Johnson, D. S., and Trick, M. A., eds., *Cliques, Coloring, and Satifiability*, volume 26 of *DIMACS Series* *in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society.

Jaumard, B.; Stan, M.; and Desrosiers, J. 1996. Tabu Search and a Quadratic Relaxation for the Satisfiability Problems. In Johnson, D. S., and Trick, M. A., eds., *Cliques, Coloring, and Satifiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society.

Johnson, D. S., and Trick, M. A., eds. 1996. *Cliques, Coloring, and Satifiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. Providence, Rhode Island: American Mathematical Society.

Kondrak, G., and van Beek, P. 1997. A Theoretical Evaluation of Selected Backtracking Algorithms. *Artificial Intelligence* 89:365–387.

Mitchell, D.; Selman, B.; and Levesque, H. 1992. Hard and Easy Distributions of SAT Problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 459–465.

Pretolani, D. 1996. Efficiency and Stability of Hypergraph SAT Algorithms. In Johnson, D. S., and Trick, M. A., eds., *Cliques, Coloring, and Satifiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society.

Resende, M. G. C., and Feo, T. A. 1996. A GRASP for Satisfiability. In Johnson, D. S., and Trick, M. A., eds., *Cliques, Coloring, and Satifiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society.

Rish, I., and Frost, D. 1997. Statistical analysis of backtracking on inconsistent csps. In Smolka, G., ed., *Principles and Practice of Constraint Programming – CP97*, 150–162r.

Spears, W. M. 1996. Simulated Annealing for Hard Satisfiability Problems. In Johnson, D. S., and Trick, M. A., eds., *Cliques, Coloring, and Satifiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society.

Van Gelder, A., and Tsuji, Y. K. 1996. Satisfiability Testing with More Reasoning and Less Guessing. In Johnson, D. S., and Trick, M. A., eds., *Cliques, Coloring, and Satifiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society.

Zurm, H. H., and Quintana, V. H. 1975. Generator Maintenance Scheduling Via Successive Approximation Dynamic Programming. *IEEE Trans. on Power Apparatus and Systems* PAS-94(2).