

A Comparison of Methods for Solving MAX-SAT Problems

Steve Joy, John E. Mitchell

Mathematical Sciences,
Rensselaer Polytechnic Institute,
Troy NY 12180 USA.

email: joys@rpi.edu, mitchj@rpi.edu

<http://www.math.rpi.edu/~mitchj>

Brian Borchers

Mathematical Sciences,
New Mexico Tech,
Socorro, NM 87801 USA

email: borchers@rpi.edu

<http://www.nmtech.edu/~borchers>

Seattle INFORMS Conference

October 25–28, 1998

Session MD29

Abstract

We compare the performance of 3 approaches to the solution of MAX-SAT problems, including a version of the Davis-Putnam-Loveland algorithm extended to solve MAX-SAT, an integer programming branch-and-cut algorithm and an algorithm for MAX-2-SAT problems based on a semidefinite programming relation.

1 Overview

- Definition of MAXSAT.
- Expressing MAXSAT as an integer program.
- Cutting planes.
- Extended Davis-Putnam-Loveland algorithm.
- Semidefinite programming relaxation of MAX2SAT.
- Computational results.
- Conclusions.

2 Definition of MAXSAT

- A propositional logic formula F can always be written in **conjunctive normal form** with m clauses and n variables.
- Each **clause** is a disjunction of a set of literals.
- Each **literal** is either a variable or the negation of a variable.
- The **formula** is the conjunction of the clauses.

- For example:

$$F = (X_1 \vee \bar{X}_2) \wedge (X_1 \vee X_2 \vee \bar{X}_3) \wedge (X_2 \vee X_3).$$

- The **satisfiability problem, SAT**:

Given a propositional logic formula F in conjunctive normal form, is there an assignment of the logical variables that makes F true?

- The **maximum satisfiability problem, MAXSAT**:

Given a propositional logic formula F in conjunctive normal form, find an assignment of the logical variables that maximizes the number of clauses in F that are true.

Computational complexity

- **SAT** is NP-Complete.
- Even if each clause has exactly **three literals**, SAT is still NP-complete.
- Some special cases are solvable in polynomial time, including **2-SAT** and **HORN-SAT**.
- **MAXSAT** is NP-Hard even if each clause has only two literals.
- There is no polynomial time approximation scheme (**PTAS**) for MAXSAT unless $P=NP$.
- It is possible to approximate MAXSAT to within a factor of 1.32 and MAX2SAT to within a factor of 1.14 in polynomial time, using a semidefinite programming approach (Goemans and Williamson, 1995).

3 An integer programming approach

- Any weighted MAXSAT instance can be expressed as an equivalent integer programming problem.
- For example:

$$C_1 = \overline{v_1} \vee \overline{v_2} \quad \text{weight 1}$$

$$C_2 = v_1 \vee v_2 \vee v_3 \quad \text{weight 4}$$

$$C_3 = v_1 \vee \overline{v_2} \quad \text{weight 3}$$

- Equivalent to:

$$\min \quad w_1 + 4w_2 + 3w_3$$

$$s.t. \quad (1 - x_1) + (1 - x_2) + w_1 \geq 1$$

$$x_1 + x_2 + x_3 + w_2 \geq 1$$

$$x_1 + (1 - x_2) + w_3 \geq 1$$

$$x_i = 0 \text{ or } 1, i = 1, \dots, 3, \quad w_i = 0 \text{ or } 1, i = 1, \dots, 3$$

- We call the added variables w_1, \dots, w_3 **weighted variables**.

Branch-and-cut

- Use a **primal heuristic** similar to GSAT (Selman *et al.*, 1992, 1996) or the heuristics of Gu (1992,1993,1994) to find a good solution.
- Solve the **linear programming relaxation** obtained by requiring $0 \leq x_i \leq 1$ for $i = 1, \dots, n$, $0 \leq w_j \leq 1$ for $j = 1, \dots, m$.
- If the solution to the relaxation is non-integral, modify it by adding **cutting planes**, or **branch**.
- Use three different types of cutting planes:
 - **Resolution cuts** (Hooker, 1988.)
 - **Odd cycle inequalities** (Cheriyān *et al.*, 1996.).
 - **Clique cuts**.
- The cuts are added **locally**, so we don't lift them to be valid throughout the branch-and-cut tree.
- **Branching**: split the relaxation into two problems, in one of which a specific variable is required to equal zero and in the other it is required to equal one.
- The algorithm is written in **C**. It uses **MINTO** to handle the branch-and-cut tree, which uses **CPLEX 4.0** to solve the linear programming relaxations.

Clique cuts

- Motivated by **pigeonhole** problems (Hooker): **hole n** asks if it is possible to place $n + 1$ pigeons in n holes without two pigeons being in the same hole.
- We describe these cuts for **2-SAT** problems:
- **Example:** Assume our instance of 2SAT contains the four literals l_1, l_2, l_3, l_4 and the six clauses:

$$l_1 \vee l_2, l_1 \vee l_3, l_1 \vee l_4, l_2 \vee l_3, l_2 \vee l_4, l_3 \vee l_4.$$

- We can think of this as giving a **clique**, with vertices labelled by the four literals, and edges given by the six clauses.
- Any satisfying truth assignment must have at least **three** of these literals true. This gives a potential cutting plane.
- This can be extended to **MAX2SAT** and general **MAXSAT** by regarding some edges in the clique as missing and modifying the cutting plane appropriately. This is really a **lifting** procedure.
- We seek inequalities that are violated by at least 0.1. The separation routine looks for literals with large values and tries to find cliques involving such variables. The cliques are built vertex-by-vertex. We reject inequalities that contain too many weighted variables.

Resolution cuts

- A classical logical procedure.
- Also implemented in branch-and-cut by Hooker and Fedjki, 1990.
- Example: Given two clauses, one containing a variable and the other containing its complement:

$$\overline{x_1} \vee x_3 \vee x_7 \vee w_9$$

$$x_2 \vee \overline{x_3} \vee x_7 \vee w_{11}$$

we can resolve to generate the following:

$$\overline{x_1} \vee x_2 \vee x_7 \vee w_9 \vee w_{11}$$

- Hooker showed that this is equivalent to a **Chvatal-Gomory** cutting plane procedure in the integer programming setting.
- Our resolution routine consists of a sequence of passes, each designed to generate resolvents satisfying certain criteria.
- Only accept resolved clauses that are **unit clauses** and that are violated by at least 0.3.

Odd cycle inequalities

- Derived by Cheriyan *et al.*, 1996, motivated by similar inequalities for MAXCUT problems.
- Example:

$$\begin{array}{rccccccc}
 x_1 & +x_2 & & & +w_{10} & +w_{11} & \geq 1 \\
 & +x_2 & -x_3 & & & +w_{11} & \geq 0 \\
 & & -x_3 & -x_4 & & +w_{12} & \geq -1 \\
 -x_1 & & & -x_4 & & & +w_{13} \geq -1
 \end{array}$$

- The first, third, and fourth constraints are odd; the second is even. Adding these constraints together, dividing by 2 and rounding coefficients we obtain:

$$x_2 - x_3 - x_4 + w_{10} + w_{11} + w_{12} + w_{13} \geq 0.$$

- Use a modified version of Dijkstra's algorithm to find cutting planes.
- Look for cuts violated by at least 0.3, and of length at most 2.
- Odd cycle cuts combine together clauses of **length 2**.
- Thus, only effective if the current relaxation contains a large number of clauses of length 2, eg, MAX2SAT problems.
- For problems with longer clauses, can only use these cuts deep in the tree.

Branching strategy

- After generating cuts and applying bounds, we branch if there are still fractional variables in the LP solution.
- The branching scheme we use is a modification of the **Jeroslow-Wang** (1990) scheme.
- Jeroslow-Wang examines the probability of satisfying all the constraints if we randomly assign values to the remaining unfixed variables. It attempts to pick a variable whose assignment will have the greatest change in this probability.
- Our approach differs from Jeroslow-Wang in that:
 1. We give considerably less weight to **singleton** clauses.
 2. We consider the effect on **both** branches.

4 Extended Davis-Putnam-Loveland

- Davis-Putnam-Loveland (1960,1978) is a classical branching and backtracking procedure for solving **SAT**.
- Borchers and Furman (1995) extended this to solve **MAXSAT** problems. (Also done independently by Wallace and Freuder (1996).)
- We maintain an **upper bound**, ub , given by the number of unsatisfied clauses in the best known solution.
- We keep track of $unsat$, the number of clauses left unsatisfied by the current partial solution.
- If $unsat > ub$, **backtrack** from the current partial solution.
- Written in **C**.
- Code uses **primal heuristic** to find good solution.
- Incorporates **unit clause tracking** when $unsat = ub - 1$.
- Uses the variable selection rule of Dubois *et al.*.

5 Semidefinite programming

This formulation is due to Goemans and Williamson, 1995.

MAX2SAT as a QP

- For each logical variable y_i , introduce a ± 1 variable x_i .
- Add a variable x_0 . We'll set $y_i = \text{TRUE}$ if $x_0 = x_i$, and FALSE otherwise.
- Let

$$v(y_k \vee y_l) := 0.25(1 + x_0x_k + 1 + x_0x_l + 1 - x_kx_l).$$

- We obtain $v(y_k \vee y_l) = 0$ if the clause is FALSE, and 1 otherwise.
- Similar formulas can be constructed for clauses involving negated literals.
- The MAX2SAT problem becomes:

$$\max\left\{\sum_{j=1}^m v(C_j) : x_i \in \{-1, 1\}, i = 0, \dots, n\right\}.$$

Formulation as an SDP

- Express QP as:

$$\max\{x^T A x + c : x_i \in \{-1, 1\}, i = 0, \dots, n\}$$

for a symmetric matrix A and a scalar c .

- Note that $x^T A x = \text{tr}(X A)$, where $X = x x^T$ and tr denotes the trace of a matrix.
- Rewrite problem as:

$$\begin{aligned} \max \quad & \text{tr}(X A) + c \\ \text{subject to} \quad & X = x x^T \\ & x_i \in \{-1, 1\}, \quad i = 0, \dots, n. \end{aligned}$$

- Notice that $X_{ii} = 1$.
- **Relax** the requirement that X be a rank 1 matrix to the requirement that it be a **positive semidefinite matrix**. Write this $X \succeq 0$.
- Obtain the semidefinite programming (SDP) relaxation of MAX2SAT:

$$\begin{aligned} \max \quad & \text{tr}(X A) + c \\ \text{subject to} \quad & X \succeq 0 \\ & X_{ii} = 1, \quad i = 0, \dots, n. \end{aligned}$$

- This relaxation can be solved in polynomial time using an interior point method.

Cutting planes

- Can **strengthen** the SDP relaxation by adding linear inequalities, as shown by Helmberg and Rendl (1998) for MAXCUT.
- Add constraints of the form:

$$\begin{aligned}X_{ij} + X_{jk} + X_{ik} &\geq -1 \\X_{ij} - X_{jk} - X_{ik} &\geq -1 \\-X_{ij} + X_{jk} - X_{ik} &\geq -1 \\-X_{ij} - X_{jk} + X_{ik} &\geq -1.\end{aligned}$$

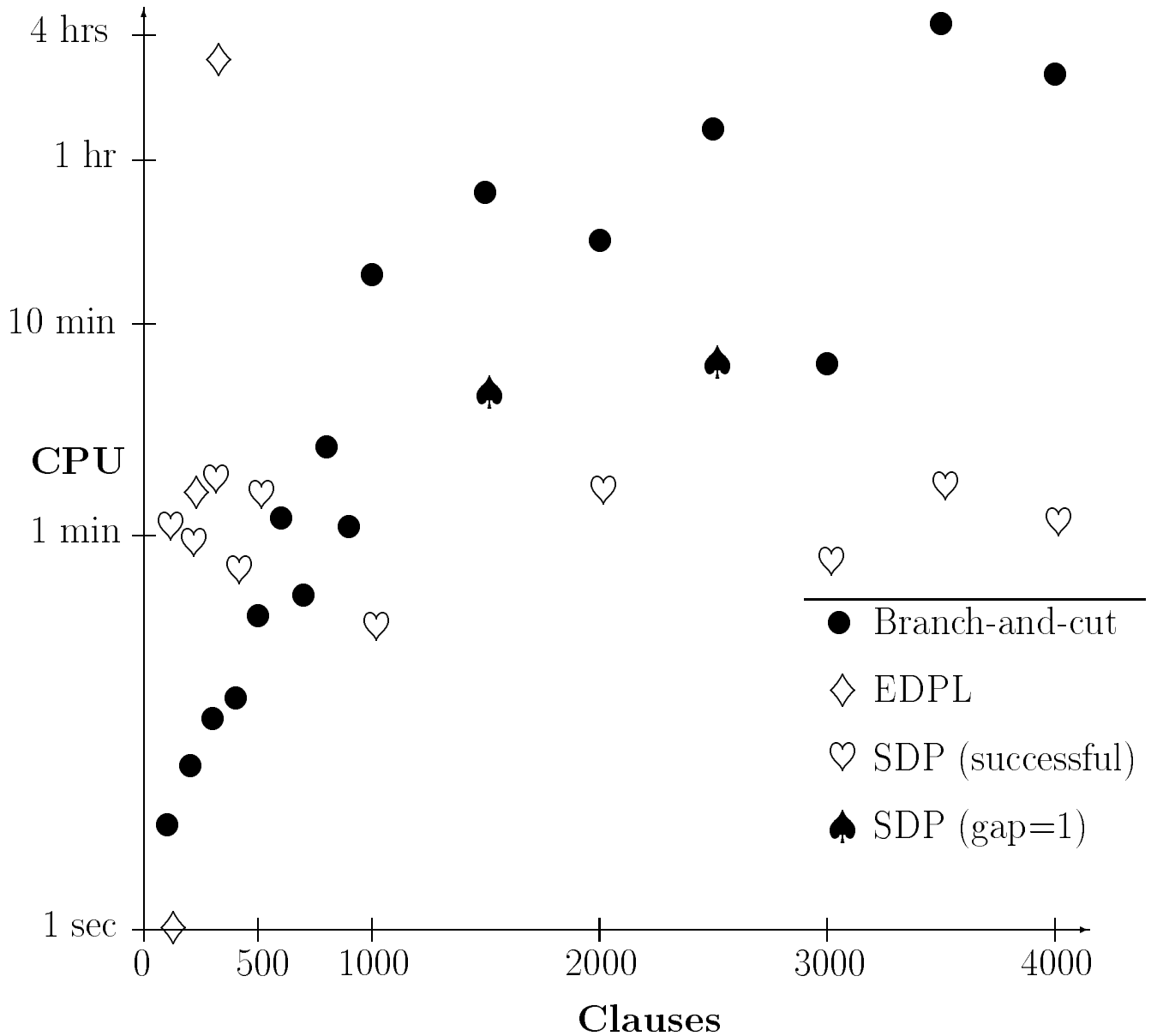
- These constraints are derived by considering possible values for $x_i x_j$, $x_j x_k$, and $x_i x_k$ for ± 1 variables.
- Add these constraints selectively, as cutting planes.

Our SDP implementation

- Written in **C**.
- Uses the primal-dual interior point semidefinite programming algorithm of Helmberg, Rendl, Vanderbei, and Wolkowicz (1996) to solve the SDP relaxations.
- Uses a primal heuristic due to Goemans and Williamson, 1995.
- Adds cutting planes of the type described above.

6 Computational results

MAX2SAT



Log(CPU time) vs number of clauses for random 50 variable MAX-2-SAT problems

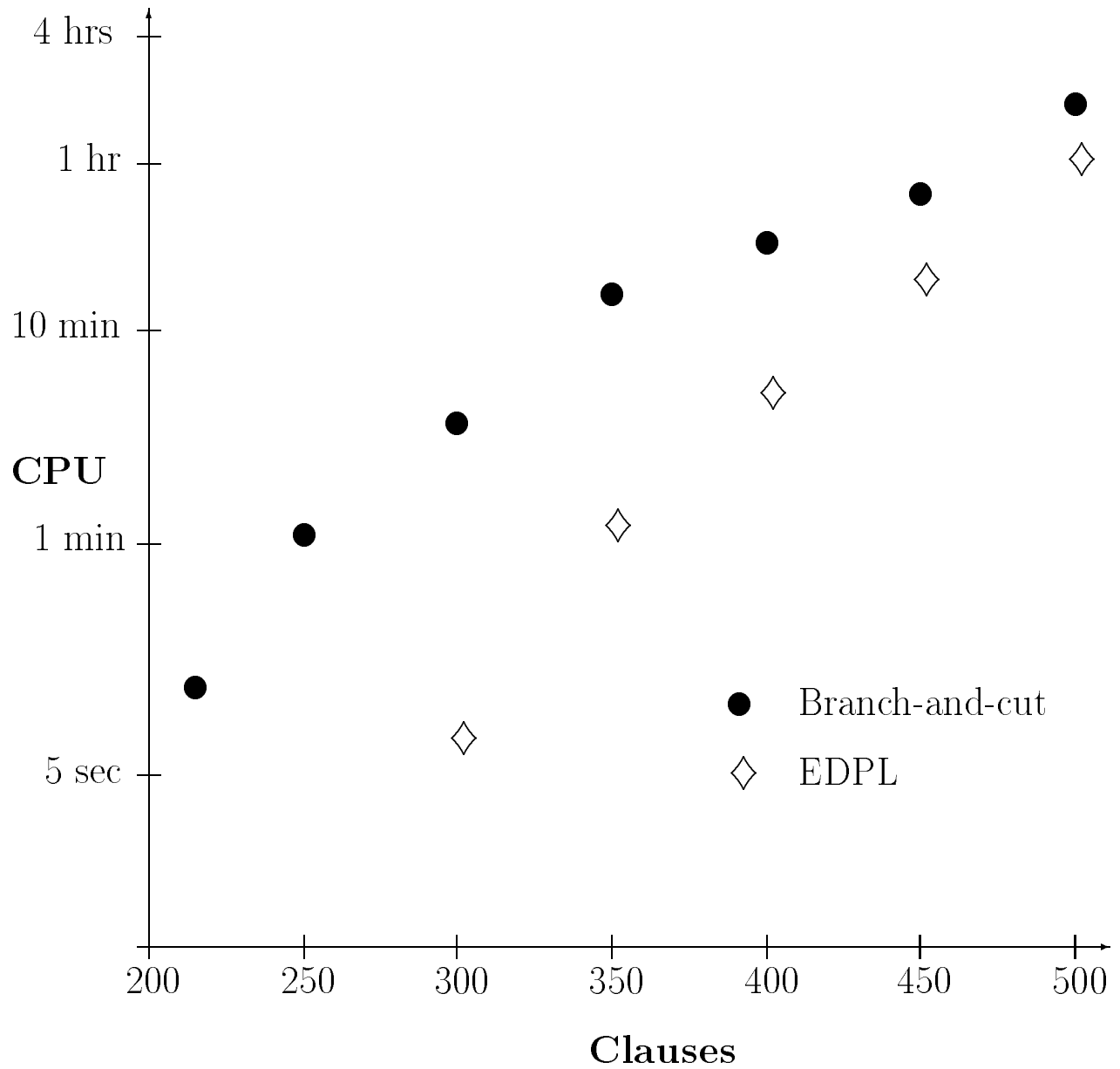
Results for 100 variable MAX-2-SAT problems

problem name	clauses	unsat. clauses	B+C		EDPL		SDP		
			CPU	nodes	CPU	backtracks	CPU	lower	upper
p2180_5	180	0	0.62	1	0.1	1	366.85	180	180
p2180_8	180	1	4.96	1	0.7	2	1015.99	179	179
p2180_6	180	1	4.89	1	0.7	47	911.65	178	178
p2180_3	180	2	5.18	1	0.7	40	1417.08	178	178
p2180_9	180	2	5.23	1	0.7	34	2497.35	178	179
p2180_2	180	2	5.29	1	0.7	10	2152.11	178	179
p2180_1	180	3	4.99	1	0.7	162	2299.32	177	178
p2180_7	180	4	5.25	1	0.8	964	1322.69	176	176
p2180_4	180	4	5.19	1	0.9	1773	2315.90	176	177
p2180_10	180	4	5.31	1	0.9	1933	1106.76	176	176
p2300_2	300	13	7.18	1	2229.1	21334438	2307.46	287	288
p2300_3	300	13	7.50	1	2497.9	23764177	2399.92	287	288
p2300_4	300	14	7.29	1	4618.5	47629271	2202.70	286	287
p2300_9	300	15	7.51	1	8746.0	85235320	1220.04	285	285
p2300_1	300	15	8.18	1	10564.3	104553113	2086.05	285	286
p2300_10	300	15	8.24	1	12693.1	125325067	2292.71	285	286
p2300_5	300	15	11.35	3	29738.0	309999407	2263.76	285	287
p2300_6	300	17	9.67	1	69325.3	704246333	2200.03	283	284
p2300_8	300	17	9.42	1	Not Run	Not Run	2123.03	283	284
p2300_7	300	20	32.33	11	Not Run	Not Run	2266.63	280	281
p2400_3	400	25	13.36	1	Not Run	Not Run	2097.99	375	376
p2400_7	400	26	30.69	11	Not Run	Not Run	2127.00	374	375
p2400_6	400	27	18.70	3	Not Run	Not Run	2134.13	373	374
p2400_4	400	28	13.58	1	Not Run	Not Run	640.15	372	372
p2400_2	400	28	18.29	3	Not Run	Not Run	2134.42	372	373
p2400_1	400	29	21.75	3	Not Run	Not Run	2157.36	371	372
p2400_5	400	29	45.13	15	Not Run	Not Run	2167.33	371	373
p2400_10	400	30	64.82	11	Not Run	Not Run	2141.70	370	372
p2400_8	400	33	71.13	11	Not Run	Not Run	2166.69	367	369
p2400_9	400	34	121.73	19	Not Run	Not Run	2074.66	365	368

Results on dense MAX-2-SAT problems

variables	clauses	B+C CPU	SDP		
			CPU	lower	upper
50	100	3.32	56.15	96	96
	200	5.70	46.61	184	184
	300	9.51	93.63	268	268
	400	12.65	34.45	355	355
	500	26.94	78.56	434	434
	1000	64.76	18.75	838	838
	1500	1933.55	250.39	1228	1229
	2000	1350.11	84.56	1614	1614
	2500	3378.69	348.00	2007	2008
	3000	523.75	38.77	2419	2419
	3500	10142.31	88.67	2783	2783
	4000	5761.59	58.76	3169	3169
	4500	16958.45	352.68	3540	3541
	5000	37241.84	298.27	3933	3933
100	1000	22915.42	5153.65	857	859
	1500	MEM	5321.79	1270	1272
	2000	MEM	1549.69	1685	1685
	2500	MEM	6422.84	2055	2060
150	1000	MEM	24445.65	1285	1297
	1500	MEM	23274.42	1309	1310
	2000	MEM	30934.79	1702	1711
	2500	MEM	21335.99	2104	2112

MAX3SAT



Log(CPU time) vs number of clauses for random 50 variable MAX3SAT problems

Results for 100 variable MAX-3-SAT problems

problem name	clauses	unsat. clauses	B+C		EDPL	
			CPU	nodes	CPU	backtracks
o430_3	430	0	0.97	1	0.2	1
o430_4	430	0	1.08	1	0.1	1
o430_2	430	0	2.19	1	0.1	1
o430_1	430	1	169.68	543	1.4	320
o430_5	430	2	1429.61	7541	5.2	8834
o500_1	500	0	3.65	1	0.2	1
o500_3	500	2	983.06	3139	2.9	3345
o500_2	500	3	2908.52	7095	31.3	87816
o500_5	500	4	7927.12	16889	203.1	735535
o500_4	500	4	9873.80	18793	162.5	572489
o575_5	575	4	6488.93	8609	151.9	491192
o575_3	575	5	13342.57	17253	602.2	2191647
o575_1	575	6	MEM	MEM	4370.9	18334787
o575_2	575	7	MEM	MEM	10488.5	47346157

Steiner tree systems

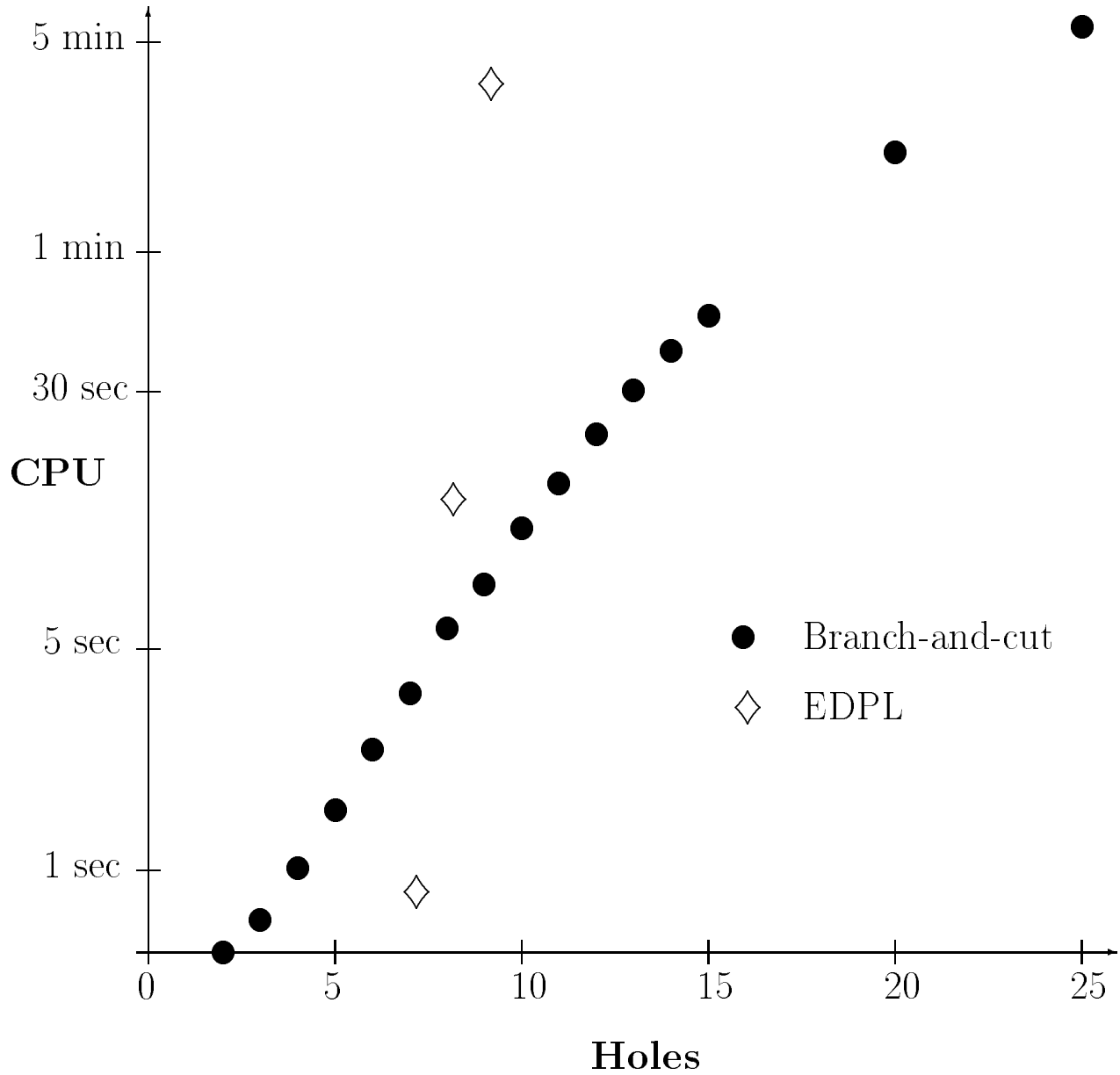
Available from ORlib. Encoding due to Kautz *et al.*, 1997.

problem name	variables	clauses	ave clause length	optimal value	B+C	
					CPU	nodes
steind2	1295	1765	1.31	220	86.35	105
steind3	1416	1885	1.25	1646	3.08	1
steind4	1499	2074	1.28	2044	4.44	1
steind5	1749	2646	1.34	3419	11.42	1
steind7	2045	2325	1.15	103	2.90	3
steind8	2166	2544	1.15	1180	3.14	1
steind9	2249	2797	1.20	1585	4.73	1
steind10	2499	3261	1.23	2219	10.46	1
steind11	5120	5882	1.17	29	467.52	333
steind12	5009	5039	1.01	42	2.57	1
steind13	5166	5499	1.06	544	4.29	1
steind14	5249	5709	1.08	740	5.81	1
steind15	5499	6202	1.11	1193	11.41	1
steind16	25032	25133	1.01	13	18.41	2
steind18	25166	25412	1.01	262	13.42	1
steind19	25249	25585	1.01	359	15.02	1
steind20	25499	26041	1.02	558	20.54	1

Results for Steiner “D” tree problems

Pigeonhole problems Hooker.

hole n asks if it is possible to place $n + 1$ pigeons in n holes without two pigeons being in the same hole.



Log(CPU time) vs Number of Pigeons for pigeon hole problems

Branch-and-cut on an instance

depth	nodes	resolution cuts/node	odd cycle cuts/node	clique cuts/node
0	1	0.000	0.000	0.000
1	2	14.000	0.000	0.000
2	4	26.750	0.500	0.000
3	8	7.750	2.375	0.250
4	16	14.000	6.688	0.063
5	32	9.906	9.313	0.219
6	64	7.469	14.984	0.109
7	128	4.656	18.430	0.297
8	254	4.961	24.047	0.335
9	457	3.569	22.571	0.267
10	585	2.829	21.043	0.251
11	412	2.415	17.944	0.204
12	179	2.302	15.391	0.196
13	60	1.733	11.050	0.233
14	22	2.000	9.363	0.227
15	4	0.000	6.500	0.000
16	1	0.000	8.000	0.000

Cuts per node in the branch and cut tree of a 50 variable, 500 clause MAX3SAT instance

7 Conclusions

- **Branch-and-cut** generates smaller trees than **EDPL**, but spends more time at each node.
- **Branch-and-cut** is very effective for **MAX2SAT** problems with a reasonable number of clauses.
- Once the **MAX2SAT** instance has a **large** number of clauses, the **semidefinite programming** approach is superior. The **SDP** relaxation appears to improve as the number of clauses increases.
- **EDPL** is superior for **MAX3SAT**. For **MAX3SAT**, the branch-and-cut tree can expand in almost a binary fashion until a sufficient number of clauses of length 2 have been generated.
- **Branch-and-cut** is superior for the **Steiner** problems and for the **Pigeonhole** problems.
- The **clique** inequalities are essential for effective solution of the **Pigeonhole** problems.

References

- [1] B. Borchers and J. Furman. A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems. Technical report, Mathematics Department, New Mexico Tech, Socorro, NM 87801, October 1995. Revised: March 1997. To appear in Journal of Combinatorial Optimization.
- [2] J. Cheriyan, W. H. Cunningham, L. Tunçel, and Y. Wang. A linear programming and rounding approach to max 2-sat. In David S. Johnson and Michael A. Trick, editors, *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge, DIMACS Series In Discrete Mathematics and Theoretical Computer Science*, volume 26, pages 395–414. AMS, 1996.
- [3] M. Davis and H. Putnam. A computing procedure for quantification theory. *J. Assoc. Comput. Mach.*, 7:201–215, 1960.
- [4] Michel X. Goemans and David P. Williamson. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *J. Assoc. Comput. Mach.*, 42:1115–1145, 1995.
- [5] J. Gu. Efficient local search for very large-scale satisfiability problems. *SIGART Bulletin*, 3, 1992.
- [6] J. Gu. Local search for satisfiability (sat) problem. *IEEE Transactions On Systems Man And Cybernetics*, 23:1108–1129, 1993.
- [7] J. Gu. Global optimization for satisfiability (sat) problem. *IEEE Transactions On Knowledge and Data Engineering*, 6:361–381, 1994.
- [8] J. Gu, P.W. Purdom, J. Franco, and B.W. Wah. Algorithms for the satisfiability (SAT) problem: A survey. In D. Du, J. Gu, and P. Pardalos, editors, *Satisfiability Problem: Theory and Applications*, volume 35 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 19–151. AMS/DIMACS, 1997.
- [9] C. Helmberg and F. Rendl. Solving quadratic (0,1)-problems by semidefinite programs and cutting planes. *Mathematical Programming*, 82:291–315, 1998.

- [10] C. Helmberg, F. Rendl, R. J. Vanderbei, and H. Wolkowicz. An interior point method for semidefinite programming. *SIAM Journal on Optimization*, 6:342–361, 1996.
- [11] J. N. Hooker. A quantitative approach to logical inference. *Decision Support Systems*, 4:45–69, 1988.
- [12] J. N. Hooker and C. Fedjki. Branch-and-cut solution of inference problems in propositional logic. *Annals of Mathematics and Artificial Intelligence*, 1:123–139, 1990.
- [13] R. G. Jeroslow and J. Wang. Solving propositional satisfiability problems. *Annals of Mathematics and AI*, 1:167–187, 1990.
- [14] H. Kautz, B. Selman, and Y. Jiang. A general stochastic approach to solving problems with hard and soft constraints. In D. Du, J. Gu, and P. Pardalos, editors, *Satisfiability Problem: Theory and Applications*, volume 35 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 573–586. AMS/DIMACS, 1997.
- [15] D. Loveland. *Automated theorem proving: A logical basis*. North-Holland, New York, 1978.
- [16] B. Selman, H. A. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In David S. Johnson and Michael A. Trick, editors, *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge, DIMACS Series In Discrete Mathematics and Theoretical Computer Science*, volume 26, pages 521–532. AMS, 1996.
- [17] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-91), Anaheim, CA*, pages 440–446, July 1992.
- [18] R. J. Wallace and E. C. Freuder. Comparative studies of constraint satisfaction and davis-putnam algorithms for maximum satisfiability problems. In David S. Johnson and Michael A. Trick, editors, *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge, DIMACS Series In Discrete Mathematics and Theoretical Computer Science*, volume 26, pages 587–615. AMS, 1996.