Satisfiability and Maximum Satisfiability Descriptions, Readings, Problems

Last revision: December 3, 1992

This a work in progress, intended as a solely to define the problems, suggest some introductory readings and offer some suggested problems. Feel free to send us any suggested additions or clarifications, or additional project suggestions. The email address for suggestions is

`challenge@dimacs.rutgers.edu`

.

Also, any researchers with recent papers or surveys are encouraged to contact that address to arrange for their papers to be made available by ftp.

1 Introduction and Problem Definitions

The *Satisfiability Problem (SAT)* and its close relative, the *Maximum Satisfiability Problem (MAX–SAT)* are central problems in artificial intelligence, logic, and computational complexity. These problems can be defined as follows:

A boolean *variable* x is a variable that can assume only the values *true* or *false* (sometimes denoted 1 or 0 respectively). A boolean formula is a combination of boolean variables using the logical connectives *not* ($\bar{x}$), *or* ($\vee$), and *and* ($\wedge$). A variable or a negation of a variable is called a *literal*. A disjunction of literals is called a *clause*.

Given a set of clauses $C_1, C_2, \ldots, C_m$ on the variables $x_1, x_2, \ldots, x_n$, the satisfiability problem is to determine if the formula

$$C_1 \wedge C_2 \wedge \ldots \wedge C_m$$

is satisfiable. That is, is there an assignment of values to the variables so that the above formula evaluates to *true*. Clearly, this requires that each C_j evaluate to *true*.

The maximum satisfiability problem is to find an assignment of values to the variables so that the largest number of clauses evaluate to *true*.

2 Suggested Readings

Both SAT and MAX–SAT have a large literature. Here we simply list a few of the relevant papers according to a rough classification scheme. These papers provide many pointers to other papers and areas of interest. A more detailed review is in preparation.

SAT: Symbolic and Heuristic Methods. Since artificial intelligence is a large generator of SAT problems, it is not surprising that symbolic methods have been generated to solve these problems. The best known of these is Robinson's resolution algorithm ([15], though see [12] for some refinements and additional description). The most widely known variant is that of Davis and Putnam [4]. Most large expert systems seem to use heuristic solution methods ([2]). One heuristic approach, based on local search, is that of Selman, Levesque and Mitchell [16] (available in the ftp directory `pub/challenge/sat/contributed/selman`).

SAT: Mathematical Programming Methods. A wonderful exposition of using such mathematical programming techniques as cutting planes and branch and bound is Hooker [8]. This work summarizes a large number of other papers and gives an overview of the field suitable for someone trained in mathematical programming. Other relevant papers include Jeroslow and Wang [10] and Hooker [9]. Kamath et. al. [11] apply interior point methods to the problem and report some computational results.

SAT: Solvable Special Cases. There are a number of known special cases that can be solved quickly. These include 2–satisfiability (each clause has at most two literals), Horn systems (that have at most one non–negated variable per clause), hidden Horn systems (that are Horn after suitable definitions of the variables) and so on. In all these cases, not only are there polynomial algorithms for satisfiability, but there are extremely fast algorithms (often linear). Methods for these problems have been used as heuristics for unstructured problems (for instance, chaining methods that work for Horn

systems are also used for more general systems, see [14]).

SAT: Probabilistic Results. There are a number of results about the likelihood of randomly generated problems being solvable as well as the likely performance of algorithms on such problems. Some suggested papers are Franco [5], Franco and Paull [6], and Chvátal and Reed [3]. A discussion of how to generate hard random problems is given in Mitchell, Selman, and Levesque [13] (available in the ftp directory `pub/challenge/sat/contributed/selman`).

MAX–SAT: Heuristic Techniques. Hansen and Jaumard [7] have a recent paper that does a nice job of summarizing heuristics available for MAX–SAT. In addition to reviewing work on the general problem, they also review heuristics for the case when there are just two literals per clause (MAX–2–SAT). The also examine some local search schemes including a variation on simulated annealing. The heuristic method of Selman et. al. [16] is actually a local search method for solving MAX–SAT (with a particular emphasis on cases where all clauses can be satisfied).

MAX–SAT: Worst Case Bounds. It is possible to find a solution to MAX–SAT that is within a constant bound of optimal (that is, is guaranteed to find a truth assignment that satisfies at least a constant number times the optimal number of clauses. Yannakakis [17] gives an algorithm that has a guaranteed ratio of 3/4. Arora et. al. [1] show that there is some threshold $c < 1$ such that no polynomial time algorithm can guarantee a solution satisfying at least c times the optimal number of clauses.

3 Suggested Projects

There are a many directions computational–oriented research could go. Here are a number of suggestions.

1. Construct interesting test beds. One idea is to use NP-completeness transformations, from, for example, from graph coloring and clique, the other problems covered by this Challenge.

2. Examine practical instances. SAT and MAX–SAT are reputed to have many applications, but there is only a limited number of real instances available. Expert systems and other logic machines should be generating many, many instances. Do these instances "look" different from

random problems? Are such instances easier or harder to solve in practice? Can aspects of such instances be embedded in random generators to allow for generating larger random instances.? Are there applications that "typically" generate very hard instances?

3. Find an application that yields "easy" instance in practice, and test behavior of exact algorithms on very large instances of this type.

4. Explore the question of satisfiability/non-satisfiability for various types of random k-SAT instances. There is already a fair amount of research on this subject, but a computational approach may provide insights that are not yet noticed.

5. Examine heuristics for MAX-SAT. Here there is a substantial theory, but perhaps less experimental data. How much better than worst-case are these in practice?

6. Selman and others have had reasonable success with local search. Can you do better with simulated annealing, tabu search and other variations?

7. Test how best to use various parallel architectures to speed up the search in exact algorithms.

8. Develop new algorithms for (exact) satisfiability, and test them.

9. Compare LP and interior point approaches with other more traditional approaches. One approach suggested in Hooker [8] is to replace general purpose linear programming codes with a specialized code that exploits the structure inherent in satisfiability problems.

10. Examine mathematical programming techniques for MAX–SAT. While computational approaches to SAT have received a lot of attention recently, little work has yet been published on similar methods for MAX–SAT. Are similar approaches too much the same to be interesting or are there some interesting variations?

References

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *Proceedings 33rd IEEE Symposium on the Foundations of Computer Science*, pages 14–23, Los Angeles, CA, 1992. IEEE Computer Society.

[2] A. Barr and E.A. Feigenbaum. *The Handbook of Artificial Intelligence.* William Kaufman, 1982.

[3] Vasek Chvátal and Bruce Reed. Mick gets some (the odds are on his side). In *Proceedings on the Foundations of Computer Science*, Pittsburgh, 1992.

[4] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[5] John Franco. On the probabilistic performance of algorithms for the satisfiability problem. *Information Processing Letters*, 23:103–106, 1986.

[6] John Franco and M. Paull. Probabilistic analysis of the davis putnam procedure for solving the satisfiability problem. *Discrete Applied Mathematics*, 5:77–87, 1983.

[7] Pierre Hansen and Brigitte Jaumard. Algorithms for the maximum satisfiability problem. *Computing*, 44:279–303, 1990.

[8] John N. Hooker. A quantitative approach to logical inference. *Decision Support Systems*, 4:45–69, 1988.

[9] John N. Hooker. Resolution vs. cutting plane solution of inference problems: Some computational results. *Operations Research Letters*, 7:1–7, 1988.

[10] Robert E. Jeroslow and J. Wang. Solving propositional satisfiability problems. *Annals of Mathematics and AI*, 1:167–187, 1990.

[11] A.P. Kamath, N.K. Karmarkar, K.G. Rmakrishnan, and M.G.C. Resende. Computational experience with an interior point algorithm on the satisfiability problem. *Annals of Operations Research*, 25:43–58, 1990.

[12] D.W. Loveland. *Automated Theorem Proving: A Logical Basis*. North–Holland, 1978.

[13] David Mitchell, Bart Selman, and Hector Levesque. Hard and easy distributions of SAT problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI–92)*, San Jose, CA, 1992.

[14] N.J. Nilsson. *Principles of Artificial Intelligence*. Tioga Publishing Company, 1980.

[15] J.A. Robinson. A machine–oriented logic based on the resolution principle. *Journal of the ACM*, 1965:23–41, 1965.

[16] Bart Selman, Hector Levesque, and David Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI–92)*, San Jose, CA, 1992.

[17] Mihalis Yannakakis. On the approximation of maximum satisfiability. In *Proceedings of the Third ACM–SIAM Symposium on Discrete Algorithms*, pages 1–9, 1992.