B

Backtracking Based *k*-SAT Algorithms

2005; Paturi, Pudlák, Saks, Zane

Ramamohan Paturi[1], Pavel Pudlák[2], Michael Saks[3], Francis Zane[4]

[1] Department of Computer Science and Engineering, University of California at San Diego, San Diego, CA, USA

[2] Mathematical Institute, Academy of Science of the Czech Republic, Prague, Czech Republic

[3] Department of Mathematics, Rutgers, State University of New Jersey, Piscataway, NJ, USA

[4] Bell Laboraties, Lucent Technologies, Murray Hill, NJ, USA

Problem Definition

Determination of the complexity of k-CNF satisfiability is a celebrated open problem: given a Boolean formula in conjunctive normal form with at most k literals per clause, find an assignment to the variables that satisfies each of the clauses or declare none exists. It is well-known that the decision problem of k–CNF satisfiability is NP-complete for $k \geq 3$. This entry is concerned with algorithms that significantly improve the worst case running time of the naive exhaustive search algorithm, which is poly$(n)2^n$ for a formula on n variables. Monien and Speckenmeyer [8] gave the first real improvement by giving a simple algorithm whose running time is $O(2^{(1-\varepsilon_k)n})$, with $\varepsilon_k > 0$ for all k. In a sequence of results [1,3,5,6,7,9,10,11,12], algorithms with increasingly better running times (larger values of ε_k) have been proposed and analyzed.

These algorithms usually follow one of two lines of attack to find a satisfying solution. Backtrack search algorithms make up one class of algorithms. These algorithms were originally proposed by Davis, Logemann and Loveland [4] and are sometimes called Davis–Putnam procedures. Such algorithms search for a satisfying assignment by assigning values to variables one by one (in some order), backtracking if a clause is made false. The other class of algorithms is based on local searches (the first guaranteed performance results were obtained by Schöning [12]). One starts with a randomly (or strategically) selected assignment, and searches locally for a satisfying assignment guided by the unsatisfied clauses.

This entry presents **ResolveSat**, a randomized algorithm for k-CNF satisfiability which achieves some of the best known upper bounds. **ResolveSat** is based on an earlier algorithm of Paturi, Pudlák and Zane [10], which is essentially a backtrack search algorithm where the variables are examined in a randomly chosen order. An analysis of the algorithm is based on the observation that as long as the formula has a satisfying assignment which is isolated from other satisfying assignments, a third of the variables are expected to occur as unit clauses as the variables are assigned in a random order. Thus, the algorithm needs to correctly guess the values of at most 2/3 of the variables. This analysis is extended to the general case by observing that there either exists an isolated satisfying assignment, or there are many solutions so the probability of guessing one correctly is sufficiently high.

ResolveSat combines these ideas with resolution to obtain significantly improved bounds [9]. In fact, **ResolveSat** obtains the best known upper bounds for k-CNF satisfiability for all $k \geq 5$. For $k = 3$ and 4, Iwama and Takami [6] obtained the best known upper bound with their randomized algorithm which combines the ideas from Schöning's local search algorithm and **ResolveSat**. Furthermore, for the promise problem of unique k-CNF satisfiability whose instances are conjectured to be among the hardest instances of k-CNF satisfiability [2], **ResolveSat** holds the best record for all $k \geq 3$. Bounds obtained by **ResolveSat** for unique k-SAT and k-SAT, for $k = 3, 4, 5, 6$ are shown in Table 1. Here, these bounds are compared with those of of Schöning [12], subsequently improved results based on local search [1,5,11], and the most recent improvements due to Iwama and Takami [6]. The upper bounds obtained by these algorithms are ex-

pressed in the form $2^{cn-o(n)}$ and the numbers in the table represent the exponent c. This comparison focuses only on the best bounds irrespective of the type of the algorithm (randomized versus deterministic).

Notation In this entry, a CNF boolean formula $F(x_1, x_2, \ldots, x_n)$ is viewed as both a boolean function and a set of clauses. A boolean formula F is a k-CNF if all the clauses have size at most k. For a clause C, write $var(C)$ for the set of variables appearing in C. If $v \in var(C)$, the *orientation* of v is positive if the literal v is in C and is negative if $\bar{v}$ is in C. Recall that if F is a CNF boolean formula on variables $(x_1, x_2, \ldots, x_n)$ and a is a partial assignment of the variables, the *restriction* of F by a is defined to be the formula $F' = F\lceil_a$ on the set of variables that are not set by a, obtained by treating each clause C of F as follows: if C is set to 1 by a then delete C, and otherwise replace C by the clause C' obtained by deleting any literals of C that are set to 0 by a. Finally, a *unit clause* is a clause that contains exactly one literal.

Key Results

ResolveSat Algorithm

The **ResolveSat** algorithm is very simple. Given a k-CNF formula, it first generates clauses that can be obtained by resolution without exceeding a certain clause length. Then it takes a random order of variables and gradually assigns values to them in this order. If the currently considered variable occurs in a unit clause, it is assigned the only value that satisfies the clause. If it occurs in contradictory unit clauses, the algorithm starts over. At each step, the algorithm also checks if the formula is satisfied. If the formula is satisfied, then the input is accepted. This subroutine is repeated until either a satisfying assignment is found or a given time limit is exceeded.

The **ResolveSat** algorithm uses the following subroutine, which takes an arbitrary assignment y, a CNF formula F, and a permutation π as input, and produces an assignment u. The assignment u is obtained by considering the variables of y in the order given by π and modifying their values in an attempt to satisfy F.

Function **Modify**(CNF formula $G(x_1, x_2, \ldots, x_n)$, permutation π of $\{1, 2, \ldots, n\}$, assignment y) $\longrightarrow$ (assignment u)
  $G_0 = G$.
  for $i = 1$ **to** n
    if G_{i-1} contains the unit clause $x_{\pi(i)}$
      then $u_{\pi(i)} = 1$
    else if G_{i-1} contains the unit clause $\bar{x}_{\pi(i)}$
      then $u_{\pi(i)} = 0$
    else $u_{\pi(i)} = y_{\pi(i)}$
    $G_i = G_{i-1}\lceil_{x_{\pi(i)}=u_{\pi(i)}}$
  end /* end for loop */
  return u;

The algorithm **Search** is obtained by running **Modify**(G, π, y) on many pairs (π, y), where π is a random permutation and y is a random assignment.

Search(CNF-formula F, integer I)
  repeat I times
    π = uniformly random permutation of $1, \ldots, n$
    y = uniformly random vector $\in \{0, 1\}^n$
    u = **Modify**(F, π, y);
    if u satisfies F
      then output(u); **exit**;
  end/* end repeat loop */
  output('Unsatisfiable');

The **ResolveSat** algorithm is obtained by combining **Search** with a preprocessing step consisting of *bounded resolution*. For the clauses C_1 and C_2, C_1 and C_2 *conflict* on variable v if one of them contains v and the other contains $\bar{v}$. C_1 and C_2 is a *resolvable pair* if they conflict on exactly one variable v. For such a pair, their *resolvent*, denoted $R(C_1, C_2)$, is the clause $C = D_1 \vee D_2$ where D_1 and D_2 are obtained by deleting v and $\bar{v}$ from C_1 and C_2. It is easy to see that any assignment satisfying C_1 and C_2 also satisfies C. Hence, if F is a satisfiable CNF formula containing the resolvable pair C_1, C_2 then the formula $F' = F \wedge R(C_1, C_2)$ has the same satisfying assignments as F. The resolvable pair C_1, C_2 is *s-bounded* if $|C_1|, |C_2| \leq s$ and $|R(C_1, C_2)| \leq s$. The following subroutine extends a formula F to a formula F_s by applying as many steps of s-bounded resolution as possible.

Resolve(CNF Formula F, integer s)
  $F_s = F$.
  while F_s has an s-bounded resolvable pair C_1, C_2
      with $R(C_1, C_2) \notin F_s$
    $F_s = F_s \wedge R(C_1, C_2)$.
  return (F_s).

The algorithm for k-SAT is the following simple combination of **Resolve** and **Search**:

ResolveSat(CNF-formula F, integer s, positive integer I)
  F_s = **Resolve**(F, s).
  Search(F_s, I).

Backtracking Based *k*-SAT Algorithms, Table 1
This table shows the exponent *c* in the bound $2^{cn-o(n)}$ for the unique *k*-SAT and *k*-SAT from the ResolveSat algorithm, the bounds for *k*-SAT from Schöning's algorithm [12], its improved versions for 3-SAT [1,5,11], and the hybrid version of [6]

k	unique *k*-SAT [9]	*k*-SAT [9]	*k*-SAT [12]	*k*-SAT [1,5,11]	*k*-SAT [6]
3	0.386 …	0.521 …	0.415 …	0.409 …	0.404 …
4	0.554 …	0.562 …	0.584 …		0.559 …
5	0.650 …		0.678 …		
6	0.711 …		0.736 …		

Analysis of ResolveSat

The running time of **ResolveSat**(F, s, I) can be bounded as follows. **Resolve**(F, s) adds at most $O(n^s)$ clauses to F by comparing pairs of clauses, so a naive implementation runs in time $n^{3s}\text{poly}(n)$ (this time bound can be improved, but this will not affect the asymptotics of the main results). **Search**(F_s, I) runs in time $I(|F| + n^s)\text{poly}(n)$. Hence the overall running time of **ResolveSat**(F, s, I) is crudely bounded from above by $(n^{3s} + I(|F| + n^s))\text{poly}(n)$. If $s = O(n/\log n)$, the overall running time can be bounded by $I|F|2^{O(n)}$ since $n^s = 2^{O(n)}$. It will be sufficient to choose s either to be some large constant or to be a *slowly growing* function of n. That is, $s(n)$ tends to infinity with n but is $O(\log n)$.

The algorithm **Search**(F, I) always answers "unsatisfiable" if F is unsatisfiable. Thus the only problem is to place an upper bound on the error probability in the case that F is satisfiable. Define $\tau(F)$ to be the probability that **Modify**(F, π, y) finds some satisfying assignment. Then for a satisfiable F the error probability of **Search**(F, I) is equal to $(1 - \tau(F))^I \le \mathrm{e}^{-I\tau(F)}$, which is at most e^{-n} provided that $I \ge n/\tau(F)$. Hence, it suffices to give good upper bounds on $\tau(F)$.

Complexity analysis of **ResolveSat** requires certain constants μ_k for $k \ge 2$:

$$\mu_k = \sum_{j=1}^{\infty} \frac{1}{j(j + \frac{1}{k-1})} .$$

It is straightforward to show that $\mu_3 = 4 - 4\ln 2 > 1.226$ using Taylor's series expansion of $\ln 2$. Using standard facts, it is easy to show that μ_k is an increasing function of k with the limit $\sum_{j=1}^{\infty}(1/j^2) = (\pi^2/6) = 1.644\ldots$

The results on the algorithm **ResolveSat** are summarized in the following three theorems.

Theorem 1 ***(i)*** *Let $k \ge 5$, and let $s(n)$ be a function going to infinity. Then for any satisfiable k-CNF formula F on n variables,*

$$\tau(F_s) \ge 2^{-(1-\frac{\mu_k}{k-1})n-o(n)} .$$

*Hence, **ResolveSat**(F, s, I) with $I = 2^{(1-\mu_k/(k-1))n+O(n)}$ has error probability $O(1)$ and running time $2^{(1-\mu_k/(k-1))n+O(n)}$ on any satisfiable k-CNF formula, provided that $s(n)$ goes to infinity sufficiently slowly.*

(ii) *For $k \ge 3$, the same bounds are obtained provided that F is uniquely satisfiable.*

Theorem 1 is proved by first considering the uniquely satisfiable case and then relating the general case to the uniquely satisfiable case. When $k \ge 5$, the analysis reveals that the asymptotics of the general case is no worse than that of the uniquely satisfiable case. When $k = 3$ or $k = 4$, it gives somewhat worse bounds for the general case than for the uniquely satisfiable case.

Theorem 2 *Let $s = s(n)$ be a slowly growing function. For any satisfiable n-variable 3-CNF formula, $\tau(F_s) \ge 2^{-0.521n}$ and so **ResolveSat**(F, s, I) with $I = n2^{0.521n}$ has error probability $O(1)$ and running time $2^{0.521n+O(n)}$.*

Theorem 3 *Let $s = s(n)$ be a slowly growing function. For any satisfiable n-variable 4-CNF formula, $\tau(F_s) \ge 2^{-0.5625n}$, and so **ResolveSat**$(F, s, I)$ with $I = n2^{0.5625n}$ has error probability $O(1)$ and running time $2^{0.5625n+O(n)}$.*

Applications

Various heuristics have been employed to produce implementations of 3-CNF satisfiability algorithms which are considerably more efficient than exhaustive search algorithms. The **ResolveSat** algorithm and its analysis provide a rigorous explanation for this efficiency and identify the structural parameters (for example, the width of clauses and the number of solutions), influencing the complexity.

Open Problems

The gap between the bounds for the general case and the uniquely satisfiable case when $k \in \{3, 4\}$ is due to a weakness in analysis, and it is conjectured that the asymptotic bounds for the uniquely satisfiable case hold in general for all k. If true, the conjecture would imply that **ResolveSat** is also faster than any other known algorithm in the $k = 3$ case.

Another interesting problem is to better understand the connection between the number of satisfying assignments and the complexity of finding a satisfying assignment [2]. A strong conjecture is that satisfiability for formulas with many satisfying assignments is strictly easier than for formulas with fewer solutions.

Finally, an important open problem is to design an improved k-SAT algorithm which runs faster than the bounds presented in here for the unique k-SAT case.

Cross References

- ▶ Local Search Algorithms for kSAT
- ▶ Maximum Two-Satisfiability
- ▶ Parameterized SAT
- ▶ Thresholds of Random k-SAT

Recommended Reading

1. Baumer, S., Schuler, R.: Improving a Probabilistic 3-SAT Algorithm by Dynamic Search and Independent Clause Pairs. In: SAT 2003, pp. 150–161
2. Calabro, C., Impagliazzo, R., Kabanets, V., Paturi, R.: The Complexity of Unique k-SAT: An Isolation Lemma for k-CNFs. In: Proceedings of the Eighteenth IEEE Conference on Computational Complexity, 2003
3. Dantsin, E., Goerdt, A., Hirsch, E.A., Kannan, R., Kleinberg, J., Papadimitriou, C., Raghavan, P., Schöning, U.: A deterministic $(2-\frac{2}{k+1})^n$ algorithm for k-SAT based on local search. Theor. Comp. Sci. **289**(1), 69–83 (2002)
4. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem proving. Commun. ACM **5**, 394–397 (1962)
5. Hofmeister, T., Schöning, U., Schuler, R., Watanabe, O.: A probabilistic 3–SAT algorithm further improved. In: STACS 2002. LNCS, vol. 2285, pp. 192–202. Springer, Berlin (2002)
6. Iwama, K., Tamaki, S.: Improved upper bounds for 3-SAT. In: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms, 2004, pp. 328–329
7. Kullmann, O.: New methods for 3-SAT decision and worst-case analysis. Theor. Comp. Sci. **223**(1–2), 1–72 (1999)
8. Monien, B., Speckenmeyer, E.: Solving Satisfiability In Less Than 2^n Steps. Discret. Appl. Math. **10**, 287–295 (1985)
9. Paturi, R., Pudlák, P., Saks, M., Zane, F.: An Improved Exponential-time Algorithm for k-SAT. J. ACM **52**(3), 337–364 (2005) (An earlier version presented in Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, 1998, pp. 628–637)
10. Paturi, R., Pudlák, P., Zane, F.: Satisfiability Coding Lemma. In: Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, 1997, pp. 566–574. Chicago J. Theor. Comput. Sci. (1999), http://cjtcs.cs.uchicago.edu/
11. Rolf, D.: 3-SAT ∈ *RTIME*(1.32971^n). In: ECCC TR03-054, 2003
12. Schöning, U.: A probabilistic algorithm for k-SAT based on limited local search and restart. Algorithmica **32**, 615–623 (2002) (An earlier version appeared in 40th Annual Symposium on Foundations of Computer Science (FOCS '99), pp. 410–414)

Best Response Algorithms for Selfish Routing

2005; Fotakis, Kontogiannis, Spirakis

Paul Spirakis
Computer Engineering and Informatics, Research and Academic Computer Technology Institute, Patras University, Patras, Greece

Keywords and Synonyms

Atomic selfish flows

Problem Definition

A setting is assumed in which n selfish users compete for routing their loads in a network. The network is an $s-t$ directed graph with a single source vertex s and a single destination vertex t. The users are ordered sequentially. It is assumed that each user plays after the user before her in the ordering, and the desired end result is a Pure Nash Equilibrium (PNE for short). It is assumed that, when a user plays (i. e. when she selects an $s-t$ path to route her load), the play is a best response (i. e. minimum delay), given the paths and loads of users currently in the net. The problem then is to find the class of directed graphs for which such an ordering exists so that the implied sequence of best responses leads indeed to a Pure Nash Equilibrium.

The Model

A *network congestion game* is a tuple $((w_i)_{i\in N}, G, (d_e)_{e\in E})$ where $N=\{1,\ldots,n\}$ is the set of users where user i controls w_i units of traffic demand. In *unweighted* congestion games $w_i=1$ for $i=1,\ldots,n$. $G(V,E)$ is a directed graph representing the communications network and d_e is the latency function associated with edge $e\in E$. It is assumed that the d_e's are non-negative and non-decreasing functions of the edge loads. The edges are called *identical* if $d_e(x)=x$, $\forall e\in E$. The model is further restricted to single-commodity network congestion games, where G has a single source s and destination t and the set of users' strategies is the set of $s-t$ paths, denoted P. Without loss of generality it is assumed that G is connected and that every vertex of G lies on a directed $s-t$ path.

A vector $P=(p_1,\ldots,p_n)$ consisting of an $s-t$ path p_i for each user i is a *pure strategies profile*. Let $l_e(P)=\sum_{i:e\in p_i} w_i$ be the load of edge e in P. The authors define *the cost* $\lambda^i_p(P)$ for user i routing her demand on