Some Explorations in SAT

Thomas C. Henderson, Amar Mitiche, Xiuyi Fan, and David Sacharny
University of Utah

UUCS-21-016

School of Computing
University of Utah
Salt Lake City, UT 84112 USA

14 July 2021

Abstract

We look at the SAT problem geometrically: for an n-variable problem, each corner of the hypercube is a possible SAT solution, and the interior points are possible Probabilistic SAT solutions. Given a Conjunctive Normal Form (CNF) sentence, $S = C_1 \wedge C_2 \wedge \ldots \wedge C_m$, over a set of Boolean variables $A = \{a_1, a_2, ..., a_n\}$, each conjunct, C_i, has at least one truth assignment that makes it false. If there are k literals in the conjunct, then there are 2^{n-k} truth assignments that make it false. A hyperplane can then be found for each conjunct which separates the solutions from the non-solutions. The intersection of the solution side of the hyperplane with the hypercube (the initial feasible region) produces a convex feasible region. Continuing this process for each conjunct, the result is a convex feasible region which may or may not contain a corner. Linear programming using the interior point method can be applied along each dimension to find the solutions with minimum and maximum values in that dimension. If neither of these has value 0 or 1, then there is no SAT solution. Moreover, our conjecture is that the interior point method can be made to find a corner solution if it exists. This is called the *Chop SAT* method. We also consider two other approaches which may prove useful in analyzing SAT: (1) count the number of solutions ruled out by each conjunct (this entails determining the number of new solutions eliminated by each conjunct), and (2) perform the interior point search in a non-Euclidean geometry (e.g., hyperbolic space). Current results are given for each approach.

1 Introduction

Suppose $\mathcal{A} = \{a_1, a_2, ..., a_n\}$ is a set of Boolean variables, and the set of $\mathcal{A}$-formulas is the inductive set of propositional well-formed formulas over $\mathcal{A}$. A Conjunctive Normal Form (CNF) sentence over $\mathcal{A}$ is defined as a conjunction of a set of disjunctions of literals.

$$S = C_1 \wedge C_2 \wedge \ldots \wedge C_m$$

$$C_i = D_{i_1} \vee D_{i_2} \vee \ldots \vee D_{i_{k_i}}$$

$$D_{i_j} = a_{i_j} or D_{i_j} = \neg a_{i_j}$$

The *Satisfiability Problem* is defined as[1]:

> Find an efficient algorithm for testing an $\mathcal{A}$-formula in CNF to determine whether it is truth-functional satisfiable.

By efficient, we understand "of polynomial complexity."

2 Basic Approach

The CNF SAT problem is cast as a linear programming problem:

Minimize $\pm \bar{\mathbf{e}}_1 \cdot x$

Subject to: $Ax \leq c$

where each constraint is given by:

$$-\alpha_i \cdot x \leq c_i$$

A solution for the SAT sentence exists iff a solution exists for the LP problem with the $x(1)$ value equal to 0 or 1.

Given a set of m conjuncts, $C_i, i = 1 : m$, each conjunct is used to produce a hyperplane of dimension $n - 1$ which separates the solutions (i.e., some subset of vertexes of the n-dimensional hypercube) from non-solutions. The hyperplane for the i^{th} conjunct is:

$$\alpha(i) \cdot x + c = 0$$

Each of these hyperplanes produces an inequality:

$$-\alpha(i) \cdot x \leq c_i$$

A matrix, A, is produced where each row is the $1 \times n$-tuple $\alpha(i)$. An $n \times 1$-vector, c, is constructed where the i^{th} element of c is c_i.

The way these hyperplanes are constructed, it is now possible to run the interior-point method for linear programming to find feasible points which minimize $f^T x$ for $x \in X$, where X is the feasible region and f is the e_1 unit vector. If there is a satisfying solution with a_1 set to 0, then the LP solution found will have 0 value in the first dimension. Next, minimize $f^T \cdot x$, where f is $-e_1$. If there is a CNF satisfying solution with a_1 set to 1, then the LP solution found will have a value of 1 in the first dimension. If neither of these is the case (i.e., no 0 value and no 1 value for the first dimension of the LP solution), then there is no satisfying solution.

This result is made possible due to two aspects of the approach:

1. The projection onto one axis allows bisection of the solutions. Although the projection onto dimension one determines the existence of a solution, the method can be iterated separately for the $x(1) = 0$ and $x(1) = 1$ subsets, respectively, in order to find a particular solution in n steps.
2. LP solves over the reals, so there is never any combinatorics of integer solutions.

Suppose a 0/1 solution is found for each dimension (i.e., for each Boolean variable). This does not guarantee a SAT solution since the individual solutions in each dimension may be on the surface of the hypercube and not at a corner. However, it is the case that if for any dimension, there is no 0 or 1 solution, then there is no SAT solution. The question is whether there exists a CNF sentence for which SAT solutions exist, yet for which linear programming does not find any of those solutions. After describing the details of this approach, we discuss directions of work which may be able to lower the complexity of finding SAT solutions.

3 Chop SAT Algorithm

Given m conjuncts, $C_i, i = 1 \ldots m$, then:

$$C_i = L_1 \vee L_2 \vee \ldots \vee L_k$$

Note that any complete truth assignment with $\neg L_1 \wedge \neg L_2 \wedge \ldots \wedge \neg L_k$ makes C_i false. Next, note that the complete conjunction set can be associated with the vertexes of the n-dimensional hypercube.

Observe that:

- If $k = n$, then this eliminates 1 solution (1 vertex).

- If $k = n - 1$, then this eliminates 2 points (on a line).
- If $k = n - 2$, then this eliminates 4 points (on a plane).
- ...
- If $k = 1$, then this eliminates half the points in the hypercube (all in a hyperplane of dimension $n - 1$.

The individual hyperplane is determined as follows. Let $A = \{1, 2, \ldots, n\}$, and $\mathcal{I} \subseteq A$. Given $C_i = L_1 \vee L_2 \vee \ldots \vee L_k$, then define α_i, the hyperplane normal vector, as follows.

$$\forall i_j \in \mathcal{I}, \alpha_i(i_j) = 1 \textit{ if } L_j \textit{ is an atom } a_{i_j}, \textit{ else } -1$$

$$\forall m \notin \mathcal{I}, \alpha_i(m) = 0$$

$$\alpha_i = \frac{\alpha_i}{\parallel \alpha_i \parallel}$$

In order to get the constant for the hyperplane equation, a point must be found on the hyperplane. This is selected so that the hyperplane cuts the edges of the hypercube at a distance ξ from the non-solution vertex. This distance depends on the number k of non-solution vertexes:

$$d = \parallel \xi \frac{\overline{\mathbf{b_k}}}{k} \parallel$$

where $\overline{b_k}$ is a k-tuple of 1's. Next:

$$\forall i_j \in \mathcal{I}, p(i_j) = 0 \textit{ if } L_i \textit{ is an atom, else } 1$$

$$\forall m \notin \mathcal{I}, p(m) = 0$$

Then p is a non-solution vertex. To find a point, q, on the hyperplane:

$$q = p + d\alpha_i$$

This allows a solution for the constant, c, in the hyperplane:

$$c_i = -(\alpha_i \cdot q)$$

This yields the hyperplane equation:

$$\alpha_i \cdot x + c = 0$$

and the resulting inequality:

$$-\alpha_i \cdot x \leq c$$

3.1 The Chop SAT Algorithm

Thus, to solve a CNF instance:

1. Find the linear inequality for each conjunct.
2. Set up an $m \times n$ matrix, A, with row i set to $-\alpha_i$ (the negative of the hyperplane normal).
3. Set up an $n \times 1$ vector b with row i set to c_i (the constant from hyperplane i).
4. Apply the interior-point method for linear programming with A and b specifying the inequalities, and with $0 \leq x \leq 1$. Minimize $f^T x$ with $x \in X$, where X is the feasible region, using $f = \mathbf{e}_1$, i.e., the unit vector in the first dimension. Call the resulting solution x_{10}.
5. Apply the interior-point method for linear programming with A and b specifying the inequalities, no equality constraints, and with $0 \leq x \leq 1$. Minimize $f^T x$ with $x \in X$, where X is the feasible region, using $f = -\mathbf{e}_1$, i.e., the unit vector in the first dimension. Call the resulting solution x_{11}.
6. If $x_{10}(1) = 0$ or $x_{11}(1) = 1$,then it is possible there is a solution for the CNF sentence, S. If $x_{10}(1) > \xi$ and $x_{11}(1) < 1 - \xi$, then there is no satisfying solution.

Steps 4 and 5 are guaranteed to find a solution with $x(1) = 0$ or $x(1) = 1$, if there is such a point in the feasible region; however, this point may be on a face of the hypercube, and not at a corner.

Algorithm Chop SAT
On input:
S: CNF sentence
On output:
res: for each dimension, the min and max values found
sol: 1 if complete SAT solution found, else 0 *begin*
A = matrix of negated hyperplane normals (1 per row)
b = vector of hyperplane constants *for each* atom $a \in$ S
d is dimension associated with a
e_d is unit vector in dimension d
x0 = linear programming solution projected on -e_d
x1 = linear programming solution projected on e_d
res(d,1) = x0
res(d,2) = x1
if x0 or x1 is complete 0/1 solution
sol = 1;
end
end

For Matlab code, see Appendix A.

Now consider the time complexity of the approach. Converting the conjuncts to hyperplanes is clearly polynomial given that there are m conjuncts, and each has at most n literals. Given the sizes of A and b, the interior-point method for linear programming requires only polynomial time (see Potra[2]).

3.2 Some Examples

3.2.1 2D One Solution

Consider the two clauses in modus ponens:

$1.a_1$

$2.\neg a_1 \vee a_2$

Then the feasible region is shown in Figure 1. The hyperplane found for conjunct 1 (with $\xi = 0.9$)

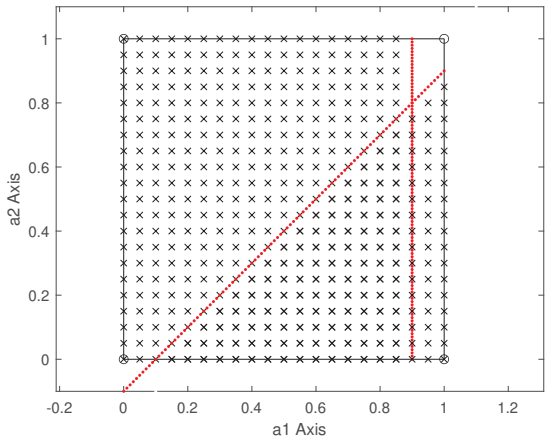

Figure 1: The Feasible Region for Modus Ponens.

is:

$$1.0a_1 + 0a_2 - 0.9 = 0$$

while the hyperplane for conjunct 2 is:

$$-0.7071a_1 + 0.7071a_2 + 0.7071 = 0$$

The solutions are:

$$x_{10}(1) = 0.9$$

and

$$x_{11}(1) = 1$$

3.2.2 2D No Solution

For a second example, consider a CNF sentence with no satisfying solution:

$$1.\neg a_1 \vee \neg a_2$$

$$2.\neg a_1 \vee a_2$$

$$3.a_1 \vee \neg a_2$$

$$4.a_1 \vee a_2$$

Then the feasible region is shown in Figure 2. The hyperplane found for conjunct 1 is:

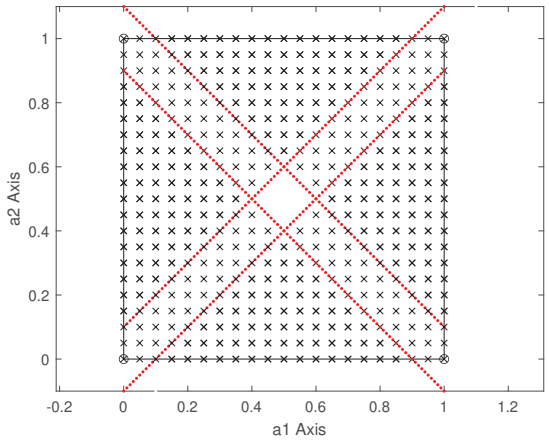

Figure 2: The Feasible Region for an Unsatisfiable CNF Sentence.

$$-0.7071a_1 - 0.7071a_2 + 0.7778 = 0$$

The hyperplane found for conjunct 2 is:

$$-0.7071a_1 + 0.7071a_2 + 0.0707 = 0$$

The hyperplane found for conjunct 3 is:

$$0.7071a_1 - 0.7071a_2 + 0.0707 = 0$$

The hyperplane found for conjunct 4 is:

$$0.7071a_1 + 0.7071a_2 - 0.6364 = 0$$

The linear programming solutions are:

$$x_{20}(1) = 0.4999$$

and

$$x_{21}(1) = 0.5001$$

indicating there is no satisfying solution for S.

3.2.3 3D Two Solutions

As a final example, consider the case with 3 variables, and such that $(\neg a_1 \wedge a_2 \wedge a_3) \vee (a_1 \wedge \neg a_2 \wedge \neg a_3)$ is true. Re-writing this in CNF yields:

$$1. : a_1 \vee a_2$$

$$2. : a_1 \vee a_3$$

$$3. : \neg a_1 \vee \neg a_2$$

$$4. : \neg a_2 \vee a_3$$

$$5. : \neg a_1 \vee \neg a_3$$

$$6. : a_2 \vee \neg a_3$$

Then the hyperplane equations are:

$$C_1 : 0.7071a_1 + 0.7071a_2 - 0.7071 = 0$$

$$C_2 : 0.7071a_1 + 0.7071a_3 - 0.7071 = 0$$

$$C_3 : -0.7071a_1 - 0.7071a_2 + 0.7071 = 0$$

$$C_4 : -0.7071a_2 + 0.7071a_3 + 0 = 0$$

$$C_5 : -0.7071a_1 - 0.7071a_3 + 0.7071 = 0$$

$$C_6 : 0.7071a_2 - 0.7071a_3 + 0 = 0$$

The linear programming solutions are:

$$x_1 = [0; 1; 1]$$

and

$$x_2 = [1; 0; 0]$$

This approach has been tested on thousands of randomly generated CNF sentences (both consistent and inconsistent) and always returned a solution where there was one, and gave the empty set where there was none. In addition, a test was made on a consistent 1000-atom, 30,000-conjunct CNF in which the solution was found in about five minutes (this was in Matlab with no special optimizations). On the other hand, for a CNF from satcompetition.org/2002 with 450 variables and 2025 clauses (each with 3 literals), *Chop SAT* produced a 0/1 in each dimension, but no complete SAT solution.

Conjecture: The linear programming interior point method can be made to find a corner point of a convex feasible region.

If this conjecture is true, then *Chop SAT* provides a polynomial-time method to solve SAT.

4 Further Ideas

4.1 Non-Solution Counting

As noted above, a conjunct on isolation eliminates 2^{n-k} solutions, where k is the number of literals in the conjunct. It is possible to solve SAT by counting the number of non-solutions, and it that number is 2^n, then the CNF sentence is not satisfiable. A clear way to do this is start the count at 0, and for each conjunct, determine the number of induced non-solutions, and then add the number of those that are not in the current set of non-solutions. The basic way to do this involves set intersection, and thus is of exponential cost. However, there may be other approaches, including geometric to this, and that is under study.

4.2 Non-Euclidean Geometry

It is possible that interior point method convergence can be improved by performing the operations in the hyperbolic geometry space. I.e., since parallel lines now intersect, if the solution trajectory follows a face, then it may eventually pass by a (Euclidean) corner since there are no corners anymore in the non-Euclidean space. Preliminary experiments are underway in lower-dimensional spaces to see the effectiveness of this strategy.

5 Conclusions and Future Work

The *Chop SAT* method provides a polynomial-time method to determine if a SAT solution exists, although there is no proof at the present time that this is guaranteed. That is, there may be a consistent CNF sentence for which no version of the interior point method provides a solution at a corner, but instead produces a solution in a hypercube face for each dimension.

Future work includes:

- Prove or disprove conjecture.
- Study non-solution counting approach.
- Determine linear programming interior point method convergence properties in non-Euclidean geometry.

6 Appendix A

```
function [res,sol] = BR_chop_SAT(KB,all)
% BR_chap_SAT - Apply Chop SAT algorithm to find SAT result
% On input:
%     KB (KB struct): CNF sentence; m clauses with field:
%       (k).clauses (1xp vector): disjunction with p literals
%           represented as signed integers
%     all (Boolean): if 0 return just one solution; else all
% On output:
%     res (2nxx array): solutions
%     sol (Boolean): 1 if a complete 0/1 solution found; else 0
% Call:
%     [res,sol] = BR_chop_SAT(KB,0);
% Author:
%     T. Henderson
%     UU
%     Summer 2021
%

ZERO_THRESH = 0.0001;

res = [];
sol = 0;
```

```
m = length(KB);
if m==0
    return
end

% Set up A and b
n = max(BR_vars(KB,[]));
res = zeros(n*2,n);
A = zeros(m,n);
b = zeros(m,1);

for i = 1:m
    clause = KB(i).clauses;
    k = length(clause);
    I = unique(abs(clause));
    alpha = zeros(n,1);
    for j = 1:k
        i_j = abs(clause(j));
        alpha(i_j) = sign(clause(j));
    end
    alpha = alpha/norm(alpha);
    b_k = ones(k,1)/k;
    d = norm(b_k);
    p = zeros(n,1);
    indexes = find(clause<0);
    atoms = abs(clause(indexes));
    p(atoms) = 1;
    q = p + d*alpha;
    c = -dot(alpha,q);
    A(i,:) = -alpha';
    b(i) = c;
end

index = 0;
for i = 1:n
    index = (i-1)*2 + 1;
    p = zeros(n,1);
    p(i) = 1;
    x1 = linprog(p,A,b,[],[],zeros(n,1),ones(n,1));
    if isempty(x1)
        res(index,:) = -ones(1,n);
    else
        res(index,:) = x1';
    end
```

```
    p(i) = -1;
    x2 = linprog(p,A,b,[],[],zeros(n,1),ones(n,1));
    index = index + 1;
    if isempty(x2)
        res(index,:) = -ones(1,n);
    else
        res(index,:) = x2';
    end
    OK1 = 1;
    for h = 1:n
        if ~isempty(x1)&x1(h)>ZERO_THRESH&x1(h)<1-ZERO_THRESH
            OK1 = 0;
        end
    end
    OK2 = 1;
    for h = 1:n
        if ~isempty(x2)&x2(h)>ZERO_THRESH&x2(h)<1-ZERO_THRESH
            OK2 = 0;
        end
    end
    if OK1==1&~isempty(x1)|(OK2==1&~isempty(x2))
        sol = 1;
        if all==0
            res = res(1:index,:);
            return
        end
    end
end
tch = 0;
```

References

[1] M.D. Davis, R. Sigal, and E.J.Weyuker. *Computability, Complexity, and Languages*. Morgan Kaufmann, San Diego, CA, 1994.

[2] F.A. Potra and S.J. Wright. Interior Point Methods. *Journal of Computational and Applied Mathematics*, 124, 2004.