Cool NAE 3-SAT

Diqiu Zhou and Cindy Le
Washington University in Saint Louis

This paper details potential methods and original discoveries on the MAX Not-All-Equal 3-Satisfiability Problem (MAX NAE-3-SAT). It provides basic insights to the problem, summarizes resources from multiple papers, and combines some devised approaches for polynomial-time approximation.

I. BACKGROUND

A. Introduction to NAE-3-SAT

A general Not-All-Equal (NAE) SAT optimization problem is to find an assignment of variables that maximizes the number of clauses satisfying the following condition:

"In any clause, there must be at least one True (T) and one False (F). "

NAE-3-SAT is a subset of the problem where there are exactly three literals in each clause.

B. Hardness of NAE-3-SAT

1. Proof of NP

The problem is in NP. A certificate of the problem is a satisfying assignment. We can check in polynomial time whether every clause in the assignment contains at least one True (T) and one False (F).

The problem is NP-hard by reduction from 3-SAT. Given an instance of 3-SAT containing m clauses, we can construct an instance of NAE-3-SAT with $2m$ clauses. For every clause of ϕ in the form $(x \vee y \vee z)$, we append two clauses to ϕ', $(x, y, a) \wedge (z, \neg a, F)$. where the variable a has a same value in both clauses. We add F to the second clause because we want the clause to include exactly three literals. The reduction runs in polynomial time $\mathrm{O}(m)$.

Hence, the problem is NP-Complete.

2. Proof of APX

MAX-3-SAT is APX-Hard and APX-Complete. Using the same reduction, we conclude that MAX NAE-3-SAT is APX-Complete as well.

3. Exact Recursion Algorithm

To solve the problem exactly, we define a recursive function that returns the maximum number of clauses satisfied by the assignment a for the first k variables. The function takes three parameters: k, a, and the total number of variables n. The recursive boundary is when all the variables have been assigned some truth values. The algorithm is illustrated below, as a call of $rec(1, [], n)$ yields an objective value.

Algorithm 1 rec(k,a,n)

```
if k==n then
return number of satisfied clauses
else
x ← rec(k + 1,a.append(True), n)
y ← rec(k + 1,a.append(False), n)
return max(x, y)
end if
```

The running time $\mathrm{O}(2^n)$ of the algorithm is exponential, which makes sense because no known exact algorithm runs in polynomial time.

C. NAE-3-SAT Benchmarks

Depending on how problem instances are generated, the family of SAT problems has two benchmarks: dealing with the crafted instances and with the randomized ones. Some carefully crafted instances have prevented the approximation from reaching better ratios. Hasta's 2001 paper[1] has shown that no algorithm approximates better than $\mathrm{O}(1 - 2^k + \epsilon)$ for MAX-k-SAT where $k \geq 3$ and $\epsilon > 0$. When $k = 3$, the upper bound of approximation ratio is $\frac{7}{8}$ for crafted instances.

The randomized instances, however, allows algorithms to reach better expected approximation ratio. The instances are generated by picking a possible clause one at a time from a pool that follows some distribution. The most commonly studied case is when the distribution is uniform. Currently, the best approximation ratio is 0.9087 by Zwick[6].

In this paper, we focus on the uniformly randomized SAT problems. In addition, all NAE-3-SAT below refer to the maximization problem of NAE-3-SAT.

II. POTENTIAL METHODS

A. Basic Randomized Algorithm

One of the algorithms is to randomly assign either True (T) or False (F) to each variable. Trivially, the resulting

assignment is feasible.

The algorithm is $\frac{3}{4}$-approximation on random NAE-3-SAT. For each clause, we have $2*2*2 = 8$ possible combinations of values for the three variables. Among the eight ways with equal probabilities, (T, T, T) and (F, F, F) do not satisfy the clause. Each clause has a probability of $(8-2)/8 = 3/4$ to be satisfied. The expected number of clauses satisfied during the approximation is $\frac{3}{4}m$ where m is the number of clauses. Because an optimal solution does no better than m, the algorithm is $\frac{3}{4}$-approximation on average.

B. A First Greedy Algorithm

A greedy algorithm for random NAE-3-SAT is to assign each value to a variable that satisfies the most clauses. To describe the algorithm in a more formal way, we first use a brute force search to determine the variables in the first clause. Then, starting from the second clause, when a variable appears the first time, we assign it a value determined by comparison. Suppose that, after assigning a value to the variable, there are a set X of clauses where the values of all three literals have been determined. Let a be the number of satisfied clauses in X when the new variable is assigned True (T) while b False (F). If $a > b$, we assign True (T) to the new variable and False (F) otherwise.

This algorithm is not good because it does not guarantee a theoretical bound for the approximation. In fact, the algorithm can produce very bad results. It is possible that, the first k clauses are not satisfied in an optimal solution and yet most of them are satisfied in the greedy algorithm.

C. A Second Greedy Algorithm

Algorithms to the normal 3-SAT problem inspire other greedy algorithms for the NAE-3-SAT problem. One of the algorithms for 3-SAT is that, if most of the literals of a variable appear in the positive form, assign it to be True (T), and assign it to be False (F) otherwise.

Unfortunately, the counterpart for NAE-3-SAT does no better than the naive randomized algorithm. In the normal 3-SAT problem, the more literals with a value of True (T), the better the results can potentially be, which gives an advantage to the deterministic algorithm over the randomized one. Nevertheless, the property does not hold true for NAE-3-SAT due to its symmetry nature. (T, T, T) is as bad as (F, F, F), both leading to an unsatisfied clause. If we flip all the values of the variables, the objective value of the problem remains the same as that before the flip.

D. Dynamic Programming Approach

A recent paper[2] adopts a dynamic programming approach to the general MAX SAT problem by decomposing clauses into a tree structure. It shows that, given a problem instance and its corresponding tree paths, the problem can be solved in polynomial time $\mathrm{O}(w^3 m(m+n))$ where w is projection-satisfiable-width, a structural parameter of the tree.

E. PTAS

A possible polynomial-time approximation scheme is to guess the variables that have appeared in at least $\frac{1}{\epsilon}$ statements in an optimal solution.

Since each variable appears in at least $\frac{1}{\epsilon}$ clauses, and that we have 3ϵ variables in the brute force search at most, the guessing process runs in $\mathrm{O}(2^{3\epsilon})$ time.

After the guess, the algorithm can use the randomized algorithm or the two greedy algorithms mentioned above to finish the rest of the assignment.

We cannot prove an improved expected ratio. At a worst case, the ratio would still be $\frac{3}{4}$ when no variables appear in that many clauses. In practice, the algorithm produces better results by guessing the correct set of variables in an optimal solution.

F. LP from MAX-CSP Reduction (Devised)

In this algorithm, we first reduce a given NAE-3-SAT instance to its corresponding MAX-CSP instance, and then encode it in the LP form for a solution. We end up with a $\frac{1}{4}$-approximation algorithm, which is not an impressive result. Nevertheless, we are able to utilize some of the techniques learned in class to make it work.

1. Conversion from MAX NAE-SAT to MAX-CSP

Given an instance of MAX NAE-SAT, we can create a MAX-CSP problem in the following way. The set of variables X in CSP consists of all the variables appeared in NAE-SAT. For the set D of variable domains, each variable x_i in $X = \{x_1, ..., x_n\}$ is either 1 for True (T) or -1 otherwise. The set of constraints C is derived from combination of literals in each clause. The sum of x_i of the three literals should fall between -1 and 1.

An solution in MAX-CSP is allowed to violate some of the constraints. The resulted reduction still represents an instance of the original maximization problem.

2. LP for solving MAX-CSP

The general form of the LP in this section is based on the work of Serna, Trevisany, and Xhafa[3]. We use a

variable z_j to denote whether a constraint C_j is satisfied. We set $z_j = 1$ when C_j is satisfied and $z_j = 0$ otherwise. We also define a indicator variable $t_{i,v}$ where $v \in \{-1, 1\}$ such that $t_{i,v} = 1$ when $x_i = v$, and $t_{i,v} = 0$ otherwise. The assignment of $t_{i,v}$ in the relaxed form of LP leads to the assignment of x_i in the LP rounding. The relaxed LP for MAX-CSP is

$$max \sum_j z_j$$

under the constraints

$$z_j = 1 \text{ when } \sum_{x_i \in C_j} x_i \in [-1, 1],$$

$$\sum_{v \in [-1,1]} t_{i,v} = 1, \forall i \in n$$

$$t_{i,v} \in [0, 1], \forall i \in n.$$

3. Rounding and Expected Ratio

After obtaining a fractional solution to the above LP (z, t), we adopt a randomized rounding. Let us denote k as the arity of each clause. In our case of NAE-3-SAT, because there are three variables in every clause, $k = 3$. Because each variable has two possible values, the MAX-CSP problem has a domain of 2.

Consider any random assignment. For $v \in \{0, 1\}$ and any integer $i \in [0, n]$, we have

$$Pr[x_i = v] = \frac{k-1}{2k} + \frac{t_{i,v}}{k} = \frac{1}{3} + \frac{t_{i,v}}{3}.$$

The assignment has an average cost of at least

$$(\frac{1}{2})^{k-1} \sum_j z_j = \frac{1}{4} \sum_j z_j.$$

We prove the conclusion by showing that any constraint C_j is satisfied with probability of at least $\frac{1}{2^{k-1}} * z_j$, which then follows the linearity of expectation. If $[x_i = v]$ occurs in a satisfied constraint C_j, we have at least one v such that $t_{i,v} = 1$ after the rounding. Thus,

$$z_j \leq t_{i,v}$$

and

$$Pr[C_j \text{ is satisfied}] \leq (\frac{k-1}{2k} + \frac{t_{i,v}}{k} z_j)^k \leq (\frac{1}{2})^{k-1} z_j.$$

This is a tight bound, which leads to an integrality gap of $2^{k-1} = 4$.

Plugging $k = 3$ into the equation $2^k = k^{O(1)}$, we obtain the approximation ratio

$$(\frac{1}{2})^{k-1} - O(1) = \frac{1}{4} - O(1).$$

4. Discussion

Although the ratio is worse than the naive randomized algorithm, the LP algorithm is still valuable in that it is a new approach to the problem. It can be a potential direction of future research.

G. LP from Strict-CSP Reduction (Devised)

Inspired by Amit Kumar's paper[5], we propose another approach in which we convert a NAE-3SAT instance to a Strict-CSP instance. We then solve its corresponding LP for a solution to the original problem.

1. Conversion from NAE-3-SAT to Strict-CSP

Unlike that of MAX-CSP, a solution to Strict-CSP must satisfy all constraints in C. Based on the means proposed by Walsh[4], we use dual encoding to convert a given instance of NAE-3-SAT to that of the Strict-CSP.

We introduce a dual variable D_i for each clause in the given NAE-3-SAT instance. The domain of D_i is a set of combinations of truth values, each of which satisfies the clause c_i. For instance, the domain D_i for the clause (x_1, x_2, x_3) is

$$\{(T, T, F), (T, F, F), (F, F, T), (T, F, T), (F, T, F), (F, T, T)\}.$$

For each variable, all literals associated with the dual variable must be consistent across clauses. For instance, for the dual variable D_1 for the clause (x_1, x_2, x_3) and the dual variable D_2 for the clause $(x_5, x_4, \neg x_3)$, the third literal in D_1 must be the negative form of that in D_2. We should keep in mind of this when converting from NAE-3-SAT to Strict-CSP.

The number of constraints is bounded by the number of occurrences of the literals, and thus by O(m) in terms of the number of clauses.

2. LP for solving Strict-CSP

Amit Kumar's work[5] has encouraged us to explore more about using LP to solve Strict-CSP.

The convex hull of a set P of points in a Euclidean plane is the smallest convex set that contains P. Let us use ConvexHull(C_e) to denote the set of assignments $\lambda \in \{-1, 1\}$ that satisfies the constraint C_e.

The LP for the reduced CSP is

$$max \sum_{i \in [1,n]} w_i$$

under the constraints

$$(x_1, x_2...) \in \text{ConvexHull}(C_e), \forall e \in [1, m],$$

$$x_i \in [-1, 1], \forall i \in [1, n],$$

$$w_i = 1 \text{ when } (x_i, x_j, x_k) \text{ assigned to clause i} \in D_i,$$

$$w_i = 0 \text{ otherwise.}$$

3. Rounding and Expected Ratio

The rounding algorithm is again based on the paper by Kumar[5]. Let us denote p as the fractional solution of the above LP. Given a constant $x > 0$, we set $p = 1/x$. Given a CSP instance $I = (C, D, \{C_e\}_{e \in C}, \{w_i\}_{i \in D})$, we can construct a solution p^x. For every $z \in 0, 1$, we construct an integral solution S by setting $S_u = z_j$ if $p^x_u = j_x$. Then, we return the solution S with the greatest objective value of all feasible solutions in $\{S_z | z \in \{0, 1\}^m\}$.

The expected approximation ratio after the rounding is $\frac{1}{2-\epsilon}$, according to Kumar[5].

4. Discussion

The algorithm gives an approximation ratio of $\frac{1}{2-\epsilon}$ with the number of constraints bounded by number of the clauses. While the algorithm still performs worse than the naive randomized algorithm, it is a potential direction of future research.

H. The SDP Approach (Researched)

According to a paper by Uri Zwick[6], there is a Semidefinite Programming Approach (SDP) of solving NAE-3-SAT.

We adopt the notation $x_{n+1} = x_i$ for $1 \leq i \leq n$, and use $w_{i,j,k} \geq 0$ to represent the weight of the clause (x_i, x_j, x_k) where $1 \leq i, j, k \leq 2k$. The solution is encoded as v_i, a set of unit vectors to a SDP relaxation. Then, the SDP is defined as follows.

$$max \sum_{i,j,k} w_{i,j,k} \frac{3 - v_i v_j - v_i v_k - v_j v_k}{4}$$

under the constraints

$$v_i v_i = 1, \forall i \in [1, n],$$

$$v_i v_{n+1} = -1, \forall i \in [1, n],$$
$$v_i v_j + v_i v_k + v_j v_k \geq -1, \forall i, j, k \in [1, 2n].$$

According to the paper[6], we can round the SDP solution using a random hyperplane. We denote $\theta_{ij} = arccos(v_i v_j)$ as the angle between vector v_i and v_j. The algorithm can possibly reach an approximation ratio of 0.87856 when for every clause (x_i, x_j, x_k) where $w_{ijk} > 0$, we have

$$\theta_{ij} = \theta_{ik} = \theta_0 \approx 2.331122$$

and

$$\theta_{jk} = 0,$$

after an effective re-arrangement of the indices.

When the approximation algorithm solves the SDP relaxation, we obtain the vectors $v_1, ...v_n$. If we rotate them outwards by a degree of γ and then round them using a random hyperplane, the algorithm yields an approximation ratio of β.

Solving the following set of equations,

$$cos^4\gamma(cos^2\theta + \frac{4}{\beta^2\pi^2}) = 1,$$

$$\frac{2arccos(cos^2\gamma cos\theta) + 2arccos(cos^2\gamma)}{2\pi} = \frac{1 - cos\theta}{2}\beta,$$

$$\frac{3}{2\pi} arccos(-\frac{1}{3}cos^2\gamma) = \beta,$$

we gain the solution $\theta \approx 2.408874$, $\gamma \approx 0.145455$ and $\beta \approx 0.908718$.

The approximation ratio $\beta \approx 0.908718$ is achieved with vectors $v_1, ..., v_n$ when for every clause $(x_i, x_j, x_k$) where $w_{ijk} > 0$, it is the case that $\{\theta_{ij}, \theta_{ik}, \theta jk\} = \{\theta, \theta, 0\}$ where $\theta = 2.408874$, or $\{\theta_{ij}, \theta_{ik}, \theta jk\} = \{arccos(-\frac{1}{3}), arccos(-\frac{1}{3}), arccos(-\frac{1}{3})\}$.

III. REFERENCE

[1] Adi Avidor, Ido Berkovitch, Uri Zwick. Improved Approximation Algorithms for MAX NAE-SAT and MAX SAT.

[2] Sigve Hortemo, Sther, Jan Arne Telle, Martin Vatshelle. Solving SAT and MaxSAT by Dynamic Programming.

[3] Maria Serna, Luca Trevisany and Fatos Xhafa. The (Parallel) Approximability of Non-Boolean Satisfability Problems and Restricted Integer Programming.

[4] Toby Walsh. SAT v CSP.

[5] Amit Kumar, Rajsekar Manokaran, Madhur Tulsiani, Nisheeth K. Vishnoi. On LP-based Approximability for Strict CSPs.

[6] Uri Zwick. Outward Rotations: a Tool for Rounding Solutions of Semidefinite Programming Relaxations, with Applications to MAX CUT and Other Problems.