Approximating Minimum Unsatisfiability of Linear Equations

Piotr Berman * Marek Karpinski †

Abstract

We consider the following optimization problem: given a system of m linear equations in n variables over a certain field, a feasible solution is any assignment of values to the variables, and the minimized objective function is the number of equations that are not satisfied. For the case of the finite field GF[2], this problem is also known as the Nearest Codeword problem. In this note we show that for any constant c there exists a randomized polynomial time algorithm that approximates the above problem, called the *Minimum Unsatisfiability of Linear Equations* (MIN-UNSATISFY for short), with $n/(c \log n)$ approximation ratio. Our results hold for any field in which systems of linear equations can be solved in polynomial time.

*Dept. of Computer Science, University of Bonn, 53117 Bonn, visiting from Pensylvania State University, University Park, PA 16802. Partially supported by DFG grant Bo 56/157-1 and NSF grant CCR-9700053, Email `berman@cse.psu.edu`

†Dept. of Computer Science, University of Bonn, 53117 Bonn. Supported in part by DFG grants KA 673/4-1 and Bo 56/157-1, DIMACS, and IST grant 14036 (RAND-APX), Email `marek@cs.uni-bonn.de`

1 Introduction

The lower approximation bounds for the Min-Unsatisfy problem over finite fields and $\mathbf{Q}$ were studied intensively in a number of papers, cf., e.g. [ABSS93], [KST97], [DKS98], [DKRS00] and [BFK00]. The papers [DKS98] and [DKRS00] yield an approximation lower bound of $n^{\Omega(1)/\log\log n}$ for finite fields. However, no good approximation algorithms other that of order n were known up to now for any field. Kannan [K01] has designed a polynomial time approximation algorithm for Min-Unsatisfy over $\mathbf{Q}$ within a factor of $n+1$ using Helly's theorem.

In this paper we present the first sublinear approximation ratio algorithm for Min-Unsatisfy problem working for any field in which systems of linear equations can be solved in polynomial time.

Our method depends on incremental randomized selection of the equations from the input system with the new choices being linearly independent from the previous ones. Once the set of selected equations forms a base of the input system, it determines a solution. We run such a procedure a number of times and select the best solution. Surprisingly, a polynomial number of runs of this procedure suffices to find a *good* approximate solution.

2 Preliminaries

We consider a certain fixed field K, and assume that we can solve a system of n linear equations in n variables over K in polynomial time.

An instance of Min-Unsatisfy is a set E of equations in n variables and the objective is to find a solution that minimizes the number of unsatisfied equations in E. Below we give some notations and definitions used in formulation and analysis of our algorithms.

- An equation $ax = b$ in n variables is represented by a pair (a, b) where $a \in K^n$ and $b \in K$.
- For a set of equations S and an $x \in K^n$, $\textsc{Sat}(S, x) = \{(a, b) \in S : ax = b\}$, $\textsc{UnSat}(S, x) = S - \textsc{Sat}(S, x)$ and $u(S, x) = |\textsc{UnSat}(S, x)|$.
- E denotes a set of equations that forms the Min-Unsatisfy instance we consider, $u(E, x)$ is the objective function and x^* is an optimal solution.

- A set of equations S is independent if the set of vectors $V = \{a : (a,b) \in S \text{ for some } b\}$ is linearly independent and $|V| = |S|$.
- A maximal independent subset of S is a *base* of S.

In this note we use several times the following lemma.

Lemma 1 *A base of* $\textsc{Sat}(E, x^*)$ *is also a base of* E.

Proof. Let B be a base of $\textsc{Sat}(E, x^*)$. Suppose that B is not a base of E. Because B is an independent set, we can find a base C of E such that $C \supset B$. Let x be the solution of system of equations C. Clearly, $\textsc{UnSat}(B, x) = \emptyset$; moreover, $\textsc{UnSat}(\textsc{Sat}(E, x^*), x) = \emptyset$ because every equation in $\textsc{Sat}(E, x^*)$ is a linear combination of equations of B. We got a contradiction, because $\textsc{UnSat}(C, x) = \emptyset$ and thus $u(E, x) \leq u(E, x^*) - |C - B| < u(E, x^*)$. ❑

3 A Randomized Approximation Algorithm

We first discuss the main subroutine GUESS(E) of our algorithm that uses a *good extension* function $\text{G}\textsc{Ext}(B)$. This function returns the set of *good extensions* of independent set B, *i.e.* $\text{G}\textsc{Ext}(B) = \{e \in E - B : B \cup \{e\} \text{ is independent}\}$. We note that a membership of an equation to $\text{G}\textsc{Ext}(B)$ can be verified by solving a linear system.

```
Guess(E)
  B ← ∅
  while GExt(B) ≠ ∅
    B ← B ∪ {random element of GExt(B)}
  return a solution of system of equations B
```

Lemma 2 *If x is the output of* **Guess**(E)*, then $u(E, x) \leq \frac{n}{d} u(E, x^*)$ with probability at least e^{-d} for any $d > 0$.*

Proof. In consecutive iterations of **Guess** set B changes from $\emptyset$ to $B_1 = \{b_1\}$, then to $B_2 = \{b_1, b_2\}$ etc. We say that b_i is a *good selection* if either $b_i \in \textsc{Sat}(E, x^*)$ or we have

$$|\text{G}\textsc{Ext}(B_{i-1})| \leq \frac{n-d}{d} u(E, x^*). \qquad (*)$$

If all selections are good, then $u(E,x) \leq \frac{n}{d} u(E,x^*)$. Indeed, if all selections belong to $\textsc{Sat}(E,x^*)$, then $\textsc{UnSat}(E,x) = \textsc{UnSat}(E,x^*)$ and thus $u(E,x) = u(E,x^*)$. Otherwise let b_i be the first selection such that $(*)$ holds.

Because $B_{i-1} \subset \textsc{Sat}(E,x^*)$, if all equations in B_{i-1} are satisfied and $B_{i-1} \cup \{e\}$ is dependent, then e is satisfied if and only if $e \in \textsc{Sat}(E,x^*)$. For this reason

$$\begin{aligned} \textsc{UnSat}(E,x) - \textrm{G}\textsc{Ext}(B_{i-1}) &\subset \textsc{UnSat}(E,x^*) \text{ and} \\ \textsc{UnSat}(E,x) \cap \textrm{G}\textsc{Ext}(B_{i-1}) &\subset \textrm{G}\textsc{Ext}(B_{i-1}), \end{aligned}$$

and thus $u(E,x) \leq \frac{d}{d} u(E,x^*) + \frac{n-d}{d} u(E,x^*) = \frac{n}{d} u(E,x^*)$.

Now it remains to show that the probability that all selections are good is at least e^{-d}. First we will estimate from below the conditional probability that selection b_i is good if all previous selections were good. If $|\textrm{G}\textsc{Ext}(B_{i-1})| \leq \frac{n-d}{d} u(E,x^*)$, then b_i is good with probability 1. Otherwise, we choose b_i in $|\textrm{G}\textsc{Ext}(B_{i-1})|$ many possible ways, of which at most $u(E,x^*)$ many do not belong to $\textsc{Sat}(E,x^*)$, thus the probability that b_i is good is at least

$$1 - \frac{u(E,x^*)}{|\textrm{G}\textsc{Ext}(B_{i-1})|} \quad \geq \quad 1 - \frac{d}{n-d}.$$

Probability that all selections are good is the product of the conditional probabilities estimated above, thus it can be estimated from below as

$$\left(1 - \frac{d}{n-d}\right)^n \quad \approx \quad \left(\left(1 - \frac{d}{n-d}\right)^{(n-d)/d}\right)^d \quad \approx \quad e^{-d}.$$

Actually, this estimate is a bit inaccurate, but our application of this inequality does not rely too much on precision, *e.g.*, the estimate $\frac{1}{2}e^{-d}$ is good enough; importantly, we consider only very small values of d as compared with n. ❑

We are going to formulate now our approximation algorithm.

```
Rand_App(E, N)
    x_best ← 0
    repeat N times
        x ← Guess(E)
        if u(E, x) < u(E, x_best)
```

$x_{\text{best}} \leftarrow x$
output x_{best}

The algorithm works by running N times the subroutine **Guess**(E) and then choosing the best solution. If a run of **Guess** delivers approximation ratio r with probability at least N^{-1}, then a run of **Rand_App** delivers ratio r with probability at least e^{-1}. If we choose $N = n^c$, then we need **Guess** to deliver ratio r with probability at least n^{-c}. Lemma 2 says that **Guess** obtains ratio n/d with probability e^{-d}. Thus we have $e^{-d} = n^{-c}$, or, equivalently, $d = c \ln n$.

Summarizing, for every c we obtain an algorithm with approximation ratio $n/(c \ln n)$. Note that if Lemma 2 is inaccurate by a factor of 2 (and it is much more accurate), we just need to double N. This proves the following theorem:

Theorem 1 *For every $c > 0$ there exists a randomized polynomial time approximation algorithm that approximates* MIN-UNSATISFY *within ratio $n/(c \ln n)$.*

For the sake of completeness we present in the next section a simple deterministic algorithm with n/c approximation ratio for any constant c.

4 A Deterministic Approximation Algorithm

We start with formulating the following lemma.

Lemma 3 *For every positive integer c there exists a polynomial time algorithm for the following subproblem:*
Input: *An equation system E and a base B of E;*
Output: *A solution x of E that minimizes $u(E, x)$ under restriction $u(B, x) < c$.*

Proof. We sketch the algorithm.

EasyCase(E, B, c)
$x_{\text{best}} \leftarrow \mathbf{0}$
for every $S \subset B$ such that $|S| < c$
for every $T \subset E - B$ such that $|T| = |S|$

```
        B' ← B − S ∪ T
        if B' is independent
            x ← a solution of system B'
            if u(E, x) < u(E, x_best)
                x_best ← x
    output x_best
```

It is easy to see that **EasyCase** runs in polynomial time, because the searching space of each of the **for** loops is polynomial in size (for a fixed c!). To see the correctness, note first that the outer **for** loop inspects every $\textsc{UnSat}(B,x)$ that is allowed under our restriction. Under assumptions that $\textsc{Sat}(B,x) = B - S$, $B - S$ is an independent subset of $\textsc{Sat}(E,x)$; thus $B - S$ is contained in some base B' of $\textsc{Sat}(E,x)$. By Lemma 1 we may assume that B' is a base of E, hence the inner **for** loop inspects all possible B'. Once we make our assumption concerning B', the value of x is determined.

Note that it is possible that B has fewer elements than n, and thus a system of equations formed by a base of E may have more than one solution. Therefore we introduce an assignment $x \leftarrow$ a solution of system B'. ❑

Theorem 2 *There exists a polynomial time algorithm that for any equation set E it returns a solution x such that $u(E,x) \leq \frac{n}{c} u(E,x^*)$.*

Proof. We describe the algorithm **Det_App** recursively.

```
Det_App(E, c)
    if E = ∅
        output 0
    B ← a base of E.
    x_1 ← EasyCase(E, B, c)
    x_2 ← Det_App(E − B, c)
    if u(E, x_1) ≤ u(E, x_2)
        output x_1
    else
        output x_2
```

To see that **Det_App** runs in polynomial time for a fixed c, note that it invokes **EasyCase** only once, and then it makes a recursive call for a smaller problem instance.

The approximation property of **Det_App** can be proven by induction. The claim is trivial if $E = \emptyset$. If $u(B, x^*) < c$, then x_1 found by **EasyCase** is optimal, hence $u(E, x_1) \leq u(E, x^*)$. Otherwise we have, by induction:

$$\begin{aligned} cu(E, x_2) = c(u(B, x_2) + u(E - B, x_2)) \leq c(n + u(E - B, x_2)) &= \\ cn + cu(E - B, x_2) \leq cn + nu(E - B, x^*) &= \\ n(c + u(E - B, x^*)) \leq n(u(B, x^*) + u(E - B, x^*)) &= \\ nu(E, x^*) & \end{aligned}$$

❑

Acknowledgments

We thank Ravi Kannan and Madhu Sudan for stimulating discussions.

References

[ABSS93] S. Arora, L. Babai, J. Stern and Z. Sweedyk, *The hardness of approximate optima in lattice, codes, and systems of linear equations*, Proc. of 34th IEEE FOCS, 1993, 724-733.

[BFK00] C. Bazgan, W. Fernandez de la Vega and M. Karpinski, *Approximability of dense instances of NEAREST CODEWORD problem,* ECCC Tech. Rep. TR00-091, 2000.

[DKS98] I. Dinur, G. Kindler and S. Safra, *Approximating CVP to within almost polynomial factors is NP-hard*, Proc. of 39th IEEE FOCS, 1998, 99-109.

[DKRS00] I. Dinur, G. Kindler, R. Raz and S. Safra, *An improved lower bound for approximating CVP*, 2000, submitted.

[K01] R. Kannan, personal communication (see also [ABSS93], p. 725).

[KST97] S. Khanna, M. Sudan and L. Trevisan, *Constraint Satisfaction: the approximability of minimization problems*, Proc. of 12th IEEE Computational Complexity, 1997, 282-296.