Final thesis

Tight Approximability Results for the Maximum Solution Equation Problem over Abelian Groups

by

Fredrik Kuivinen

LITH-IDA-EX--05/049--SE

2005-06-01

 www.FirstRanker.com

www.FirstRanker.com

www.FirstRanker.com

Final thesis

Tight Approximability Results for the Maximum Solution Equation Problem over Abelian Groups

by **Fredrik Kuivinen**

LITH-IDA-EX--05/049--SE

Supervisor : **Gustav Nordh**
Department of Computer and Information Science at Linköpings universitet

Examiner : **Peter Jonsson**
Department of Computer and Information Science at Linköpings universitet

Abstract

In the maximum solution equation problem a collection of equations are given over some algebraic structure. The objective is to find an assignment to the variables in the equations such that all equations are satisfied and the sum of the variables is maximised. We give tight approximability results for the maximum solution equation problem when the equations are given over finite abelian groups. We also prove that the weighted and unweighted versions of this problem have asymptotically equal approximability thresholds. Furthermore, we show that the problem is equally hard to solve as the general problem even if each equation is restricted to contain at most three variables and solvable in polynomial time if the equations are restricted to contain at most two variables each. All of our results also hold for the generalised version of maximum solution equation where the elements of the group are mapped arbitrarily to non-negative integers in the objective function.

Keywords : systems of equations, finite groups, NP-hardness, approximation

Acknowledgements

I would like to thank my supervisor Gustav Nordh and my examiner Peter Jonsson for all the help and useful discussions during the work with this thesis. I would also like to thank Peter Hackman, for ideas on how the solutions to linear system of equations over a ring is structured, and Johan Håstad, for quickly answering my questions regarding one of his papers.

Contents

Chapter 1

Introduction

Problems related to solving equations over various algebraic structures have been studied extensively during a large time frame. The most fundamental problem is, perhaps, EQN which is the problem of: given an equation, does it have a solution? That is, is it possible to assign values to the variables in the equation such that the equation is satisfied? Goldmann and Russell [9] studied this problem for finite groups. They showed that EQN is **NP**-complete for all non-solvable groups and solvable in polynomial time for nilpotent groups.

A problem related to EQN is EQN*. In EQN* a collection of equations are given and the question is whether or not there exists an assignment to the variables such that all equations are satisfied. For finite groups Goldmann and Russell [9] have shown that this problem is solvable in polynomial time if the group is abelian and **NP**-complete otherwise. Moore et al. [16] have studied this problem when the equations are given over finite monoids. The same problem have been studied for semigroups [14, 22] and even universal algebras [15].

Another problem is the following: given a over-determined system of equations, satisfy as many equations as possible simultaneously. This problem have been studied with respect to approximability by Håstad [10]. He proved optimal inapproximability bounds for the case where the equations are given over a finite abelian group. Håstad's result has later on been generalised by Engebretsen et al. [6] to cover finite non-abelian groups as well. Those results uses the PCP theorem [1] which has been used to prove a number of inapproximability results. Other problems that have been studied which are related to this area is #EQN* (counting the number of solutions to a system of equations) [18] and EQUIV-EQN* and ISO-EQN* (deciding whether two systems of equations are equivalent or isomorphic, respectively) [17].

In this paper we study the following problem: given a system of equations over a finite abelian group, find the best solution. With "best solution" we mean a solution (an assignment to the variables that satisfies all equations) that maximises the sum of the variables. We call this problem MAXIMUM SOLUTION EQUATION (here after denoted by MAX SOL EQN).

A problem that is similar to MAX SOL EQN is NEAREST CODEWORD.[1] In this problem we are given a matrix A and a vector $\boldsymbol{b}$. The objective is to find a vector $\boldsymbol{x}$ such that the hamming weight (the number of positions in the vector that differs from 0) of $A\boldsymbol{x}-\boldsymbol{b}$ is minimised. The decision version of a restricted variant[2] of this problem was proved to be **NP**-complete by Bruck and Noar [4]. Later on Feige and Micciancio [8] proved inapproximability results for the same restricted problem. Arora et al. [2] proved that NEAREST CODEWORD over $GF(2)$ is not approximable within $2^{\log^{1-\epsilon} n}$ for any $\epsilon > 0$ unless $\mathbf{NP} \subseteq DTIME(n^{poly(\log n)})$. NEAREST CODEWORD is interesting because it has practical applications in the field of error correcting codes.

MAX SOL EQN is parametrised on the group we are working with and a map from the elements of the group to non-negative integers. The map is used in the objective function to compute the measure of a solution. Our main result give tight approximability results for MAX SOL EQN for every finite abelian group and every map from the elements of the group to non-negative integers. That is, we prove that for every finite abelian group and every map from group elements to non-negative integers there is a constant, α, such that MAX SOL EQN is approximable within α but not approximable within $\alpha - \epsilon$ in polynomial time for any $\epsilon > 0$ unless

[1] This problem is sometimes called MLD for MAXIMUM LIKELIHOOD DECODING.

[2] The problem we are referring to is NEAREST CODEWORD with preprocessing. See [4] for a definition.

$\mathbf{P} = \mathbf{NP}$. We also show that the weighted and the unweighted versions of this problem are asymptotically equally hard to approximate. All our hardness results hold even if the instances are restricted to have at most three variables per equation. We also prove that this is tight since with two variables per equation the problems are solvable to optimum in polynomial time.

Our work may be seen as a generalisation of Khanna et al.'s [13] work on the problem MAX ONES($\mathcal{F}$) in the sense that we study larger domains. However, their work is not restricted to equations over finite groups which the results in this paper are. Nevertheless, they give a 2-approximation algorithm for MAX ONES($\mathcal{F}$) when $\mathcal{F}$ is affine. We prove that, unless $\mathbf{P} = \mathbf{NP}$, this is tight. (MAX ONES($\mathcal{F}$) when $\mathcal{F}$ is an affine constraint family is equivalent to a specific version of MAX SOL EQN.)

The structure of this thesis is as follows, in the first (this) chapter we give an introduction to the problem. We begin with a background of the area and present some previous results. In the second section we give a general introduction to the theory of computational complexity and its relation to optimisation and approximation problems. We then go on and state some preliminaries where we define our notation. Our problem, called MAX SOL EQN, is then formally defined together with the results that we have obtained.

In Chapter 2 we prove our inapproximability results for MAX SOL EQN. That is, we prove that if $\mathbf{P} \neq \mathbf{NP}$ then there do not exist any polynomial time approximation algorithms for MAX SOL EQN with a performance ratio strictly less than some α.

We also want to bound the approximability from above. That is, we want to say "MAX SOL EQN is approximable within α", for some constant α. The easiest way to do this is to construct an approximation algorithm for MAX SOL EQN and then prove that the performance ratio for the algorithm is α. This is what we do in Chapter 3.

The results in Chapter 2 say things about one version (the weighted version) of MAX SOL EQN and Chapter 3 say things about another version (the unweighted version) of MAX SOL EQN. In Chapter 4 we prove that the weights do not really matter, the approximability threshold for the weighted and unweighted versions of MAX SOL EQN are asymptotically equal.

Finally, in Chapter 5, we prove our main results. The proof combines the results from the previous chapters. The last section in the final chapter contains a short discussion about possible future work related to the work in this thesis.

1.1 A Short Introduction to Complexity Theory

This section contains a short introduction to complexity theory and its relation to optimisation and approximation problems. Readers familiar with those concepts may want to skip this part of the thesis. For a more detailed presentation see, e.g., [3].

In this thesis we are going to study a specific optimisation problem. In an optimisation problem we are, in general, given a set of variables, a set of constraints over those variables and an objective function. The goal is to assign values from some domain to the variables such that the constraints are fulfilled and the objective function is either maximised or minimised. A well known optimisation problem is the linear programming problem (here after called LP). In the LP problem we are given a set of linear inequalities over some variables. The goal is to find an assignment to the variables such that the inequalities are satisfied and a given linear combination of the variables is maximised (or minimised). The LP problem is well studied and can, for example, be solved with the Simplex algorithm. The set of values that can be assigned to a variable in an optimisation problem is called the domain of the problem. In the LP case the domain is the set of real numbers.

Another example of a optimisation problem is MAX 2SAT. In MAX 2SAT we are given a set of disjunctions over a set of variables. Each disjunction contains exactly two literals. An example of a possible disjunction is thus $\neg x \vee y$. The goal is to assign truth values (either true or false) to the variables such that the maximum number of disjunctions are satisfied. In this case the domain consists of two values, true and false.

In the LP case it turns out to be possible to find an optimal solution fast (we will define what we mean with "fast" soon). For some other problems, for example, MAX 2SAT it is probably not possible to find optimal solutions fast. When it is not possible to find an optimal solution fast it is natural to ask if we can find a "good" solution fast, for some definition of "good". This is formalised with approximation algorithms. An approximation algorithm is a fast algorithm that produces a solution to an optimisation problem and gives some sort of guarantee about the quality of the solution. The guarantee can, for example be, (for an maximisation problem) "the measure of a solution returned by the approximation algorithm will never be less than 50% of the measure of the optimal solution". The value 50% is called the *performance ratio* of the algorithm. In the MAX 2SAT case there exists an approximation algorithm that return solutions who's measure is at least 93% of the measure of the optimal solution. [7]

Example 1.1
Maximise $3x_1 + 5x_2$ subject to the following constraints

$$x_1 + 2x_2 \leq 8$$
$$-x_1 + x_2 \geq -5$$

In this example we have an *instance* of the LP problem. In this instance the objective is to maximise $3x_1 + 5x_2$ subject to the constraints in the example. The optimal solution to this instance is $x_1 = 6$, $x_2 = 1$ which gives us the measure $3 \cdot 6 + 5 \cdot 1 = 23$. An approximation algorithm for the LP problem with the guarantee that it will return solutions that have a measure that is at least 50% of the optimal measure could, for example, return the solution $x_1 = 0$, $x_2 = 4$ to the instance in this example. This solution has the measure 20 and is larger than the required $0.5 \cdot 23$. It could not, however, return $x_1 = 1$, $x_2 = 1$ since this solution has the measure 8 which is smaller than $0.5 \cdot 23$. Neither could it return $x_1 = 0$, $x_2 = 10$ because those values do not satisfy the constraints, they are not a *feasible solution* to the instance.

In complexity theory it is often said that an algorithm is "fast" or "practical" if the running time of the algorithm is bounded by a polynomial in the size of the input.

In this thesis we will study a specific optimisation problem with respect to approximability. We will give an approximation algorithm for our problem, and we will prove that it (probably) do not exists any approximation algorithms that are better than our approximation algorithm. Our inapproximability results are of the following form, "MAX SOL EQN is not approximable within $\alpha - \epsilon$ for any $\epsilon > 0$, unless $\mathbf{P} = \mathbf{NP}$", where α is some well defined number. If, for example $\alpha = 2$, then the meaning of a theorem of the above mentioned form is that it is unlikely that there exists an approximation algorithm that can generate solutions to MAX SOL EQN such that the solutions always are better than 50% of the optimum value. Results of this form bound the approximability of a certain problem from below, they say that it is (probably) not possible to approximate MAX SOL EQN within a constant smaller than α.

One of the most interesting and most well studied open questions in complexity theory is whether or not $\mathbf{P} = \mathbf{NP}$. It is widely believed that $\mathbf{P} \neq \mathbf{NP}$, but no one has managed to prove that. Informally one could state this question in the following way: For all problems where it is possible to verify a solution fast, is it equally hard to find solutions as it is to verify a given solution? $\mathbf{P}$ contains all problems that are solvable in polynomial time, i.e., fast and $\mathbf{NP}$ contains all problems where it is possible to verify a given solution in polynomial time. Intuitively it seems that it is easier to verify a solution than to find a solution. Consider, for example, the problem of finding a proper colouring of a map, using only three colours. A colouring of a map is considered proper if two adjacent countries have different colours. A trivial way to get a proper colouring is to assign a unique colour to each country. But if we are restricted to use only three colours, say red, green and blue is there a fast way to decide if it is possible to find a proper colouring to a given map using only those colours? It is easy to verify a solution to the map colouring problem, just check if every country in the map have a different colour compared to its neighbours. However, it seems to be very hard to come up with a *fast* way to find out if there exists a proper colouring to a given map. One way is to try every possible combination of colour assignments, however the running time of this method will grove exponentially with the number of countries in the map and it is therefore not considered fast. The map colouring problem is usually called GRAPH COLOURABILITY and has been proved to be **NP**-

complete by Stockmeyer [20]. This means that if someone comes up with a fast algorithm for the map colouring problem then we would have $\mathbf{P} = \mathbf{NP}$.

The "unless $\mathbf{P} = \mathbf{NP}$" part of our inapproximability results thus states that our inapproximability results only holds if $\mathbf{P} \neq \mathbf{NP}$. If it turns out that $\mathbf{P} = \mathbf{NP}$ then the results in this thesis would become useless, because for every problem studied here it would exist a fast algorithm which could find the optimum.

1.2 Preliminaries

We assume that the reader has some basic knowledge of complexity theory. We will nevertheless briefly state some fundamental definitions of optimisation problems and approximation in this section, see Section 1.1 for a brief introduction or [3] for a more detailed presentation of complexity theory. We also assume that the reader has some basic knowledge of group theory. More specifically the theory of abelian groups. For an introduction to abstract algebra the reader is referred to [12] and [21].

An *optimisation problem* has a set of admissible input data, called the *instances* of the problem. Each instance has a set of *feasible solutions*. The optimisation problem also has a function of two variables, an instance and a feasible solution, that associates an integer with each such pair. This function denotes the *measure* of the solution. The *goal* of an optimisation problem is to find a feasible solution that either maximises or minimises the measure for a given instance.

An **NPO** problem is an optimisation problem where instances and feasible solutions can be recognised in polynomial time, feasible solutions are polynomially bounded in the input size and the measure can be computed in polynomial time. We will only study **NPO** maximisation problems in this thesis.

We will denote the measure of our problems with $m(I, s)$, where I is an instance and s is a feasible solution. The optimum for an instance I of some problem (which problem we are talking about will be clear from the context) is designated by $\text{OPT}(I)$. We say that a maximisation problem Π is r-*approximable* if there exists a polynomial time algorithm A such that for every instance I of Π, $m(I, A(I)) \geq \text{OPT}(I)/r$. Equivalently, we say that the *performance ratio* of A is r if this inequality holds. The same terminology is used if there exists a randomised polynomial time algorithm where the expected value of the measure is greater than $\text{OPT}(I)/r$. That is, if there exists a randomised polynomial time algorithm A' such that for every instance I of Π, $\mathrm{E}[m(I, A'(I))] \geq \text{OPT}(I)/r$, then Π is said to be r-approximable.

In reductions we will work with two different problems simultaneously. The objects associated with the problem that the reduction is from will be denoted by symbols without $'$ and objects associated with the other problem will be denoted by symbols with $'$. Thus, for example, the measuring function of the problem that the reduction starts with will be denoted by $m(I, s)$ and the measuring function of the other problem will be denoted by $m'(I', s')$.

For a random variable X and a set S we use the notation $X \sim U(S)$ to denote that X is uniformly distributed over S. That is, $X \sim U(S)$ means that for every $x \in S$ we have $\Pr[X = x] = 1/|S|$.

We use the standard definition of $o(\cdot)$. That is, given two functions $f(n)$ and $g(n)$, we say that $f(n)$ is in $o(g(n))$ if $f(n)/g(n) \to 0$ as n tends to infinity. [23] Hence, we have in particular, that if $f(n)$ is in $o(1)$ then $f(n) \to 0$ as n tends to infinity.

For a finite abelian group $G = (D, +)$ we have

$$G \cong \mathbf{Z}_{p_1^{\alpha_1}} \times \cdots \times \mathbf{Z}_{p_n^{\alpha_n}}$$

for some integer n, primes $p_1, \ldots, p_n$ and integers $\alpha_1, \ldots, \alpha_n$. See e.g. Theorem 11.3 in [12]. In the subsequent parts of this thesis we will assume that, unless explicitly stated otherwise, the group G is defined as above. We will also identify the group with its domain, i.e., we will sometimes treat G as a set such that $G = D$. We see the elements in G as vectors of integers. Position number i in each such vector is an element of $\mathbf{Z}_{p_i^{\alpha_i}}$. For a group G we denote its identity element with 0_G. We will use addition as the group operator. Every group dealt with in this text is finite.

For a group G and a subgroup $G' \subseteq G$ of this group we denote the coset, C, of G' with representative $c \in G$ as $G' + c$. That is,

$$G' + c = C = \{x + c \mid x \in G'\}.$$

For a function $f : X \to \mathbf{N}$ and a set $S \subseteq X$ we use the notations $f_{\max}(S)$ and $f_{\text{sum}}(S)$ for the following quantities,

$$f_{\max}(S) = \max_{x \in S} f(x) \qquad f_{\text{sum}}(S) = \sum_{x \in S} f(x).$$

We will sometimes use $f_{\max}$ and f_{sum} as a shortening of $f_{\max}(X)$ and $f_{\text{sum}}(X)$, respectively. Those notations will only be used when they are well defined.

We use "mod" as a modulos operator. For an integer a and a positive integer b we define "a mod b" as follows,

$$c = a \bmod b \iff 0 \leq c < b \quad \text{and} \quad c \equiv a \pmod{b}.$$

Note that there is a large difference between "a mod b", which is defined to be the unique integer that lies between 0 and $b-1$ (inclusive) and is congruent to a modulo b, and "$a \equiv b \pmod{m}$" which states that a is congruent to b modulo m.

To describe our algorithms we use a simple pseudo code syntax. It is mostly self-documenting but we will make some comments of it here. We use $\leftarrow$ as the assignment operator. For a matrix A the expression $Rows(A)$ denotes the number of rows in A and $Cols(A)$ denotes the number of columns in A. For a set S the expression $Rand(S)$ is a random element from S, more precisely, for every $x \in S$ the value of the expression $Rand(S)$ is x with probability $1/|S|$.

1.3 Definitions and Results

We are going to study the following problem in this thesis.

Definition 1.1. Weighted Maximum Solution Equation(G, g) *where G is a group and $g : G \to \mathbf{N}$ is a function, is denoted by* W-Max Sol Eqn(G, g). *An instance of* W-Max Sol Eqn(G, g) *is defined to be a triple (V, E, w) where,*

- *V is a set of variables.*

- *E is a set of equations of the form $w_1 + \ldots + w_k = 0_G$, where each w_i is either a variable, an inverted variable or a group constant.*
- *w is a weight function $w : V \to \mathbf{N}$.*

The objective is to find an assignment $f : V \to G$ to the variables such that all equations are satisfied and the sum

$$\sum_{v \in V} w(v)g(f(v))$$

is maximised.

Note that the function g and the group G are not parts of the input. Thus, W-MAX SOL EQN(G, g) is a problem parameterised by G and g. We will also study the unweighted problem, MAX SOL EQN(G, g), which is equivalent to W-MAX SOL EQN(G, g) with the additional restriction that the weight function is equal to 1 for every variable in every instance. The collection of linear equations in an instance of W-MAX SOL EQN(G, g) can also be represented in the standard way as an integer-valued matrix, A, and a vector of group elements, $\boldsymbol{b}$. If the variables are called $x_1, \ldots, x_m$ we can then, with $\boldsymbol{x} = (x_1, \ldots, x_m)^T$, use $A\boldsymbol{x} = \boldsymbol{b}$ as an equivalent form of the sets V and E in the definition above.

Due to Goldmann and Russell's result [9] that solving systems of equations over non-abelian groups is **NP**-hard, it is **NP**-hard to find feasible solutions to MAX SOL EQN(H, g) if H is non-abelian. It is therefore sufficient to only study MAX SOL EQN(H, g) where H is abelian.

To describe our results we need the following concept.

Definition 1.8 (Coset-Validity). *Let G be an abelian group. A non-empty set $B \subseteq G$ is* coset-valid *with respect to G if there exists a matrix A, a vector $\boldsymbol{b}$, a vector of variables $\boldsymbol{x} = (x_1, \ldots, x_m)^T$ such that the system of equations $A\boldsymbol{x} = \boldsymbol{b}$ restricts the values that x_1 can have such that those values form a subgroup, G', of G. Furthermore, there exists a group element $c \in G$ such that $B = G' + c$.*

That is, the set

$$G' = \{x_1 \mid A\boldsymbol{x} = \boldsymbol{b}\}$$

is a subgroup of G. Furthermore, there exists a group constant, $c \in G$, such that

$$B = \{c + x_1 \mid A\boldsymbol{x} = \boldsymbol{b}\}.$$

If those conditions are fulfilled then B is coset-valid with respect to G. Note that B is a coset of G' with representative c.

Given a group G there is always at least one set that is coset-valid with respect to G, namely G itself.

The main result of this thesis is the following theorem about the approximability of Max Sol Eqn(G, g).

Theorem 1.1 (Main). *For every finite abelian group G and every function $g : G \to \mathbf{N}$,* Max Sol Eqn(G, g) *is approximable within α where*

$$\alpha = \max \left\{ \frac{g_{\max}(B)}{g_{\mathrm{sum}}(B)} |B| \;\middle|\; B \text{ is coset-valid with respect to } G \right\}.$$

Furthermore, for every finite abelian group G and every non-constant function $g : G \to \mathbf{N}$ Max Sol Eqn(G, g) *is not approximable within $\alpha - \epsilon$ for any $\epsilon > 0$ unless $\mathbf{P} = \mathbf{NP}$.*

We will prove Theorem 1.1 in Chapter 5. Note that if g is a constant function then every feasible solution has the same measure and finding an optimum is solvable in polynomial time.

We will also prove that the approximability threshold for W-Max Sol Eqn(G, g) is asymptotically equal to the approximability threshold for Max Sol Eqn(G, g). That is, we will prove that W-Max Sol Eqn(G, g) is approximable within $\alpha + o(1)$ where the $o(\cdot)$-notation is with respect to the size of the instance. Furthermore, we will prove that W-Max Sol Eqn(G, g) is not approximable within $\alpha - \epsilon$ for any $\epsilon > 0$, unless $\mathbf{P} = \mathbf{NP}$.

Chapter 2

Inapproximability

In this chapter we are going to prove inapproximability results for Max Sol Eqn. We will begin with a section with some preliminaries, we use a few other problems in our inapproximability proofs which are presented in Section 2.1, we also introduce a special kind of reduction.

We will then go on and prove an inapproximability result for one of the new problems, namely Maximum Expression, which is defined in Section 2.1. To do this we use Håstad's [10] inapproximability results for Max-Ek-Lin-G, the latter problem is also defined in Section 2.1.

In Section 2.3 we use the inapproximability results of Maximum Expression in a gap-preserving reduction to prove an inapproximability bound for Max Sol Eqn(G, g). This bound turns out to be tight for some groups G and some functions $g : G \to \mathbf{N}$, but not for all such combinations. We will then use this result as a stepping stone to prove our final inapproximability result in Section 2.4. The proof of the final result relies on the observation that for some combinations of groups, G, and functions, g, it is possible to construct a linear system of equations that induce a subgroup of G which is, in a sense made clear below, hard to approximate.

This latter result is the main theorem of this chapter. It is formally stated as follows:

Theorem 2.1 (Main Inapproximability Theorem). *For every finite* *abelian group G and every non-constant function $g : G \to \mathbf{N}$ it is not possible to approximate* W-MAX SOL EQN(G, g) *within $\alpha - \epsilon$ where*

$$\alpha = \max \left\{ \frac{g_{\max}(B)}{g_{\text{sum}}(B)} |B| \;\middle|\; B \text{ is coset-valid with respect to } G \right\}$$

for any $\epsilon > 0$ unless $\mathbf{P} = \mathbf{NP}$.

This theorem turns out to be a tight inapproximability result for W-MAX SOL EQN(G, g). That is, the theorem states that it is not possible to approximate W-MAX SOL EQN(G, g) within $\alpha - \epsilon$ for some α and any $\epsilon > 0$ unless $\mathbf{P} = \mathbf{NP}$. We will, in Chapter 3 and Chapter 4 prove that there exists an α-approximation algorithm for W-MAX SOL EQN(G, g).

2.1 Preliminaries

We will prove our inapproximability results with a special kind of reduction, namely a gap-preserving reduction introduced by Arora in [1]. The definition is as follows.

Definition 2.1 (Gap-preserving reduction [1]). *Let Π and Π' be two maximisation problems and $\rho, \rho' > 1$. A* gap-preserving reduction with parameters c, ρ, c', ρ' *from Π to Π' is a polynomial time algorithm f. For each instance I of Π, f produces an instance $I' = f(I)$ of Π'. The optima of I and I', satisfy the following properties:*

- *if $\text{OPT}(I) \geq c$ then $\text{OPT}(I') \geq c'$, and*
- *if $\text{OPT}(I) \leq c/\rho$ then $\text{OPT}(I') \leq c'/\rho'$.*

Gap-preserving reductions are useful because if for every language in **NP** there is a polynomial time reduction to the maximisation problem Π such that YES instances are mapped to instances of Π of measure at least c and NO instances to instances of measure at most c/ρ, then a gap-preserving reduction from Π to Π' implies that finding ρ'-approximations to Π' is **NP**-hard. [1]

Definition 2.2. MAXIMUM EXPRESSION *over the abelian group G is denoted by* MAX EXPR(G). *An instance of* MAX EXPR(G) *is defined to be $I = (V, E)$ where,*

- *V is a set of variables, and*
- *$E = \{e_1, \ldots, e_m\}$ is a set of expressions. Each expression e_i is of the form $c_i + w_{i1} + w_{i2} + \ldots$ where $w_{i1}, w_{i2}, \ldots$ are either variables or inverted variables and c_i is a group constant.*

The objective is to find an assignment $f : V \to G$ and a group element $x \in G$ such that the maximum number of expressions in E evaluate to x when the variables in the expressions are assigned values according to f.

The inapproximability of the following problem is the starting point for our results in this chapter.

Definition 2.3 **(**MAX-Ek-LIN-G **[10]).** *An instance of* MAX-Ek-LIN-G *is defined to be (V, E) where*

- *V is a set of variables, and*
- *E is a set of linear equations over the group G with exactly k variables in each equation.*

The objective is to find an assignment $f : V \to G$ such that the maximum number of equations in E are satisfied.

The following theorem can be deduced from the proof of Theorem 5.9 in [10]. (In [10] the theorem is first proved for the case $k = 3$ there is then, on page 827, a hint on how this proof can be generalised to an arbitrary k. However, it appears that the proof which is suggested in [10] do not work. A slight modification of Theorem 5.9 in [10] do give the desired result, though. [11])

Theorem 2.2. *For every problem Π in* **NP** *there is a polynomial time reduction from instances I of Π to instances $I' = (V, E)$ of* MAX-Ek-LIN-G *such that*

- *if I is a* YES *instance then at least $(1-\delta)|E|$ equations can be satisfied, and*

- *if I is a* NO *instance then no assignment satisfies more than $|E|(1+\delta)/|G|$ equations*

where δ is an arbitrary constant such that $0 < \delta < 1$. Furthermore, no equation in E contains any variables in their inverted form. That is, all occurrences of the variables are non-inverted.

2.2 Inapproximability of MAX EXPR

The inapproximability results we need for MAX EXPR is proved in this section.

Lemma 2.1. *For every problem Π in* **NP** *there is a polynomial time reduction from instances I of Π to instances $I' = (V, E)$ of* MAX EXPR(G) *such that*

- *if I is a* YES *instance then* $\text{OPT}(I') \geq (1-\delta)|E|$, *and*
- *if I is a* NO *instance then* $\text{OPT}(I') \leq |E|(1+\delta)/|G|$

where δ is an arbitrary constant such that $0 < \delta < 1$. Furthermore, every expression in E has exactly k variables and $\gcd(k, p_i) = 1$ for every i, $1 \leq i \leq n$.

Proof. Choose $k > 1$ such that $\gcd(k, p_i) = 1$ for every i, $1 \leq i \leq n$. We could, for example, choose k as $k = 1 + \prod_{i=1}^{n} p_i$.

We will prove the theorem with a reduction from MAX-Ek-LIN-G. Theorem 2.2 makes this a suitable approach. Given an instance J of an arbitrary problem Π in **NP**, reduce J to an instance, $I = (V, E)$, of MAX-Ek-LIN-G with the reduction in Theorem 2.2. We will construct an instance $I' = (V, E')$ of MAX EXPR(G) from I.

Every equation e_j in E is of the form $x_1 + \ldots + x_k = c_j$. For every e_j add the expression e'_j, which we define as, $x_1 + \ldots + x_k - c_j$ to E'.

According to Theorem 2.2 we know that either $\text{OPT}(I) \geq |E|(1-\delta)$ or $\text{OPT}(I) \leq |E|(1+\delta)/|G|$.

Case 1: ($\text{OPT}(I) \geq (1-\delta)|E|$) Let f be an assignment such that $m(I, f) \geq (1-\delta)|E|$. The same assignment used on I' will give $m'(I', f) \geq m(I, f) = (1-\delta)|E|$. (At least $(1-\delta)|E|$ of the expressions will evaluate to 0_G.)

Case 2: ($\text{OPT}(I) \leq |E|(1+\delta)/|G|$) Assume that we have an assignment $f' : V \to G$ to I' such that $m'(I', f') > |E|(1+\delta)/|G|$. Then there exists an element $a \in G$ such that more than $|E|(1+\delta)/|G|$ expressions in I' evaluate to a when f' is used.

As $\gcd(k, p_i) = 1$ for every i, $1 \leq i \leq n$ the integer k has a multiplicative inverse in every $\mathbf{Z}_{p_i^{\alpha_i}}$. Therefore, the equation $kq = -a$ has a solution q in G. We can now construct an new assignment, f, to the variables in I and I'.

$$f(v) = f'(v) + q$$

If we use f on I' we get the following value for the expressions that evaluated to a under f': (we are abusing our notation here; $f(Q)$ where Q is an equation or an expression means that the variables in Q shall be assigned values according to f)

$$\begin{aligned} f(e'_j) &= f(x_1) + \ldots + f(x_k) - c_j \\ &= kq + f'(x_1) + \ldots + f'(x_k) - c_j \\ &= kq + a = -a + a = 0_G \end{aligned}$$

Under f' we had more than $|E|(1+\delta)/|G|$ expressions which evaluated to a, now under f all those expressions evaluate to 0_G. However, that more than $|E|(1+\delta)/|G|$ expressions in I' evaluate to 0_G is equivalent to that more than $|E|(1+\delta)/|G|$ equations in I are satisfied. We have constructed an assignment f such that $m(f, I) > |E|(1+\delta)/|G|$. This contradicts our initial premise that $\text{OPT}(I) \leq |E|(1+\delta)/|G|$, the assignment f' can therefore not exist. Hence, $\text{OPT}(I') \leq |E|(1+\delta)/|G|$.

To conclude the proof note that we have proven that if $\text{OPT}(I) \geq |E|(1-\delta)$ then $\text{OPT}(I') \geq |E|(1-\delta)$ and if $\text{OPT}(I) \leq |E|(1+\delta)/|G|$ then $\text{OPT}(I') \leq |E|(1+\delta)/|G|$. This, together with Theorem 2.2, gives us the desired result. □

2.3 A First Inapproximability Result for MAX SOL EQN

In this section we prove a first inapproximability result for MAX SOL EQN. This result will be used in Section 2.4 to prove a stronger inapproximability result for MAX SOL EQN.

Lemma 2.2. *For any finite abelian group* G *and any non-constant function* $g : G \to \mathbf{N}$ *it is not possible to approximate* W-MAX SOL EQN(G, g) *within* $\alpha - \epsilon$ *where*

$$\alpha = |G|\frac{g_{\max}}{g_{\text{sum}}}$$

for any $\epsilon > 0$*, unless* $\mathbf{P} = \mathbf{NP}$.

Proof. We will prove the lemma with a gap-preserving reduction from MAX EXPR(G). Given an instance, J, of an arbitrary problem Π in $\mathbf{NP}$, reduce J to an instance, $I = (V, E)$, of MAX EXPR(G) with the reduction in Lemma 2.1. We will use I to construct an instance $I' = (V, E', w')$ of W-MAX SOL EQN(G, g). According to Lemma 2.1 every expression e_j in E is of the form $x_1 + \ldots + x_k + c_j$. Furthermore, we have $\gcd(k, p_i) = 1$ for every i, $1 \leq i \leq n$.

For every $e_j \in E$ add the equation e'_j, defined as $x_1 + \ldots + x_k + c_j = z_j$ to E', where z_j is a fresh variable. Let $w'(z_j) = 1$ for all $1 \leq j \leq |E|$ and $w'(\cdot) = 0$ otherwise.

We claim that the procedure presented above is a gap-preserving reduction from MAX EXPR(G) to MAX SOL EQN(G, g) with parameters

$$\begin{aligned} c &= (1-\delta)|E|, \\ c' &= (1-\delta)|E|g_{\max}, \text{ and} \\ \rho &= |G|\frac{1-\delta}{1+\delta}. \end{aligned}$$

Where δ is the constant from Lemma 2.1. The last parameter, ρ', is specified below. According to Lemma 2.1 we know that either $\text{OPT}(I) \geq |E|(1-\delta) = c$ or $\text{OPT}(I) \leq |E|(1+\delta)/|G| = c/\rho$.

Case 1: $(\text{OPT}(I) \geq (1-\delta)|E|)$ Let f be an assignment such that $m(I, f) \geq (1-\delta)|E|$ and let $a \in G$ be the element that most expressions in E evaluate to under f. Let b be an element in G such that $g(b) = g_{\max}$ and let q be the element in G such that $kq = -a + b$, such a q exists because $\gcd(k, p_i) = 1$ for every i, $1 \leq i \leq n$. Construct an assignment f' as follows: let $f'(x) = f(x) + q$ for every $x \in V$ and let $f'(z_j)$ be the value in G such that equation e'_j is satisfied.

It is clear that every equation in E' is satisfied by f'. Furthermore, note that for the expressions in E where $f(e_j) = a$ holds we have, for the corresponding equation e'_j,

$$\begin{aligned} f'(e'_j) &\iff f'(x_1) + \ldots + f'(x_k) + c_j = f'(z_j) \\ &\iff kq + f(x_1) + \ldots + f(x_k) + c_j = f'(z_j) \\ &\iff kq + a = f'(z_j) \\ &\iff f'(z_j) = b. \end{aligned}$$

Hence, for every expression e_j that evaluated to a under f the variable z_j gets the value b. As $g(b) = g_{\max}$ we get

$$m'(f', I') \geq (1 - \delta)|E'|g_{\max} = c'.$$

Case 2: ($\text{OPT}(I) \leq |E|(1+\delta)/|G|$) For any assignment f' to I' any subset of the z_j variables that have been assigned the same value must contain at most $\lfloor |E|(1+\delta)/|G| \rfloor$ variables. (Otherwise we would have $m(I, f') > |E|(1+\delta)/|G|$, which contradicts our assumption.) The measure of any assignment, f', to I' is then bounded by

$$\begin{aligned} m'(I', f') &\leq \sum_{d \in G} \left\lfloor |E| \frac{1+\delta}{|G|} \right\rfloor g(d) \\ &\leq |E| \frac{1+\delta}{|G|} \sum_{d \in G} g(d) \\ &\leq |E| \frac{1+\delta}{|G|} g_{\text{sum}} \end{aligned}$$

Let h denote the quantity on the right hand side of the inequality above. We want to find the largest ρ' that satisfies $\text{OPT}(I') \leq c'/\rho'$. If we choose ρ' such that $c'/\rho' = h$ then $\text{OPT}(I') \leq c'/\rho'$ because of $\text{OPT}(I') = m'(I', f') \leq h$.

$$\begin{aligned} \rho' = \frac{c'}{h} &= \frac{(1-\delta)|E|g_{\max}}{|E|\frac{1+\delta}{|G|}g_{\text{sum}}} \\ &= |G| \frac{g_{\max} - \delta g_{\max}}{g_{\text{sum}} + \delta g_{\text{sum}}} \end{aligned}$$

Now, given a fixed but arbitrary $\epsilon > 0$ we can choose $0 < \delta < 1$ such that

$$\rho' > |G|\frac{g_{\max}}{g_{\text{sum}}} - \epsilon = \alpha - \epsilon.$$

Note that due to the assumption that g is non-constant we have $|G|g_{\max}/g_{\text{sum}} > 1$. The gap-preserving reduction implies that it is **NP**-hard to find ρ'-approximations to W-MAX SOL EQN$(G,\ g)$, and as $\rho' > \alpha - \epsilon$ we have the desired result. □

2.4 Inapproximability of MAX SOL EQN

We are now ready to prove the main inapproximability theorem.

Proof (Of Theorem 2.1). We will begin with an outline of the proof. Let I' be an arbitrary instance of W-MAX SOL EQN$(G',\ g')$, where G' is a new group and $g' : G' \to \mathbf{N}$ is a new function, both of them will soon be defined. We will then prove that W-MAX SOL EQN$(G',\ g')$ is not approximable within $\alpha - \epsilon$ for any $\epsilon > 0$ unless $\mathbf{P} = \mathbf{NP}$. As the final step we will transform I' to an essentially equivalent instance I of W-MAX SOL EQN$(G,\ g)$. That is, for every solution s to I we can construct a solution s' to I' in polynomial time such that $m(I, s) = m'(I', s')$ and vice versa.

If we could approximate W-MAX SOL EQN$(G,\ g)$ within some ratio $\beta < \alpha$ we would be able to approximate W-MAX SOL EQN$(G',\ g')$ within β too, because given an instance, I', of W-MAX SOL EQN$(G',\ g')$ we can transform it into an essentially equivalent instance, I, of W-MAX SOL EQN$(G,\ g)$ and find a β-approximate solution, s, to this instance. This solution, s, can then be transformed into a β-approximate solution, s', to I' (due to the relation between I and I', they are essentially equivalent). We will now prove the theorem.

Let A be a matrix, $\boldsymbol{b}$ be a vector, c a group constant and $\boldsymbol{x} = (x_1, \ldots, x_m)^T$ a vector of variables such that

$$B = \{x_1 + c \mid A\boldsymbol{x} = \boldsymbol{b}\} \quad \text{and} \quad \alpha = \frac{g_{\max}(B)}{g_{\text{sum}}(B)}|B|.$$

The objects A, $\boldsymbol{b}$ and c do clearly exist due to the definition of coset-validity.

Let the group G' be defined as follows

$$G' = \{x_1 \mid A\boldsymbol{x} = \boldsymbol{b}\}.$$

That G' is a subgroup of G follows from the definition of coset-validity.

We define $g' : G' \to \mathbf{N}$ as $g'(x) = g(x + c)$. Note that $g'_{\max} = g_{\max}(B)$ and $g'_{\text{sum}} = g_{\text{sum}}(B)$. Hence, according to Lemma 2.2, MAX SOL EQN(G', g') is not approximable within $\alpha - \epsilon$ for any $\epsilon > 0$ unless $\mathbf{P} = \mathbf{NP}$.

An instance, $I' = (V', E', w')$, of W-MAX SOL EQN(G', g') can be transformed into an instance, $I = (V, E, w)$, of W-MAX SOL EQN(G, g) in the following way, assume that $V' = \{x_1, \ldots, x_{m'}\}$.

Let

$$V = V' \cup \{y_{ij} \mid 1 \leq i \leq m', 1 \leq j \leq m'\} \cup \{z_i \mid 1 \leq i \leq m'\}.$$

For each variable x_i in V add the equations

$$\begin{aligned} A\begin{pmatrix} y_{i1} \\ \vdots \\ y_{im'} \end{pmatrix} &= \boldsymbol{b} \\ x_i &= y_{i1} \\ z_i &= x_i + c \end{aligned}$$

to the set E''. Those equations will force the x_i variables to be assigned values that are in G'. Finally we let $E = E' \cup E''$. The weight function, w, is constructed as follows, for $1 \leq i \leq m'$, $w(z_i) = w'(x_i)$ otherwise $w(\cdot) = 0$.

Given a solution $s : V \to G$ to I we can construct a solution $s' : V' \to G'$ to I' with the property that $m(s, I) = m'(s', I')$. Note that the equations in E'' force the x_i variables to be assigned values that are contained in G'. Hence, for every feasible solution s to I we have that $s(x_i) \in G'$ for all i such that $1 \leq i \leq m$. Let s' be constructed as $s'(x_i) = s(x_i)$ for all i such that $1 \leq i \leq m$. Note that s' must be a feasible solution to I' as we have included the equations in E' in E. The measure of s and s' are then related to each other in the following way,

$$\begin{aligned} m(I,s) &= \sum_{i=1}^{m} w(z_i)g(s(z_i)) \\ &= \sum_{i=1}^{m} w'(x_i)g(s(x_i)+c) \\ &= \sum_{i=1}^{m} w'(x_i)g'(s'(x_i)) = m(I',s'). \end{aligned}$$

If we are given a solution, r' to I' we can in a similar way construct a solution r to I such that $m(I,r) = m'(I',r')$.

This concludes the proof, if we can approximate W-MAX SOL EQN(G, g) within some factor then we can also approximate W-MAX SOL EQN(G', g') within the same factor. □

Chapter 3

Approximability

Our approximability results for MAX SOL EQN(G, g) will be proven in this chapter. We will present a randomised approximation algorithm, called APPROX-SOLUTION, that just picks a feasible solution at random. The somewhat complicated part turns out to be the analysis of this algorithm. We will show that the performance ratio of this algorithm is equal, up to an arbitrary additive constant, to the inapproximability ratio of Theorem 2.1. That is, we will prove the following theorem.

Theorem 3.1 (Main Approximability Theorem). APPROX-SOLUTION *is an* α*-approximation algorithm for* MAX SOL EQN(G, g)*, where*

$$\alpha = \max \left\{ \frac{g_{\max}(B)}{g_{\text{sum}}(B)} |B| \;\middle|\; B \textit{ is coset-valid with respect to } G \right\}.$$

3.1 Algorithm Overview

The algorithm consists of four parts, TRANSFORM-MATRIX, REMOVE-ROWS, RANDOM-SOLUTION and APPROX-SOLUTION. As we are working with the abelian group G we can divide the input into n independent system of equations, each one over a group of the form $\mathbf{Z}_{p_i^{\alpha_i}}$. This is done in APPROX-SOLUTION. Each such system of equations is then given to

TRANSFORM-MATRIX which restructures the system of equations to a specific form. This restructuring is then continued in REMOVE-ROWS. The restructuring is performed in a way which preserves the solutions to the system of equations. Finally the decomposed and restructured systems of equations is fed to RANDOM-SOLUTION which generates a random solution. Note that the sequence TRANSFORM-MATRIX and REMOVE-ROWS is run once for each group which G is composed of. Hence, they will in total be run n times where n is the number of groups that G is composed of. The n system of equations generated by TRANSFORM-MATRIX and REMOVE-ROWS is then used by RANDOM-SOLUTION to find one solution to the original system of equations over G.

3.2 Matrix Restructuring

The goal of this chapter is to present an algorithm which restructures a system of equations, given on matrix form $A\boldsymbol{x} = \boldsymbol{b}$, into an other equivalent system of equations, $A'\boldsymbol{x} = \boldsymbol{b}'$, where the matrix A' satisfies certain properties.

The structure of this chapter is as follows, we begin with a description of what kind of restructuring we want to do to the system of equations. We then continue with a section with some mathematical preliminaries that we will use in the subsequent parts of the chapter. In the final two sections we present the two restructuring algorithms TRANSFORM-MATRIX and REMOVE-ROWS, we also prove their correctness.

At the end of the restructuring algorithms we want that $A' = (a'_{ij})$ satisfies the properties listed below. (Remember that we are working over $\mathbf{Z}_{p^\alpha}$ here.)

(i) $a'_{ij} = 0$ for $i > j$.

(ii) For every i we either have $a'_{ii} = p^k$ for some k or $a'_{ii} = 0$, furthermore for $i \leq j$ we either have $a'_{ii} \leq a'_{jj}$ or $a'_{ii} = p^k$ and $a'_{jj} = 0$ for some k.

(iii) For every row i and every element a'_{ij}, if $j > i$ then a'_{ij} is a multiple of a'_{ii}.

The transformation from A to A' is done by Transform-Matrix. The algorithm Remove-Rows will then do some further restructuring, which is not captured by the properties above.

3.2.1 Preliminaries

We need the following lemma in this section. It is a fundamental result about the existence of a multiplicative inverse. The proof can be found in any introduction to the theory of numbers. It can, for example, easily be deduced from Theorem 4.10 in [19].

Lemma 3.1. *For a given x the equation*

$$xy \equiv 1 \pmod{p^\alpha} \tag{3.1}$$

has one unique solution (i.e., all solutions to (3.1) *are congruent modulo p^α) if $p \nmid x$, furthermore if $p \mid x$ then no solutions exist to* (3.1).

When a solution exists to (3.1) it is called the inverse of x modulo p^α and the solution is denoted by x^{-1}.

3.2.2 Transform-Matrix

Given a system of equations over $\mathbb{Z}_{p^\alpha}$, for some prime p and integer α, on matrix form, $A\boldsymbol{x} = \boldsymbol{b}$, it is easy to see that any combination of the following elementary operations creates a new system of equations which have the same set of solutions as the original one.

1. Interchanging two rows of A and the corresponding elements in $\boldsymbol{b}$. That is, reordering the equations.

2. Interchanging two columns of A. That is, reordering the variables.

3. Adding a multiple of row i to row j and adding the same multiple of b_i to b_j, where $i \neq j$.

4. Multiplying a row of A and the corresponding element in $\boldsymbol{b}$ by a constant c such that $p \nmid c$.

We will only prove that operation number 4 preserves the set of solutions to the system of equations.

Proof (Of 4.). Let

$$\sum_{i=1}^{m} a_i x_i = q \tag{3.2}$$

be an equation over $\mathbf{Z}_{p^\alpha}$, where a_i are integers, x_i are variables and q is a group constant. As $p \nmid c$ there exists, according to Lemma 3.1, an integer c^{-1} such that $cc^{-1} = 1$.

If (3.2) holds then

$$c \sum_{i=1}^{m} a_i x_i = cq \iff \sum_{i=1}^{m} c a_i x_i = cq \tag{3.3}$$

clearly also holds. Furthermore, if (3.3) holds then

$$c^{-1} \sum_{i=1}^{m} c a_i x_i = c^{-1} cq \iff \sum_{i=1}^{m} a_i x_i = q$$

also holds. Hence, (3.2) is equivalent to (3.3). □

With the elementary operations we will soon see that it is always possible to transform the matrix A to a matrix $A' = \left(a'_{ij}\right)$ that satisfies properties (i)–(iii) listed in Section 3.2.

The algorithm TRANSFORM-MATRIX is supposed to, given a matrix A, a column vector $\boldsymbol{b}$ with as many rows as there are rows in A, a prime p and an integer α return a matrix A' that satisfies (i)–(iii) from Section 3.2 and a vector $\boldsymbol{b}'$ such that the system of equations $A\boldsymbol{x} = \boldsymbol{b}$ is equivalent to $A'\boldsymbol{x} = \boldsymbol{b}'$ over $\mathbf{Z}_{p^\alpha}$. Intuitively TRANSFORM-MATRIX transforms A to an upper triangular matrix of a special form.

Algorithm 1: TRANSFORM-MATRIX(A, $\boldsymbol{b}$, p, α)

1 Reduce A modulo p^α.
2 $l \leftarrow 1$
3 **For** c **from** 1 **to** $Rows(A)$ **do**
4   Let $i \geq c$ and $j \geq c$ be indices in A such that $p^l \nmid a_{ij}$.
5   **If** *no such indices exists* **then**
6   **If** $l = \alpha$ **then**
            ▷ At this stage every element a_{ij} with $i \geq c$ is equal to zero.
7   **Return** $(A, \boldsymbol{b})$
8   **Else**
9             $l \leftarrow l + 1$
10   **Goto** 4
11   **end**
12   **end**
13   Interchange row i and row c of A and interchange b_i and b_c.
14   Interchange column j and column c of A.
15   At this stage we have $a_{cc} = sp^{l-1}$ for some s such that $p \nmid s$. Multiply row c in A and $\boldsymbol{b}$ with s^{-1}.
16   Reduce row c in A and $\boldsymbol{b}$ modulo p^α.
17   **For** r **from** $c + 1$ **to** $Rows(A)$ **do**
18   We have $a_{rc} = tp^{l-1}$ for some t. Subtract t times row c from row r. Reduce row c modulo p^α.
19   **end**
20 **end**
21 **Return** $(A, \boldsymbol{b})$

We will now describe how TRANSFORM-MATRIX works. On lines 1 and 2 we reduce the matrix, A, modulo p^α and initialise l to 1. l is a variable that will be used to keep track of the current exponent of p we are working with. On line 4 we look for an element, on a row that we have not yet processed, which is not divisible by p^l. If $l = l_0$ at some point in the algorithm then we know that there are no elements in the non-processed part of the matrix which is not divisible by $p^{l_0 - 1}$, because such elements would have been chosen on line 4 in some previous iteration of the algorithm, when l held a smaller value. Therefore, if we find an element which is not divisible by p^l we know that it must be divisible by p^{l-1}. If we do not find such an element we increase l and try again. When we have found our non-divisible-by-p^l element we rearrange the matrix such that this element is positioned on the diagonal, this element is now denoted by a_{cc}. The repositioning is done on lines 13 and 14. On line 15 we multiply the row with an appropriate value to get a power of p in the diagonal element a_{cc}. Finally, on lines 17–19 we add an appropriate multiple of the current row (row c) to every row below to get zeros in the column below a_{cc}.

It is easily verified that TRANSFORM-MATRIX runs in polynomial time.

We prove the correctness of TRANSFORM-MATRIX in the following lemma.

Lemma 3.2 (Correctness of TRANSFORM-MATRIX**).** TRANSFORM-MATRIX *always returns a matrix, A', that satisfies the properties (i)–(iii). Furthermore, if* TRANSFORM-MATRIX *returns $(A', \boldsymbol{b}')$ on input A, $\boldsymbol{b}$, p and α then the system of equations $A\boldsymbol{x} = \boldsymbol{b}$ and $A'\boldsymbol{x} = \boldsymbol{b}'$ have the same set of solutions over $\mathbf{Z}_{p^\alpha}$ (except a possible reordering of the variables).*

Proof. The second part of the lemma holds because the only modifications done to A and $\boldsymbol{b}$ by the algorithm are elementary operations. The multiplication on row 15 do not create any problems because as $p \nmid s$, there exists an inverse s^{-1} to s, hence $ss^{-1} = 1$. However, the only elements in $\mathbf{Z}_{p_i^{\alpha_i}}$ that have inverses are those which are not multiples of p and as s is an inverse to s^{-1} we must have $p \nmid s^{-1}$ (this follows from Lemma 3.1). We will prove the first part of the lemma with the following loop invariants.

L_1: At the beginning of line 4 the $Rows(A) \times c - 1$ upper left sub matrix of A satisfies (i).

L_2: At the beginning of line 4 the $c - 1 \times c - 1$ upper left sub matrix of A satisfies (ii).

L_3: At the beginning of line 4 the $c - 1 \times Cols(A)$ upper left sub matrix of A satisfies (iii).

When $c = 1$ the loop invariants are vacuously true. Now assume that the loop invariants are true for $c = c_0$. We will prove that they are also true for $c = c_0 + 1$.

Case 1: (There are indices $i, j \geq c$ in A such that $p^l \nmid a_{ij}$.)

L_1: For all r such that $c + 1 \leq r \leq Rows(A)$ we will have $a_{rc} = tp^{l-1}$ for some t, because if $p^{l-1} \nmid a_{rc}$ then those indices would have been chosen on line 4 in some previous iteration of the algorithm and l would not have been increased to its present value. As we have $a_{rc} = tp^{l-1}$ lines 14–15 will clearly make the matrix satisfy $a_{rc} = 0$ for $c + 1 \leq r \leq Rows(A)$ and as we have assumed that L_1 holds for $c \leq c_0$ the loop invariant L_1 must hold for $c = c_0 + 1$ too.

L_2: We will have $a_{cc} = sp^{l-1}$ for some s such that $p \nmid s$ on line 15. (We must have $p^{l-1} \mid a_{cc}$ because otherwise this matrix element would have been chosen in some previous iteration of the algorithm before l was increased to its present value, furthermore we cannot have $p \mid s$ because then we would not have had $p^l \nmid a_{ij}$ on line 4.) As $p \nmid s$, s do have a multiplicative inverse in $\mathbf{Z}_{p^\alpha}$. After we multiply row c with s^{-1} we will have $a_{cc} = p^{l-1}$. Note that a_{cc} will not be modified any more by the algorithm, hence as L_2 is true for $c \leq c_0$ it is also true for $c \leq c_0 + 1$.

L_3: To prove L_3 for this case, assume that there is an index j such that $j > c$ and $a_{cc} \nmid a_{cj}$. Then we must have $l > 1$, because otherwise we would have $a_{cc} = 1$, which implies $a_{cc} \mid a_{cj}$. However, l would not have been increased to its present value if there was an element, a_{cj}, in the matrix such that $a_{cc} = p^{l-1} \nmid a_{cj}$. We conclude that the element a_{cj} cannot exist.

Case 2: (There are no indices $i, j \geq c$ in A such that $p^l \nmid a_{ij}$.) If $l = \alpha$ every element a_{ij} with $j \geq c$ must be equal to zero, as the matrix has been reduced modulo p^α and there did not exist any indices $i, j \geq c$ such that $p^\alpha = 0 \nmid a_{ij}$. As we have assumed that L_1, L_2 and L_3 holds for $c \leq c_0$, L_1, L_2 and L_2 must, in this case, hold for the entire matrix.

Assume that $l < \alpha$. The variable l will then be increased and sooner or later we will either get case 1 or the first part of case 2. With that, we are done with case 2.

The loop will terminate on either line 7 or on line 21. We have already considered the first case above. In the second case the loop terminated because $c = Rows(A)$, the loop invariants then tells us that A satisfies (i)–(iii).

□

3.2.3 REMOVE-ROWS

Property (ii) and (iii) implies that if we have $a_{ii} = 0$ for some i we get two different cases.

Case 1: ($b_i = 0$) Row i expresses an equation that do not constrain the variables in any way and can hence be removed. That is, row i expresses an equation of the form

$$\sum_{i=1}^{m} a_i x_i = b_i$$

with $a_i = 0$ for $1 \leq i \leq m$ and $b_i = 0$.

Case 2: ($b_i \neq 0$) Row i expresses an equation that do not have any solutions. That is, row i expresses an equation of the form

$$\sum_{i=1}^{m} a_i x_i = b_i$$

with $a_i = 0$ for $1 \leq i \leq m$ and $b_i \neq 0$. The entire system of equations is therefore unsolvable.

The algorithm REMOVE-ROWS takes care of this extra transformation. From the discussion above we get the following lemma.

 www.FirstRanker.com

Algorithm 2: REMOVE-ROWS(A, $\boldsymbol{b}$)

```
For r from Rows(A) down to 1 do
    If a_rr = 0 then
        If b_r = 0 then
            Remove row r from A
            Remove row r from b
        Else
            Return "no solutions"
        end
    Else
        Return (A, b)
    end
end
Return (A, b)
```

Lemma 3.3 (Correctness of REMOVE-ROWS**).** *Given a matrix A that satisfies (i)–(iii) from Section 3.2 and a vector $\boldsymbol{b}$, if $A\boldsymbol{x} = \boldsymbol{b}$ has at least one solution,* REMOVE-ROWS *returns a new matrix A' and vector $\boldsymbol{b}'$ such that $A\boldsymbol{x} = \boldsymbol{b}$ is equivalent to $A'\boldsymbol{x} = \boldsymbol{b}'$. If $A\boldsymbol{x} = \boldsymbol{b}$ do not have any solutions then* REMOVE-ROWS *will either return a new matrix A' and vector $\boldsymbol{b}'$ such that $A'\boldsymbol{x} = \boldsymbol{b}'$ do not have any solutions or it will return "no solutions".*

3.3 RANDOM-SOLUTION

A consequence of the fact that finite abelian groups can be seen as direct products of groups of the form $\mathbf{Z}_{p^\alpha}$ for some prime p and integer α is that a system of equations over G can be decomposed into n systems of equations over the groups $\mathbf{Z}_{p_i^{\alpha_i}}$ for i, $1 \leq i \leq n$. Furthermore, as the elements in the group G can be seen as vectors where the element in position i in the vector is an element in $\mathbf{Z}_{p_i^{\alpha_i}}$, the vector $\boldsymbol{b}$ can be decomposed into n vectors, $\boldsymbol{b}^{(1)}, \ldots, \boldsymbol{b}^{(n)}$, such that

$$\boldsymbol{b} = \left(\boldsymbol{b}^{(1)}, \ldots, \boldsymbol{b}^{(n)}\right)$$

where $\boldsymbol{b}^{(i)}$ is a vector with elements in $\mathbf{Z}_{p_i^{\alpha_i}}$.

Hence, given A and $\boldsymbol{b}$ we can decompose $\boldsymbol{b}$ into $\boldsymbol{b}^{(1)}, \ldots, \boldsymbol{b}^{(n)}$ and give, for each i such that $1 \leq i \leq n$, A, $\boldsymbol{b}^{(i)}$, p_i and α_i to TRANSFORM-MATRIX. We will then end up with n matrices, let us call them ${A'}^{(1)}, \ldots, {A'}^{(n)}$, and n vectors, called ${\boldsymbol{b}'}^{(1)}, \ldots, {\boldsymbol{b}'}^{(n)}$.

As TRANSFORM-MATRIX only does solution-preserving operations on A and $\boldsymbol{b}$ we can, given ${A'}^{(1)}, \ldots, {A'}^{(n)}$ and ${\boldsymbol{b}'}^{(1)}, \ldots, {\boldsymbol{b}'}^{(n)}$, generate solutions to $A\boldsymbol{x} = \boldsymbol{b}$. This is precisely what the algorithm presented in the following section does. Intuitively RANDOM-SOLUTION generates a random solution by back substitution.

3.3.1 The Algorithm

Algorithm 3: RANDOM-SOLUTION$\left(\left(A^{(1)},\dots,A^{(n)}\right),\left(\boldsymbol{b}^{(1)},\dots,\boldsymbol{b}^{(n)}\right)\right)$

▷ All matrices $A^{(1)},\dots,A^{(n)}$ have the same number of columns.

1 $c \leftarrow \mathit{Cols}\left(A^{(1)}\right)$
2 **For** j **from** 1 **to** n **do**
3     $r \leftarrow \mathit{Rows}\left(A^{(j)}\right)$
4   **For** i **from** $r+1$ **to** c **do**
5         $B_i^{(j)} \leftarrow \mathbf{Z}_{p_j^{\alpha_j}}$
6         $x_i^{(j)} \leftarrow \mathit{Rand}\left(B_i^{(j)}\right)$
7   **end**
8   **For** i **from** r **down to** 1 **do**
9         $e_i^{(j)} \leftarrow b_i^{(j)} - \sum_{k=i+1}^{c} a_{ik}^{(j)} x_k^{(j)}$
10         $B_i^{(j)} \leftarrow \left\{ q \;\middle|\; q \in \mathbf{Z}_{p_j^{\alpha_j}},\ a_{ii}^{(j)} q \equiv e_i^{(j)} \pmod{p_j^{\alpha_j}} \right\}$
11   **If** $B_i^{(j)} = \emptyset$ **then**
12   **Return** *"no solutions"*
13   **end**
14         $x_i^{(j)} \leftarrow \mathit{Rand}\left(B_i^{(j)}\right)$
15   **end**
16 **end**
17 **For** i **from** 1 **to** c **do**
18     $x_i \leftarrow \left(x_i^{(1)},\dots,x_i^{(n)}\right)$
19 **end**
20 **Return** $(x_1,\dots,x_c)$

RANDOM-SOLUTION is supposed to be given n matrices $A^{(1)},\dots,A^{(n)}$ and n vectors $\boldsymbol{b}^{(1)},\dots,\boldsymbol{b}^{(n)}$, those matrices and vectors come from TRANSFORM-MATRIX followed by REMOVE-ROWS. The matrices therefore satisfies the properties (i)–(iii) in Section 3.2.

On line 1 in we assign the number of columns in the matrices to c. This is done to shorten the notation somewhat in the subsequent parts of the algorithm. The first for loop, on lines 2–16, loops over the different groups that G is composed of. We generate a solution to the system of equations over each such group independently of each other. On lines 4–7 we generate values to the last $c-r-1$ variables. They are not constrained in any way by any other variables and can therefore be chosen freely from the entire group. On lines 8–15 we generate values for the variables that are constrained by other variables. We do this "backwards", i.e., we start with variable number r and go down to 1, this is because variable number $i-1$ may depend upon variable number i, we therefore need the value of variable number i before we know which values can be assigned to variable number $i-1$. However, due to the form of the transformed matrix variable number i will never depend upon variable number $i-1$. This observation is the fundamental idea behind RANDOM-SOLUTION. In the last for loop on lines 17–19 we compose the different group solutions to one solution over G.

It is easily verified that RANDOM-SOLUTION runs in polynomial time. To prove the correctness of RANDOM-SOLUTION we need a couple of technical lemmas, which are presented in the following section. In will use those lemmas in Section 3.3.3 to prove the correctness of RANDOM-SOLUTION.

3.3.2 Technical Lemmas

We need two lemmas to prove the results in this chapter. The first one is about the number of solutions to a specific congruence. The second lemma is about the distribution of a linear combination of certain random variables.

Lemma 3.4. *A congruence of the form*

$$xp^n \equiv c \pmod{p^\alpha}$$

for some prime p, positive integer α and non-negative integer n have,

1. *exactly p^n incongruent solutions if $n < \alpha$ and $p^n \mid c$, and*

2. *no solutions if $n > 0$ and $p^n \nmid c$.*

Note that congruences modulo p^α is equivalent to equations over the group $\mathbf{Z}_{p^\alpha}$. Each congruence class modulo p^α may be seen as one group element in $\mathbf{Z}_{p^\alpha}$. We will therefore be able to use this lemma when we prove results about equations over groups of the form $\mathbf{Z}_{p^\alpha}$.

Proof (Of 1). As $p^n \mid c$ then there exists an integer m such that $mp^n = c$.

$$\begin{array}{ll} xp^n \equiv c \pmod{p^\alpha} & \iff \\ p^\alpha \mid xp^n - mp^n & \iff \\ \exists k : kp^\alpha = xp^n - mp^n & \iff \\ \exists k : x = kp^{\alpha-n} + m & \end{array}$$

The last statement gives us the p^n incongruent solutions, they are generated by $k = 0, \ldots, k = p^n - 1$. □

Proof (Of 2).

$$\begin{array}{ll} xp^n \equiv c \pmod{p^\alpha} & \iff \\ p^\alpha \mid xp^n - c & \iff \\ \exists k : kp^\alpha = xp^n - c & \iff \\ \exists k : c = xp^n - kp^\alpha & \iff \\ \exists k : c = p^n(x - kp^{\alpha-n}) & \end{array}$$

Hence, the only possibility for a solution to exist is if $p^n \mid c$, but we assumed $p^n \nmid c$ and therefore no solution can exist. □

We will also need the following lemma about the distribution of linear combinations of certain uniformly distributed random variables.

Lemma 3.5. *Let A and B be subgroups of $H = \mathbf{Z}_{p^\alpha}$ for some prime p and integer α. Given two constants, $a, b \in \mathbf{N}$ and two independent random variables $X \sim U(A)$ and $Y \sim U(B)$, define the random variable Z as $Z = aX + bY$. We will then have $Z \sim U(C)$ for some subgroup C of H.*

Proof. For every subgroup of H there is a non-negative integer, r, such that the subgroup is equal to $\{p^r x \mid 0 \leq x < p^{\alpha-r}\}$.

Assume that k and l are integers such that $A = \{p^k x \mid 0 \leq x < p^{\alpha-k}\}$ and $B = \{p^l x \mid 0 \leq x < p^{\alpha-l}\}$. Due to the observation above those integers exists. Furthermore, let a', b', n and m be the integers such that $a = a'p^n$ and $b = b'p^m$ where $p \nmid a', b'$. Such integers exists for every integer a and b.

We introduce new random variables, X' and Y', such that $X = p^k X'$ and $Y = p^l Y'$, we then have $X' \sim U(\{0, \ldots, p^{\alpha-k} - 1\})$ and $Y' \sim U(\{0, \ldots, p^{\alpha-l} -$

$1\}$). Furthermore, we can now express the random variable Z as

$$Z = aX + bY = a'p^n X + b'p^m Y = a'p^{n+k}X' + b'p^{m+l}Y'. \tag{3.4}$$

Assume, without loss of generality, that $n + k \leq m + l$. We can then rewrite (3.4) as,

$$Z = p^{n+k}\left(a'X' + b'p^{m+l-n-k}Y'\right).$$

If $n + k \geq \alpha$ then $Z = U(\{0_H\})$ (i.e., $\Pr[Z = 0_H] = 1$). This follows from the fact that every integer which is divisible by p^α is congruent with 0 modulo $\mathbf{Z}_{p^\alpha}$. This Z and the trivial subgroup $C = \{0_H\}$ of H give us the desired result for this case. Now assume that $n + k < \alpha$ and let $C = \{xp^{n+k} \mid 0 \leq x < p^{\alpha-n-k}\}$. For each $xp^{n+k} \in C$ the probability that Z equals xp^{n+k} is,

$$\begin{aligned}
&\Pr\left[Z \equiv xp^{n+k} \pmod{p^\alpha}\right] &=\\
&\Pr\left[p^{n+k}\left(a'X' + b'p^{m+l-n-k}Y'\right) \equiv xp^{n+k} \pmod{p^\alpha}\right] &=\\
&\Pr\left[a'X' + b'p^{m+l-n-k}Y' \equiv x \pmod{p^{\alpha-n-k}}\right].
\end{aligned} \tag{3.5}$$

Note that $a'X' \bmod p^{\alpha-n-k} \sim U(\{0, \ldots, p^{\alpha-n-k} - 1\})$, this implies, together with the fact that Y' is independent of $a'X'$, that the probability

$$\Pr\left[a'X' \equiv x \pmod{p^{\alpha-n-k}}\right] = p^{-(\alpha-n-k)} = p^{n+k-\alpha}$$

is equal to (3.5). We conclude that $Z \sim U(C)$. □

3.3.3 Correctness

In the following lemma we will prove the correctness of RANDOM-SOLUTION. We will use variable names from the algorithm in this lemma, hence c, r and $x_i^{(j)}$ refer to those variables in the algorithm.

This lemma is the main ingredient in our approximability result for MAX SOL EQN(G, g). What we really want to prove is that the variables $x_i^{(j)}$ will always be uniformly distributed over a coset of some subgroup of G. At a first look on RANDOM-SOLUTION this seems to be a trivial statement, those variables are clearly uniformly distributed because they are assigned values with a statement such as $Rand(S)$, for some set S! However, note that the set S may depend on previous choices made by the algorithm. It is therefore not clear that those variables will be uniformly distributed. However, the following lemma and its proof tells us that this always is the case.

Lemma 3.6 (Correctness of RANDOM-SOLUTION**).** *If* RANDOM-SOLUTION *is given an instance of* MAX SOL EQN(G, g) *with at least one feasible solution that have been fed through* TRANSFORM-MATRIX *and* REMOVE-ROWS *then for all i and j such that $1 \leq i \leq c$ and $1 \leq j \leq n$ the following entities exists,*

- *a subgroup $G_i^{(j)}$ of $\mathbf{Z}_{p_j^{\alpha_j}}$,*
- *a random variable $Z_i^{(j)}$, and*
- *a group constant $c_i^{(j)} \in \mathbf{Z}_{p_j^{\alpha_j}}$.*

Furthermore, those entities and the variable $x_i^{(j)}$, satisfy the following properties

- *$x_i^{(j)} = Z_i^{(j)} + c_i^{(j)}$, and*
- *$Z_i^{(j)} \sim U\left(G_i^{(j)}\right)$.*

The central part of the lemma is the last two bullet points, which tells us that $x_i^{(j)}$ is uniformly distributed over some coset of some subgroup of $\mathbf{Z}_{p_j^{\alpha_j}}$. We will now go on with the proof.

Proof. Fix j. We will prove the lemma with induction on i. The induction hypothesis we will use is:

There exists random variables $Y_1^{(j)}, \ldots, Y_c^{(j)}$ such that for every i', $i \leq i' \leq c$ the following entities exists,

- a subgroup $G_{i'}^{(j)}$ of $\mathbf{Z}_{p_j^{\alpha_j}}$,
- integers $d_{1,i'}^{(j)}, \ldots, d_{c,i'}^{(j)}$, and

- group constants $c_{i'}^{(j)} \in \mathbf{Z}_{p_j^{\alpha_j}}$.

Those entities satisfy the following properties

- $x_{i'}^{(j)} = c_{i'}^{(j)} + \sum_{k=i'}^{c} d_{k,i'}^{(j)} Y_k^{(j)}$, and
- $Y_{i'}^{(j)} \sim U\left(G_{i'}^{(j)}\right)$, furthermore each $Y_{i'}^{(j)}$ is independent of every $Y_{i''}^{(j')}$ such that $i' \neq i''$ or $j \neq j'$.

The hypothesis is clearly true for all i such that $r+1 \leq i \leq c$. (To see this let $G_i^{(j)} = \mathbf{Z}_{p_j^{\alpha_j}}$, $c_i^{(j)} = 0$, $d_{i,i}^{(j)} = 1$ and $d_{i,i'}^{(j)} = 0$ for i', $i+1 \leq i' \leq c$.) Assume that the hypothesis is true for $i = i_0$. We will prove that the hypothesis is true for $i = i_0 - 1$. With $i = i_0 - 1 \leq r$ we will, on line 10 in RANDOM-SOLUTION, have

$$e_i^{(j)} = b_i^{(j)} - \sum_{k=i+1}^{c} a_{ik}^{(j)} x_k^{(j)}.$$

Property (ii) and (iii) of A implies that there is an integer q such that $p_j^q = a_{ii}^{(j)} \mid a_{ik}^{(j)}$ for all k such that $i+1 \leq k \leq c$. (We cannot have $a_{ii}^{(j)} = 0$, because REMOVE-ROWS removes such rows.) Furthermore, if $p_j^q \nmid b_i^{(j)}$ then $p_j^q \nmid e_i^{(j)}$.

We can actually assume that $p_j^q \mid e_i^{(j)}$, because if we have $p_j^q \nmid e_i^{(j)}$ then according to Lemma 3.4, the equation

$$a_{ii}^{(j)} x_i^{(j)} \equiv e_i^{(j)} \pmod{p_j^{\alpha_j}}$$

do not have any solutions and we have assumed that we are working with an instance that do have at least one feasible solution.

The values that can be assigned to $x_i^{(j)}$ are one of the solutions to the equation

$$\begin{aligned} p_j^q x_i^{(j)} &\equiv e_i^{(j)} && \pmod{p_j^{\alpha_j}} \iff \\ p_j^q x_i^{(j)} &\equiv b_i^{(j)} - \textstyle\sum_{k=i+1}^{c} a_{ik}^{(j)} x_k^{(j)} && \pmod{p_j^{\alpha_j}}. \end{aligned}$$

The solutions to this equation can, according to Lemma 3.4, be written as

$$x_i^{(j)} = Wp_j^{\alpha_j - q} + p_j^{-q}\left(b_i^{(j)} - \sum_{k=i+1}^{c} a_{ik}^{(j)} x_k^{(j)}\right) \tag{3.6}$$

for $W = 0, \ldots, W = p_j^q - 1$. As we have assumed the induction hypothesis (3.6) can be rewritten as

$$x_i^{(j)} = Wp_j^{\alpha_j - q} + p_j^{-q} b_i^{(j)} - \left(\sum_{k=i+1}^{c} D_k Y_{k,j} + p_j^{-q} a_{ik}^{(j)} c_k^{(j)}\right) \tag{3.7}$$

for some integers $D_{i+1}, \ldots, D_k$. Line 14 of RANDOM-SOLUTION picks W uniformly at random from $\{0, \ldots, p_j^q - 1\}$, furthermore this choice is independent of everything that has happened before. Hence we can see W as a random variable such that $W \sim U(\{0, \ldots, p_j^q - 1\})$, furthermore W is independent of every $Y_k^{(j)}$ for all $k \geq i+1$ and j. Equation (3.7) can be rewritten as

$$x_i^{(j)} = Wp^{\alpha_j - q} - \sum_{k=i+1}^{c} D_k Y_k^{(j)} + p_j^{-q} b_i^{(j)} - p_j^{-q} \sum_{k=i+1}^{c} a_{ik}^{(j)} c_k^{(j)}.$$

Which is exactly what we want to prove, because the set

$$G' = \left\{ Wp_j^{\alpha_j - q} \;\middle|\; W \in \{0, \ldots, p_j^q - 1\} \right\}$$

is a subgroup of $\mathbf{Z}_{p_j^{\alpha_j}}$ since G' just is all multiples of $p_j^{\alpha_j - q}$ that exists in $\mathbf{Z}_{p_j^{\alpha_j}}$. Now, let $H_i^{(j)} = G'$, $Y_i^{(j)} = Wp_j^{\alpha - q}$, $d_{k,i}^{(j)} = -D_k$ for k, $i+1 \leq k \leq c$, $d_{i,i}^{(j)} = 1$, and, finally,

$$c_i^{(j)} = p_j^{-q} b_i^{(j)} - p_j^{-q} \sum_{k=i+1}^{c} a_{ik}^{(j)} c_k^{(j)}.$$

This completes the induction step.

It remains to prove that the induction hypothesis implies the lemma we want to prove. By repeatedly applying Lemma 3.5 to

$$x_i^{(j)} = c_i^{(j)} + \sum_{k=i+1}^{c} d_{k,i}^{(j)} Y_k^{(j)}$$

we get the desired result. To see this consider the first two terms in the sum, $d_{i+1,i}^{(j)} Y_{i+1}^{(j)}$ and $d_{i+2,i}^{(j)} Y_{i+2}^{(j)}$. Let Q_1 be defined as $Q_1 = d_{i+1,i}^{(j)} Y_{i+1}^{(j)} + d_{i+2,i}^{(j)} Y_{i+2}^{(j)}$. Lemma 3.5 then tells us that Q_1 is uniformly distributed over some subgroup of $\mathbf{Z}_{p_j^{\alpha_j}}$. We can then apply Lemma 3.5 again on Q_1 and the third term in the sum, $d_{i+3,i}^{(j)} Y_{i+3}^{(j)}$, to define a new random variable, Q_2, as the sum of Q_1 and the third term. Due to Lemma 3.5 Q_2 will also be uniformly distributed over some subgroup of $\mathbf{Z}_{p_j^{\alpha_j}}$. Continuing in the same manner we will get

$$x_i^{(j)} = c_i^{(j)} + Q_{c-i-1}$$

where Q_{c-i-1} is uniformly distributed over some subgroup of $\mathbf{Z}_{p_j^{\alpha_j}}$. As the last step let $Z_i^{(j)} = Q_{c-i-1}$. □

3.4 APPROX-SOLUTION

In this section we present the final part of the approximation algorithm, APPROX-SOLUTION. APPROX-SOLUTION uses TRANSFORM-MATRIX, REMOVE-ROWS and RANDOM-SOLUTION to find an approximate solution to an instance $I = (A, \boldsymbol{b})$ of MAX SOL EQN(G, g).

We begin with presenting the algorithm in this section, in Section 3.4.1 we prove its correctness and, finally, in Section 3.4.2 we analyse the performance of the algorithm.

Algorithm 4: Approx-Solution($A, \boldsymbol{b}$)

```
1 For i from 1 to n do
2     (A'^(i), b'^(i)) ← Transform-Matrix(A, b^(i), p_i, α_i)
3     A'^(i) ← Remove-Rows(A'^(i), b'^(i))
4 end
5 Return Random-Solution((A'^(1), ..., A'^(n)), (b'^(1), ..., b'^(n)))
```

On lines 4–4 the matrix A is transformed with both Transform-Matrix and Remove-Rows. Note that for each group which G is composed of an individual matrix ${A'}^{(i)}$ is produced. The result of the for loop is then fed to Random-Solution to produce an approximate solution to the instance.

The for loop on lines 4–4 runs n times. As n is independent of the size of the input and Transform-Matrix and Remove-Rows are polynomial time algorithms, the for loop is also a polynomial time algorithm. As the last step Approx-Solution invokes Random-Solution, and the latter is a polynomial time algorithm, we can conclude that Approx-Solution is a polynomial time algorithm.

3.4.1 Correctness

The correctness of Approx-Solution is given by the following lemma.

Lemma 3.7 (Correctness of Approx-Solution). *Given an instance, $I = (A, \boldsymbol{b})$, of* Max Sol Eqn$(G,\ g)$ *then*

1. Approx-Solution *will not return a non-feasible solution,*
2. *if there is at least one feasible solution to I then* Approx-Solution *will not return "no solutions", and*
3. *for every feasible solution to I there is a non-zero probability that it will be returned by* Approx-Solution*.*

Proof (Part 1). A consequence of the fact that finite abelian groups can be seen as direct products of groups of the form $\mathbf{Z}_{p^\alpha}$ for some prime p and integer α is that a system of equations over G can be decomposed into

n systems of equations over the groups $\mathbf{Z}_{p_i^{\alpha_i}}$ for i, $1 \leq i \leq n$. This is what APPROX-SOLUTION does on lines 4–4. Furthermore, Lemma 3.2 and Lemma 3.3 says that lines 4–4 do not alter the set of feasible solutions.

TRANSFORM-MATRIX returns an equivalent system of equations with the property that the matrix is upper triangular. This makes back substitution a valid approach to find solutions, which is exactly what RANDOM-SOLUTION does. It is therefore clear that RANDOM-SOLUTION will not return a non-feasible solution. □

Proof (Part 2). "no solutions" is returned on two places, on line 2 in REMOVE-ROWS and on line 12 in RANDOM-SOLUTION. Lemma 3.3 says that REMOVE-ROWS will only return "no solutions" if there do not exist any feasible solutions to I. RANDOM-SOLUTION returns "no solutions" if $B_i^{(j)} = \emptyset$. Lemma 3.4 tells us that this happens if and only if $a_{ii}^{(j)} \nmid e_i^{(j)}$.

Property (iii) of $A^{(j)}$ implies that we will have $a_{ii}^{(j)} \nmid e_i^{(j)}$ if and only if $a_{ii}^{(j)} \nmid b_i^{(j)}$. As $b_i^{(j)}$ is independent of the random choices that RANDOM-SOLUTION does we will only get $a_{ii}^{(j)} \nmid b_i^{(j)}$ if there are no feasible solutions to the instance I. □

Proof (Part 3). As argued in the proof of part 1 of this lemma, lines 4–4 of APPROX-SOLUTION cannot cause any trouble because they only transform the system of equations to a set of systems of equations which are equivalent to the system we started with.

Let us assume that A have c columns and r rows, furthermore assume that $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_c$ is a feasible solution to I (with $\boldsymbol{y}_i = \left(y_i^{(1)}, \ldots, y_i^{(n)}\right)$ for all i, $1 \leq i \leq c$ as usual). Then for $r + 1 \leq i \leq c$ and $1 \leq j \leq n$ we must have $y_i^{(j)} \in \mathbf{Z}_{p_j^{\alpha_j}}$, due to lines 4–7 of RANDOM-SOLUTION it is then a non-zero probability that $x_i^{(j)} = y_i^{(j)}$ for those i:s and j:s.

Line 10 of RANDOM-SOLUTION finds, in iteration i, all group elements that satisfies equation number i. As $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_c$ is a feasible solution it must satisfy every equation, $y_i^{(j)}$ must therefore be one of the group elements that satisfies equation i. We conclude that we will have $\Pr\left[x_i^{(j)} = y_i^{(j)}\right] > 0$ on line 14. □

3.4.2 Performance Analysis

We are now almost ready to analyse the performance of APPROX-SOLUTION. We need the following lemma in the performance analysis.

Lemma 3.8. *Given three sequences of integers, $a_1, \ldots, a_w$, $b_1, \ldots, b_w$ and $b'_1, \ldots, b'_w$. Then*

$$\sum_{i=1}^{w} a_i b'_i \leq \left(\max_{1 \leq i \leq w} b'_i / b_i \right) \cdot \sum_{i=1}^{w} a_i b_i. \tag{3.8}$$

Proof. Let us introduce d and e defined as

$$d/e = \max_{1 \leq i \leq w} b'_i / b_i$$

such that $d = b'_i$ and $e = b_i$ for some i. Now we have

$$(a_1 b_1 + \ldots + a_w b_w) \frac{d}{e} = a_1 \frac{b_1 d}{e} + \ldots + a_w \frac{b_w d}{e}. \tag{3.9}$$

Let us now compare the coefficients in front of one of the a_i values in this sum with the coefficient in front of the same value in the left hand side of (3.8). We get

$$\frac{d}{e} \geq \frac{b'_i}{b_i} \Rightarrow \frac{b_i d}{e} \geq b'_i.$$

The first inequality follows from the fact that d/e is the greatest such ratio. Note that b'_i is the coefficient in front of a_i in the left hand side of (3.8) and $b_k d/e$ is the coefficient in front of a_k on the right hand side of (3.8). Hence we get the desired result. □

We are now ready to prove the main theorem of this chapter, which really is the performance ratio of APPROX-SOLUTION.

Proof (Of Theorem 3.1). Let $I = (A, \boldsymbol{b})$ be an instance of MAX SOL EQN(G, g). Assume that A have r rows and c columns.

Lemma 3.6 says that for every j, $1 \leq j \leq n$ and i, $1 \leq i \leq c$ there are subgroups, $G_i^{(j)} \subseteq \mathbf{Z}_{p_j^{\alpha_j}}$, and group constants $c_i^{(j)} \in \mathbf{Z}_{p_j^{\alpha_j}}$ such that

$$x_i^{(j)} \sim U\left(G_i^{(j)} + c_i^{(j)}\right),$$

where $x_i^{(j)}$ are variables from RANDOM-SOLUTION.

Let us define the following groups G_i and group constants c_i

$$G_i = G_i^{(1)} \times \cdots \times G_i^{(n)} \qquad c_i = \left(c_i^{(1)}, \ldots, c_i^{(n)}\right)$$

for all i, $1 \leq i \leq c$. We then get

$$\left(x_i^{(1)}, \ldots, x_i^{(n)}\right) \sim U(G_i + c_i) \tag{3.10}$$

for all i, $1 \leq i \leq n$. (From now on we will use x_i to denote the left hand side of the expression above.) Furthermore, if $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_c$ is a feasible solution then $\boldsymbol{y}_i \in G_i + c_i$ for all i, $1 \leq i \leq c$. This follows from (3.10), part 1 of Lemma 3.7 (APPROX-SOLUTION do not return non-feasible solutions) and part 3 of Lemma 3.7 (for every feasible solution there is a non-zero probability it will be returned by APPROX-SOLUTION).

The previous argument together with the fact that the variables x_i for i, $1 \leq i \leq c$ are constrained by a system of equations implies that the set $G_i + c_i$ is coset-valid with respect to G. We will therefore have

$$\alpha \geq \frac{g_{\max}(G_i + c_i)}{g_{\text{sum}}(G_i + c_i)} |G_i| \tag{3.11}$$

for every i, $1 \leq i \leq n$. (Note that we have $|G_i + c_i| = |G_i|$.)

We are now ready to analyse APPROX-SOLUTION. Let S^* be defined as follows,

$$S^* = \sum_{i=1}^{c} g_{\max}(G_i + c_i).$$

From the discussion above we know that there is a non-zero probability that APPROX-SOLUTION will return an optimal solution. Let $s : \{x_1, \ldots, x_c\} \to G$ denote an optimal solution. As relation (3.10) holds we must have, for every i, $1 \leq i \leq c$, $g(s(x_i)) \leq g_{\max}(G_i + c_i)$. It follows that $S^* \geq$ OPT.

Let S denote the measure of a solution computed by APPROX-SOLUTION.

$$\begin{aligned} \mathrm{E}[S] &= \mathrm{E}\left[\sum_{i=1}^{c} g(x_i)\right] = \sum_{i=1}^{c} \mathrm{E}[g(x_i)] \\ &= \sum_{i=1}^{c} \frac{g_{\mathrm{sum}}(G_i + c_i)}{|G_i|} \\ &\geq S^*/\alpha \geq \textsc{opt}/\alpha \end{aligned}$$

The last inequality follows almost directly from (3.11) and Lemma 3.8 if $a_i = 1$, $b_i' = g_{\max}(G_i+c_i)$ and $b_i = g_{\mathrm{sum}}(G_i+c_i)/|G_i|$. To use Lemma 3.8 we only have to note that $\alpha \geq |G_i| g_{\max}(G_i + c_i)/g_{\mathrm{sum}}(G_i + c_i)$ for i, $1 \leq i \leq c$ (i.e., we do not necessarily have $\alpha = |G_i| g_{\max}(G_i + c_i)/g_{\mathrm{sum}}(G_i + c_i)$ for some i, but this fact is not a problem). □

Chapter 4

Weighted vs. Unweighted Problems

In Chapter 2 we proved that it is not possible to approximate W-Max Sol Eqn(G, g) better than a certain constant unless $\mathbf{P} = \mathbf{NP}$. We then proved that Max Sol Eqn(G, g) is approximable within the same constant. The main difference between those two results are that we have proved our inapproximability result for the weighted problem and our approximability result for the unweighted problem. We will, in this chapter, show that the approximability thresholds for the weighted and unweighted problems are asymptotically equal. This result is formally formulated in the following theorem.

Theorem 4.1. *If* Max Sol Eqn(G, g) *is approximable within in r, then* W-Max Sol Eqn(G, g) *is approximable within $r + o(1)$, where the $o(\cdot)$-notation is with respect to the size of the instance.*

In the context of Theorem 4.1 the $o(\cdot)$-notation means that if it possible to approximate Max Sol Eqn(G, g) within r then, for sufficiently large instances, it is possible to approximate W-Max Sol Eqn(G, g) within $r + \epsilon$ for any fixed $\epsilon > 0$.

The proof of Theorem 4.1 is divided into two parts. In Section 4.1 we prove that the weighted version of our problem, W-Max Sol Eqn, is in a specific sense at least weakly approximable. We then prove, in Section 4.2, that if W-Max Sol Eqn is weakly approximable then the approximation thresholds of Max Sol Eqn and W-Max Sol Eqn must be asymptotically equal.

4.1 Weak Approximability

In this section we will prove a lemma which says that W-Max Sol Eqn(G, g) is at least weakly approximable, i.e., there is a $p(n)$-approximation algorithm for some polynomial $p(n)$. We will then use this lemma to do a reduction from W-Max Sol Eqn(G, g) to Max Sol Eqn(G, g).

We use the terminology from [13] and say that if a problem Π is r-approximable and there exists a polynomial $p(n)$ such that $r \leq p(|I|)$ for every instance I then Π is in **poly-APX**.

The proof of the following lemma is based on the proof of Lemma 6.2 in [13].

Lemma 4.1. *For every finite abelian group G and every function $g : G \to \mathbf{N}$, W-Max Sol Eqn(G, g) is in* **poly-APX**.

Proof. An instance, $I = (V, E, w)$, of W-Max Sol Eqn(G, g) can be seen as a system of equations, given by $A\boldsymbol{x} = \boldsymbol{b}$ over G. It is well known that the problem of finding solutions to a linear system of equations over G is solvable in polynomial time. See, e.g., Theorem 1 in [9].

Let $V = \{x_1, \ldots, x_m\}$ and assume that $w(x_1) \geq w(x_2) \geq \ldots \geq w(x_m)$. We also assume that there is some $x \in G$ such that $g(x) > 0$. If this is not the case then W-Max Sol Eqn(G, g) is trivially in **poly-APX**.

We claim that Weakly-Approximate is a $|V|g_{\max}$-approximate algorithm. The running time of the algorithm is $O(|G|nf(n))$ where $f(n)$ is the time taken to solve one system of linear equations. This is polynomial in the input size $n = |I|$. Let x_i be the first variable the algorithm finds which can be set to a value such that $g(x_i) > 0$. The measure of this solution is then at least $w(x_i)$. Furthermore, $\textsc{opt}(I) \leq |V|g_{\max}w(x_i)$. $\square$

Algorithm 5: WEAKLY-APPROXIMATE(A, $\boldsymbol{b}$)

```
For i from 1 to m do
    Foreach y ∈ G such that g(y) > 0 do
        Create a new system of equations, A'x = b', which consists of
        Ax = b and the equation x_i = y.
        If A'x = b' is solvable then
            Return x, such that A'x = b'
        end
    end
end
▷ If we reach this line no feasible solution exists.
```

4.2 Weights Do Not Matter (Much)

In this section we will prove the main result of this chapter, i.e., Theorem 4.1. To do this we use a specific type of approximation preserving reduction, an AP-reduction, which is defined below.

Definition 4.1 (AP-reducibility [13]). *For a constant $\beta > 0$ and two* **NPO** *problems Π and Π', we say that Π is β*-AP-reducible *to Π', denoted $\Pi \leq_{\text{AP}}^{\beta} \Pi'$, if two polynomial time computable functions F and H exists such that the following holds:*

1. *For any instance I of Π, $F(I)$ is an instance of Π'.*

2. *For any instance I of Π, and any feasible solution s' for $F(I)$, $H(I, s')$ is a feasible solution for I.*

3. *For any instance I of Π and any $r \geq 1$, if s' is an r-approximate solution for $F(I)$, then $H(I, s')$ is an $(1+(r-1)\beta+o(1))$-approximate solution for I, where the $o(\cdot)$-notation is with respect to $|I|$.*

The definition of AP-reducibility looks complicated, but what it really means is that if $A \leq_{\text{AP}}^{\beta} B$ for some problems A and B and we have an r-approximation algorithm for B then we also have an $(1 + (r-1)\beta +$

$o(1)$)-approximation algorithm for A. In particular, if $A \leq^1_{\mathrm{AP}} B$ and B is approximable within r then A is approximable within $r + o(1)$.

The notation W-MAX SOL EQN$_p(G,\ g)$, used in the proof of Lemma 4.2 below, denotes W-MAX SOL EQN$(G,\ g)$ with the additional restriction that the weight function is bounded by a polynomial. That is, there exists a polynomial $p(n)$ such that for every instance $I = (V, E, w)$

$$\sum_{v \in V} w(v) \leq p(|I|).$$

The proof of the following lemma is based on Lemma 3.11 in [13], which in turn is based on Theorem 4 in [5].

Lemma 4.2. *For any finite abelian group G and function $g : G \to \mathbf{N}$, if* W-MAX SOL EQN$(G,\ g)$ *is in* **poly-APX***, then* W-MAX SOL EQN$(G,\ g)$ *1-AP-reduces to* MAX SOL EQN$(G,\ g)$.

Proof. The proof will be in two parts. In the first part, we will reduce W-MAX SOL EQN$(G,\ g)$ to W-MAX SOL EQN$_p(G,\ g)$ with the additional restriction that the weights are strictly greater than zero. In the second part, we reduce this restricted version of W-MAX SOL EQN$_p(G,\ g)$ to MAX SOL EQN$(G,\ g)$.

Given an instance $I = (V, E, w)$ of W-MAX SOL EQN$(G,\ g)$, we will construct a new weight function, w', and use this to define an instance $I' = (V, E, w')$ of W-MAX SOL EQN$_p(G,\ g)$.

Let A be a $p(x)$-approximation algorithm for W-MAX SOL EQN$(G,\ g)$ and let $t = m(I, A(I))$, i.e., t is the measure of the solution returned by the algorithm A on I. Let $M = g_{\max}$, $n = |I|$ and $N = Mnp(n)(np(n) + 1)$. We define a new scaled down weight function w'' such that

$$w''(v) = \left\lfloor \frac{w(v)N}{t} \right\rfloor + 1$$

for every $v \in V$. Finally let $w'(v) = \min\{w''(v), Np(n)+1\}$. It is clear that I' is an instance of W-MAX SOL EQN$_p(G,\ g)$ because w' is polynomially bounded.

We will now prove that if $w''(v) > w'(v)$ for some $v \in V$, then no feasible solution to I (or I') can have assigned v a value such that $g(v) > 0$.

$$w''(v) > w'(v) \Rightarrow \left\lfloor \frac{w(v)N}{t} \right\rfloor + 1 > Np(n) + 1 \Rightarrow w(v) > tp(n)$$

As the measure of a solution with $g(v) > 0$ would be at least $w(v)$, the last inequality contradicts the assumption that A is a $p(x)$-approximation algorithm and hence we have $g(v) = 0$. This implies that we have $\text{OPT}(I') \geq (N/t)\text{OPT}(I)$.

We can now construct algorithm H in the AP-reduction.

Algorithm 6: H(I, s')

```
If m(I, s') > m(I, A(I)) then
    Return s'
Else
    Return A(I)
end
```


Assume that s' is an r-approximate solution for I'. If $r \geq p(n)$, then the returned solution is clearly an r-approximate solution to I. Below we prove that even if $r \leq p(n)$, then the returned solution is a $(r+1/n)$-approximate solution to I. The measure of the returned solution is at least

$$\begin{aligned}
m(I, s') &= \sum_{v \in V} s'(v)w(v) \\
&\geq \frac{t}{N} \sum_{v \in V} s'(v) \left\lfloor \frac{w(v)N}{t} \right\rfloor \\
&\geq \frac{t}{N} \sum_{v \in V} s'(v) \left(\left\lfloor \frac{w(v)N}{t} \right\rfloor + 1 \right) - \frac{t}{N} Mn \\
&= \frac{t}{N} m(I', s') - \frac{t}{N} Mn \geq \frac{t}{N} \left(\frac{\textsc{opt}(I')}{r} - Mn \right) \\
&\geq \frac{\textsc{opt}(I)}{r} - \frac{Mnt}{N} \geq \frac{\textsc{opt}(I)}{r} - \frac{Mn\textsc{opt}(I)}{N} \\
&\geq \textsc{opt}(I) \left(\frac{1}{r} - \frac{1}{nr^2 + r} \right) = \frac{\textsc{opt}(I)}{r + 1/n}.
\end{aligned}$$

Note that the weights are not only bounded by a polynomial but they are also strictly greater than zero. With that the first step of the reduction is finished.

All that remains to be done is to complete the reduction to Max Sol Eqn(G, g). As the sum of the weights is bounded by a polynomial it is possible to replicate each variable a suitable number of times. Assume that $V = \{v_1, \ldots, v_m\}$, then for each variable $v_i \in V$ with weight $w'(v_i)$ introduce $w'(v_i) - 1$ fresh variables,

$$\left\{ v_i^{(j)} \;\middle|\; 1 \leq j \leq w'(v_i) - 1 \right\},$$

and the equations

$$\left\{ v_i = v_i^{(j)} \;\middle|\; 1 \leq j \leq w'(v_i) - 1 \right\}.$$

(Note that no fresh variables or equations are introduced if $w'(v_i) = 1$.) This procedure will create an instance, I'', of Max Sol Eqn(G, g) which is essentially equivalent to the original instance I' in the sense that given a solution, s', to I' it is possible to construct a solution, s'', to I'' in polynomial time such that $m'(I', s') = m''(I'', s'')$ and vice versa. □

We are now ready to prove Theorem 4.1

Proof (Of Theorem 4.1). The AP-reduction of Lemma 4.2 do exists due to Lemma 4.1. Hence, given an r-approximation algorithm to Max Sol Eqn(G, g) we can construct an $(r + o(1))$-approximation algorithm to W-Max Sol Eqn(G, g). □

Chapter 5

Conclusion

In this chapter we will put together the results from Chapters 2, 3 and 4 to prove the main result of this thesis. We begin with a summary of what we have done in the previous chapters where we repeat the main theorem of each chapter. After that we prove the main theorem of the thesis (Theorem 1.1). When we have proved our main result we state some results of a few variants of MAX SOL EQN, and finally we will give some ideas for possible future work in this line of research.

In Chapter 2 we proved the following result about the inapproximability of W-MAX SOL EQN(G, g).

Theorem 2.1 (Main Inapproximability Theorem). *For every finite abelian group G and every non-constant function $g : G \to \mathbf{N}$ it is not possible to approximate* W-MAX SOL EQN(G, g) *within $\alpha - \epsilon$ where*

$$\alpha = \max \left\{ \frac{g_{\max}(B)}{g_{\text{sum}}(B)} |B| \;\middle|\; B \textit{ is coset-valid with respect to } G \right\}$$

for any $\epsilon > 0$ unless $\mathbf{P} = \mathbf{NP}$.

Chapter 3 contained our approximability results for MAX SOL EQN(G, g). The main result was the following theorem.

Theorem 3.1 (Main Approximability Theorem). APPROX-SOLUTION *is an α-approximation algorithm for* MAX SOL EQN(G, g)*, where*

$$\alpha = \max\left\{\frac{g_{\max}(B)}{g_{\text{sum}}(B)}|B| \;\middle|\; B \textit{ is coset-valid with respect to } G\right\}.$$

In Chapter 4 we proved that the difference between MAX SOL EQN(G, g) and W-MAX SOL EQN(G, g) is in fact quite small. This result was summarised in the following theorem.

Theorem 4.1. *If* MAX SOL EQN(G, g) *is approximable within in r, then* W-MAX SOL EQN(G, g) *is approximable within $r + o(1)$, where the $o(\cdot)$-notation is with respect to the size of the instance.*

We will now use those results to prove the main theorem of this thesis, which we repeat here for completeness.

Theorem 1.1 (Main). *For every finite abelian group G and every function $g : G \to \mathbf{N}$,* MAX SOL EQN(G, g) *is approximable within α where*

$$\alpha = \max\left\{\frac{g_{\max}(B)}{g_{\text{sum}}(B)}|B| \;\middle|\; B \textit{ is coset-valid with respect to } G\right\}.$$

Furthermore, for every finite abelian group G and every non-constant function $g : G \to \mathbf{N}$ MAX SOL EQN(G, g) *is not approximable within $\alpha - \epsilon$ for any $\epsilon > 0$ unless $\mathbf{P} = \mathbf{NP}$.*

Proof. The approximation algorithm in Theorem 3.1 is the first part of Theorem 1.1.

Lemma 4.1 says that if we can find r-approximate solutions for MAX SOL EQN(G, g), then we can find $(r + o(1))$-approximate solutions for W-MAX SOL EQN(G, g). Hence, if we can find $\alpha - \delta$ approximations for some $\delta > 0$ for MAX SOL EQN(G, g) then we can find $(\alpha - \delta + o(1))$-approximate solutions for W-MAX SOL EQN(G, g). However, as the sizes of the instances grow we will, at some point, have $-\delta + o(1) < 0$ which means that we would be able to find $(\alpha - \epsilon)$-approximate solutions, where $\epsilon > 0$, for W-MAX SOL EQN(G, g). But Theorem 2.1 says that this is not possible. Therefore, MAX SOL EQN(G, g) is not approximable within $\alpha - \delta$, for any $\delta > 0$, unless $\mathbf{P} = \mathbf{NP}$. □

The situation is almost the same for W-Max Sol Eqn(G, g). We have an α-approximate algorithm for Max Sol Eqn(G, g) (Theorem 3.1) therefore, due to Lemma 4.1 we have a ($\alpha + o(1)$)-approximate algorithm for W-Max Sol Eqn(G, g). Furthermore, it is not possible to approximate W-Max Sol Eqn(G, g) within $\alpha - \epsilon$ for any $\epsilon > 0$ (Theorem 2.1).

All our hardness results holds for equations with at most three variables per equation. If we are given an equation with n variables where $n > 3$, we can reduce this equation to one equation with $n - 1$ variables and one equation with 3 variables in the following way: Given the equation $x_1 + \ldots + x_n = c$ where each x_i is either a variable or an inverted variable and c is a group constant, introduce the equation $z = x_1 + x_2$ where z is a fresh variable. Furthermore replace the original equation with the equation $z + x_3 + \ldots + x_n = c$. Let the weight of z be zero. Those two equations are clearly equivalent to the original equation in the problem W-Max Sol Eqn(G, g). The proof of Lemma 4.1 do not introduce any equations with more than two variables, so we get the same result for Max Sol Eqn(G, g).

If the instances of W-Max Sol Eqn(G, g) are restricted to have at most two variables per equation then the problem is tractable. The following algorithm solves this restricted problem in polynomial time.

A system of equations where there are at most two variables per equation can be represented by a graph in the following way: let each variable be a vertex in the graph and introduce an edge between two vertices if the corresponding variables appear in the same equation. It is clear that the connected components of the graph are independent subsystems of the system of equations. Hence, finding the optimum of the system of equations is equivalent to finding the optimum of each of the subsystems that corresponds to the connected components. To find the optimum of one such subsystem, choose a variable, x, and assign a value to it. This assignment will force assignments of values to every other variable in the subsystem. The optimum can be found by testing every possible assignment of values to x. If this is done for every independent subsystem the optimum for the entire system of equations will be found in polynomial time.

We have given tight approximability results for the maximum solution equation problem over finite abelian groups. One natural generalisation of our work might be to investigate the (in)approximability of this problem when the variables are constrained by some other relation than equations over a finite group. This perspective leads to a family of Constraint Satisfaction Problems that are parameterised on the constraint family. From this point of view there are many open problems. A start might be to try to characterise which constraint families give rise to tractable problems.

Bibliography

[1] Sanjeev Arora. *Probabilistic checking of proofs and hardness of approximation problems*. PhD thesis, 1995.

[2] Sanjeev Arora, Laszlo Babai, Jacques Stern, and Z. Sweedyk. The hardness of approximate optima in lattices, codes, and systems of linear equations. *J. Comput. Syst. Sci.*, 54(2):317–331, 1997.

[3] Daniel Pierre Bovet and Pierluigi Crescenzi. *Introduction to the theory of complexity*. Prentice Hall International (UK) Ltd., 1994.

[4] Jehoshua Bruck and Moni Naor. The hardness of decoding linear codes with preprocessing. *IEEE Transactions on Information Theory*, 36(2):381–385, 1990.

[5] Pierluigi Crescenzi, Riccardo Silvestri, and Luca Trevisan. To weight or not to weight: Where is the question? In *ISTCS 96': Proceedings of the 4th Israeli Symposium on Theory of Computing and Systems*, pages 68–77. IEEE, 1996.

[6] Lars Engebretsen, Jonas Holmerin, and Alexander Russell. Inapproximability results for equations over finite groups. *Theor. Comput. Sci.*, 312(1):17–45, 2004.

[7] Uriel Feige and Michel Goemanst. Approximating the value of two power proof systems, with applications to max 2sat and max dicut. In *ISTCS '95: Proceedings of the 3rd Israel Symposium on the Theory*

of Computing Systems (ISTCS'95), page 182, Washington, DC, USA, 1995. IEEE Computer Society.

[8] Uriel Feige and Daniele Micciancio. The inapproximability of lattice and coding problems with preprocessing. *J. Comput. Syst. Sci.*, 69(1):45–67, 2004.

[9] Mikael Goldmann and Alexander Russell. The complexity of solving equations over finite groups. *Inf. Comput.*, 178(1):253–262, 2002.

[10] Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001.

[11] Johan Håstad. Personal communication, 2005.

[12] Thomas W. Judson. *Abstract Algebra, Theory and Applications*. PWS Publishing Company, 1994.

[13] Sanjeev Khanna, Madhu Sudan, Luca Trevisan, and David P. Williamson. The approximability of constraint satisfaction problems. *SIAM J. Comput.*, 30(6):1863–1920, 2000.

[14] Ondrej Klíma, Pascal Tesson, and Denis Thérien. Dichotomies in the complexity of solving systems of equations over finite semigroups. Technical Report TR04-091, Electronic Colloq. on Computational Complexity, 2004.

[15] Benoit Larose and Laszlo Zadori. Taylor terms, constraint satisfaction and the complexity of polynomial equations over finite algebras. Submitted for publication.

[16] Cristopher Moore, Pascal Tesson, and Denis Thérien. Satisfiability of systems of equations over finite monoids. In *Mathematical Foundations of Computer Science 2001, 26th International Symposium, MFCS 2001*, volume 2136 of *Lecture Notes in Computer Science*, pages 537–547. Springer, 2001.

[17] Gustav Nordh. The complexity of equivalence and isomorphism of systems of equations over finite groups. In *Mathematical Foundations of*

Computer Science 2004, 29th International Symposium, MFCS 2004, volume 3153 of *Lecture Notes in Computer Science*, pages 380–391. Springer, 2004.

[18] Gustav Nordh and Peter Jonsson. The complexity of counting solutions to systems of equations over finite semigroups. In *Computing and Combinatorics, 10th Annual International Conference, COCOON 2004*, volume 3106 of *Lecture Notes in Computer Science*, pages 370–379. Springer, 2004.

[19] Kenneth H. Rosen. *Elementary Number Theory and its Application*. Addison-Wesley, 4 edition, 2000.

[20] Larry Stockmeyer. Planar 3-colorability is polynomial complete. *SIGACT News*, 5(3):19–25, 1973.

[21] Per-Anders Svensson. *Abstrakt algebra*. Studentlitteratur, 2001.

[22] Pascal Tesson. *Computational Complexity Questions Related to Finite Monoids and Semigroups*. PhD thesis, 2003.

[23] Eric W. Weisstein. Asymptotic notation. In *MathWorld*. Wolfram Research, Inc., 2005.

Avdelning, Institution
Division, Department

TCSlab,
Dept. of Computer and Information Science
581 83 LINKÖPING

Datum
Date

2005-06-01

Språk
Language

☐ Svenska/Swedish
☒ Engelska/English
☐ ______

Rapporttyp
Report category

☐ Licentiatavhandling
☒ Examensarbete
☐ C-uppsats
☐ D-uppsats
☐ Övrig rapport
☐ ______

ISBN
—

ISRN
LITH-IDA-EX--05/049--SE

Serietitel och serienummer
Title of series, numbering

ISSN
—

URL för elektronisk version

http://www.ep.liu.se/exjobb/ida/2005/dd-d/049/

Titel Snäva approximerbarhetsresultat för maxlösningsproblemet över abelska grupper

Title Tight Approximability Results for the Maximum Solution Equation Problem over Abelian Groups

Författare Fredrik Kuivinen
Author

Sammanfattning
Abstract

In the maximum solution equation problem a collection of equations are given over some algebraic structure. The objective is to find an assignment to the variables in the equations such that all equations are satisfied and the sum of the variables is maximised. We give tight approximability results for the maximum solution equation problem when the equations are given over finite abelian groups. We also prove that the weighted and unweighted versions of this problem have asymptotically equal approximability thresholds. Furthermore, we show that the problem is equally hard to solve as the general problem even if each equation is restricted to contain at most three variables and solvable in polynomial time if the equations are restricted to contain at most two variables each. All of our results also hold for the generalised version of maximum solution equation where the elements of the group are mapped arbitrarily to non-negative integers in the objective function.

Nyckelord
Keywords systems of equations, finite groups, NP-hardness, approximation