

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

Order Number 9130343

Parallel computational methods in integer linear programming

Lin, Yu, Ph.D.

City University of New York, 1991

Copyright ©1991 by Lin, Yu. All rights reserved.

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

A

PARALLEL COMPUTATIONAL METHODS IN INTEGER LINEAR PROGRAMMING

by

YU LIN

A dissertation submitted to the Graduate Faculty in Computer
Science in partial fulfillment of the requirements for the degree
of Doctor of Philosophy, The City University of New York.

1991

© 1991

YU LIN

ALL RIGHTS RESERVED

This manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

March 19, 1991

Victor Pan

Date

Chair of Examining Committee

March 19, 1991

T. C. Wesselkamper

Date

Executive Officer

Professor Theodore Brown

Professor Kenneth McAloon

Professor Victor Pan (Chair)

Professor Thomas Wesselkamper

Supervisory Committee

The City University of New York

ABSTRACT

PARALLEL COMPUTATIONAL METHODS IN INTEGER LINEAR PROGRAMMING

by

YU LIN

Advisor: Professor Victor Pan

Our aim in this thesis is to tackle the most significant unknown questions about parallel computational methods in integer linear programming. These questions involve two important groups of problems: problems with a fixed number of variables and problems with equality constraints only (linear Diophantine equations). One asks whether these classes of problems are in NC, i.e., whether they can be solved by a log-space uniform system of polynomially many parallel processors in polylog time. In both cases, we obtain partial answers to those questions.

In the first case, we concentrate on the problems where the number of free variables is two. We show that integer linear programming with two variables is NC-equivalent to a strong form of the greatest common divisor problem: all the intermediate results of Euclidean algorithm plus the output of an iterated modulo function with the remainder sequence from the Euclidean algorithm as arguments. The best previously known results are special cases of this. Furthermore, we show that the iterated modulo function with general arguments is P-complete.

In the second case we obtain local results (in the sense of number theory). We show that solving linear equations over discrete valuation ring (with adequate computability conditions) is in ZNC. This has application to the linear Diophantine equation problem, as well as to other problems. In particular, we show that vector p -adic approximation and finding a p -component of a finitely generated Abelian group are in ZNC. We also show that finding a maximal independent set in a linear matroid is in NC.

ACKNOWLEDGEMENTS

I wish to express my deep gratitude to Professor Victor Pan for his guidance, help, and encouragement. Professor Pan spent an enormous amount of time and energy with me during the preparation of this thesis. He gave many helpful corrections and suggestions from the first draft to the final version. He also encouraged and supported my efforts through countless hours of discussions.

I am indebted to Professor Thomas Wesselkamper, who, after reading my first research attempts, urged me to clarify many points and thus helped me make my ideas more precise and rigorous. Professor Wesselkamper also spent many hours, reading this thesis, helped me to correct the language errors, and provided great assistance in complying with the formal requirements.

My thanks also to Professor Theodore Brown and Professor Kenneth McAloon, my supervisory committee, for their efforts and time.

I owe my many thanks to Professors James Cox, Rohit Parikh, Carol Tretkoff, Stathis Zachos for seminars and discussions which have been of valuable inspiration in the early research related to this thesis.

TABLE OF CONTENTS

1. Introduction	1
1.1 Integer Linear Programming and Parallel Computation	1
1.2 Previous Known Results	2
1.3 Main Results of this Thesis	7
Part A: Integer Linear Programming with Two Variables (2-ILP)	16
2. The Relative Euclidean Algorithm	17
2.1 Introduction	17
2.2 The Relative Euclidean Algorithm	18
2.3 Approximation Lemmas	23
2.4 REUGCD and Opt-2-ILP are NC-equivalent	29
2.5 Conclusion and the Hashing Function	46
3. The Complexity of the Hashing Function	47
3.1 The Definition of the Hashing Function MOD*	48
3.2 MOD* is P-complete	48
3.3 A Special Model of the Turing Machine	48
3.4 Proof of Theorem 3.2.1	52
3.5 An Example	54
Part B: Linear Diophantine Equations: the Local Methods	57
4. A ZNC Algorithm for Solving Systems of Linear Equations over an NC-Constructible Discrete Valuation Ring	58
4.1 Preliminaries	58
4.2 Complexity and NC-constructible Discrete Valuation Rings	64
4.3 Systems of Linear Equations over a Discrete Valuation Ring	66
4.4 The Basis of a Column Space over a Discrete Valuation Ring	68
4.5 The Gram Determinant	71
4.6 An Algorithm for Finding the Basis of a Column Space over a Discrete Valuation Ring	72
4.7 Discussion of the Algorithm	75
5. Applications: Parallel Algorithms for Related Problems	77
5.1 Introduction	77
5.2 Solving Systems of Local Linear Diophantine Equations is in ZNC	77
5.3 Finding a Minimum Weighted Basis in a Linear Matroid is in NC	80

5.4 Finding the Basis of $\text{col}(A) \cap \text{col}(B)$ is in ZNC	84
5.5 p-adic Approximation is in ZNC	88
5.6 Finding a Coherent Basis of two Free R-Modules $A \subseteq B$ over an NC-constructible Discrete Valuation Ring is in ZNC	95
5.7 Finding the p-component of an Abelian Group is in ZNC	102
6. Open Problems and Future Research	107
Appendix	109
Bibliography	114

LIST OF FIGURES

Figures	
2.1	20
2.2	29
2.3	31
2.4	33
2.5	33
2.6	35
2.7	37
2.8	38
2.9	40
2.10	42
2.11	43
2.12	45
3.1	49
3.2	50
3.3	51
3.4	51

Chapter 1

INTRODUCTION

1.1. Integer Linear Programming and Parallel Computation

In this thesis, we investigate parallel computational methods for problems of integer linear programming. This is a problem which has attracted many researchers because of its significant applications in many areas. However, little is known about its parallel complexity. The general integer linear programming problem (abbreviated as ILP) is NP-complete ([2], [26], [33], [54]). Parallel computational methods, on the other hand, are designed to speed up polynomial algorithms to polylog time at the price of employing polynomially many processors in parallel. The class of problems solvable in this way is called NC. The analogous classes of problems, in the solution of which, random elements are allowed in different settings are called RNC (the class of problems that can be solved by NC algorithms with one-side error), and ZNC (the class of problems that can be solved by NC algorithms with zero error). We will give more precise definitions of RNC and ZNC in the beginning of the next section. In the theory of parallel computation, a problem is considered *well solved* if it is in NC or RNC or ZNC ([19], [24], [40]).

We concentrate on those cases of integer linear programming that are known to have polynomial algorithms, and we explore the relevant parallel computational methods. There are two prominent classes of polynomial time ILP problems:

(A) ILP Problems with a fixed number of variables, and

(B) ILP Problems with equality constraints (systems of linear Diophantine equations).

The polynomial algorithms for the class (A) are given by Lenstra in 1983 ([32], [37], [44]). The problems of class (B) are also in P. To see this, we recall the theorem that every matrix over $\mathbf{Z}$ is equivalent to a diagonal matrix ([9]). The proof of this theorem given in [9], Chapter XI, gives the desired algorithm.

We study these two groups of problems from the point of view of parallel computation. In section 2 of this introduction, we review the previously known results in these two areas and define the related problems. In section 3, we summarize the main results of our research. Open problems and conjectures are presented in Section 4.

1.2. Previously Known Results

The model of parallel computation we use is the shared memory parallel random access machine, known for short as the PRAM. In this model, one is given an arbitrary large collection of identical processors (random access machines) and a separate collection of memory cells; any processor can access any memory cell in unit time. The processors are assumed to know the size of their input, but the program controlling them is the same for all sizes of the input. The number of processors used for a given computation may vary depending on the size of the input, but may not change during the computation.

The above definition still admits variations in details (we shall not survey the different possibilities, see [24], [40], [57]). For our purpose, we make the following specifications:

(a) We work with a PRIORITY model, i.e., each processor is labelled by a different natural number and the processor with the lowest number has the priority in the case of several processors trying to write into the same cell. Concurrent reading is allowed (known as CREW model).

(b) The instruction set of the RAMs consists of additions and subtractions, binary boolean operations, left and right shifts, and conditional branching on zero. Indirect memory access is also allowed.

The class of all problems with integer parameter n (the size of the input) which can be solved in time $T(n)$ by a PRAM consisting of $P(n)$ processors is denoted by $\text{PRAM}(P(n), T(n))$. The class NC, which is of main interest to us, is defined as follows:

$$\text{NC} = \text{PRAM}(\text{poly}(n), \text{polylog}(n))$$

More precisely, the right hand side is the union of all the classes $\text{PRAM}(O(n^a), O(\log^b n))$ where a, b run through the natural numbers, and $\log^b n = (\log n)^b$.

If we want to allow randomness in parallel computation, we endow each processor with its own random number generator, by which it can draw independently, in constant time, a number from the numbers fitting in a memory cell with a uniform probability distribution. These numbers are private; if a processor wants to share its random number with another processor it must write it to a memory cell. The numbers drawn at individual processors are independent.

When analyzing the complexity of randomized parallel algorithms, it is usual to consider problems which allow the answer "There Is No Solution". For example, the problem of finding a perfect matching in a bipartite graph is of this type. A *zero error*

(Z) algorithm gives the correct answer to the given problem (i.e., whether there is a solution or not) with probability greater than $\frac{1}{2}$ and reports failure otherwise. A *one-sided error (R) algorithm* gives a solution with probability greater than $\frac{1}{2}$, provided that a solution exists; it reports failure in any other case ([24], [40]). We define classes $ZPRAM(P(n), T(n))$, $RPRAM(P(n), T(n))$ analogously as $PRAM(P(n), T(n))$, replacing classical algorithms by zero-error (respectively one-side error) randomized algorithms. We now put

$$ZNC = ZPRAM(poly(n), polylog(n))$$

$$RNC = RPRAM(poly(n), polylog(n))$$

As is usual in complexity theory, we define a $(P(n), T(n))$ -reduction in a PRAM model from a problem A_1 to a problem A_2 as follows. In addition to its usual power, the PRAM has an extra access to an oracle which solves the problem A_2 for an input currently in a memory cell in one unit of time. Then, for an instance I_1 of A_1 of input size n , the PRAM gives the answer to I_1 within the time bound $T(n)$ and processor bound $P(n)$. NC-reduction is defined as $(P(n), T(n))$ -reduction where $P(n)$ is polynomial and $T(n)$ is polylogarithmic ([40]).

The complexity class P is defined as the set of problems solvable in polynomial time by a deterministic Turing machine. A P-complete problem is a problem A with the property that every problem in P has an NC-reduction to A . Two problems A and B are NC-equivalent iff A has an NC-reduction to B and B has an NC-reduction to A .

The notion of P-completeness was defined by analogy with the older notion of NP-completeness, which has proved to be of great use in computer science. One can show that a problem A is P-complete if any problem in P can be reduced to A by a (sequential) log-space algorithm (defined via Turing machine using log storages). In this case, we sometimes speak of problems that are P-complete with respect to log-space reduction.

It is possible to view P-complete problems as the polynomial problems "most unlikely to be in NC" in a way similar to that in which NP-complete problems are those in NP which are "most unlikely to be in P". There is, of course, similarity as in the P-NP situation, the probability of the existence of "intermediate" problems which are neither in NC nor P-complete.

The greatest progress in the parallel treatment of the ILP problems with a fixed number of variables was achieved in the case of 2 variables. *Integer linear programming* (ILP) is the problem of computing an integer vector x that maximizes $\{c^T x \mid Ax \leq b\}$, where the entries of A , b , and c are given integers. We use k -ILP, for all $k \geq 1$, to denote the set of integer linear programming problems with at most k variables.

In the studies of k -ILP, researchers in the field have tried to prove that k -ILP is either in NC, or it is P-complete. However, it is not even known whether 2-ILP, the simplest non-trivial case of a k -ILP, is in NC. Deng noted in [21] and [22] that a special case of 2-ILP, the greatest common divisor of 2 integers (GCD), is not known to be in NC, either. In investigating the complexity of the GCD, one naturally turns to the oldest approach to the problem, the Euclidean algorithm. Deng has proved that the

problem of computing all the intermediate results of Euclidean algorithm (EUGCD) is NC-reducible to 2-ILP ([21], [22]). However, it is not even known whether computing the EUGCD is in NC. Therefore, it is interesting to look for those parallel algorithms for 2-ILP which use certain advanced arithmetic subroutines (such as GCD) as basic operations. The most optimistic approach is to conjecture the existence of an NC-reduction from 2-ILP to GCD. This is equivalent to establishing an NC-equivalence relation between GCD and 2-ILP.

Systems of linear Diophantine equations (SLDE) have many applications, an outstanding one being the classification of finitely generated abelian groups [9]. Similarly to 2-ILP, the general SLDE problem includes computing the greatest common divisor problem as a special case, which means that its solution is not easier than the solution of the GCD. One may therefore approach the SLDE problem by trying to reduce it to GCD. This, however, is not the approach adopted in this thesis. We prefer to break up the SLDE problem into "local parts" (defined in the next section) and to find fast parallel algorithms for those parts (rather than just reductions of SLDE to other problems).

The SLDE problem belongs to a more general class of problems: systems of linear equations (SLE) and other operations of linear geometry (LG) over a ring R [6]. This is a very general and complicated class of problems. The simplest case for which a very simple sequential polynomial algorithm for LG is known (the Gauss elimination), is the case in which R is a field. Gauss elimination works in an arbitrary field regardless of its characteristic or its being ordered. The method consists of choosing pivots and eliminating the entries of the columns (respectively, rows), one at a time, of

a given coefficient matrix, just as when applying the Gauss elimination method for the real numbers ($\mathbf{R}$). NC algorithms for basic problems of linear geometry over fields are known, due to Berkowitz [8], Chistov [16], Csanky [20] and Pan [51], [53].

1.3. Main Results of this Thesis

The present thesis consists of two parts. Part A is devoted to integer linear programming with two variables. Part B is devoted to linear Diophantine equations.

Part A: This part consists of two chapters. In *Chapter 2*, we reduce 2-ILP to a stronger version of EUGCD: the full output of EUGCD, together with the output of the iterated modulo function MOD^* with arguments $(c, a_1, a_2, \dots, a_n)$, where $(a_1, a_2, \dots, a_n)$ is the remainder sequence of the EUGCD and c is an arbitrary (fixed) number. MOD^* is defined as:

$$MOD^*(x, x_1, x_2, \dots, x_n) = (((\dots ((x \bmod x_1) \bmod x_2) \dots) \bmod x_n)$$

We call this problem the relative Euclidean algorithm problem, REUGCD. We show that, opt-2-ILP, the optimization version (in the sense of [54]) of 2-ILP and REUGCD are NC-equivalent (denoted by $\text{Opt-2-ILP} \equiv_{NC} \text{REUGCD}$).

Furthermore, we can prove that the iterated modulo function MOD^* with arbitrary arguments is P-complete. In *Chapter 3*, we show that MOD^* is P-complete (in effect, P-complete with respect to log-space) by simulating a Turing machine computation through an evaluation of MOD^* with arbitrary arguments.

This, of course, is not a proof that 2-ILP is P-complete, since the sequence $(a_1, a_2, \dots, a_n)$ is not a general one. REUGCD does not evaluate MOD^* with general

arguments. We may say that the question of complexity of 2-ILP reduces to asking "How general are the sequences produced by REUGCD?".

Part B: We proceed to treat the problem of SLE and LG in rings which have, in a way, the next simplest structure to fields: the so-called *discrete valuation rings* ([6], to be defined later in this section). We find ZNC algorithms for many relevant SLE and LG problems over discrete valuation rings. We begin with some basic definitions necessary to describe adequately the results of Part B of the present thesis. The definitions given in this section are generally known (with the exception of the concept of an NC-constructible ring). The corresponding references are included together with the introduction of the concepts.

To avoid confusion, from now on by $\mathbf{Z}$, $\mathbf{Q}$, $\mathbf{R}$ (bold face) we mean the rings of integers, rationals and reals respectively; plain text letters, including Z , Q and R , mean other entities.

Let Q be a field. Let $Q^* = Q - \{0\}$ be the multiplicative group of Q ; and let $\mathbf{Z}$, be the set of integers. A *discrete valuation* on Q (see [6]) is a function $v: Q^* \rightarrow \mathbf{Z}$ with the following properties:

- (a) $v(ab) = v(a) + v(b)$
- (b) $v(a+b) \geq \min\{v(a), v(b)\}$
- (c) v is onto

To avoid restrictions on a, b in (a), (b), we put formally $v(0) = +\infty$ to extend v to the whole of Q and the range of v to $\mathbf{Z} \cup \{+\infty\}$. We define the *discrete valuation ring* with respect to Q, v to be

$$R = \{x \in Q \mid v(x) \geq 0\}.$$

(Now we consider $+\infty$ to belong to the range of v because we want 0 to belong to R .)

The above definition should be understood as follows: whenever we have a field Q and a function $v: Q^* \rightarrow \mathbb{Z}$ satisfying (a), (b), (c), we call v a discrete valuation on Q . The ring R as defined above is the corresponding valuation ring. Thus, only a function defined on a field may be called a discrete valuation according to our definition (this follows the terminology of Atiyah and MacDonald [6]). Note that we do not actually assert that for a given field Q , a discrete valuation v exists. Nor do we say that if it exists, it is unique. In order for a valuation on Q to exist, Q has to be infinite (by (c)). We do not, however, assert that every infinite field has a valuation. There are fields of characteristic $p > 0$ which have valuations. For example, let $Q = \mathbb{Z}_p(x)$ (the field of rational functions in one indeterminate x over the residue field $\mathbb{Z}_p$; $\mathbb{Z}$, the set of integers, and p a prime). Then Q has characteristic p . A valuation v on Q is given by

$$v\left(\frac{P(x)}{S(x)}\right) = \max \{n : x^n \mid P(x)\} - \max \{m : x^m \mid S(x)\}$$

where $P(x)$ and $S(x)$ are polynomials over $\mathbb{Z}_p$, x not identically zero.

If a valuation v exists on a field Q , it is typically not unique. As an example, let $Q = \mathbb{Q}$ be the set of rationals, p a prime, a valuation on $\mathbb{Q}$ is defined as

$$v\left(\frac{a}{b}\right) = \max \{i : p^i \mid a\} - \max \{j : p^j \mid b\}$$

where a, b are integers and $b \neq 0$. The local corresponding discrete valuation ring (called *integers localized at p*) is

$$\mathbb{Z}_{(p)} = \left\{ \frac{a}{b} \mid a, b \in \mathbb{Z} \text{ \& } \neg (p \mid b) \right\}.$$

Every prime p gives a different valuation on $\mathbb{Q}$. Linear equations over the ring $\mathbb{Z}_{(p)}$ may be called "local linear Diophantine equations", or "linear Diophantine equations one prime at a time".

The reason we think that discrete valuation rings are "simple" is that each of the discrete valuation rings has only one prime ideal.

A *matroid* ([54]) (E, σ) is a system σ of subsets of a finite set E such that the following axioms hold:

- (1) σ is closed under subset operation: i.e., if $A \in \sigma$, and $B \subseteq A$, then $B \in \sigma$.
- (2) for each $M \subseteq E$, any two inclusion-maximal sets $X, Y \in \sigma$ with the property $X, Y \subseteq M$ satisfy $|X| = |Y|$, where $|X|$ denotes the cardinality of X .

A *basis* of a matroid (E, σ) is any inclusion-maximal $X \in \sigma$.

A matroid (E, σ) is called *graphic* if there is a graph $G=(V, E)$, such that σ is the set of forests of graph G . A matroid (E, σ) is called *linear* if there is a field F and a matrix A over F such that E is the set of columns of A and σ is the system of all linearly independent sets of columns of A . It can be proved that any graphic matroid is linear (since a graph can be defined by its incidence matrix).

We may now start describing the main results of Part B of this thesis.

In *Chapter 4*, we present a ZNC algorithm for finding a basis of an R -module generated by column vectors of a matrix over a discrete valuation ring R . This leads to a ZNC algorithm for solving systems of linear equations over discrete valuation rings. Because we are dealing with general discrete valuation rings R , we have to talk about the way elements of R will be represented in a binary code. We deal with this techni-

cal question by specializing to rings R whose elements may be represented in the binary code in a way satisfying certain basic axioms. Such rings R are called *NC-constructible* (see Section 4.2). The basic axioms of NC-constructible rings R are as follows. Let Q be the fraction field of R .

- (1) Computing a product $A_1 \cdots A_n$ of n matrices $A_1, \dots, A_n$ over $Q[x]$ (the ring of polynomials with one variable over the field Q) is in NC;
- (2) Computing the valuation of an element $a \in Q$ is in NC;
- (3) For an encoding of size n of an element $a \in Q$, we may find in NC an encoding of size $n+c$ of $-a$, where c is a constant;
- (4) For an encoding of size n of an element $a \in Q$, $|v(a)| \leq cn$ (where c is a constant).

Chapter 5 is devoted to the application of the algorithm given in *Chapter 4*. The basic application is given in Section 5.2: we show that solving systems of linear Diophantine equations in $\mathbf{Z}_{(p)}$ is in ZNC.

In section 5.3, we present an NC algorithm for finding a Basis of the Minimum Weight (BMW) in a linear matroid. The problem of finding a BMW in a matroid (E, σ) is specified by giving a function $h: E \rightarrow \mathbf{N}$ (the set of all non-negative integers). A basis of the minimum weight is then a basis such that

$$h(x) = \sum_{e \in X} h(e)$$

is minimal possible. NC algorithms for BMW in a graphic matroids are known, e.g., see [19]. It seems to be an interesting problem whether BMW over an arbitrary matroid is in NC.

Sections 5.4, 5.5, 5.6 are devoted to finding ZNC algorithms to certain basic problems of linear geometry over discrete valuation rings. In Section 5.4, we show that finding a basis of the intersection of column spaces of two matrices A, B over an NC-constructible (see the definition later) discrete valuation ring R is in ZNC. This problem is defined as follows:

We assume that A, B are two matrices over a discrete valuation ring R with the same number of rows. The column space of A (respectively B) is defined as the set of all R -valued (here R is the discrete valuation ring mentioned above) linear combinations of the columns of A (respectively B). A basis of the intersection is a finite set S of column vectors over R of the same dimension as the number of rows of A, B , such that every common element of both column spaces of A, B can be expressed uniquely as an R -valued linear combination of elements of S . (Such a basis may not exist if R is an arbitrary ring, but does exist if R is a discrete valuation ring.)

In Section 5.5, we show that a (generalized) p -adic approximation is in ZNC. Let $x = (x_1, \dots, x_k)^T$ be a column vector over an NC-constructible discrete valuation ring R with valuation v . We define the valuation of x to be

$$v(x) = \min\{v(x_i) \mid 1 \leq i \leq k\}$$

The generalized p -adic approximation problem is defined as follows: Given columns $a_1, \dots, a_n, b$ of dimension k over R , find coefficients $\lambda_i \in R$ such that

$$v(b - \sum_{i=1}^n \lambda_i a_i) \text{ be maximal possible.}$$

We find a ZNC-algorithm for solving this problem. As an application, we consider the classical p -adic approximation problem:

Let $a_1, \dots, a_n, b$ be k -dimensional column vectors over $\mathbf{Z}$. Find integers $\lambda_i \in \mathbf{Z}$ such that $b - \sum_{i=1}^n \lambda_i a_i$ is divisible by the largest possible power p^j of p (p prime). It may occur that no maximal j exists: There may be systems $(\lambda_1, \dots, \lambda_n)$ of integers which make $b - \sum_{i=1}^n \lambda_i a_i$ divisible by arbitrary large powers of p . We have a ZNC-algorithms which either finds the optimal system $(\lambda_1, \dots, \lambda_n)$ or reports that no such system exists.

In Section 5.6, we present ZNC algorithm for finding a coherent basis of two free R -modules $M_1 \subseteq M_2$ over an NC-constructible discrete valuation ring R . Let $M_1 \subseteq M_2$ be free R -module, R a discrete valuation ring. It can be shown (see [9], Chapter XI) that there exists a basis B of M_2 and a mapping $g: B \rightarrow \mathbf{N}_0 \cup \{+\infty\}$ such that

$$\{p^{g(b)}b \mid b \in B \& g(b) \neq \infty\}$$

is a basis of M_1 . Such a basis B is called a coherent basis of the free R -module M_2 with respect to the submodule M_1 . We have a ZNC algorithm for finding a coherent basis for two free R -modules $M_1 \subseteq M_2$ given as column spaces of a matrix over R and a submatrix with the same number of rows.

In Section 5.7, we show that finding the p -component of an arbitrary finitely generated Abelian group is in ZNC. More precisely, this problem is defined as follows:

A finitely generated Abelian group A is given by a system of generators $z_1, \dots, z_n$ and by the relations

$$a_{11}z_1 + \dots + a_{1n}z_n = 0$$

(*)

$$a_{m_1}z_1 + \cdots + a_{m_n}z_n = 0$$

(Actually, A is the quotient group of the free Abelian group over the generators $z_1, \dots, z_n$, when factorized through the smallest congruence generated by the relations (*).) It can be shown (see [9]) that for each such A , there is a sequence of non-negative integers $m_1, \dots, m_n$, such that for $1 \leq i \leq n-1$, $m_i \mid m_{i+1}$ and A is isomorphic to

$$\mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_n}$$

here each m_i is allowed to be 0 and, if any $m_i = 0$, then all $m_j = 0$ for all $i < j \leq n$. By convention, $0 \mid 0$ and $\mathbb{Z}_0 = \mathbb{Z}$.

The p -component $A_{(p)}$ (p is prime) of A provides partial information about A . It is defined as follows:

Let $k \leq n$ be the largest such that $m_k \neq 0$. Then $A_{(p)}$ is isomorphic to

$$\mathbb{Z}_{p^{v_1}} \times \cdots \times \mathbb{Z}_{p^{v_k}} \times \mathbb{Z}_{(p)} \times \cdots \times \mathbb{Z}_{(p)}$$

where p^{v_i} is the largest power of p dividing m_i . We show that finding the above expression for $A_{(p)}$ from (*) is in ZNC. That is, we find the number k and the numbers $v_1, \dots, v_k$. (Note that $\mathbb{Z}_{(p)}$ occurs $n-k$ times in the above expression.)

We end this section with pointing out the significance of having a fast algorithm for solving systems of linear Diophantine equations over $\mathbb{Z}_{(p)}$. Notice that $\mathbb{Z} \subseteq \mathbb{Z}_{(p)}$. Therefore if a system has a solution in $\mathbb{Z}$, it must have a solution in $\mathbb{Z}_{(p)}$. The converse is not true in general. The following implications, however, are true: Let A be the matrix of the system. Assume that A is square.

(i) If $\det(A)$ is a power of the prime p then the system has a solution over $\mathbb{Z}$ if and

only if it has a solution over $\mathbf{Z}_{(p)}$.

(ii) The system has a solution over $\mathbf{Z}$ if and only if it has a solution over $\mathbf{Z}_{(p)}$ for all primes p dividing $\det(A)$.

The problem of solving general systems of linear Diophantine equations is not known to be in NC or ZNC or RNC (recall that this includes the GCD problem). Our approach, however, shows that the decision version of the problem (existence of a solution) is in ZNC for systems whose determinants have an explicit decomposition into polynomially many prime factors. Such systems do occur in practice.

Part A:**Integer Linear Programming with Two Variables (2-ILP)**

Chapter 2

THE RELATIVE EUCLIDEAN ALGORITHM

2.1. Introduction

In this chapter we compare the parallel complexity of integer linear programming in dimension 2 with the parallel complexity of a stronger version of the Euclidean algorithm for finding the greatest common divisor of two integers.

Integer linear programming (ILP) is the problem of computing, for a fixed $m \times n$ matrix A , m -column b and n -column c , an integer vector x which maximizes $c^T x$ under the constraint $Ax \leq b$. We use the symbol k -ILP, for all $k > 0$, to denote integer linear programming with at most k variables. Integer linear programming is NP-complete [26] while k -ILP is in P [40].

In further studies of k -ILP, researchers in the field tried to prove that it is either in NC (the class of problems solvable in parallel in $O(\log^c n)$ time with $O(n^d)$ processors, see Section 1.2), or it is P-complete. However, it is not even known whether 2-ILP, the simplest non-trivial case of k -ILP is in NC or P-complete. This situation is similar to the dilemma we have with the problems that can not be proved to be either in P or NP-complete.

While the complexity of 2-ILP is still open, some work has been done on the relative question concerning the parallel time complexity of 2-ILP, i.e., the NC-reductions between 2-ILP and other related problems, whose complexity with respect

to P-complete problem or NC is also unknown. Among such problems the problem of computing the intermediate results of Euclidean algorithm for the greatest common divisor of two integers (EUGCD) has been studied by Anderson and Mayr [5], and Deng [21], [22] in the past.

We prove that the optimization version of linear programming (see [54]) with two variables, Opt-2-ILP, is NC-equivalent to a stronger version of EUGCD, which we call REUGCD (see next section for the definition). It was previously known that the original Euclidean algorithm problem, EUGCD, can be NC-reduced to Opt-2-ILP [21], [22]. In effect, EUGCD is a special case of our REUGCD. This can be shown directly, although it is not immediate in our setting.

2.2. The Relative Euclidean Algorithm

2.2.1. Definition: Given input $a, b, c \in \mathbb{N}$, and $a > b > 0$, we define the following recurrence relations:

$$(2.2.1.1.) \ a_0 = a, \ a_1 = b, \ a_{i+1} = \lceil \frac{a_{i-1}}{a_i} \rceil a_i - a_{i-1}$$

$$(2.2.1.2.) \ c_1 = c \bmod a, \ c_{i+1} = c_i - \lfloor \frac{c_i}{a_i} \rfloor a_i$$

$$(2.2.1.3.) \ k_0 = 0, \ k_1 = 1, \ k_{i+1} = \lceil \frac{a_{i-1}}{a_i} \rceil k_i - k_{i-1}$$

$$(2.2.1.4.) \ l_1 = 0, \ l_{i+1} = l_i + \lfloor \frac{c_i}{a_i} \rfloor k_i$$

Also, let

$$(2.2.1.5.) \ n = \min\{k \mid a_k = 0\}$$

$$(2.2.1.6.) S_1 = \{i \in \{1, \dots, n\} \mid \lceil \frac{a_{i-1}}{a_i} \rceil > 2\}, S_2 = \{i \in \{1, \dots, n\} \mid \lfloor \frac{c_i}{a_i} \rfloor > 0\}$$

2.2.2. Remarks:

(a) The graphical interpretation of the above definition is as follows: imagine a circle of circumference a , and two points "O" and "Z" on the circle. The relative distance from "O" to "Z" clockwise is $c \bmod a$. A frog, Y, jumps along the circle clockwise, moving a constant distance b ahead during each jump (one unit of time), starting from "O". the a_i 's are the successively improving approximations of Y to "O" from the right, the k_i 's are the discrete amount of time needed to achieve these approximations, the c_i 's represent the successively improving approximations of Y to "Z" from the left, the l_i 's are the corresponding discrete amount of time needed to achieve the approximation c_i 's (Figure 2.1).

(b) (2.2.1.1) is close to the classical version of the Euclidean algorithm

$$(2.2.1.7.) r_0 = a, r_1 = b, r_{i+1} = r_{i-1} - \lfloor \frac{r_{i-1}}{r_i} \rfloor r_i.$$

which we will call the "left algorithm", to be compared with the "right algorithm" (2.2.1.1).

There are direct (although not trivial) formulas relating the remainder sequence of the algorithm (2.2.1.1) and the algorithm (2.2.1.7). We choose to work with the algorithm (2.2.1.1), because of its close relation with Diophantine approximation (see Remark (a) and also Lemma 2.3.1 below).

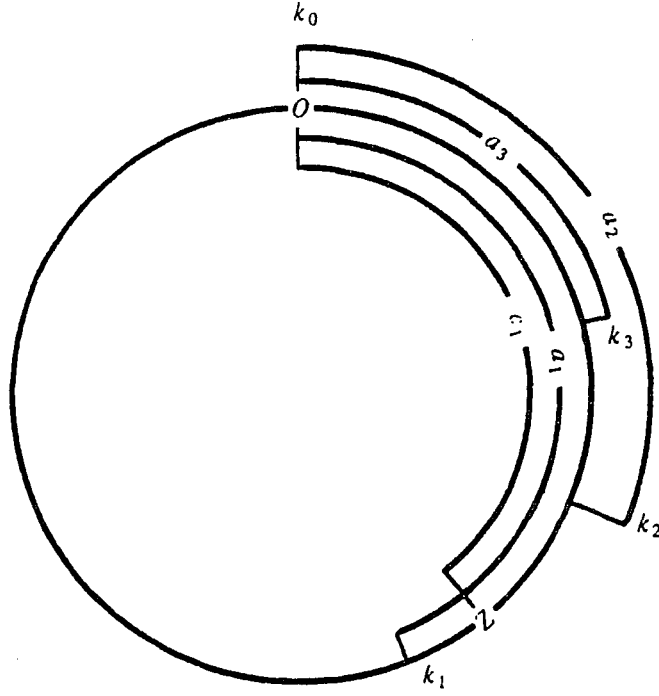


Figure 2.1. Circumference of the circle $=a=a_0$, $b=a_1$ and $c \bmod a=c_1$

To give an example of the connection between the different versions of the Euclidean algorithm, we show how the remainder sequence of (2.2.1.1) is expressed in terms of the remainder sequence of (2.2.1.7). We can show that the two sequences

$$a_1, a_2, a_3, a_4, \dots$$

and

$$r_1, r_1 - r_2, r_1 - 2r_2, \dots, r_1 - \left\lfloor \frac{r_1}{r_2} \right\rfloor r_2 = r_3,$$

$$r_3 - r_4, r_3 - 2r_4, \dots, r_3 - \left\lfloor \frac{r_3}{r_4} \right\rfloor r_4 = r_5,$$

$$r_5 - r_6, r_5 - 2r_6, \dots, r_5 - \left\lfloor \frac{r_5}{r_6} \right\rfloor r_6 = r_7,$$

coincide, so that $a_1 = r_1$, $a_2 = r_1 - r_2$, $\dots$, $a_{\left\lfloor \frac{r_1}{r_2} \right\rfloor} = r_3$, $a_{\left\lfloor \frac{r_1}{r_2} \right\rfloor + 1} = r_3 - r_4$, $\dots$.

The left algorithm converges in sequential linear time in the input size, that is, in the sequential time $O(\log(a+b))$ (see [43]). The left and right approaches can be combined into the following two-sided (an accelerated) Euclidean algorithm:

input: integers $a > b > 0$

output: $\gcd(a, b)$ and intermediate results

Euclidean Algorithm:

begin

$s_0 := a; s_1 := b; i := 0;$

repeat

$i := i + 1;$

output $s_{i+1} := \min(s_{i-1} - s_i \lfloor \frac{s_{i-1}}{s_i} \rfloor, s_i \lceil \frac{s_{i-1}}{s_i} \rceil - s_{i-1});$

until $s_{i+1} = 0;$

end;

(c) As a matter of fact, (2.2.1.3) defines a linear recurrence and thus can be immediately computed in NC given a sequence in (2.2.1.1). Similarly, (2.2.1.4) can be computed in NC (by means of the prefix sum algorithm) given the sequences (2.2.1.1), (2.2.1.2) and (2.2.1.3) (see [24], [40]).

(d) We have a problem with the right algorithm, however, if performed in a straightforward way, it may take exponential time in terms of $\log(a+b)$. This situation occurs when $\lceil \frac{a_{i-1}}{a_i} \rceil = 2$. For example.

$$a_0 = a = 2000, a_1 = b = 1999, a_2 = 1998, a_4 = 1997, \dots, \text{ then } n = 2000.$$

This can be resolved by forming a linear recurrence relation

$$(2.2.1.8.) \ a_{i+j} = 2a_{i+j-1} - a_{i+j-2}$$

The solution to this linear recurrence is the following linear function:

$$(2.2.1.9.) \ a_{i+j} = a_{i-1} - (j+1)(a_{i-1} - a_i)$$

Thus, using the values of a_{i-1}, a_i , we may calculate all values of a_{i+j} for $j \geq 1$ explicitly. Of course, these values only agree with the stages of the Euclidean algorithm while $\lceil \frac{a_{i-1}}{a_i} \rceil = 2$, i.e., $a_{i+j} \geq 0$. But the largest j for which $a_{i+j} \geq 0$ may be easily

found directly using (2.2.1.9):

$$j = \lfloor \frac{a_{i-1}}{a_{i-1} - a_i} \rfloor - 1$$

and $\lceil \frac{a_{i+j}}{a_{i+j+1}} \rceil$ will be greater than 2.

The sets S_1 and S_2 are introduced to prevent the algorithm from computing an exponential amount of redundant information. Knowing $\{(i, a_i, c_i) | i \in S_1 \cup S_2\}$, one may immediately restore (i, a_i, c_i) for $i \notin S_1 \cup S_2$ by an NC-algorithm.

Lemma 2.2.4 shows that S_1, S_2 are linear in size. As a corollary, our version of "one-sided" Euclidean algorithm, when improved by the use of linear recurrence, is in linear time.

2.2.3. Lemma: if $a_i \geq \frac{1}{2}a_{i-1}$ ($\lceil \frac{a_{i-1}}{a_i} \rceil \leq 2$) and $\lfloor \frac{c_i}{a_i} \rfloor > 0$, then $c_{i+1} < \frac{1}{2}c_i$.

Proof: We have $c_i = c_{i-1} - \lfloor \frac{c_{i-1}}{a_{i-1}} \rfloor a_{i-1}$, hence $c_i < a_{i-1}$, and $a_i \geq \frac{1}{2}a_{i-1} > \frac{1}{2}c_i$. Since

$c_{i+1} = c_i - \lfloor \frac{c_i}{a_i} \rfloor a_i < c_i - a_i < \frac{1}{2}c_i$, we have $c_{i+1} < \frac{1}{2}c_i$, i.e., c_i 's decrease by more than $\frac{1}{2}$

as long as a_i 's decrease by less than $\frac{1}{2}$. In another word, $S_1 \cup S_2$ remains linear in size as long as S_1 is linear in size. $\square$

2.2.4. Lemma: S_1 and S_2 are both linear in size.

Proof. $S_1 = \{i \in \{0, 1, \dots, n\} \mid \lceil \frac{a_{i-1}}{a_i} \rceil > 2\}$. By the definition of a_i 's, we know that

$|S_1| = O(\log a)$. From Lemma 2.2.3, $S_1 \cup S_2$ is linear in size as long as S_1 is linear in size, therefore S_2 is also linear in size. $\square$

2.2.5. Definition: We define the relative Euclidean algorithm problem (REUGCD) to compute the following information

$$\{(l_i, c_i, k_i, a_i) \mid i \in S_2\} \cup \{(l_n, c_n)\}, \text{ where } S_2 = \{i \in \{1, \dots, n\} \mid \lfloor \frac{c_i}{a_i} \rfloor > 0\}.$$

2.2.6. Theorem: $\text{REUGCD} \equiv_{NC} \text{Opt-2-ILP}$.

This is the major result of this section. The proof of 2.2.6 follows from the following lemmas and propositions. Note that we do not consider all the data in 2.2.1 a part of the output of REUGCD. It can, in effect, be shown that the data in 2.2.1 are equivalent to Opt-2-ILP.

2.3. Approximation Lemmas

The following two lemmas are reinterpretations of standard results in Diophantine approximation (see [15], [46]).

2.3.1. Lemma: Let a set $Q(a, b, c)$ consist of all the pairs of integers $(k, c - bk - al)$ such that $k \in \mathbb{N}_0$, $l \in \mathbb{Z}$, $c - bk - al \geq 0$ and the following condition is satisfied:

For any other pairs of integers $(\bar{k}, c - b\bar{k} - a\bar{l})$ such that $\bar{k} \in \mathbb{N}_0$, $\bar{l} \in \mathbb{Z}$, $c - b\bar{k} - a\bar{l} \geq 0$ and, moreover, $c - b\bar{k} - a\bar{l} < c - bk - al$, we have $\bar{k} > k$.

Then $Q(a, b, c)$ consists of all the pairs of integers $(l_i + qk_i, c_i - qa_i)$ where $i \in S_2$, $q \in \mathbb{N}_0$

and $q \leq \lfloor \frac{c_i}{a_i} \rfloor$ (see Definition 2.2.1).

Remark: We note that this lemma is essentially a mathematically rigorous way of stating the idea of Remark (a) in 2.2.2.

Proof of Lemma 2.3.1: To prove the statement of Lemma 2.3.1, we first prove the following. Let, for $x \in \mathbb{Z}$ and $y \in \mathbb{N}$, $x \bmod y$ denote the remainder of x modulo y (a number from $\{0, 1, \dots, y-1\}$). Let

$$Q(a, b) = \{k \in \mathbb{N} \mid (\forall \bar{k} \in \mathbb{N})(\bar{k} < k \rightarrow \bar{k}b \bmod a > kb \bmod a)\}.$$

Then

$$(2.3.1.1.) \quad Q(a, b) = \{k_i \mid 1 \leq i \leq n\}$$

$$(2.3.1.2.) \quad a_i = k_i b \bmod a \text{ for } i > 0.$$

We first prove (2.3.1.2). We proceed by induction on i . For $i=1, 2$ the statement is easy to check. Assuming it is valid for $i, i-1$, we have integers x, y such that

$$k_i b - xa = a_i$$

$$k_{i-1} b - ya = a_{i-1}$$

Multiplying the first equality by $\lceil \frac{a_{i-1}}{a_i} \rceil$ and subtracting the second one, we get

$$k_{i+1} b - (x \lceil \frac{a_{i-1}}{a_i} \rceil - y) a = a_{i+1}.$$

Since, by definition, $0 \leq a_{i+1} \leq a_i$, we have

$$a_{i+1} = k_{i+1}b \pmod{a}$$

as required. (2.3.1.2) is proved.

Now we prove (2.3.1.1). Note that, by (2.3.1.2), all we need to prove is

(2.3.1.3.) For $k_i < l < k_{i+1}$, $a_i < lb \pmod{a}$.

Now (2.3.1.3) will be proved by induction on i . Again, it is easy to check for $i=0$.

Now assume that (2.3.1.3) is valid for $i-1, i-2, \dots, 0$, but violated for i . Thus, there is an l such that

$$k_i < l < k_{i+1} \text{ and } a_i \geq lb \pmod{a}.$$

Select the largest $q \in \mathbb{N}$, such that

$$qk_i - k_{i-1} \leq l.$$

Note that $1 \leq q < \lceil \frac{a_{i-1}}{a_i} \rceil$. Then, by q being the largest,

$$(2.3.1.4.) \quad 0 \leq l - (qk_i - k_{i-1}) < k_i.$$

Now by assumption, there is a number z ,

$$(2.3.1.5.) \quad 0 \leq z \leq a_i, (k_i - l)b \equiv z \pmod{a}.$$

On the other hand, by (2.3.1.2), there is a number t ,

$$(2.3.1.6.) \quad a_i \leq t < a_{i-1}, (k_{i-1} - (q-1)k_i)b \equiv t \pmod{a}.$$

Subtracting (2.3.1.5) from (2.3.1.6), we get

$$a_{i-1} > t - z = (l - (qk_i - k_{i-1}))b \pmod{a},$$

which, together with (2.3.1.4), violates the induction hypothesis. Thus, (2.3.1.3) is proved and so is (2.3.1.1).

We are now ready to prove the statement of Lemma 2.3.1. Without loss of generality, $0 < l < a$. By (2.3.1.2) it suffices to prove the following.

(2.3.1.7.) Let $z \in \mathbb{N}_0$ and $zb \bmod a < c$, let $i \in \mathbb{N}_0$ be the smallest such that $a_i + (zb \bmod a) \leq c$ and let $t \in \mathbb{N}_0$ be smallest such that $t > z$ and $zb \bmod a \leq tb \bmod a \leq c$. Then $t = z + k_i$.

(Indeed, it is easy to see that the number z run through first coordinates of elements of $\mathcal{Q}(a, b, c)$. The calculation of the second coordinate is an easy induction analogous to our previous proof of (2.3.1.2).)

Now to prove (2.3.1.7), note that since t is assumed to be minimal $> z$ such that $zb \bmod a \leq tb \bmod a \leq c$ and since

$$(k_i + z)b \bmod a = a_i + (zb \bmod a) \leq c,$$

we definitely have $t \leq k_i + z$. Thus, all we need to show is

$$(2.3.1.8.) \quad t \geq k_i + z.$$

Suppose that $t < k_i + z$. Then

$$0 \leq (t - z)b \bmod a = tb \bmod a - zb \bmod a \leq c - (zb \bmod a) < a_{i-1}.$$

On the other hand, as assumed,

$$0 < (t - z) < k_i.$$

Now either $(t - z) < k_{i-1}$ which violates $k_{i-1} \in \mathcal{Q}(a, b)$, or $k_{i-1} < (t - z) < k_i$ which violates (2.3.1.3) for $l = t - z$. The contradiction proves (2.3.1.8) and consequently (2.3.1.7) and the statement Lemma 2.3.1. $\square$

For a set X in the plane, denote by $\text{CONV}(X)$ the convex hull of X .

2.3.2. Lemma: Let $C(a, b, c)$ be the set of all the pairs of integers (k, l) such that $k \geq 0$ and $bk + al \leq c$. Let, further, a set $\bar{C}(a, b, c)$ consists of all the points

$$V_i = (l_i, (c - c_i - bl_i)/a), i \in S_2 \cup \{n\}.$$

Denote by $\bar{\bar{C}}(a, b, c)$ the set of all the pairs $(z_1 + t, z_2 + u)$ where $(z_1, z_2) \in \bar{C}(a, b, c)$, $t \geq 0$ and $at + bu \leq 0$. Then

$$\text{CONV}(C(a, b, c)) = \text{CONV}(\bar{\bar{C}}(a, b, c)).$$

Remark: Actually, $\bar{\bar{C}}$ is obtained from $\bar{C}(a, b, c) = \{V_i | i \in S_2 \cup \{n\}\}$ by adding with each V_i every point in the angle $\{(x, y) | bx + ay \leq 0 \text{ \& } x \geq 0\}$. The set $C(a, b, c)$, on the other hand, contains all the integral points in the angle $\{(x, y) | by + ax \leq c \text{ \& } x > 0\}$.

The significance of Lemma 2.3.2 is that it characterizes the convex hull of the set of all integral points in an angle; this allows us to reduce a problem of 2-ILP to a problem of 2-LP. In fact, it shows that the optimum can be selected among the points V_i 's.

The situation we will be interested in, however, is still slightly more complicated, since the general 2-ILP problem reduces to 2-ILP problem whose feasibility sets are triangles, not angles.

Proof of Lemma 2.3.2: Let

$$\bar{Q}(a, b, c) = \{(z_1 + t, z_2 + u) | (z_1, z_2) \in Q(a, b, c) \text{ \& } t \geq 0 \text{ \& } at + bu \leq 0\},$$

where $Q(a, b, c)$ was defined in Lemma 2.3.1. Lemma 2.3.1 implies that

$$(2.3.2.1.) \quad C(a, b, c) \subseteq \bar{Q}(a, b, c).$$

On the other hand, obviously,

$$Q(a, b, c) \subseteq C(a, b, c).$$

and hence

$$(2.3.2.2.) \quad \overline{\overline{Q}} \subseteq \text{CONV}(C(a, b, c)).$$

(Here we are exploiting the fact that every angle whose vertex is integral and whose bounding lines have equations with integral coefficients is the convex hull of its integral points.)

Now (2.3.2.2.) implies

$$\text{CONV}(\overline{\overline{Q}}(a, b, c)) \subseteq \text{CONV}(C(a, b, c)),$$

where (2.3.2.1) implies

$$\text{CONV}(C(a, b, c)) \subseteq \text{CONV}(\overline{\overline{Q}}(a, b, c)).$$

Hence, $\text{CONV}(C(a, b, c)) = \text{CONV}(\overline{\overline{Q}}(a, b, c))$. It is easy to check that

$$\text{CONV}(\overline{\overline{C}}(a, b, c)) = \text{CONV}(\overline{\overline{Q}}(a, b, c)).$$

(for each element of $C(a, b, c)$ is on a line spanned by the points V_i, V_{i+1} for some i).

Lemma 2.3.2 is proved. $\square$

2.3.3. Remarks:

Lemma 2.3.1 and Lemma 2.3.2 are approximation theorems. Figure 2.2 gives the graphic forms of the lemmas. Lemma 2.3.1, in effect, is another way of viewing the circle: c_i 's are approximations of c by multiples of b modulo a :

$$c_i + bl_i = c \pmod{a}$$

$$c_i + bl_i + am = c \quad (m \text{ is an unknown integer, possibly negative})$$

$$c_i = c - bl_i - am$$

We would like $Q(a, b, c) = \{(l_i, c_i) | i \in S_2 \cup \{n\}\}$, but we have to account for the intermediate steps.

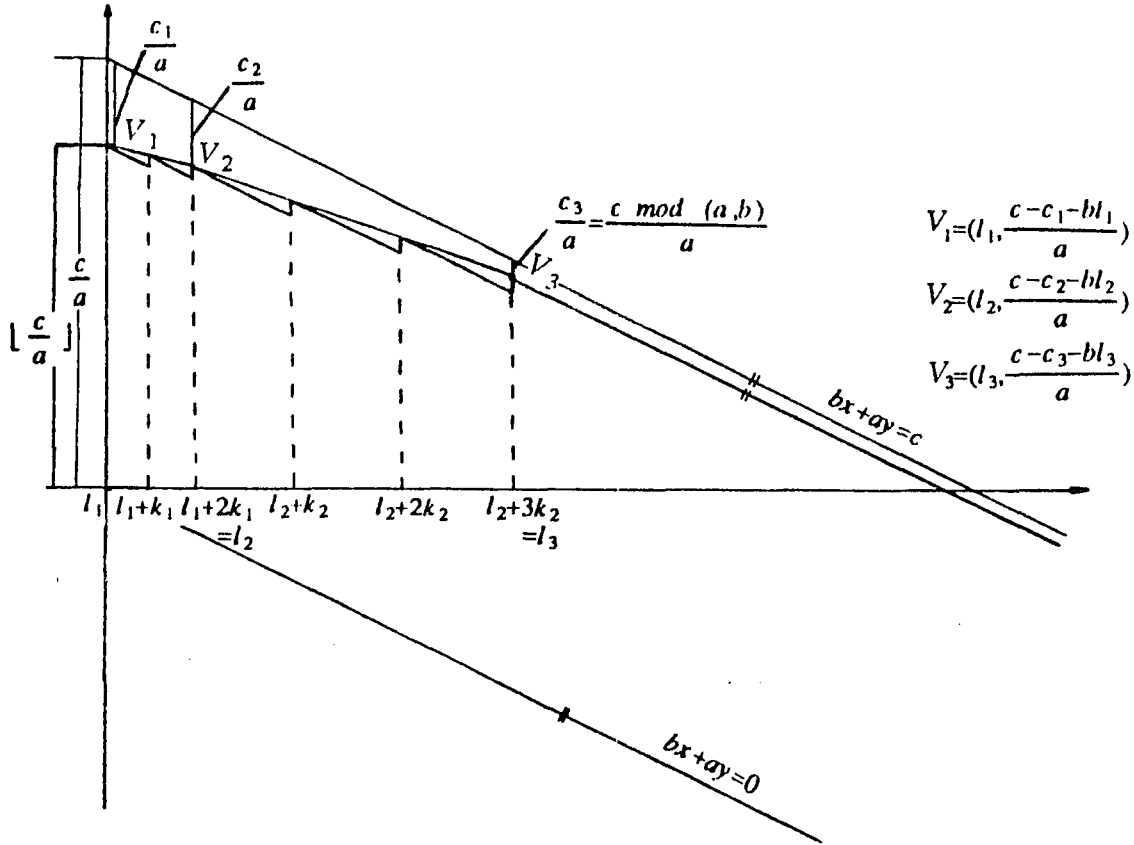


Figure 2.2. Graphical explanation of approximation lemmas

2.4. REUGCD and Opt-2-ILP are NC-equivalent

We next proceed to prove Theorem 2.2.6.

2.4.1. Proposition: Opt-2-ILP is NC-reducible to REUGCD.

The proof of Proposition 2.4.1 will be divided into several lemmas.

2.4.1.1. Lemma: 2-ILP is NC-reducible to 2-ILP in a triangle.

Proof: It was proved in [1] that triangulating a convex polygon is in NC. Thus, if the feasibility region of a given 2-ILP problem is bounded, we triangulate it and solve the problem in each of the triangles separately, subsequently selecting the best of the

optima.

Given a general Opt-2-ILP problem P with no restriction on the feasibility region, we may use 2-LP in a standard way to either (a) decide that P is infeasible or (b) decide that P is unbounded or (c) find a Opt-2-ILP problem equivalent to P with a bounded feasibility region. We note that 2-LP is in NC (see [4], [21]). $\square$

Remark: A similar approach is also used in [21] and [22]. We note that [21] deals with a special case only. [22] deals with the general case, but the algorithm given there is known to be wrong (in the sense that it is not in NC).

2.4.1.2. Lemma: Opt-2-ILP in a triangle is NC-reducible to the subproblem of Opt-2-ILP consisting of those integer linear programs whose feasibility region is a triangle ABC satisfying the following conditions:

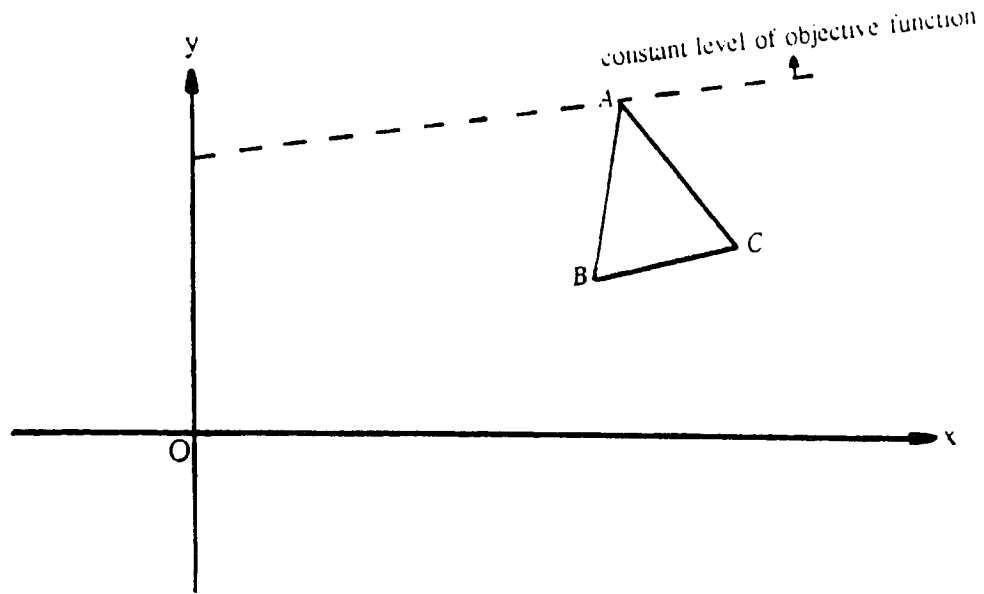
- (1) $A=(0, \frac{c}{a})$, $B=(0, q)$ where c, a are positive integers, and $\frac{c}{a} > q$.
- (2) There are integers x, y, b such that $C=(x, y)$, $x > 0, a > b > 0$ and $bx + ay = c$.

Proof: In Figure 2.3, the dotted line indicates the constant level of the objective function of an integer linear program P whose feasibility set is a triangle ABC (not necessarily satisfying the conditions (1), (2)). Thus the point A is an optimum of the corresponding linear program P' . We transform the triangle ABC to a "nice" position so that the conditions (1) and (2) are satisfied. The transformations must belong to $GL_2(\mathbb{Z})$ to preserve the set of integral points.

Recall that

$$GL_2(\mathbb{Z}) = \left\{ \begin{pmatrix} a & b \\ c & d \end{pmatrix} \mid a, b, c, d \in \mathbb{Z} \text{ \& } ad - bc = \pm 1 \right\}$$

An element in $GL_2(\mathbb{Z})$ maps integral points exactly to integral points, since if

Figure 2.3. An arbitrary triangle ABC

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

then

$$A \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax+by \\ cx+dy \end{bmatrix}$$

is an integral point. Also $A^{-1} \in GL_2(\mathbb{Z})$, since

$$A^{-1} = \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \text{ then } \det(A^{-1}) = ad - bc = \pm 1.$$

In fact, we transform the triangle ABC so that AB overlaps with the Y -axis. A remains the optimum point of correspondingly transformed linear program.

Let $A - B = \begin{bmatrix} z \\ t \end{bmatrix} = \vec{AB}$ and we want to find

$$M = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \in GL_2(\mathbb{Z})$$

such that

$$\begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{pmatrix} \begin{pmatrix} z \\ t \end{pmatrix} = \begin{pmatrix} 0 \\ v \end{pmatrix}$$

i.e.

$$m_{11}z + m_{12}t = 0$$

$$m_{21}z + m_{22}t = v$$

$$m_{11}m_{22} - m_{12}m_{21} = \pm 1$$

Let $v = \gcd(z, t)$, we can find m_{21}, m_{22} by the Euclidean algorithm. Then

$$m_{11} = \frac{t}{\gcd(z, t)}, m_{12} = -\frac{z}{\gcd(z, t)},$$

$$m_{11}m_{22} - m_{12}m_{21} = \frac{t}{\gcd(z, t)}m_{22} + \frac{z}{\gcd(z, t)}m_{21} = 1$$

So we can actually find $M \in GL_2(\mathbb{Z})$ to transform triangle ABC into triangle $A'B'C'$ in Figure 2.4. By a proper shift, S , we can make $A'B'$ overlap with Y-axis. Let $M_1 = SM$, then M_1 transform triangle ABC into triangle $A''B''C''$, Figure 2.5.

During the whole transformation process, A'' remains the optimum of the correspondingly transformed linear program and the optimal value remains the same as well. The reasons are that the objective function is transformed in the dual way.

Let $f(x, y) = kx + ly$ be the objective function, then

$$kx + ly = \begin{pmatrix} k & l \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} k & l \end{pmatrix} M_1^{-1} M_1 \begin{pmatrix} x \\ y \end{pmatrix}$$

$M_1 \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x' \\ y' \end{pmatrix}$ is the optimum point in the new coordinate system in Figure 2.5, which is the image of the original optimum point. So $\begin{pmatrix} k & l \end{pmatrix} M_1^{-1}$ is easy to calculate ($M_1^{-1} \in GL_2(\mathbb{Z})$ is the inverse of M_1) and it represents the direction of the constant level of the transformed objective function. Therefore, the optimum point and the

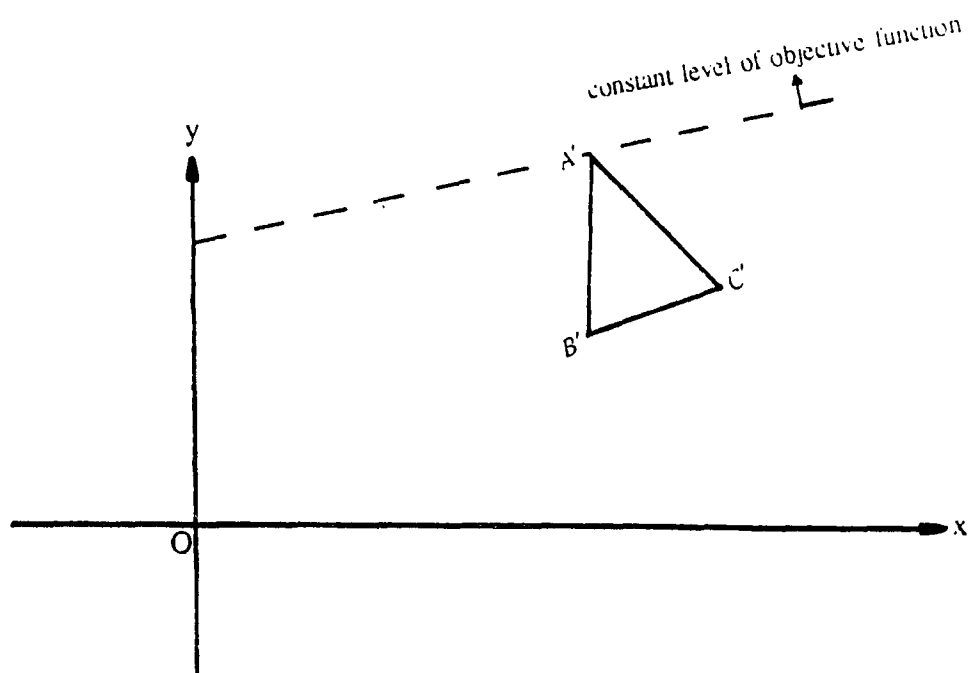


Figure 2.4. Triangle ABC rotated to triangle $A'B'C'$

optimal value are transformation invariants.

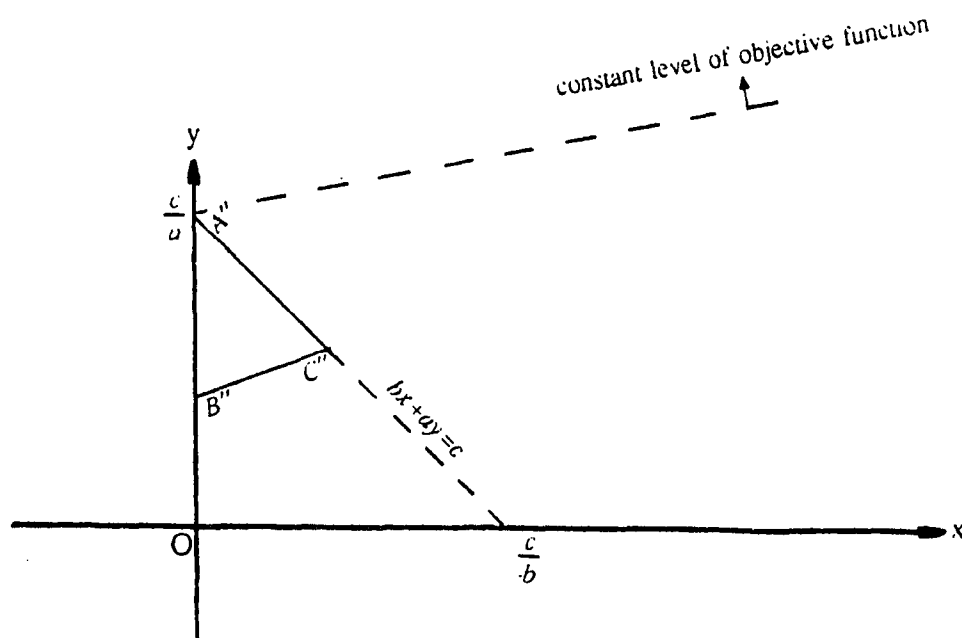


Figure 2.5. Triangle $A'B'C'$ shifted to triangle $A''B''C''$

Another observation is that we can make sure that $a > b$ in the equation of $A''C'' : bx + ay = c$ (Figure 2.5). Let the slope of $A''C''$ be $k_{A''C''}$, then all we need to prove is that $-\frac{\Pi}{4} < k_{A''C''} \leq 0$.

The modular group is defined to be $SL_2(\mathbb{Z}) = \left\{ \begin{pmatrix} a & b \\ c & d \end{pmatrix} \mid ad - bc = 1 \right\}$.

Let $\alpha, \beta \in SL_2(\mathbb{Z})$ be

$$\alpha = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}, \quad \beta = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix},$$

then $\alpha^{-1} = \beta$.

Let $C - A = \begin{pmatrix} g \\ h \end{pmatrix}$, then

$$(a) \quad \alpha \begin{pmatrix} g \\ h \end{pmatrix} = \begin{pmatrix} g \\ h - g \end{pmatrix}$$

$$(b) \quad \beta \begin{pmatrix} g \\ h \end{pmatrix} = \begin{pmatrix} g \\ h + g \end{pmatrix}.$$

We can always assume that $g > 0$, otherwise multiply by $\begin{pmatrix} -1 & 0 \\ 0 & +1 \end{pmatrix} \in GL_2(\mathbb{Z})$. If $h \geq g$,

then multiply by $\alpha^{\lfloor \frac{h}{g} \rfloor}$. if $h < 0$, then multiply by $\beta^{\lfloor \frac{-h}{g} \rfloor}$. Thus, the conditions (1),

(2) are satisfied if the triangle ABC is replaced by $A''B''C''$. Lemma 2.4.1.2 is proved.

□

We shall now consider an integer linear program P with feasibility set equal to a triangle ABC such that the optimum of the associated linear program P' is A . By Lemma 2.4.1.2, we may further make the following assumptions:

$$(1) \quad \overleftrightarrow{AB} = \{(x, y) \mid x = 0\}$$

$$(2) \quad \exists a, b, c \in \mathbb{Z}^+ (a > b > 0 \& \overleftrightarrow{AC} = \{(x, y) \mid bx + ay = c\})$$

Figure 2.6 shows the set $\overline{C}(a, b, c)$ and its convex hull, $CONV(\overline{C}(a, b, c))$ (shadowed

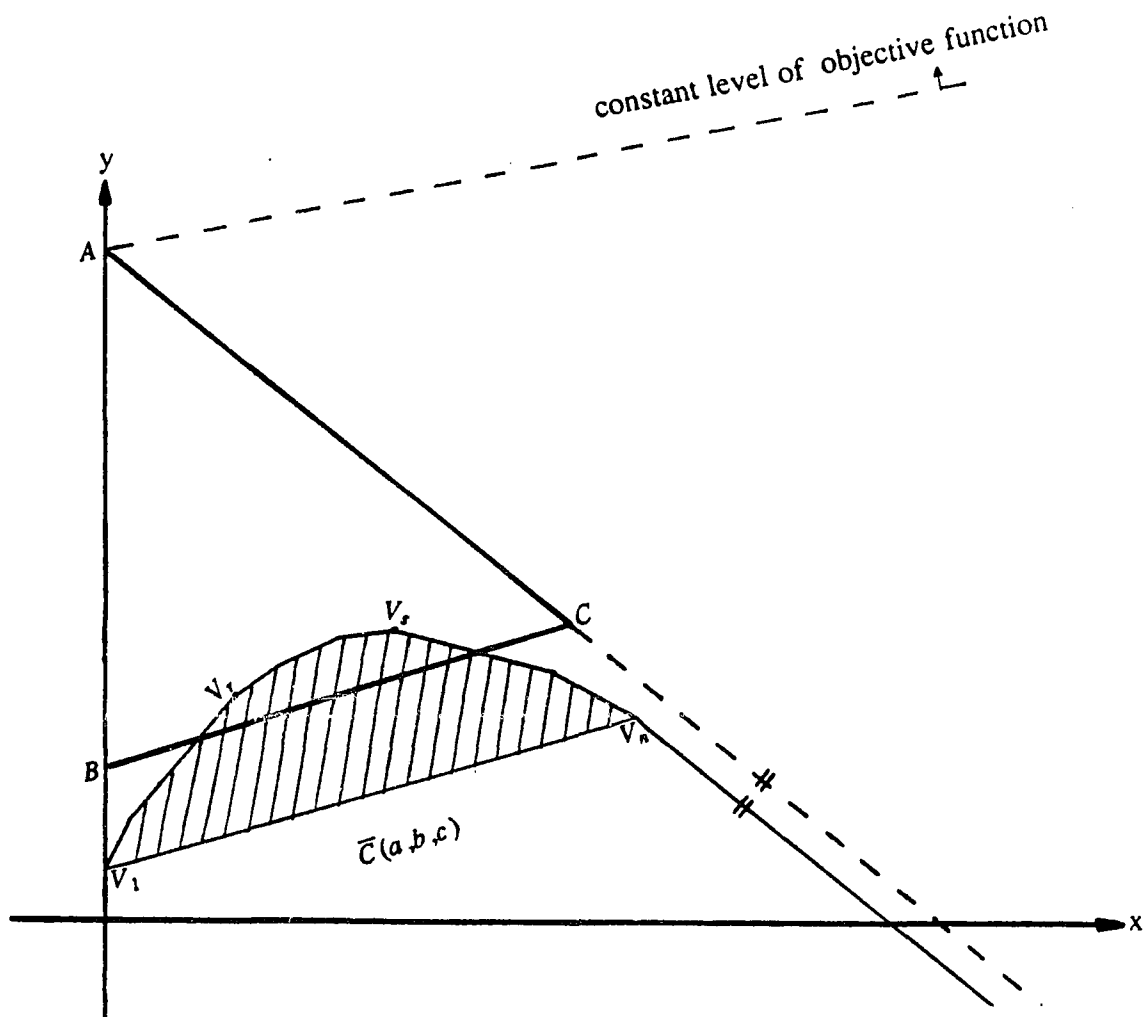


Figure 2.6. $\bar{C}(a, b, c)$ and $CONV(\bar{C}(a, b, c))$

area). Using a binary search, we can compute the numbers

$$r = \min \{i \mid V_i \text{ is feasible}\}$$

$$s = \max \{i \mid V_i \text{ is feasible}\}$$

by an NC-algorithm.

2.4.1.3. Lemma: The optimal solution of the integer linear program P can be found among the points V_i , $r \leq i \leq s$ and the intermediate integral points on the line segments $V_{r-1}V_r$ (if $r > 0$) and V_sV_{s+1} (if $s < n$).

Proof: We assume $\gcd(a,b)|c$ and $a > c \geq 0$ (note that this causes no loss of generality, since it can be always achieved by a shift.) Formally, from the fact that A is the optimum for the corresponding linear program P' , we can conclude this:

(I) The optimum of our integer linear program P is equal to the best feasible solution in $\bar{C}(a,b,c)$ (see Lemma 2.3.2 and Figure 2.6).

We assume for simplicity, $r > 1, s < n$ (in the other cases, some work is spared). By Lemma 2.3.1, the integral points on the segment $V_{r-1}V_r$ are

$$w_t = (l_{r-1} + tk_{r-1}, (c - c_{r-1} + ta_{r-1} - b(l_{r-1} + tk_{r-1}))/a)$$

where $t \in N_0$ and $t \leq \lfloor \frac{c_r}{a_r} \rfloor$ (note that $w_0 = V_{r-1}, w_{\lfloor \frac{c_r}{a_r} \rfloor} = V_r$).

Denote $w_t = (x_t, y_t)$, let $T = \min \{t \mid w_t \text{ is feasible}\}$. By Lemma 2.3.1, any point $\bar{w} = (x, y)$ with $x, y \in \mathbb{Z}$, and $x_{T-1} < x < x_T$, $bx + ay < bx_{T-1} + ay_{T-1}$ satisfies

$$bx + ay < bx_{T-1} + ay_{T-1} \text{ and hence}$$

(II) $\bar{w}$ is not feasible (Figure 2.7).

So no integral point (x, y) with $x < x_T$ is feasible.

Dually, the integral points on the segment $V_s V_{s+1}$ are

$$u_t = (l_s + tk_s, (c - c_s + ta_s - b(l_s + tk_s))/a)$$

where $t \in N_0$ and $t \leq \lfloor \frac{c_{s+1}}{a_{s+1}} \rfloor$ (note that $u_0 = V_s, u_{\lfloor \frac{c_{s+1}}{a_{s+1}} \rfloor} = V_{s+1}$).

Denote $u_t = (x_t, y_t)$. Let $Q = \min \{t \mid u_t \text{ is feasible}\}$. By Lemma 2.3.1, any point $\bar{u} = (x, y)$ with $x, y \in \mathbb{Z}$, $x_Q < x < x_{Q+1}$ and $bx + ay \leq c$ satisfies

$$bx + ay < bx_Q + ay_Q$$

and hence (using the fact that A' is an optimal solution of the corresponding linear

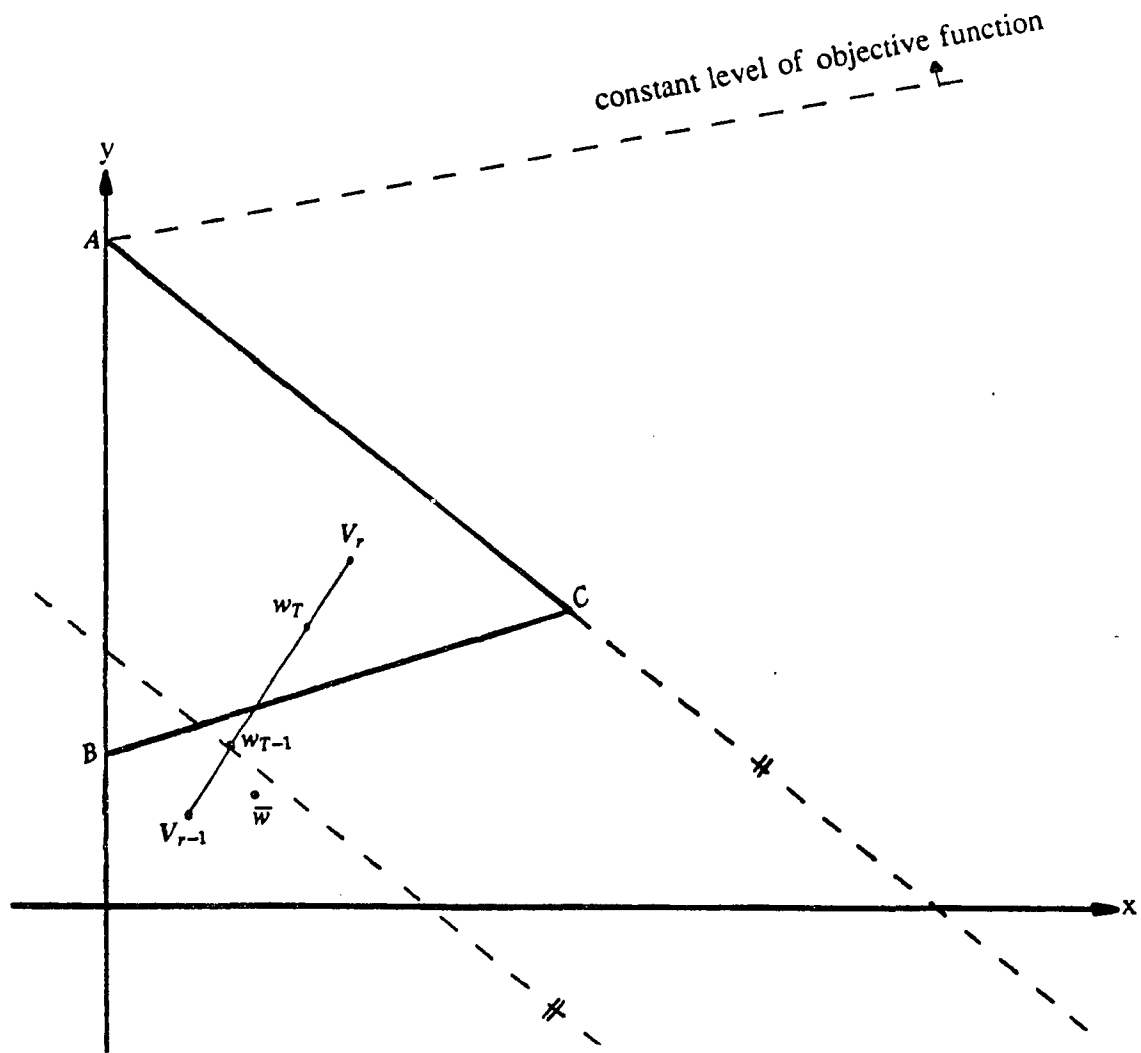


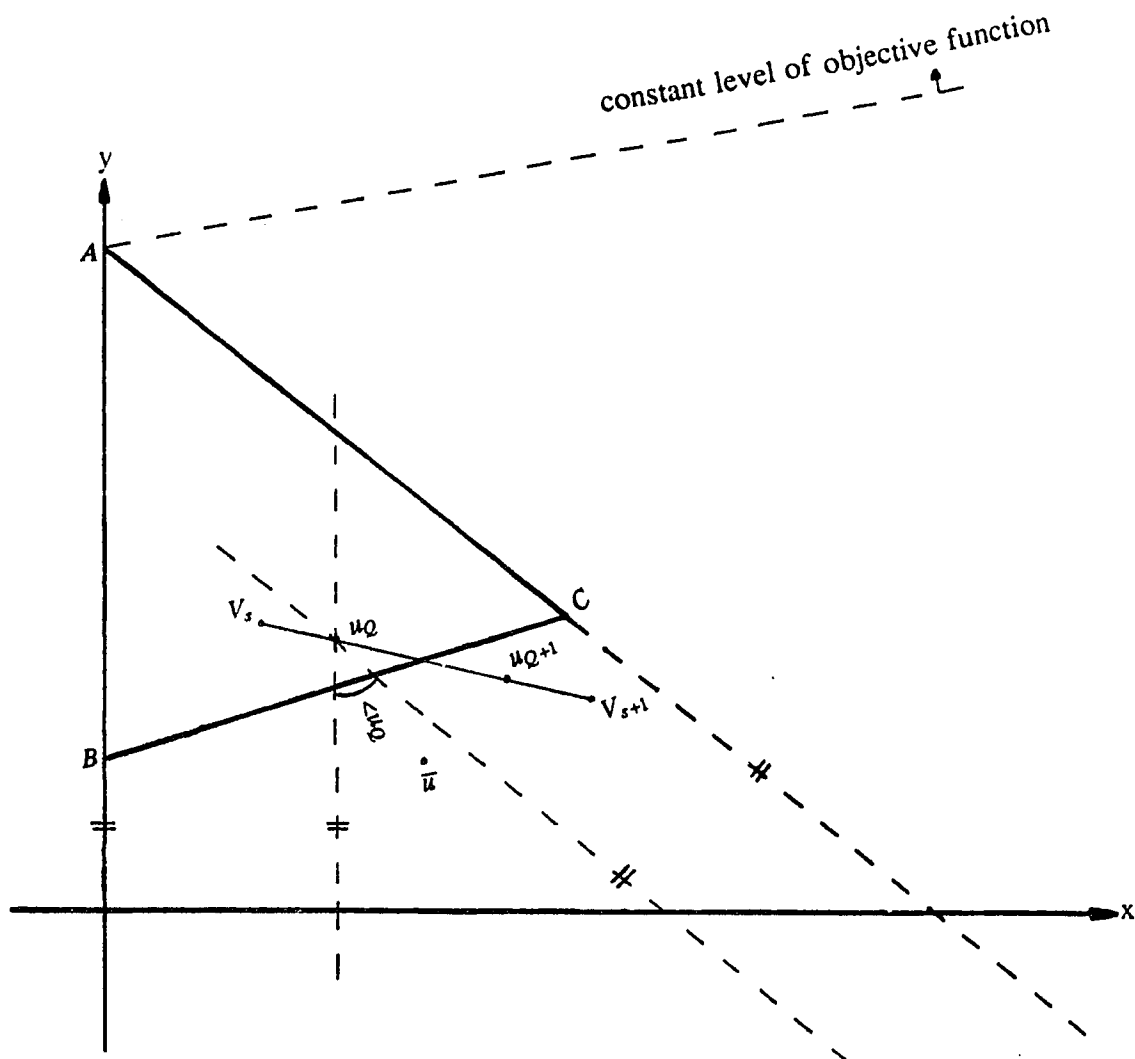
Figure 2.7. $\bar{w}$ is not a feasible solution

program P')

(III) the solution $\bar{u}$ is worse than u_Q (Figure 2.8).

Another way to look at this case is that, $\angle A$ is homothetic to $\angle u_Q$. $\bar{u} \in \angle u_Q$. As A' is the optimum of the linear program in $\angle A'$, u_Q is the optimum in $\angle u_Q$. Thus $\bar{u}$ is not better.

From (I), (II) and (III), the optimal solution of our integer linear program is the best solution among


$$S = \{w_T, u_Q\} \cup \{V_r \mid r \leq t \leq s\}.$$

Proof of Proposition 2.4.1:

We may restrict our attention to the setting described before the statement of Lemma 2.4.1.3, By Lemma 2.4.1.3, however, in this situation the optimum may be selected by comparing polynomially many candidates (which is in NC). Note that

REUGCD was designed in such a way that it allowed an immediate computation of the V_i 's and the intermediate points. $\square$

2.4.2. Proposition: REUGCD is NC-reducible to Opt-2-ILP.

Proof. Given $a, b, c \in \mathbb{Z}$, $a > b$, we need to restore $\{(l_i, c_i, k_i, a_i) | i \in S_2\} \cup \{(l_n, c_n)\}$,

where $S_2 = \{i \in \{1, \dots, n\} | \lfloor \frac{c_i}{a_i} \rfloor > 0\}$ using Opt-2-ILP. We are going to use the

sequence

$$(2^{2^0}, 2^{2^1}, 2^{2^2}, \dots, 2^{2^{\log K}})$$

where $K = \log(ab)$. By double squaring

$$(2^{2^0})^2 = 2^{2^1}$$

$$(2^{2^1})^2 = 2^{2^2}$$

$$(2^{2^2})^2 = 2^{2^3}$$

$$(2^{2^{\log K - 1}})^2 = 2^{2^{\log K}},$$

the sequence $(2^{2^0}, 2^{2^1}, 2^{2^2}, \dots, 2^{2^{\log K}})$ can be computed sequentially in $\log K$ steps. From

this sequence, we can compute the sequence

$$(2^1, 2^2, \dots, 2^K)$$

in $\log K$ time using K processors:

each processor $P_i (1 \leq i \leq K)$

i) express i in binary representation in $\log i \leq \log K$ steps;

$$i = \sum_{j \in D} 2^j, D = \{0, 1, 2, \dots, \log K\}$$

ii) compute $2^i = \prod_{j \in D} 2^j$ by at most $\log K$ multiplications;

Consider $D = \{x \geq 0, y \geq 0, bx + ay \leq c\}$, Figure 2.9. We are to find V_j 's, therefore to restore the data for REUGCD. For $v \in \mathbb{Z}^2$, we use the notation $v = (x_v, y_v)$.

input: polygon $D = \{x \geq 0, y \geq 0, bx + ay \leq c\}$ and sequence $(2^1, 2^2, \dots, 2^K)$, where $K = \log(ab)$

output: V_j 's, as defined in Lemma 2.3.2, and V_j 's immediate successor V_j^*

Algorithm: Finding V_j 's

Step 1: for $i := 1$ to K do in parallel

$$1.1 D_i := \{(x, y) \mid c(1 - \frac{1}{2^i}) \leq bx + ay \leq c, x \geq 0\};$$

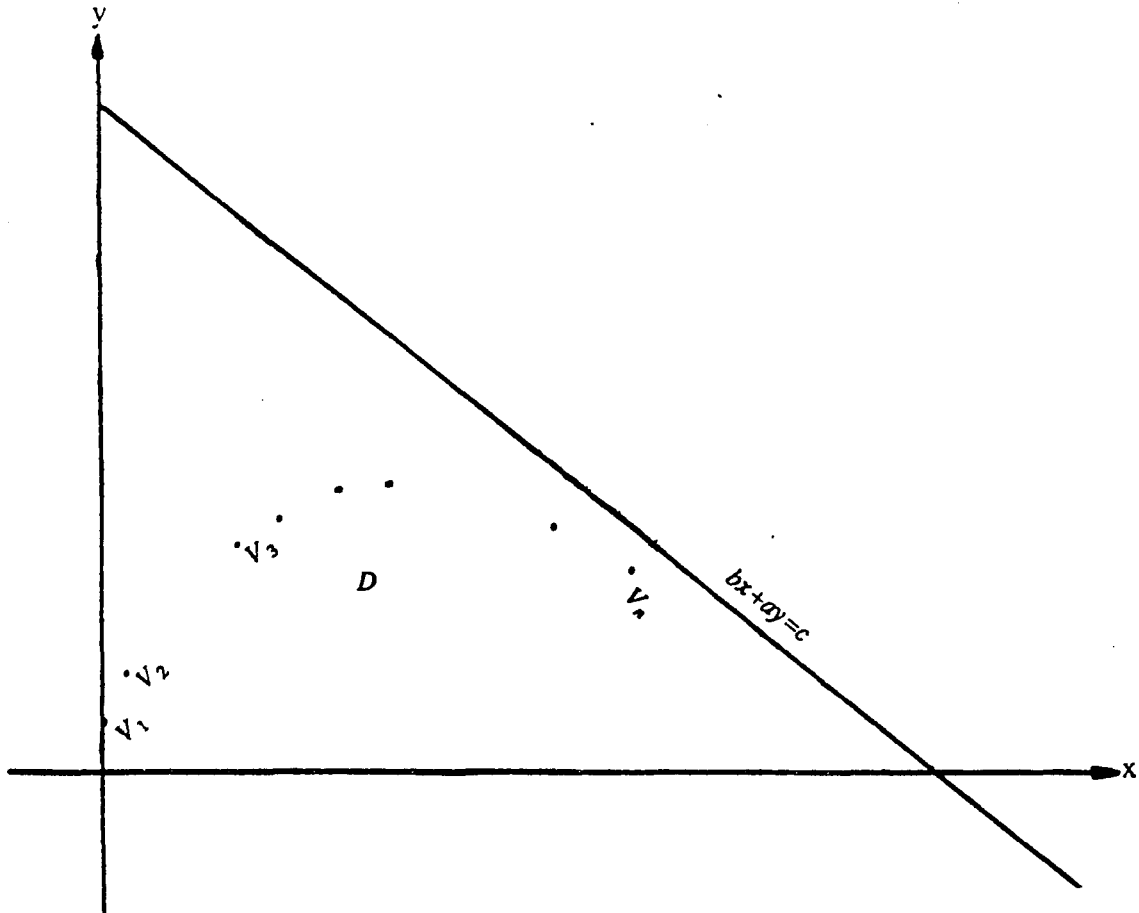


Figure 2.9. We need to restore V_j 's inside polygon D

1.2 $T_i := (x_{T_i}, y_{T_i}) :=$ the minimum of $c - bx - ay$, $(x, y) \in D_i$;

1.3 $D_i' := \{(x, y) \mid x \geq x_{T_i} + \varepsilon, bx_{T_i} + ay_{T_i} \leq bx + ay \leq c\}$; $\{\varepsilon$ is a sufficiently small positive constant}

1.4 $T_i' := (x_{T_i'}, y_{T_i'}) :=$ the minimum of $c - bx - ay$ in $(x, y) \in D_i'$;

1.5 $\bar{x}_i := x_{T_i'} - x_{T_i}$; $\bar{y}_i := y_{T_i'} - y_{T_i}$;

1.6 $t_i := \max\{t \in \mathbb{Z} \mid a(y_{T_i} + t\bar{y}_i) + b(x_{T_i} + t\bar{x}_i) \leq c\}$;

1.7 $T_i'' := (x_{T_i} + t_i\bar{x}_i, y_{T_i} + t_i\bar{y}_i)$;

1.8 if $x_{T_i} - \bar{x}_i < 0$ then $\{T_i$ is a $V_j\}$ $V_i := T_i$

else $\{T_i''$ is a $V_j\}$ $V_i := T_i''$;

1.9 $V := V \cup \{V_i\}$;

end {for}; (Figure 2.10 and Figure 2.11.)

Step 2: Sort the elements in V into $V = \{\tilde{V}_1 < \tilde{V}_2 < \dots < \tilde{V}_m\}$ by x-coordinates; (Note that $1 \leq m \leq K$)

Step 3: for $i := 1$ to $m - 1$ do in parallel

3.1 $\tilde{D}_i := \{(x, y) \mid ay_{\tilde{V}_i} + bx_{\tilde{V}_i} \leq bx + ay \leq c, x \geq x_{\tilde{V}_i} + \varepsilon, y \geq 0\}$;

3.2 $\tilde{V}_i^* :=$ the minimum of $c - bx - ay$ in $(x, y) \in \tilde{D}_i$;

$\{\tilde{V}_i^*$ is the intermediate successor of $\tilde{V}_i\}$

3.3 $V^* := V^* \cup \{\tilde{V}_i^*\}$;

end {for}; (Figure 2.12.)

Step 4: We may assume V^* is already represented in a sorted fashion (by the x-coordinate) as V is;

Step 5: Output sorted V and V^* ;

end;

Now we can calculate l_i, c_i, k_i, a_i : Since $V_i = (x_{V_i}, y_{V_i}) = (l_i, (c - c_i - bl_i)/a)$, we have

$$(2.4.2.1) \quad l_i = x_{V_i};$$

$$(2.4.2.2.) \quad c_i = c - (ay_{V_i} + bx_{V_i});$$

Let $V_i^* = (x_{V_i^*}, y_{V_i^*})$ be the intermediate successor of V_i .

$$(2.4.2.3.) \quad k_i = x_{V_i^*} - l_i;$$

$$(2.4.2.4.) \quad a_i = c_i + ay_{V_i^*} + b(l_i + k_i) - c;$$

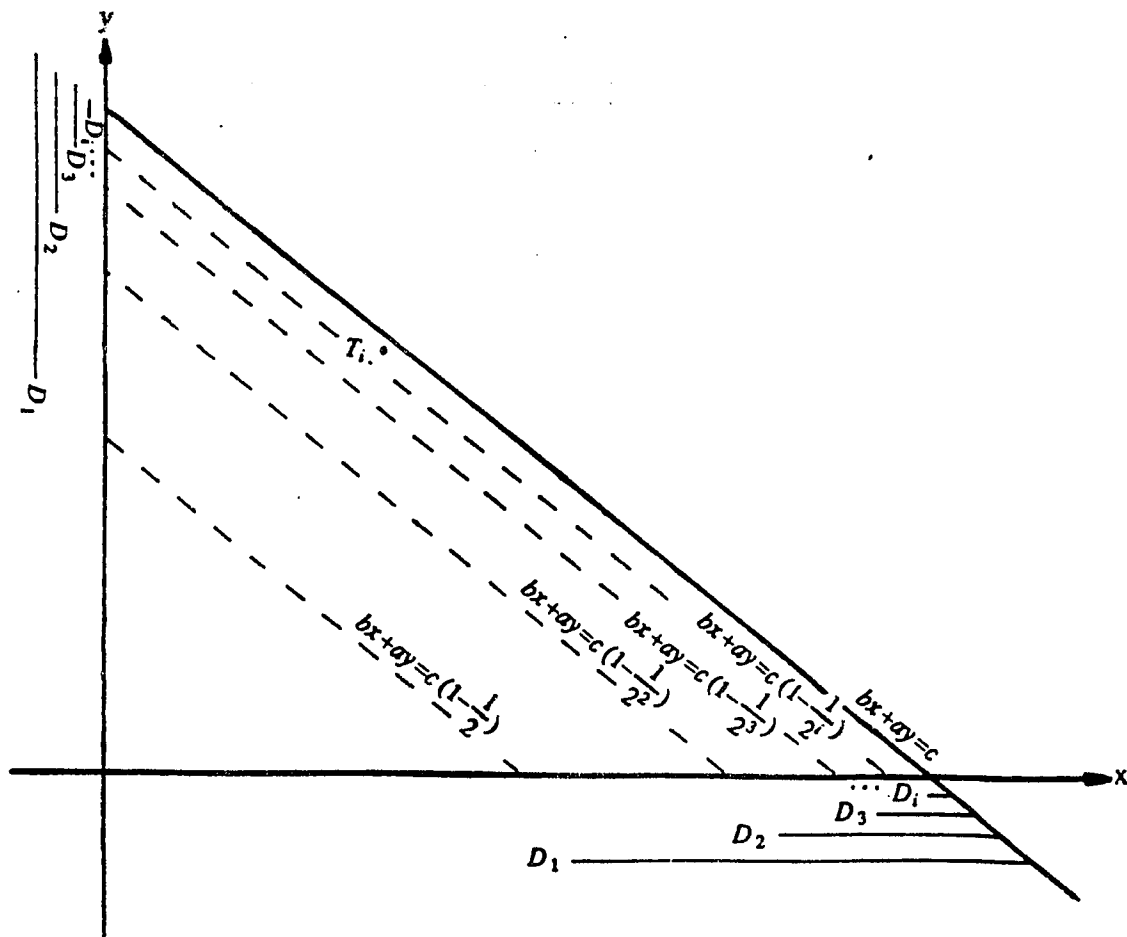


Figure 2.10. T_i is a V_j or an intermediate point in (V_{j-1}, V_j)

The correctness and complexity of the algorithm is discussed in the following sections. Therefore, we have proved that REUGCD is NC-reducible to Opt-2-ILP. $\square$

2.4.3. Correctness of the Algorithm

It is immediate from the definitions that the formulas for the V_j 's as well as the ones for the related data were correct. What we need to show is that none of the V_j 's was omitted. In effect, such an omission could happen only if two different V_j 's lay in

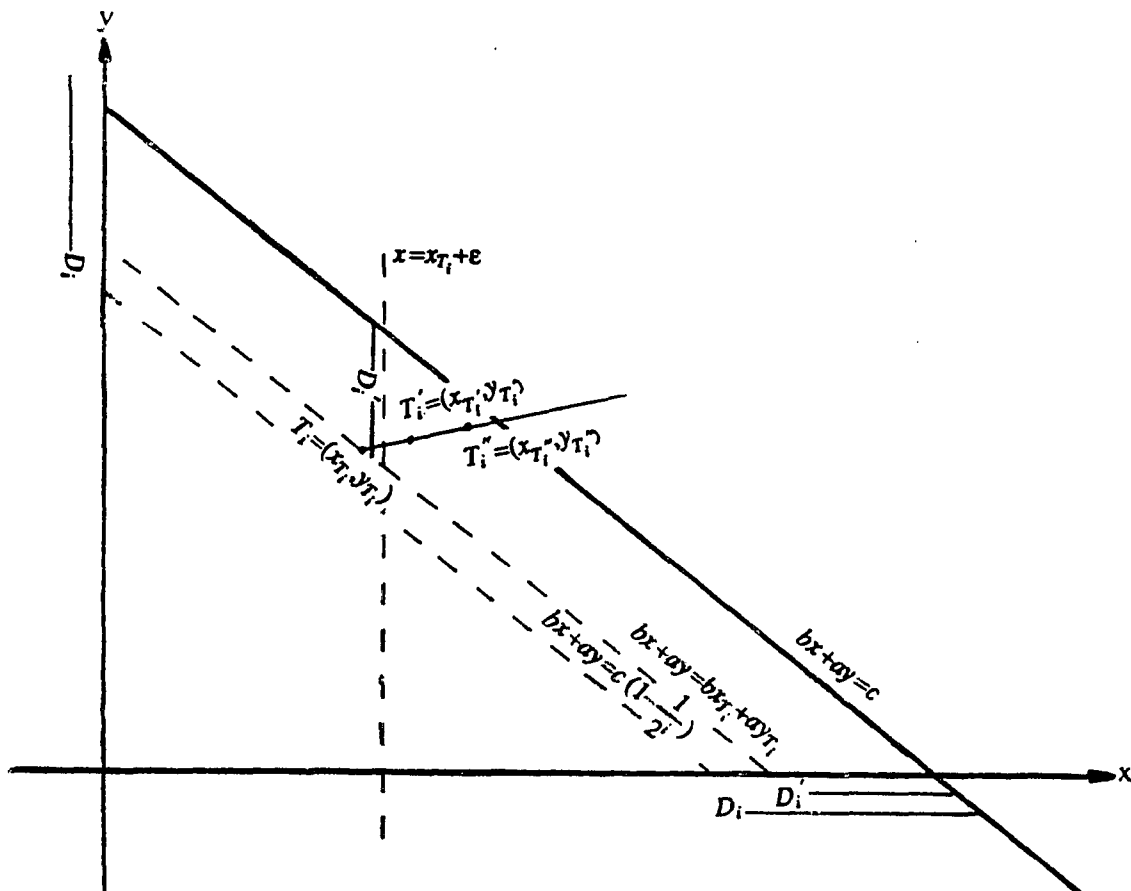


Figure 2.11. Either T_i is a V_j or T_i'' is a V_j

the same $D_i - D_{i+1}$. This, however, would directly lead to

$$(2.4.3.1.) \ c_{j+1} \geq \frac{1}{2}c_j \text{ while } \lfloor \frac{c_j}{a_j} \rfloor > 0$$

for some j . (Recall the definition of V_i in Lemma 2.3.2 and Step 1.1 of 2.4.2. It gives

$$c_j = c - ay_{V_j} - bx_{V_j},$$

$$\frac{c}{2^i} \leq c_j, \ c_{j+1} \leq \frac{c}{2^{i+1}}.)$$

Now recall that

$$c_{j+1} = c_j - \lfloor \frac{c_j}{a_j} \rfloor a_j.$$

This gives, in particular,

$$c_{j+1} = c_j - \lfloor \frac{c_j}{a_j} \rfloor a_j < a_j.$$

then,

$$c_{j+1} \leq c_j - a_j \leq c_j - c_{j+1}$$

and $2c_{j+1} < c_j$, contradicting (2.4.3.1).

2.4.4. Analysis of the Algorithm

The above algorithm may be implemented by a PRAM with polynomially many processors using the standard arithmetical operations [40] and using Opt-2-ILP as an oracle. The computation time of this PRAM is as follows:

- Generating $(2^1, 2^2, \dots, 2^K)$ takes $O(\log K)$
- Step 1: 1.2, 1.5 and 1.7 use 2-ILP and the rest of sub-steps take $O(1)$ time
- Step 2: The sorting of V can be done in $O(\log K)$ using K processors
- Step 3: 3.2 uses 2-ILP and other steps are done in $O(1)$
- Step 4: same as Step 2

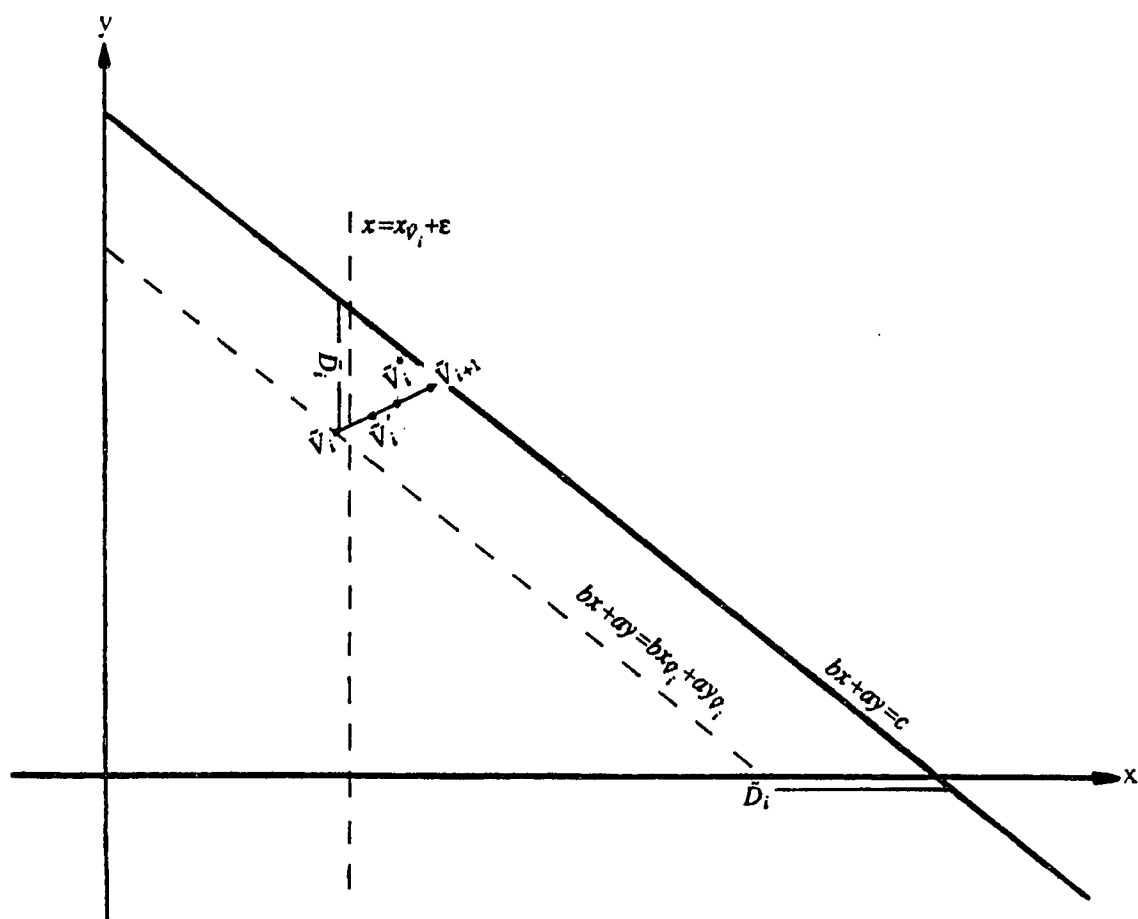


Figure 2.12. $|\overline{\tilde{V}_i \tilde{V}_i}| = |\overline{\tilde{V}_{i+1} \tilde{V}_i^*}|$

-Step 5: $O(1)$

Therefore $O(\log K)$ is the time complexity of the above algorithm.

2.5. Conclusion and the Hashing Function

The proof of Theorem 2.2.6 follows from Lemma 2.3.1, Lemma 2.3.2, Proposition 2.4.1 and Proposition 2.4.2. Therefore, we have proved that REUGCD and Opt-2-ILP are NC equivalent.

One sees that actually Opt-2-ILP is equivalent to EUGCD plus computation of MOD^* (see 3.1 for definition) with arguments $(a_1, \dots, a_n)$ arising from the Euclidean algorithm. Thus the complexity of Opt-2-ILP is, in a way, a measure of the generality of the sequence $(a_1, a_2, \dots, a_n)$.

Chapter 3

THE COMPLEXITY OF THE HASHING FUNCTION

3.1. The Definition of the Hashing Function MOD*

Recall that for an integer a and a positive integer b there are integers x, r such that

$$a = bx + r$$

and

$$0 \leq r \leq b$$

The integer r is called the remainder of a modulo b and denoted by

$$a \bmod b.$$

3.1.1. Definition: Let $x, a_1 \geq a_2 \geq \cdots \geq a_n > 0$ be integers. Define

$$\begin{aligned} &MOD^*(x, a_1, a_2, \cdots, a_n) \text{ to be} \\ &(\cdots ((x \bmod a_1) \bmod a_2) \cdots) \bmod a_n \end{aligned}$$

To avoid confusion, we emphasize that in this chapter, $a_1, \cdots, a_n$ is an *arbitrary* sequence of integers satisfying $a_1 \geq a_2 \geq \cdots \geq a_n$. Thus, the a_i 's are not required to be a remainder sequence of the Euclidean algorithm as in (2.2.1.1). Note, however, that in the special case when the a_i 's do come from (2.2.1.1), the number c_n computed by (2.2.1.1) is equal to $MOD^*(c, a_1, a_2, \cdots, a_{n-1})$.

3.2. MOD^* is P-complete

3.2.1. Theorem: MOD^* is P-complete with respect to log-space reduction.

The proof of this Theorem is given in Section 3.4. The following section is devoted to a preliminary construction.

3.3. A Special Model of the Turing Machine

3.3.1. Our Turing Machine Model

We consider a special Turing machine model. The model has only one internal state (non-final state) and an alphabet $\Sigma = \{0, 1, \dots, l-1\}$ (where l is a sufficiently large enough constant and is discussed later). Upon reading a cell (of the tape) which currently contains a symbol $\bar{p}$, the machine is programmed to execute an action which we denote by $c_{\bar{p}}$. The action may be selected from one of the following types:

$A(\bar{q})$: subtract $\bar{q}$ from the current cell

$B^+(\bar{q})$: move to the right and add $\bar{q}$ to the next cell

$B^-(\bar{q})$: move to the left and add $\bar{q}$ to the next cell

H: halt

We call a machine of the above form *correct* if its program never causes it to overflow or underflow, thereby producing an unreachable symbol (a symbol not in Σ) on the tape.

3.3.2. Lemma: A general Turing machine T can be simulated by a Turing machine T' in the above form in real time. More precisely, there is a constant c (possibly depending on T) such that for every step of T , T' makes at most c steps.

Proof: A general Turing machine model is a 7-tuple

$$M_G = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

where $\delta: Q \times \Sigma \rightarrow Q \times \Sigma \times \{L, R\}$ (see [33]).

Our simulation machine

$$M_S = (\{r\}, \Sigma_1, \Gamma_1, \delta_1, r, B_1, Halt)$$

has 5 tracks on its tape (Figure 3.1): one flag track, T_1 , two auxiliary tracks: T_2, T_3 , and two information tracks: T_4, T_5 .

Information tracks are used to record the transitions of the simulated machine, while auxiliary tracks keep a copy of the previous state and symbol of the simulated machine. If the head is on the current cell, flag = 1, otherwise flag = 0. Initially the 4th track contains the input string of M_G and the flag of the corresponding initial cell is

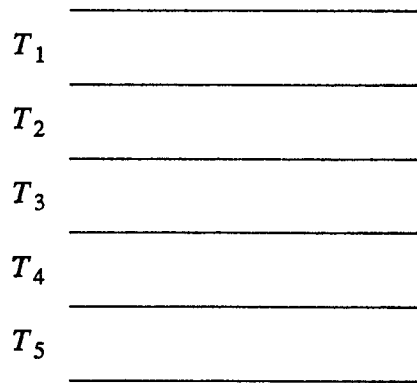


Figure 3.1. Tape configuration of the simulation machine.

set

to 1; the 5th track of the initial cell contains the initial state of M_G , q_0 , other tracks and cells are all blank. See Figure 3.2.

M_G :

A	B	A	B	A	A
---	---	---	---	---	---

M_S :

1					
A	B	A	B	A	A
q_0					

Figure 3.2. Inputs to the simulated machine M_G and simulating machine M_S

During the simulation, each cell of M_S is an encoding that can be interpreted as a 5-digit number in a base p , with first track the most significant bit (note that we may put $p = \max(|Q|, |\Gamma|)$, therefore $l = p^5 - 1$, where l was mentioned earlier).

Case 1: if M_G has a transition

$$\delta(s_i, A_j) = (s_k, A_l, R)$$

M_S simulates it by the following actions:

(1) (flag = 1) Change flag to 0 and copy the contents of the information tracks to the auxiliary tracks: T_4 to T_2 and T_5 to T_3 . Modify the information tracks according to the transition above so that T_5 contains the new state and T_4 the new symbol of the simulated machine. Stay in the same cell. This is an $A(\bar{q})$ operation since the most

significant bit is decreased from 1 to 0. See Figure 3.3.

1	0		0	0
*	*		A_j	*
*	*	$\Rightarrow$	s_i	*
A_j	A_w		A_l	A_w
s_i	*		s_k	*

Figure 3.3. First sub-step of simulation.

(2) (flag = 0) Read auxiliary tracks of the current cell and move to the right, set the flag of next cell to 1, and change the 5th track of the next cell to s_k . This is a $B^+(\bar{q})$ operation, since the most significant bit is increased from 0 to 1. See Figure 3.4.

Case 2: if M_G has a transition

$$\delta(s_i, A_j) = (s_k, A_l, L)$$

Similar to Case 1, except move to left and this is a $B^-(\bar{q})$ operation.

Case 3: if M_G halts, M_S halts too. $\square$

0	0		0	1
A_j	*		A_j	*
s_i	*	$\Rightarrow$	s_i	*
A_l	A_w		A_l	A_w
s_k	*		s_k	s_k

Figure 3.4. Second sub-step of simulation.

3.4. Proof of Theorem 3.2.1

Our aim is to simulate Turing machines by the MOD^* function. By Lemma 3.3.2, we can work in our model. From a polynomial time Turing machine T (in our model) and its input I , we need to construct arguments $x, a_1, \dots, a_n$ of the MOD^* function. The requirements on our construction may be summarized in the following two conditions

- (1) The total number of bits in the numbers $x, a_1, \dots, a_n$ is bounded by a polynomial in the size of I . Also, $x, a_1, \dots, a_n$ can be constructed from the input I by a Turing machine operating in log-space.
- (2) The output of T can be recovered from $MOD^*(x, a_1, \dots, a_n)$ by a Turing machine operating in log-space.

We present the construction below. The fulfillment of the conditions (1), (2) is immediate.

Choose a base $k=2l+1$, denote $exp(x)=k^x$. Consider a polynomial time Turing machine T , i.e., on any input of length z , T halts no later than after $p=p(z)$ steps (where $p(z)$ is a polynomial in z).

For any input $X=x_1x_2 \dots x_n$ of T , let

$$\tilde{X}=111\dots 1(l+x_1+1)(x_2+1) \dots (x_n+1)111\dots 1$$

where there are p 1's in $111\dots 1$. Put

$$q=2p+n, x=\tilde{X}exp(pq)$$

We shall consider variables

$$t \in \{1, \dots, l\}$$

$$s \in \{0, 1, \dots, q-1\}$$

$$u \in \{1, 2, \dots, p\}$$

We define numbers $b(t, s, u)$ and $d(t, s, u)$ by

$$b(t, s, u) = \begin{cases} (l+t) \exp(qu+s) - (l+t-\bar{q}) \exp(q(u-1)+s) & \text{if } c_{t-1} = A(\bar{q}) \\ (l+t) \exp(qu+s) - \exp(q(u-1)+s) - (l+\bar{q}) \exp(q(u-1)+s-1) & \text{if } c_{t-1} = B^+(\bar{q}) \\ (l+t) \exp(qu+s) - \exp(q(u-1)+s) - (l+\bar{q}) \exp(q(u-1)+s+1) & \text{if } c_{t-1} = B^-(\bar{q}) \\ (l+t) \exp(qu+s) - \exp(q(u-1)+s) & \text{if } c_{t-1} = H \end{cases}$$

$$d(t, s, u) = \exp(qu+s) - \exp(q(u-1)+s)$$

Now construct numbers $a_1, \dots, a_N$ where $N=2lpq$ by the following program:

```

for  $u := p$  to 1 by -1 do
    for  $s := (q-1)$  to 0 by -1 do
        begin
            for  $t := l$  to 1 by -1 do
                 $a_{N-(2l((u-1)q+s)+l+(t-1))} := b(t, s, u);$ 
                for  $t := l$  to 1 by -1 do
                     $a_{N-(2l((u-1)q+s)+(t-1))} := d(t, s, u)$ 
                end;
            end;
        end;
    end;
end;

```

Actually " u " denotes the step of the calculation, " s " indicates the position of the current cell on which we are working and " $t-1$ " is the content of the current cell. It is easy to see that the output of T is

$$\text{MOD}^*(x, a_1, a_2, \dots, a_N) - (\exp(q)-1)/2l.$$

We now show that the construction of the numbers $x, a_1, a_2, \dots, a_N$ from the machine T and its input is in log-space (see [2], [33] for definition). Note that the

machine T may be assumed constant, since we are only interested in (log-space)-reducing a fixed polynomial problem to MOD^* . Moreover, we may assume that we are working in the base k : If we want to work in another base m , (typically $m=2$), we simply choose a t such that $m^t > k$ and observe that the base can be changed from k to m^t without making any change in the simulation. Now in base k , the number $b(t, s, u)$ and $d(t, s, u)$ can be calculated by for-loops with constant internal memory (apart from the for-loop variable). The same is true about the calculation of the numbers $a_1, a_2, \dots, a_N$ using $b(t, s, u)$ and $d(t, s, u)$ as procedures. Our assertion now follows from the fact that we only need logarithmic space to store the for-loop variable. (The calculation of x from the input is trivial.)

This implies that, given an instance of a polynomial time Turing machine in our model, we can construct, in log-space, an instance of MOD^* such that the output of the Turing machine can be recovered from

$$MOD^*(x, a_1, \dots, a_N)$$

in log-space. Therefore, the conditions (1) and (2) are met and MOD^* is P-complete with respect to log-space reduction. $\square$

3.5. An Example

To see how this really works, we present an example. Assume a polynomial time ($p(x)=x+1$) Turing machine, M_1 , in our model has alphabet $\Sigma = \{0, 1, 2, 3\}$, $l = 4$, and

$$c_0 = c_1 = B^+(1)$$

$$c_2 = Halt, \quad c_3 = A(3)$$

On input 301, M_1 calculates as follows ($\underline{d}$ denotes that the current head position is

under d , etc.)

301

001

011

012

Halt

Based on this information, we have

$$n = \text{length of input} = 3, p = p(z) = z+1 = 4,$$

$$q = 2p + n = 11, k=2*l+1=9, \tilde{X}=11118121111$$

$$x = \tilde{X} \exp(pq) =$$

$$(11118121111)(00000000000)(00000000000)(00000000000)(00000000000)$$

Each step of the calculation is simulated by the construction of a series of a_i 's, according to the previous formula. In most steps, of course, a_i is greater than the corresponding remainder of $x_i = \text{MOD}^*(x, a_1, \dots, a_{i-1})$, this serves for the purpose of selecting only those a_i 's which actually simulate the computation of the Turing machine. We list below the changes of the remainder of x as they happen, discarding those steps where the remainder of x does not change.

$$(11118121111)(00000000000)(00000000000)(00000000000)(00000000000)$$

$$(01118121111)(10000000000)(00000000000)(00000000000)(00000000000)$$

$$(00118121111)(11000000000)(00000000000)(00000000000)(00000000000)$$

$$(00018121111)(11100000000)(00000000000)(00000000000)(00000000000)$$

$$(00008121111)(11110000000)(00000000000)(00000000000)(00000000000)$$

$$(00000121111)(11115000000)(00000000000)(00000000000)(00000000000)$$

```

(00000021111)(11115100000)(00000000000)(00000000000)(00000000000)
(00000001111)(11115120000)(00000000000)(00000000000)(00000000000)
(00000000111)(11115121000)(00000000000)(00000000000)(00000000000)
(00000000011)(11115121100)(00000000000)(00000000000)(00000000000)
(00000000001)(11115121110)(00000000000)(00000000000)(00000000000)
(00000000000)(11115121111)(00000000000)(00000000000)(00000000000)

```

We have now simulated one step of the computation of the machine M_1 . The other steps of the simulation work in the same way. Below, we list only the remainder of x after each *complete* step of computation, thus discarding even more intermediate steps of the simulation.

```

(00000000000)(00000000000)(11111621111)(00000000000)(00000000000)
(00000000000)(00000000000)(00000000000)(11111271111)(00000000000)
(00000000000)(00000000000)(00000000000)(00000000000)(11111231111).

```

Part B:

Linear Diophantine Equations: the Local Methods

Chapter 4

A ZNC ALGORITHM FOR SOLVING SYSTEMS OF LINEAR EQUATIONS OVER AN NC-CONSTRUCTIBLE VALUATION RING

4.1. Preliminaries

4.1.1. Rings and Modules

A *ring* R is a set with two binary operations (addition and multiplication) such that

- (1) R is an Abelian group with respect to addition (so that R has a zero element, denoted by 0, and every $x \in R$ has an (additive) inverse, $-x$).
- (2) Multiplication is associative ($(xy)z = x(yz)$) and distributive over addition

$$x(y+z) = xy + xz, (y+z)x = yx + zx.$$

We consider only rings which are commutative:

- (3) $xy = yx$ for all $x, y \in R$,

and have a multiplicative identity element (denoted by 1):

- (4) $\exists 1 \in R$ such that $x \cdot 1 = 1 \cdot x$ for all $x \in R$. The identity element is unique.

In this thesis, a ring is a commutative ring with identity.

A *module* over a ring R is an Abelian group M (on which A acts linearly) together with a map $R \times M \rightarrow M$ such that for $\lambda, \mu \in R$, $a, b \in M$

$$\lambda(a+b) = \lambda a + \lambda b$$

$$(\lambda + \mu)a = \lambda a + \mu a$$

$$\lambda(\mu a) = (\lambda\mu)a$$

$$1 \cdot a = a$$

An R -module M is called *finitely generated* if there is a subset $\{z_1, \dots, z_n\} \subseteq M$ such that every element $x \in M$ can be (not necessarily uniquely) expressed in the form

$$x = \sum_{i=1}^n \lambda_i z_i$$

with $\lambda_i \in R$.

A *basis* of a module M over a ring R is a subset $S \subseteq M$ such that each element of M can be *uniquely* expressed in the form

$$\lambda_1 m_1 + \dots + \lambda_k m_k$$

with $\lambda_1, \dots, \lambda_k \in R$, $m_1, \dots, m_k \in S$.

Of course, a general module over a general ring may not have a basis. A module which has a basis is called *free*.

If R is a field then an R -module is the same as a vector space over R . This is due to the fact that every module over a field has a basis. Thus, every module over a field is free.

A ring R is called an *integral domain* if, for $r, s \in R$,

$$r \cdot s = 0 \text{ implies } (r = 0 \text{ or } s = 0).$$

If R is an integral domain, we can construct the *fraction field* Q of R : Q is the set of equivalence classes of the set

$$\left\{ \frac{r}{s} \mid r, s \in R \text{ \& } s \neq 0 \right\}$$

modulo the equivalence relation

$$\frac{r}{s} \sim \frac{t}{u} \text{ if } ru = st.$$

One can see that Q , indeed, is a field. For details, see [6].

As a matter of fact, the construction of the fraction field generalizes as follows.

Let R be a ring, a set $S \subseteq R$ is called *multiplicative* if

(1) $0 \notin S$

(2) If $a, b \in S$ then $a \cdot b \in S$

If S is a multiplicative set in R , define a ring $R[S^{-1}]$ to be the equivalence classes of the set

$$\left\{ \frac{r}{s} \mid r \in R \text{ \& } s \in S \right\}$$

modulo the equivalence relation

$$\frac{r}{s} \sim \frac{p}{t} \text{ if there is } u \in S \text{ such that } utr = usp.$$

It can be shown that $R[S^{-1}]$, indeed, is a ring [6].

We now turn to an important special case. An *ideal* of a ring R is a set $I \subseteq R$ satisfying the following conditions:

(a) $1 \notin I$

(b) If $x \in I$ and $y \in R$, then $x \cdot y \in I$

(c) If $x, y \in I$, then $x + y \in I$

An ideal is called a *prime ideal* (or simply *prime*) if the following condition holds:

(d) If $x, y \in R$ and $x \cdot y \in I$ then $x \in I$ or $y \in I$.

An ideal I is called a *maximal ideal* if any ideal J in R with $J \supset I$ actually satisfies $J = I$.

If I is a prime ideal, then the set

$$S = R - I = \{x \in R \mid x \notin I\}$$

is multiplicative in R . We put

$$R_I = R[S^{-1}].$$

The process of passing from R to R_I is called *localization* at I . The ring R_I is called " R localized at I ". A ring R is called *local* if it has a unique maximal ideal. It is easy to see that if I is a prime ideal in R , R_I has only one maximal ideal $m = \{ \frac{r}{s} \mid r \in I \text{ \& } s \in R - I \}$, therefore R_I is local (see [6]). An element in a ring R is called *unit* if there is a $y \in R$ such that $x \cdot y = 1$. In a local ring, it is easy to see that the maximal ideal consists exactly of those elements which are not units.

Let R be a ring, M be an R -module and let I be a prime ideal in R . One can also define an R_I -module M_I (called " M localized at I ") as follows: M_I is the set of equivalence classes of the set

$$\{ \frac{m}{r} \mid m \in M \text{ \& } r \in R - I \}$$

modulo the equivalence relation $\sim$ given by

$$\frac{m}{r} \sim \frac{n}{s} \text{ iff } (\exists t) \ t \in R - I \text{ such that } mst = nrt.$$

The module operations are the obvious ones.

Let R be a ring and let $x \in R$ be an element which is not a unit. Then there is a ideal

$$(x) = \{x \cdot r \mid r \in R\}$$

This ideal is called the *principal ideal* in R corresponding to x . We note that princi-

pal ideals may or may not be prime (see the example in 4.1.5).

We describe yet another construction which involves a ring R and its ideal I . This time, I does not have to be prime. We define the *quotient ring* R/I as the set of classes

$$\{x+I = \{x+a \mid a \in I\} \mid x \in R\}.$$

We define operations on R/I by

$$(x+I)+(y+I)=(x+y)+I$$

$$(x+I) \cdot (y+I) = xy + I.$$

It is easy to see that these operations are well defined and that, indeed, R/I with these operations is a ring.

It can also be shown that the ideal I is maximal if and only if R/I is an integral domain; and I is prime if and only if R/I is a field (*class field*) [6].

If R is an integral domain with a fraction field Q and $A = (a_1, \dots, a_n)$ is a matrix over fraction field Q , we consider the R -module

$$M = \left\{ \sum_{i=1}^n r_i a_i \mid r_i \in R \right\}$$

(with the obvious operations). We call this R -module the *column space*, $\text{col}(A)$, of A .

For more detailed information on modules, see [6], [9].

4.1.2. Definition: Let Q be a field. A *valuation* on Q is a function $v: Q^* \rightarrow \mathbb{Z}$ (where $Q^* = Q - \{0\}$) with the following properties:

$$(a) \quad v(ab) = v(a) + v(b)$$

$$(b) \quad v(a+b) \geq \min\{v(a), v(b)\}$$

(c) v is onto

To avoid additional assumptions in the arguments a, b in (a), (b), we allow $v(0) = +\infty$.

The convention $v(0) = +\infty$ will also be adopted in the sequel. Thus from now on,

$$v: Q \rightarrow \mathbb{Z} \cup \{+\infty\}.$$

4.1.3. Definition: The *discrete valuation ring* R associated with the field Q , and valuation v is

$$R = \{x \in Q \mid v(x) \geq 0\}.$$

It can be proved easily that the ring R is a regular local ring of dimension 1 with *maximal ideal*

$$m = \{x \in R \mid v(x) > 0\}$$

Note that $m^2 = \{x \in R \mid v(x) > 1\}$ and $m - m^2 = \{x \in R \mid v(x) = 1\}$.

4.1.4. Definition: Q in 4.1.1 is called the *fraction field* of R . The corresponding *class field* is

$$R/m = \{m + i \mid i \in R\}$$

In addition, the projection from R to R/m is $\pi: R \rightarrow R/m$.

4.1.5. $\mathbb{Z}_{(p)}$ - Integers Localized at p

As an example, $Q = \mathbb{Q}$ (rational field), for any $a = \frac{r}{s} \in Q$, let

$$v(a) = \max\{i \mid p^i \mid r\} - \max\{j \mid p^j \mid s\}$$

Then the discrete valuation ring

$$R = \mathbb{Z}_{(p)} = \{x \in \mathbb{Q} \mid v(x) \geq 0\} = \left\{ \frac{r}{q} \in \mathbb{Q} \mid \neg (p \mid q) \right\} \text{ and}$$

$$R/m = \mathbb{Z}_{(p)}/(p) = \mathbb{Z}/(p).$$

We call $\mathbb{Z}_{(p)}$ the ring of "integers localized at p ". However, if $R = \mathbb{Z}$, the prime ideal (p) is prime if and only if p is a prime number (or 0).

Our main algorithm finds a basis of a free R -module given as a column space of a matrix A over Q , where R is a discrete valuation ring and Q is its fraction field.

4.2. Complexity and NC-constructible Discrete Valuation Rings

In this section, R denotes a discrete valuation ring and Q its fraction field. During discussions of the complexity of our algorithms, we want to concentrate on the cases in which the discrete valuation rings are $R = \mathbb{Z}_{(p)}$, $R = \mathbb{Q}[x]_{(x)}$ (the set of rational functions in x whose denominator is not divisible by x ; this notation comes from the fact that R is the localization of the ring $\mathbb{Q}[x]$ at the prime ideal $(x) = \{xz \mid z \in \mathbb{Q}[x]\}$). Even in these cases, we encounter the difficulty of not having a good way of encoding the elements of R unambiguously. Taking a fraction $\frac{a}{b}$ as an example, where $a, b \in \mathbb{Z}$, converting $\frac{a}{b}$ to a canonical form involves the calculating of $\gcd(a, b)$, which is not known to be in NC.

We therefore work in the following setting: elements of R , Q are encoded as sequences of 0's and 1's, where elements may have, in general, infinitely many encodings. The questions about size (complexity) consequently refers to the encodings (as opposed to elements of R , Q). We have to require that certain operations over R (respectively Q) be in NC. For our purposes, the following assumptions are sufficient:

(1) Computing a product $A_1 \cdots A_n$ of n matrices $A_1, \cdots, A_n$ over $\mathbb{Q}[x]$ (the ring of

polynomials with one variable over the field Q) is in NC;

(2) Computing the valuation of an element $a \in Q$ is in NC;

(3) For an encoding of size n of an element $a \in Q$, we may find in NC an encoding of size $n+c$ of $-a$, where c is a constant;

(4) For an encoding of size n of an element $a \in Q$, $|v(a)| \leq cn$ (where c is a constant).

A discrete valuation ring R satisfying (1), ... , (4) is called *NC-constructible*.

Using Chistov's algorithm ([16]), we see that finding the determinant of a matrix over an arbitrary field F and the inverse of a regular matrix over F are both in NC. By Mulmuley's algorithm [49], we can then also find the rank of matrices over F and consequently bases of column spaces over F in NC.

The reason we require (1) as it stands (as opposed to its weaker version just for Q) is that we need to perform certain calculations in the rings $R[p^{\frac{1}{n}}]$ (see Appendix) and their fraction fields, rather than in R . These require multiplication of matrices over $R[p^{\frac{1}{n}}]$ to be in NC. The fulfillment of this requirement easily follows from (1).

All discrete valuation rings in the sequel are assumed NC-constructible. In the special cases of Q , $Q(x)$, NC-constructibility can be established by exploiting passage to common denominator (product of the denominators involved).

4.3. Systems of Linear Equations over a Discrete Valuation Ring

4.3.1. Systems of Linear Equations

A system of linear equations over a ring R is a system

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2$$

.....

$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m$$

where $a_{ij}, b_i \in R$ are given constants and x_i 's are unknowns, $i=1, \dots, m$, $j=1, \dots, n$. We can also write a system of linear equations in the matrix form $Ax=b$. A solution of the system is a column vector $x=(x_1, x_2, \dots, x_n)^T \in R^n$ which satisfies all the equations.

On one hand, when R is a field F , we have a system of linear equations over a field (SLE(F)). There are known NC algorithms to solve SLE(F) over fields [40].

On the other hand, if $R = \mathbb{Z}$, the integer ring, the system is called a system of linear Diophantine equations, SLDE, which is known to be in P (see [9], Chapter XI). However, it is not known whether SLDE is in NC, or is P-complete.

We remark that solving systems of linear equations over a *general* ring (or even a general integral domain) is a profound problem for which no algorithm is known.

4.3.2. Systems of Linear Equations with Basis

Suppose we represent a system of linear equations over a discrete valuation ring R in the matrix form

$$(4.3.2.1.) \quad Ax=b$$

where A is a given $m \times n$ matrix over R , and b a given m -column vector over R and x is n -column vector. Let the columns of A be $a_1, \dots, a_n$. Suppose $\text{col}(A)$ (as defined in 4.1.5) has a basis consisting of certain columns of A , say $a_1, \dots, a_k$ ($k \leq m$) (This generally happens if R is a field, but not in the case $R=\mathbb{Z}$.) then the system has a solution if and only if b can be expressed as a linear combinations of $a_1, \dots, a_k$. If so, then a general solution $x=(x_1, \dots, x_n)^T$ is obtained by picking $x_{k+1}, \dots, x_n$ arbitrarily and calculating $x_1, \dots, x_k$ as the coefficients of the vector

$$b - \sum_{i=k+1}^n x_i a_i$$

in the basis $a_1, \dots, a_k$.

4.3.3. Solving Systems of Linear Equations over Discrete Valuation Rings

Our goal is to find an algebraic ZNC algorithm for solving the system (4.3.2.1) in the case where R is a discrete valuation ring. We are only concerned with the cases where R, Q have representations such that arithmetic operations are in NC, e.g., the case of $R=\mathbb{Z}_{(p)}$, $Q=\mathbb{Q}$, $R=\mathbb{Q}[x]_{(x)}$, $Q=\mathbb{Q}[x]$. We therefore specialize to NC-constructible rings (see 4.2). We show that if R is a discrete valuation ring then the case of 4.4.2 always occurs (see Theorem 4.4.2). Moreover, R has a fraction field Q . We may assume that the rows of A are independent over Q (passing to a row basis over Q is an equivalent operation for solving linear equations over R and is in NC (see [40], 4.2 and the proof of Theorem 4.4.2).

If the rows of A are independent and $a_1, \dots, a_k$ is a basis of the column space (this can be achieved by permuting the columns) then $k=m$ and $C=(a_1, \dots, a_m)$ is a

regular invertible matrix over Q . Thus, determining whether $b \in \text{col}(A)$ consists of checking if $C^{-1}b$ is a matrix over R . (It is, of course, always a matrix over Q .) If the answer is positive, we may also calculate

$$(x_1, \dots, x_m)^T = (C^{-1}(b - \sum_{i=m+1}^n x_i a_i))^T$$

All of these operations (linear algebra over Q) are in NC.

We have therefore reduced the problem of solving linear equations over discrete valuation rings to finding bases of column spaces. This problem is treated in the remainder of the chapter.

4.4. The Basis of a Column Space over a Discrete Valuation Ring

4.4.1. Lemma: For a square matrix $A=(a_{ij})$ over R , the following are equivalent:

- (1) A is invertible over R ;
- (2) $v(\det A) = 0$;
- (3) $\pi A = (\pi(a_{ij}))$ is a regular matrix over R/m .

Proof: see [6]. $\square$

4.4.2. Theorem: Let R be a discrete valuation ring with fraction field Q , let A be a matrix over Q . Then $\text{col}(A)$ is a free R -module which has a basis consisting of columns of A .

Proof: We may assume that

$$A = \begin{bmatrix} a_{11} & \cdot & \cdot & \cdot & a_{1n} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{m1} & \cdot & \cdot & \cdot & a_{mn} \end{bmatrix}$$

where $a_{ij} \in R$, $i=1, \dots, m$, and $j=1, \dots, n$. Also, without loss of generality, assume that all the rows in A are independent over Q . Let, also,

$$a_i = (a_{1i}, \dots, a_{mi})^T$$

Now we are going to find the basis of $\text{col}(A)$ by the following procedure (recall that row operations over Q are equivalent in the sense that they do not change the indices of the columns in the basis):

(0) Set A as the "active matrix";

for $i:=1$ to m do

(1) reset the current "active matrix" M as

$$M := \begin{bmatrix} a_{ii} & \cdot & \cdot & \cdot & a_{in} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{mi} & \cdot & \cdot & \cdot & a_{mn} \end{bmatrix}$$

(2) Choose an element in the active matrix that has the minimum valuation as pivot.

Without loss of generality, it is a_{ii} ;

(3) Perform equivalent row operations over R on M to make all the elements in the column below the pivot 0. (Recall that the pivot has the minimum valuation.) After these operations, M remains a matrix over R ;

(4) Divide the i -th row by a_{ii} . Again M remains a matrix over R as a_{ii} had the minimum valuation;

(5) Delete the i -th row and i -th column from the "active matrix"

end for;

At the end, A has turned into a matrix A' of the form:

$$(U|D)$$

where U is a $m \times m$ up triangle matrix with all diagonal entries equal to 1, and D is a $m \times (n-m)$ matrix. Note that both U and D are matrices over R .

By further row operations over R , we turn A' into a matrix A'' of the form:

$$(I_m|D')$$

where D' is a $m \times (n-m)$ matrix over R .

Rewrite $A'' = BA$ where B is a regular matrix over Q . Now $B(a_1, \dots, a_m) = I_m$ and hence $\{Ba_1, \dots, Ba_m\}$ is a basis of $\text{col}(BA)$. Consequently, $\{a_1, \dots, a_m\}$ is a basis of $\text{col}(A)$. (Recall that the indices of the columns would in general change if we drop the assumption in (2) that the pivot is a_{ii} .) $\square$

4.4.3. Theorem: Let A be an $m \times n$ matrix over R , which have independent rows. Let v be the minimum valuation of a determinant of an $m \times m$ submatrix of A (such that any other $m \times m$ submatrix B of A satisfies $v(\det B) \geq v$). Then the columns of any $m \times m$ submatrix of A whose determinant has valuation v forms a basis of $\text{col}(A)$.

Proof. By the proof of Theorem 4.4.2, without loss of generality, $A = BA''$, where $A'' = (I_m | D)$, D is a matrix over R .

Now the determinant of an $m \times m$ submatrix C of A has minimum valuation among $m \times m$ submatrices of A if and only if $\det(BC)$ has minimum valuation among determinants of square submatrices of A'' , i.e., if and only if $v(\det(BC)) = 0$. By Lemma 4.4.1, this occurs if and only if BC is invertible over R . If BC is invertible, then

$$C^{-1}A = C^{-1}(B^{-1}A'') = (C^{-1}B^{-1})A''$$

is a matrix over R and consequently, the columns of C form a basis of $\text{col}(A)$. $\square$

4.5. The Gram Determinant

4.5.1. Definition: Let A be a $m \times n$ matrix over R , where $m \leq n$. The Gram determinant of A , $g(A)$, is defined as

$$g(A) = \det(A \cdot A^T).$$

4.5.2. Theorem:

$$g(A) = \sum \{ (\det B)^2 \mid B \text{ is an } m \times m \text{ submatrix of } A \}.$$

Proof: see [13], [25]. $\square$

4.5.3. Theorem:

$$v(g(A)) \geq 2 \cdot \min \{ v(\det B) \mid B \text{ is an } m \times m \text{ submatrix of } A \}.$$

If there is a *unique* $m \times m$ submatrix of A whose determinant has the minimum valuation, then equality holds.

Proof: directly from the definition of the Gram determinant, Theorem 4.5.2 and the axioms of valuation. $\square$

4.6. An Algorithm for Finding the Basis of a Column Space over a Discrete Valuation Ring

4.6.1. Statistics Lemma: Let $\mathbf{C}$ be any nonempty collection of subsets of $\{1, 2, \dots, N\}$. Let $w(1), w(2), \dots, w(N)$ be independent random variables, each with uniform distribution over $\{0, 1, 2, \dots, 2N\}$. Associate with each set $S \subseteq \{1, 2, \dots, N\}$ a weight

$$W(S) = \sum_{i \in S} w(i)$$

Then with probability greater than $\frac{1}{2}$, the family $\mathbf{C}$ contains a unique set of minimum weight (See [50], c.f. [40]). $\square$

4.6.2. Application of the Statistics Lemma to Our Algorithm

By Theorem 4.4.2, our goal is to find a $m \times m$ submatrix of A that has the minimum valuation of its determinant. First notice that if there is a unique $m \times m$ submatrix B with minimum valuation of its determinant, then Theorem 4.5.3 gives an easy method of finding B : just check, for column a_i of A , if $v(g(A)) < v(g(A - \{a_i\}))$. Those columns for which this occurs form the submatrix B .

So we discuss the case where B is not unique. The statistics lemma is used to isolate *one* $m \times m$ submatrix that has the minimum valuation of its determinant.

Let N in the Lemma 4.6.1 be the number of columns in A , which is n ; $\mathbf{C}$ in Lemma 4.6.1 the set of columns of all $m \times m$ submatrices of A that have minimum valuation v_0 of their determinants (cf. Theorem 4.4.3). Let t be a sufficiently large integer to be specified later. In order to apply Lemma 4.6.1, we need to assign random

weights to the sets in C . Randomly choose variables w_i from $[0, 1, 2, \dots, 2n]$, for $i=1, \dots, n$, and multiply the i -th column a_i of A by $p^{\frac{w_i}{t}}$, where $p \in m - m^2$ and $v(p)=1$ (we could always choose such p , since v is onto by definition). Call this new matrix A' .

As a matter of fact, we are now working in a new ring $R[p^{\frac{1}{t}}]$ to which the entries of A' belong. The arithmetic operations and complexity of the operations are discussed in Appendix. By guaranteeing that the complexity of the basic operations in the ring $R[p^{\frac{1}{t}}]$ are in NC, our algorithm remains a ZNC algorithm (see 4.7).

Let v be the valuation in R . A valuation u in $R[p^{\frac{1}{t}}]$ is defined such that $u(p^{\frac{1}{t}})=1$, and for any $a \in R[p^{\frac{1}{t}}]$,

$$u(a) = t \cdot v(a)$$

By lemma 4.6.1, with probability greater than $\frac{1}{2}$, there is a unique set S_0 in C with minimum weight $W(S_0)$.

If $S \subseteq \{1, \dots, n\}$ is a set of cardinality m , let B_S be the corresponding $m \times m$ submatrix of A and B'_S be the corresponding submatrix of A' . We have

$$u(\det(B'_S)) = t \cdot v(\det(B_S)) + W(S)$$

(Note that $v(p)=1$ as assumed.) Now since $S \in C$ if and only if $\det(B_S) \neq v_0$. In that case,

$$(4.6.2.1) \quad u(\det(B'_S)) = t \cdot v_0 + W(S)$$

If $S \notin C$ then $v(\det(B_S)) > v_0$ and hence

$$(4.6.2.2) \quad u(\det(B'_S)) \geq t \cdot (v_0 + 1) = t \cdot v_0 + t$$

If we can make sure that

$$(4.6.2.3) \quad W(S) < t \text{ is always true}$$

then, for $S \in \mathcal{C}, T \notin \mathcal{C}$, we necessarily have

$$(4.6.2.4) \quad u(\det B'_S) < u(\det B'_T).$$

Thus, with probability greater than $\frac{1}{2}$, A' has a unique $m \times m$ submatrix B'_S whose determinant has the minimum valuation. Moreover, by (4.6.2.4), we automatically have $S \in \mathcal{C}$. This gives us the desired algorithm.

Now (4.6.2.2) and (4.6.2.3) are secured by letting

$$t > \max\{W(S)\} = m \cdot 2n \geq 2n^2.$$

(Note that t has to be of polynomial size with respect to n .)

4.6.3. The Algorithm

Input: An $m \times n$ matrix A over a discrete valuation ring R and its valuation v .

Output: A basis B of $\text{col}(A)$ over R consisting of columns of A

Algorithm:

Step 0: Find a maximum Q -independent set of rows of A , reduce A to the submatrix consisting of these rows;

Step 1:

Step 1.1: choose $t > 2n^2$ and a $p \in R$, such that $v(p) = 1$;

for $i := 1$ to n do in parallel

Step 1.2: choose $w(i)$ from $[0, 1, 2, \dots, 2n]$ at random;

Step 1.3: multiply column i by $p^{-\frac{w_i}{t}}$ to form A'

end for;

Step 2: {We assume at this point that there is a unique $m \times m$ submatrix of A' whose determinant has minimum valuation}

Step 2.1: calculate Gram determinant

$$g(A') := \det(A' \cdot A'^T);$$

Step 2.2: calculate the minimum valuation of the $m \times m$ submatrix

$$u_0 := \frac{u(g(A'))}{2};$$

For $i := 1$ to n do in parallel

Step 2.3: $A'_i := A' - \{a_i\}$;

Step 2.4: if $u(\det(A'_i \cdot A'^T_i)) > 2u_0$ then $B := B \cup \{a_i\}$

end for;

Step 3: {check to see if B is really a basis over R }

Step 3.1: find B^{-1} ;

Step 3.2: calculate $N := B^{-1} \cdot A$;

Step 3.3: for all the $n-m$ columns that are not in I_m check in parallel to see if they are columns over R . If yes, output B as a basis over R .

4.7. Discussion of the Algorithm

First of all, by Theorem 4.4.2, there is always a basis formed by m columns of A . This theorem guarantees the "Las Vegas" (ZNC) nature of our random algorithm: a solution always exists. The Gram determinant helps us to find the minimum valuation

of the determinant of an $m \times m$ submatrix of A .

If such matrix is unique, we only need run the deterministic part of our algorithm (Steps 0 and 2). Otherwise, we have to assign random variables to each column and form the matrix A' . The uniqueness of the $m \times m$ submatrix in A' with minimum valuation of its determinant depends on the random assignment of weight to each column of A . We can find such a basis only with probability greater than $\frac{1}{2}$. We therefore have to check in Step 3 if the set obtained is really a basis over R . This is true if and only if those columns of $B^{-1}A$ that are not columns of I_m are in R .

Chapter 5

APPLICATIONS: PARALLEL ALGORITHMS FOR RELATED PROBLEMS

5.1. Introduction

In this chapter, we discuss the applications of the algorithm 4.6.3. Our ZNC algorithm for solving systems of linear equations over NC-constructible discrete valuation rings gives rise to ZNC or NC algorithms for the following problems:

- (a) Solving systems of linear Diophantine equations over $\mathbf{Z}_{(p)}$ (5.2)
- (b) Finding a weighted basis in a linear matroid (an NC-algorithm, 5.3)
- (c) Finding a basis of the intersection of column spaces of two matrices over an NC-constructible discrete valuation ring (5.4)
- (d) p -adic approximation (5.5)
- (e) Finding a coherent basis of two free R -modules $A \subseteq B$ over an NC-constructible discrete valuation ring (5.6)
- (f) Finding the p -component of an Abelian group (5.7)

5.2. Solving Systems of Local Linear Diophantine Equations is in ZNC

5.2.1. Systems of Local Linear Diophantine Equations

A system of linear Diophantine equations (SLDE) is a system of linear equations

(5.2.1.1.) $Ax=b$

with A, b matrices over $\mathbf{Z}$ (the set of integers). A solution to the (5.2.1.1) is an n -tuple $(x_1, \dots, x_n) \in \mathbf{Z}^n$, satisfying all the equations in (5.2.1.1).

While the complexity of solving general SLDE is not known to be in NC or RNC or ZNC, we investigate the case of finding solutions in a local ring, $\mathbf{Z}_{(p)}$. As discussed in Section 4.1, $\mathbf{Z}_{(p)}$ is a discrete valuation ring with fraction field $\mathbf{Q}$ (the set of rationals) and

$$\mathbf{Z}_{(p)} = \left\{ \frac{r}{q} \in \mathbf{Q} \mid \neg (p \mid q) \right\}.$$

Algorithm 4.6.3 together with the contents of 4.3 immediately give a ZNC-algorithm for finding a $\mathbf{Z}_{(p)}$ -solution of the system 5.2.1.1. The algorithm (with previously described algorithms used as subroutines) is described below.

Input: The system (5.2.1.1)

Output: A basis of the $\mathbf{Z}_{(p)}$ -module consisting of all the $\mathbf{Z}_{(p)}$ -solutions of (5.2.1.1) (a free $\mathbf{Z}_{(p)}$ -module by 4.3.3 and Theorem 4.4.2).

Algorithm:

Step 0: By eliminating dependent rows, assure that rows of A are linearly independent over $\mathbf{Q}$ (this can be done in NC by algorithms of Chistov [16] and Mulmuley [49]).

Step 1: Find a basis of $\text{col}(A)$ consisting of columns of A (by algorithm 4.6.3). After permuting columns of A , if necessary, we may assure that the first m columns $a_1, \dots, a_m$ of A form the basis of $\text{col}(A)$. Let C be the submatrix of A (invertible of $\mathbf{Q}$) formed by the columns of $a_1, \dots, a_m$.

Step 2: Check if $C^{-1}b$ is a matrix over $\mathbf{Z}_{(p)}$. If no, no solution exists. If so, let

$e_i = (0, \dots, 0, 1, \dots, 0)^T$ (with 1 in the i -th place). Let $C' = \begin{bmatrix} C^{-1} \\ 0 \end{bmatrix}$ (the last $n-m$ rows

are 0). The basis of the $\mathbf{Z}_{(p)}$ -module of solutions of (5.2.1.1) is

$$\begin{aligned} &e_{m+1} + C'(b - Ae_{m+1}) \\ &\dots\dots \\ &e_n + C'(b - Ae_n). \end{aligned}$$

5.2.2. The Significance of a solution in $\mathbf{Z}_{(p)}$

We notice that $\mathbf{Z} \subseteq \mathbf{Z}_{(p)}$. If (5.2.1.1) has a solution in $\mathbf{Z}$, it must have a solution in $\mathbf{Z}_{(p)}$. The following implications are also true (assuming A is square):

- (i) If $\det(A)$ is a power of the prime p then (5.2.1.1) has a solution over $\mathbf{Z}$ if and only if it has a solution over $\mathbf{Z}_{(p)}$.
- (ii) The system (5.2.1.1) has a solution over $\mathbf{Z}$ if and only if it has a solution over $\mathbf{Z}_{(p)}$ for all primes p dividing $\det(A)$.

(The statements (i), (ii) follow from the diagonalization of matrices over $\mathbf{Z}$, see [9], Chapter XI.)

Note that the above remarks do not completely solve the question of the parallel complexity of systems of linear Diophantine equations over $\mathbf{Z}$. The difficulties are (1) that we do not get the set solution of (ii); (2) there are exponentially many primes between $1, 2, \dots, \det(A)$ and finding the prime decomposition of $\det(A)$ may take exponential time. Therefore our result is particularly useful if $\det(A)$ is known as a product of powers of a small number of distinct primes.

If A is not square, say A is $m \times n$.

Case 1: $m < n$. Using Gram determinant technique. By Theorem 4.5.2, we can see that the $g(A)$ (the Gram determinant of A) and the determinant of a basis B of A have the

same prime decomposition.

Case 2: $m > n$. Reduce to the case $m \leq n$ by eliminating the redundant equations. This involves row operations over the fraction field Q of R .

5.3. Finding a Minimum Weighted Basis in a Linear Matroid is in NC

5.3.1. Matroids

A *matroid* (E, σ) (see [54]) is a system σ of subsets of a finite set E such that the following axioms hold:

- (1) σ is closed under subsets: i.e., if $A \in \sigma$, $B \subseteq A$, then $B \in \sigma$.
- (2) for each $M \subseteq E$, any two inclusion-maximal $X, Y \in \sigma$ with the property $X, Y \subseteq M$ satisfy $|X| = |Y|$.

A *basis* of a matroid (E, σ) is any inclusion-maximal $X \in \sigma$.

Now assume $h: E \rightarrow \mathbb{N}$ is a function. The Basis of Minimum Weight (abbreviated BMW) problem over the matroid (E, σ) asks for a basis $X \in \sigma$ such that

$$h(X) = \sum_{e \in X} h(e)$$

is minimal possible. An easy polynomial algorithm for BMW is the "greedy algorithm".

A matroid (E, σ) is called *linear* if there is a field F (hereby assumed NC-constructible) and a matrix A over F such that E is the set of columns of A and σ is the system of all linearly independent sets of columns of A .

In this section we show an NC-algorithm for BMW in a linear matroid. Note that, by the properties of the greedy algorithm (see [54]), all that relevant is the order

of the weights, which may also be considered different. Therefore, there is no restriction in assuming that the weights are different integers given in unary. Upon these assumptions, the BMW is unique.

5.3.2. Definition: Let $Q = F(x)$, where F is the field mentioned in 5.3.1, the valuation v on Q is defined as

$$v\left(\frac{p(x)}{q(x)}\right) = \max\{i \mid x^i \mid p(x)\} - \max\{j \mid x^j \mid q(x)\}$$

The discrete valuation ring R corresponding to Q , v is

$$R = \left\{ \frac{p(x)}{q(x)} \mid \gcd(q(x), x) = 1 \right\}.$$

The ring R may be denoted by $(F[x])_{(x)}$ (the ring $F[x]$ localized at the maximal ideal (x)).

5.3.3. The Algorithm

Input: A matrix A over a field F consisting of n columns, $A = (a_1, a_2, \dots, a_n)$, and associated with each column a_i a weight $h(i)$

Output: A basis of $\text{col}(A)$ with minimum sum of weights

Algorithm:

Step 1: for $i := 1$ to n do in parallel

multiply a_i by $x^{h(i)}$;

Step 2: Let $\bar{A}$ be the matrix over R resulting from Step 1, i.e.

$$\bar{A} := (a_1 x^{h(1)}, \dots, a_n x^{h(n)})$$

Step 3: run Step 2 (the deterministic part) of Algorithm 4.6.3 on $\bar{A}$, let $\bar{B}$ be the result, i.e., a basis of $\text{col}(\bar{A})$ consisting of certain columns of $\bar{A}$;

Step 4: Consider the set B of columns of A corresponding to the set $\bar{B}$ of columns of A . B is the basis of A of minimum weight.

Note that in the above algorithm, $\bar{A}$ is a matrix over $R=(F[x])_{(x)}$. Using the algorithm 4.6.3, we found basis of $\text{col}(\bar{A})$, $\bar{B}$, whose determinant has the minimum valuation among all the maximal square submatrices of $\bar{A}$. Let B be the corresponding set of columns of $\bar{B}$ in A , then

$$\det(\bar{B}) = x^{\sum_{i \in B} h(i)} \cdot \det(B), \text{ thus}$$

$$v(\det(\bar{B})) = \sum_{i \in B} h(i) \text{ is minimum.}$$

Therefore B is a minimum weighted basis of A . This establishes the correctness of the above algorithm.

In effect, by the uniqueness assertion in 5.3.1, the basis $\bar{B}$ with determinant of minimum valuation is unique and consequently we only need to use the deterministic part of algorithm 4.6.3. We conclude that our algorithm for finding a BMW of a linear matroid is in NC.

5.3.4. Minimum Spanning Tree

There is a particular class of linear matroids called *graphic matroids*. A *graphic matroid* is a matroid (E, σ) for which there exists an undirected connected graph $G=(V, E)$ (V is the set of vertices, E is the set of edges) such that σ is the set of all subsets S of the edge set E such that (V, S) is a forest (disjointed union of trees). For more details, see [54].

It is known that every graphic matroid may be identified with a linear matroid as follows ([54]): First choose an orientation of each edge (labeling one of its vertices as the "beginning vertex" and the other one as the "end vertex"). Let $V = \{v_1, \dots, v_m\}$, $E = \{e_1, \dots, e_n\}$. We define an $m \times n$ matrix $A = (a_{ij})$ as follows:

$$a_{ij} = \begin{cases} 1 & \text{if } v_i \text{ is the beginning vertex of } e_j \\ -1 & \text{if } v_i \text{ is the end vertex of } e_j \\ 0 & \text{otherwise} \end{cases}$$

A is called the *incident matrix* of the graph G . Let an $(m-1) \times n$ matrix $\tilde{A}$ be obtained from A by omitting the last row (corresponding to the vertex v_m). Note that the columns of $\tilde{A}$ (as well as the columns of A) are in bijective correspondence with E (the edges of G). Now it can be shown that a set of columns of $\tilde{A}$ is linearly independent over $\mathbb{Q}$ if and only if the corresponding set of edges forms a forest. This shows that E, σ is, indeed, a linear matroid.

In the beginning of this section, we assumed that the graph G was connected. This is really an unnecessary assumption. For non-connected graph, the matrix $\tilde{A}$ is constructed from A by omitting one row corresponding to a vertex in each connected component of G .

The BMW over a graphic matroid is equivalent to the minimum (weighted) spanning tree (abbreviated MST) problem in a graph. We note that NC-algorithms for MST are known (see [19]). However, there seems to be no apparent way of generalizing those algorithms (in [19]) beyond graphic matroids.

5.4. Finding the Basis of $\text{col}(A) \cap \text{col}(B)$ over an NC-constructible Discrete Valuation Ring is in ZNC

One of the basic problems of linear geometry over a field F is finding the intersection of two vector spaces V_1, V_2 . In order to investigate the computational complexity of this problem, we have to specify the format of the input and output. We thus note that a sufficiently general situation is achieved when V_1, V_2 are column spaces of two matrices with the same number of rows. Recall that for a matrix A over F with columns $a_1, \dots, a_n$ the column space of A is the vector space

$$\text{col}(A) = \left\{ \sum_{i=1}^n \lambda_i a_i \mid \lambda_i \in F \right\}.$$

Thus, assume that

$$V_i = \text{col}(A_i), \text{ for } i=1,2.$$

Let the columns of A_1 be $a_1, \dots, a_{n_1}$ and let the columns of A_2 be $b_1, \dots, b_{n_2}$. To find a basis of solutions of the (underdetermined) system of linear equations

$$A_1 x = A_2 y$$

for an n_1 -column unknown vector x an n_2 -column unknown vector y . However, there are known NC algorithms for finding bases of underdetermined systems of linear equations over fields (see [16], [40], [49] and Chapter 4 above). We thus conclude that finding an intersection of two vector spaces over a field is in NC.

In this section, we wish to investigate this problem over an NC-constructible discrete valuation ring R . To do this, however, we have to go back to the ring-theoretic concepts introduced in 4.1.1. Instead of vector spaces V_1, V_2 , we have to talk about free R -modules M_1, M_2 . Due to Theorem 4.4.2, it is sufficient to represent

free R -modules as column spaces of two matrices A, B with same number of rows. Recall from 4.1.1 that the column space of a matrix A over R with columns $a_1, \dots, a_n$ is

$$\text{col}(A) = \left\{ \sum_{i=1}^n \lambda_i a_i \mid \lambda_i \in R \right\}$$

Thus, one problem reduces to finding two bases of the R -module $\text{col}(A) \cap \text{col}(B)$ for two matrices A, B with the same number of rows. In this section, we show that this problem is in ZNC.

5.4.1. A Basis of $\text{col}(A) \cap \text{col}(B)$

Let A, B be $m \times n_A, m \times n_B$ matrices over the field Q respectively. We can assume that the rows of $C = A \cup B$ are independent over Q . First we find the bases of $\text{col}(A)$ and $\text{col}(B)$ over the discrete valuation ring R (by algorithm 4.6.3); this is in ZNC. We can assume that both A and B have Q -independent columns. We proceed to find a basis of $\text{col}(A) \cap \text{col}(B)$ over R .

5.4.2. The Algorithm

Apply algorithm 4.6.3 on C to find a basis of $\text{col}(C)$ over R , consisting of columns of C . Let this basis be $\bar{C}$. After multiplying C by $\bar{C}^{-1}$ from the left (equivalent row operations) and a possible permutation of columns within A and B , C turns into a matrix of the following form:

$$(5.4.1.1.) \quad (a_1, \dots, a_r \mid e_1, \dots, e_m \mid b_1, \dots, b_t)$$

where $(a_1, \dots, a_r, e_1, \dots, e_{n_A-r}) = C^{-1} \cdot A$, and $(e_{n_A-r+1}, \dots, e_m, b_1, \dots, b_t) = C^{-1} \cdot B$,

$t = n_A + n_B - r - m$. Now let

$$k=n_A-r,$$

$$a_i=(a_{1i}, \dots, a_{mi})^T, b_j=(b_{1j}, \dots, b_{mj})^T,$$

$$a'_i=(0, \dots, 0, a_{(k+1)i}, \dots, a_{mi})^T, b'_j=(b_{1j}, \dots, b_{kj}, 0, \dots, 0)^T,$$

where $i=1, \dots, r$, and $j=1, \dots, t$.

We claim that $(\bar{C}a'_1, \dots, \bar{C}a'_r, \bar{C}b'_1, \dots, \bar{C}b'_t)$ is a basis of $\text{col}(A) \cap \text{col}(B)$ over R . Here, we put $\bar{a}_i=\bar{C}a'_i$, and $\bar{b}_i=\bar{C}b'_i$.

5.4.3. Lemma: $(\bar{a}_1, \dots, \bar{a}_r, \bar{b}_1, \dots, \bar{b}_t) \in \text{col}(A) \cap \text{col}(B)$.

Proof: It is sufficient to show that

$$(a'_1, \dots, a'_r, b'_1, \dots, b'_t) \in \text{col}(\bar{C}^{-1}A) \cap \text{col}(\bar{C}^{-1}B).$$

We have

$$a'_i = a_i - \sum_{j=1}^k a_{ji} e_j, \text{ hence } a'_i \in \text{col}(\bar{C}^{-1}A).$$

$$a'_i = \sum_{j=k+1}^m a_{ji} e_j, \text{ hence } a'_i \in \text{col}(\bar{C}^{-1}B).$$

Similarly for b'_i . $\square$

5.4.4. Lemma: $(\bar{a}_1, \dots, \bar{a}_r, \bar{b}_1, \dots, \bar{b}_t)$ generate $\text{col}(A) \cap \text{col}(B)$ over R .

Proof. Again, we work with $a'_i, b'_i, \bar{C}^{-1}A, \bar{C}^{-1}B$. For any $x \in \text{col}(\bar{C}^{-1}A) \cap \text{col}(\bar{C}^{-1}B)$

$$\begin{aligned} x &= \sum_{i=1}^r s_i a'_i + \sum_{i=1}^k u_i e_i \\ &= \sum_{i=1}^{m-r} \bar{s}_i b'_i + \sum_{i=k+1}^m \bar{u}_i e_i \end{aligned}$$

a'_i 's are R -generated by e_i 's, $i=k+1, \dots, m$, b'_i 's are R -generated by e_i 's, $i=1, \dots, k$, and e_i 's are independent, therefore

$$\sum_{i=1}^r s_i a'_i = \sum_{i=k+1}^m \bar{u}_i e_i$$

$$\sum_{i=1}^k u_i e_i = \sum_{i=1}^{m-r} \bar{s}_i b'_i$$

$$x = \sum_{i=1}^r s_i a'_i + \sum_{i=1}^{m-r} \bar{s}_i b'_i. \quad \square$$

5.4.5. Lemma: $(\bar{a}_1, \dots, \bar{a}_r, \bar{b}_1, \dots, \bar{b}_t)$ are independent over Q .

Proof: We show that $(a'_1, \dots, a'_r, b'_1, \dots, b'_t)$ are Q -independent. For any two subspaces, V, W of a vector space over Q ,

$$\dim(V) + \dim(W) = \dim(V+W) + \dim(V \cap W)$$

where $V+W$ denotes the linear hull of $V \cup W$.

So

$$\dim(a'_1, \dots, a'_r, b'_1, \dots, b'_t) =$$

$$\dim(\text{col}(A) \cap \text{col}(B)) = \dim(\text{col}(A)) + \dim(\text{col}(B)) - \dim(\text{col}(A) + \text{col}(B)) =$$

$$n_A + n_B - m = r + t \quad (\text{by the definition of } t)$$

Thus $a'_1, \dots, a'_r, b'_1, \dots, b'_t$ generate a vector space over Q of dimension $r+t$ and hence are independent. $\square$

5.4.6. Theorem: $(\bar{a}_1, \dots, \bar{a}_r, \bar{b}_1, \dots, \bar{b}_t)$ is a basis of $\text{col}(A) \cap \text{col}(B)$ over R .

Proof: Follows directly from Lemmas 5.4.3, 5.4.4, 5.4.5. $\square$

5.5. p -adic Approximation is in ZNC

The problem of (vector) p -adic approximation problem is defined as follows: let $a_1, \dots, a_n, b$ be integral column vectors of dimension k (which simply means k -tuples of integers written in columns). We look for integers $\lambda_1, \dots, \lambda_n$ such that

$$(*) \quad b - \sum_{i=1}^n \lambda_i a_i \text{ is divisible by the largest possible power } p^s \text{ of } p,$$

p being a fixed prime number. In this section, we show that vector p -adic approximation is in ZNC.

Our first objective is to reduce the vector p -adic approximation problem to a similar problem in the discrete valuation ring $\mathbf{Z}_{(p)}$. Before we do this, however, we analyze the circumstances in which the problem (*) lacks a solution. It is obvious that the only way this can happen is that, for each $j \geq 0$, there is an n -tuple $\lambda^{(j)} = (\lambda_1^{(j)}, \dots, \lambda_n^{(j)})$ such that $p^j \mid (b - \sum_{i=1}^n \lambda_i^{(j)} a_i)$. One way this can happen is that the system of linear Diophantine equations

$$(**) \quad \sum_{i=1}^n a_i x_i = b$$

is solvable. However, this is not the only way. For example, consider $p=2$, $k=1$, $n=1$, $a_1=3$, $b=5$. Put $\lambda_1^{(j)} = -\frac{5}{3}(2^{4j}-1)$ (It is easy to see this is an integer). Moreover, obviously,

$$2^j \mid (b - \lambda_1^{(j)} a_1), \quad j \geq 0.$$

On the other hand, $a_1 x_1 = b$ is the system

$$3x=5$$

which obviously has no integral solution. We therefore see that the unsolvability of (*)

is implied by, but not equivalent to the solvability of $(**)$ in $\mathbf{Z}$. We may therefore ask for a necessary and sufficient condition for the unsolvability of $(*)$. It turns out that such a necessary and sufficient condition is the solvability of $(**)$ in $\mathbf{Z}_{(p)}$. This is one of the motivations for introducing the ring $\mathbf{Z}_{(p)}$. The above statement as well as other aspects of reducing p -adic approximation in $\mathbf{Z}$ to p -adic approximation in $\mathbf{Z}_{(p)}$ are established in Section 5.5.1.

5.5.1. Reducing p -adic Approximation in $\mathbf{Z}$ to p -adic Approximation in $\mathbf{Z}_{(p)}$

We first define p -adic approximation in $\mathbf{Z}_{(p)}$ in the same way as $(*)$, with the exception that $\lambda_1, \dots, \lambda_n$ are allowed to be elements of $\mathbf{Z}_{(p)}$. This is consistent with the general concept of p -adic approximation in a discrete valuation ring as defined below in 5.5.2. The desired connection between p -adic approximation in $\mathbf{Z}$ and $\mathbf{Z}_{(p)}$ is established by the following theorem.

5.5.1.1. Theorem: Let $b, a_1, \dots, a_n$ be integral column vectors of dimension k .

(1) The problem $(*)$ is unsolvable in $\mathbf{Z}$ if and only if the system of linear equations

$$\sum_{i=1}^n x_i a_i = b \text{ is solvable in } \mathbf{Z}_{(p)}.$$

(2) Let $(\mu_1, \dots, \mu_n)$ be the solution of $(*)$ in $\mathbf{Z}_{(p)}$ and let $s < \infty$ be the largest integer

such that $(p^s | b - \sum_{i=1}^n \mu_i a_i)$ in $\mathbf{Z}_{(p)}$. Then there are integers $(\lambda_1, \dots, \lambda_n)$ such that

$$p^s | (\lambda_i - \mu_i) \text{ in } \mathbf{Z}_{(p)} \text{ for } 1 \leq i \leq n.$$

Moreover, any such n -tuple of integers $(\lambda_1, \dots, \lambda_n)$ is a solution of $(*)$ in $\mathbf{Z}$.

Remarks: Before we prove Theorem 5.5.1.1, we show that how it leads to an NC-reduction of the problem $(*)$ in $\mathbf{Z}$ to the problem $(*)$ in $\mathbf{Z}_{(p)}$. In the sequel, we show

that the problem (*) in $\mathbf{Z}_{(p)}$ (and, in fact, in any NC-constructible discrete valuation ring) is in ZNC. Thus, we conclude that vector p -adic approximation over $\mathbf{Z}$ is in ZNC. The algorithm for an NC-reduction of the problem (*) in $\mathbf{Z}_{(p)}$ is presented below.

5.5.1.2. The Algorithm

This algorithm described here reduces p -adic approximation over $\mathbf{Z}$ to a p -adic approximation over $\mathbf{Z}_{(p)}$.

Input: Integer column vectors $b, a_1, \dots, a_n$ of dimension k .

Output: The integral solution $(\lambda_1, \dots, \lambda_n)$ of (*) and the number s or the information that no solution exists.

Algorithm:

Step 1: Check whether the system $\sum_{i=1}^n x_i a_i = b$ is solvable in $\mathbf{Z}_{(p)}$ (using the algorithm from Section 5.2). If so, output "No Solution Exists". If not, continue.

Step 2: Find the solution $(\mu_1, \dots, \mu_n)$ of the problem (*) in $\mathbf{Z}_{(p)}$ and the number s .

Let $\mu_i = \frac{s_i}{q_i}$, $s_i, q_i \in \mathbf{Z}$, $\neg (p \mid q_i)$.

{Comment. The number s is less than or equal to the number t discussed in Theorem 5.5.3 and Section 5.5.5 below and therefore, in particular, polynomial in the size of the input. By Theorem 5.5.1.1, we now need to solve the congruences

$$\lambda_i q_i \equiv s_i \pmod{p^s}$$

in $\mathbf{Z}$ and output the n -tuple $(\lambda_1, \dots, \lambda_n)$. This is dealt with in the subsequent steps.)

Step 3: Find a number ϵ , $0 < \epsilon < p$, such that

$$\varepsilon q_i \equiv 1 \pmod{p}.$$

Note that p is constant so that ε may be found by the Euclidean algorithm. Put $q = \varepsilon q_i$.

Step 4: Express

$$q = \sum_{j=0}^u \beta_j p^j, \quad 0 \leq \beta_j < p, \quad \beta_j \in \mathbb{Z}.$$

This can be done by a divide and conquer algorithm in NC. Note that the size of q is polynomial and hence so is u . Also, by step 3,

$$\beta_0 = 1.$$

Step 5: Let $Q(x) = \sum_{j=0}^u \beta_j x^j$. Expressing $\frac{1}{Q(x)}$ by a Taylor series

$$\frac{1}{Q(x)} = \sum_{j=0}^{\infty} \gamma_j x^j,$$

the numbers γ_j are integers and satisfy the linear recurrence relation

$$\gamma_{-u+1} = \cdots = \gamma_{-2} = \gamma_{-1} = \gamma_0$$

$$\gamma_0 = 1$$

$$\gamma_k = -\sum_{j=1}^n \beta_j \gamma_{k-j} \quad \text{for } k \geq 1.$$

Using dynamic programming, we can find $\gamma_0, \dots, \gamma_{s-1}$ in NC.

Step 6: We have

$$Q(x) \left(\sum_{j=0}^{s-1} \gamma_j x^j \right) \equiv 1 \pmod{x^s}$$

in $\mathbb{Z}[x]$, therefore

$$Q(p) \left(\sum_{j=0}^{s-1} \gamma_j p^j \right) \equiv 1 \pmod{p^s}$$

in $\mathbb{Z}$. We have $Q(p) = q$ and $q = \varepsilon q_i$. Put

$$\lambda_i = \varepsilon s_i \left(\sum_{j=0}^{s-1} \gamma_j p^j \right).$$

We conclude that

$$\lambda_i q_i = s_i q \left(\sum_{j=0}^{s-1} \gamma_j p^j \right) \equiv s_i \pmod{p^s}$$

as required.

Step 7: Output the numbers $(\lambda_1, \dots, \lambda_n), s$.

Proof of Theorem 5.5.1.1: We say that b is s -approximable by $a_1, \dots, a_n$ in $\mathbf{Z}$ (respectively in $\mathbf{Z}_{(p)}$) if there are $\lambda_1, \dots, \lambda_n \in \mathbf{Z}$ (respectively $\in \mathbf{Z}_{(p)}$) such that

$$p^s \mid (b - \sum_{i=1}^n \lambda_i a_i).$$

We claim that s -approximability in $\mathbf{Z}$ and $\mathbf{Z}_{(p)}$ for $s < \infty$ are equivalent. In fact, this is obvious from the fact that for each $\mu \in \mathbf{Z}_{(p)}$ and $s \in \mathbf{N}$ there is a $\lambda \in \mathbf{Z}$ with $p^s \mid (\mu - \lambda)$.

Thus the only thing that remains to be shown to establish Theorem 5.5.1.1 is the fact that if b is s -approximable by $a_1, \dots, a_n$ in $\mathbf{Z}_{(p)}$ for each $s \in \mathbf{N}$ then the equation

$$\sum_{i=1}^n x_i a_i = b.$$

has a solution in $\mathbf{Z}_{(p)}$. This follows from the subsequent discussion of p -adic approximation in discrete valuation rings. $\square$

5.5.2. The p-adic Approximation Problem in a Discrete Valuation Ring R

Let R be an NC-constructible valuation ring with fraction field Q , valuation v and maximal ideal m . Let $x=(x_1, \dots, x_k)^T$ be a column vector over R , the valuation on x be $v(x)=\min\{v(x_i)|1 \leq i \leq k\}$.

Given columns $a_1, \dots, a_n, b$ of dimension k over field Q , find the coefficients $\lambda_i \in R$ such that

$$v(b - \sum_{i=1}^n \lambda_i a_i) \text{ be maximal.}$$

Choose an element $p \in m - m^2$. Without loss of generality we can assume that $a_1, \dots, a_n, b$ are column vectors over R (because we can always multiply the columns by a constant power of p and obtain a new set of columns in R) and that they are R -independent.

5.5.3. Theorem: Let $B=(a_1, \dots, a_n, p^t e_1, \dots, p^t e_k)$. The largest possible value of t such that $p^t e_i$ is a part of a basis of $\text{col}(B)$ consisting of columns of B is $\leq \frac{1}{2}v(g(A))$, where $A=(a_1, \dots, a_n)$.

Proof: Let $\bar{A}$ be a basis of B , say $\bar{A}=(p^t e_1, \dots, p^t e_i, a_1, \dots, a_{k-i})$. Note that $A=\bar{A}X$, where X is a matrix over R .

$$g(A)=\det(\bar{A}X X^T \bar{A}^T)=\det(\bar{A})g(X)\det(\bar{A}^T)=(\det(\bar{A}))^2 g(X)$$

$$v(g(A)) \geq 2v(\det(\bar{A})) + v(g(X)) \geq 2v(\det(\bar{A})) \geq 2t$$

Therefore $t \leq \frac{1}{2}v(g(A))$. $\square$

If $t > \frac{1}{2}v(g(A))$, $a_1, \dots, a_n$ generate $\text{col}(B)$ and we either find an exact solution

$(\lambda_1, \dots, \lambda_n)$ such that $v(b - \sum_{i=1}^n \lambda_i a_i) = \infty$, i.e., $b = \sum_{i=1}^n \lambda_i a_i$, or no solution at all.

5.5.4. The Algorithm

Algorithm:

Step 1: for $t := 1$ to $\lceil \frac{v(g(A))}{2} \rceil$ do in parallel

Step 1.1. Use algorithm 4.6.3 to find a basis of

$(a_1, \dots, a_n, p^t e_1, \dots, p^t e_k)$ over R . Let this basis be

$(a_1, \dots, a_r, p^t e_{r+1}, \dots, p^t e_k)$;

Step 1.2. solve system of linear equations

$$(5.5.4.1.) \sum_{i=1}^r \lambda_i a_i + p^t \sum_{j=r+1}^k \delta_j e_j = b \text{ in } R;$$

Step 1.3. If there is a solution in Step 1.2, then record the current value of

t and $\lambda_1, \dots, \lambda_r$

end for;

Step 2: find the largest t recorded in Step 3, and output this t and the corresponding

$\lambda_1, \dots, \lambda_r$. (Note $\lambda_{r+1}, \dots, \lambda_n$ are 0.)

5.5.5. Discussion of the Algorithm

Finding a solution of the system (5.5.4.1.) is equivalent to finding a solution of the system

$$(5.5.5.1.) \sum_{i=1}^n \lambda_i a_i + p^t \sum_{j=1}^k \delta_j e_j = b$$

as $a_1, \dots, a_r, p^t e_{r+1}, \dots, p^t e_k$ form a basis. Now (5.5.5.1) is, by definition,

equivalent to

$$v(b - \sum_{i=1}^n \lambda_i a_i) \geq t.$$

The upper bound of t equals $\frac{1}{2}v(g(A))$ and is within polynomial range. For,

$$v(g(A)) = v(\det(AA^T)) \leq \log_p(\det(AA^T))$$

$$\leq \log_p(n! (\max\{|(AA^T)_{ij}|\})^n)$$

$$= \log_p n! + n \cdot \log_p \max\{|(AA^T)_{ij}|\}.$$

The above statement is true for $R = \mathbb{Z}_{(p)}$. For an arbitrary NC-constructible ring R , an analogous estimate is based on the axioms (1), ..., (4) in 4.2 (a bound for the determinant of a matrix follows from (1) and the analysis of Chistov's algorithm). This guarantees that our algorithm may be implemented so that to use only polynomially many processors.

5.6. Finding a Coherent Basis of two Free R -Modules $A \subseteq B$ over an NC-constructible Discrete Valuation Ring is in ZNC

5.6.1. Theorem: Let $M_1 \subseteq M_2$ be finitely generated free R -modules (R is a discrete valuation ring). Then there exists a basis B for M_2 and a mapping $g: B \rightarrow \mathbb{N}_0 \cup \{+\infty\}$ such that

$$\{p^{g(b)}b \mid b \in B \text{ \& } g(b) \neq \infty\}$$

is a basis for M_1 .

Proof: by combining the results in [6] and [9]. $\square$

The coherent basis problem is to find such a basis B . The basis B of M_2 is called a *coherent basis* of M_2 with respect to $M_1 \subseteq M_2$, the basis

$$\{p^{g(b)}b \mid b \in B \& g(b) \neq \infty\}$$

is called a *coherent basis* of M_1 with respect to $M_1 \subseteq M_2$. Without loss of generality, we can assume M_2 is the space of all columns over R of dimension n and $M_1 = \text{col}(A)$, where $A = (a_1, \dots, a_l)$ with independent columns of dimension n .

5.6.2. Lemma: Let $A = (a_1, \dots, a_l)$ be a matrix over R . For each $1 \leq j \leq l$, let

$$v_j = \max \{v(\sum_{i=1}^{j-1} r_i a_i + a_j) \mid r_i \in R\}$$

Assume that $v_1 \leq v_2 \leq \dots \leq v_l$, let k be the largest number such that $v_k \neq \infty$, then a

coherent basis for $\text{col}(A)$ consists of elements $\bar{a}_j = a_j + \sum_{i=1}^{j-1} r_{ij} a_i$, $r_{ij} \in R$, $j=1, \dots, k$,

such that $v(\frac{\bar{a}_j}{p^{v_j}}) = 0$.

Proof: Let $b_j = \frac{\bar{a}_j}{p^{v_j}}$, $j=1, \dots, k$. In fact $(b_1, \dots, b_j)$ should be a basis of $\text{col}(I_j)$. We

prove this fact by induction on j .

Assume that $(b_1, \dots, b_{j-1})$ contain an R -invertible square submatrix. If $(b_1, \dots, b_j)$ does not contain an R -invertible square submatrix, then by Lemma 4.4.1, $\pi(b_1, \dots, b_j)$ is not regular in R/m (the class field), therefore

$$\sum_{q=1}^j \bar{s}_q \pi(b_q) = 0 \text{ in } R/m, \bar{s}_q \in R/m$$

where $\pi(b_q)$ is the coordinate wise projection of b_q . In addition, $\bar{s}_j \neq 0$, otherwise

$$\sum_{q=1}^{j-1} \bar{s}_q \pi(b_q) = 0 \text{ in } R/m$$

contradicting the induction hypothesis.

Choose $s_q \in R$ such that $\pi(s_q) = \bar{s}_q$, for $q=1, \dots, j$. Then

$$\sum_{q=1}^j \pi(s_q) \pi(b_q) = 0 \text{ in } R/m, \text{ i.e. } \pi\left(\sum_{q=1}^j s_q b_q\right) = 0$$

By the definition of the class field R/m ,

$$v\left(\sum_{q=1}^j s_q b_q\right) > 0,$$

$\bar{s}_j = \pi(s_j) \neq 0$, therefore $v(s_j) = 0$. Compute:

$$v_j = v(\bar{a}_j) \geq v\left(\sum_{q=1}^j s_q p^{v_j - v_q} \bar{a}_q\right) = v\left(\sum_{q=1}^j s_q p^{v_j} \cdot \frac{\bar{a}_q}{p^{v_q}}\right) \geq v_j + v\left(\sum_{q=1}^j s_q b_q\right) > v_j$$

A contradiction. $\square$

5.6.3. Definition: An ordering of columns of A which satisfies $v_1 \leq v_2 \leq \dots \leq v_n$ is called a *canonical ordering*.

5.6.4. Theorem: Let $A = (a_1, \dots, a_l)$ be an $n \times l$ matrix. We assume that $(a_1, \dots, a_l, p^t e_1, \dots, p^t e_n)$ have a basis $\bar{A} = (a_1, \dots, a_l, p^t e_{j+1}, \dots, p^t e_n)$, where t is a non-negative integer. Let B be the matrix consisting of the first j rows of $(a_1, \dots, a_l)$. Let C be the matrix consisting of the last $n-j$ rows of $\bar{A}^{-1} \cdot (a_{j+1}, \dots, a_l)$, then

$$(5.6.4.1.) \quad v_q^{(B)} = v_q^{(A)} \leq t, \quad 1 \leq q \leq j$$

$$(5.6.4.2.) \quad v_{q-j}^{(C)} = v_q^{(A)} - t, \quad j+1 \leq q \leq l$$

Consequently, if the columns of B, C are in canonical order, so are the columns in A .

Proof: First prove (5.6.4.1).

Since $\bar{A}$ is a basis, there exist matrices M, N over R such that

$$(5.6.4.3.) (a_1, \dots, a_j)M + (p^t e_{j+1}, \dots, p^t e_n)N = (p^t e_1, \dots, p^t e_j)$$

Take the first j row equations of (5.6.4.3). We have

$$BM = p^t I_j, M = p^t B^{-1}$$

Rewrite (5.6.4.3) as

$$(a_1, \dots, a_j)M = p^t ((e_1, \dots, e_j) - (e_{j+1}, \dots, e_n)N).$$

Then

$$p^t (a_1, \dots, a_j)M = (a_1, \dots, a_j)p^t B^{-1}, \text{ hence } (a_1, \dots, a_j)B^{-1} \text{ is a matrix}$$

over R , therefore

$$(a_1, \dots, a_j) = WB$$

for some matrix W over R . Consequently, the rows of B form a R -basis of the row space of $(a_1, \dots, a_j)$, $A = WB$.

Now for any column z of dimension j over R (we are most interested in the case $z = (z_1, \dots, z_{j-1}, 1)^T$) obviously $v(Az) \leq v(Bz)$, as B is a part of A and also

$$v(Az) = v(WBz) \geq v(Bz).$$

Thus, $v(Az) = v(Bz)$. We conclude that $v_q^{(B)} = v_q^{(A)}$, $1 \leq q \leq j$. This concludes the proof of (5.6.4.1), except for the inequality.

Let $B = (b_1, \dots, b_j)$, also $v(z) = 0$. Suppose

$$v(Bz) > t$$

then

$$t = t + v(z) = v(p^t B^{-1} Bz) = v(MBz) \geq v(Bz) > t,$$

a contradiction. Hence we have proved (5.6.4.1) completely.

Now we proceed to prove (5.6.4.2). Let D, E be matrices over R such that

$$(5.6.4.4.) (a_{j+1}, \dots, a_l) = (a_1, \dots, a_j)D + p^t(e_{j+1}, \dots, e_n)E.$$

Then

$$(a_{j+1}, \dots, a_l) = \bar{A} \begin{bmatrix} D \\ E \end{bmatrix} = \bar{A} \cdot \bar{A}^{-1} (a_{j+1}, \dots, a_l)$$

Rewrite (5.6.4.4.) as

$$(a_{j+1}, \dots, a_l) = (a_1, \dots, a_j)D + p^t \begin{bmatrix} 0 \\ C \end{bmatrix}.$$

Thus, if d_q is the q -th column of D ($j+1 \leq q \leq l$), and c_q the q -th column of $\begin{bmatrix} 0 \\ C \end{bmatrix}$, then

$$(5.6.4.5.) a_q - (a_1, \dots, a_j)d_q = p^t c_q,$$

$$v\left(\sum_{s=1}^{q-1} \lambda_s c_s + c_q\right) = v\left(p^{-t} \left(\sum_{s=j+1}^{q-1} \lambda_s a_s + a_q\right) + \sum_{i=1}^j u_i a_i\right)$$

where $u_i = -\left(\sum_{s=j+1}^{q-1} \lambda_{is} d_s + d_q\right)$. Thus,

$$v_{q-j}^{(C)} \leq v_q^{(A)} - t, \quad q = j+1, \dots, l.$$

On the other hand, let

$$v(\bar{a}_q) = v\left(a_q + \sum_{i=1}^{q-1} r_i a_i\right) = v_q, \quad j+1 \leq q \leq l.$$

Then

$$\bar{a}_q = a_q + \sum_{i=1}^{q-1} r_i a_i = p^t c_q + \sum_{i=j+1}^{q-1} r_i c_i p^t + \sum_{i=1}^j s_i a_i, \quad r_i, s_i \in R \text{ (by 5.6.4.5.)}$$

As the first j rows of $p^t c_q + p^t \sum_{i=j+1}^{q-1} r_i c_i$ are 0, we conclude that the valuation of

the first j rows of $\sum_{i=1}^j s_i a_i \geq v_q^{(A)}$. But since the first j rows of $(a_1, \dots, a_j)$ form a

basis of the row space of $(a_1, \dots, a_j)$, we have

$$v\left(\sum_{i=1}^j s_i a_i\right) \geq v_q^{(A)} \text{ and}$$

$$v_q^{(A)} \leq v(\bar{a}_q - \sum_{i=1}^j s_i a_i) = v(p^t c_q + \sum_{i=j+1}^{q-1} r_i c_i p') =$$

$$t + v(c_q + \sum_{i=j+1}^{q-1} r_i c_i) \leq t + v_q^{(C)}.$$

Thus we have proved (5.6.4.2). $\square$

5.6.5. Lemma: Let w be the valuation on $R[\sqrt{p}]$. Then

$$\max\{v(\sum_{i=1}^{k-1} r_i a_i + a_k) \mid r_i \in R\} = \frac{1}{2} \max\{w(\sum_{i=1}^{k-1} \bar{r}_i a_i + a_k) \mid \bar{r}_i \in R[\sqrt{p}]\}.$$

Proof. Since $R \subseteq R[\sqrt{p}]$,

$$\max\{v(\sum_{i=1}^{k-1} r_i a_i + a_k) \mid r_i \in R\} \leq$$

$$\frac{1}{2} \max\{w(\sum_{i=1}^{k-1} r_i a_i + a_k) \mid r_i \in R[\sqrt{p}]\}.$$

For any $r'_i, r''_i \in R$,

$$w(\sum_{i=1}^{k-1} (r'_i + r''_i \sqrt{p}) a_i + a_k) =$$

$$w(\sum_{i=1}^{k-1} r'_i a_i + a_k + \sqrt{p} \sum_{i=1}^{k-1} r''_i a_i) \leq$$

$$w(\sum_{i=1}^{k-1} r'_i a_i + a_k),$$

since $w(\sum_{i=1}^{k-1} r'_i a_i + a_k) \neq w(\sqrt{p} \sum_{i=1}^{k-1} r''_i a_i)$. $\square$

Our algorithm works in the ring $R[\sqrt{p}]$ (see the Appendix), rather than R . This is because Theorem 5.6.4 separates A into two matrices, one of which has the diagonal factors of valuation $\leq t$ and the other one the factors of valuation $\geq t$. Thus, the factors with valuation t may be mixed in both matrices. By setting $t = \text{integer} + \frac{1}{2}$ we make

sure that no such mixing occurs. The correctness of this procedure is guaranteed by the above lemma.

5.6.6. The Algorithm:

Input: a column independent matrix $A=(a_1, \dots, a_l)$

Output: a coherent basis

Algorithm:

Step 1: Find an upper bound b of valuations of diagonal factors as in 5.5.3 (see the inequality in (5.6.4.1.));

Step 2: for $t := \lfloor \frac{b}{2} \rfloor + \frac{1}{2}$ do

find a basis of $(a_1, \dots, a_n, p^t e_1, \dots, p^t e_n)$ by algorithm 4.6.3; recursively run the same algorithm in parallel for B and C (see Theorem 5.6.5);

Step 3: Step 2 finds a canonical ordering of columns of A . Use p -adic approximation and Lemma 5.6.2 to find a coherent basis of $\text{col}(A)$;

Step 4: Find a set of rows of A forming a basis of the row space of A . The vectors of the remaining row coordinates form the part of the coherent basis of $\text{col}(I_n)$, $\text{col}(A)$ which does not hit $\text{col}(A)$ (see Theorem 5.6.1).

Finally similar remarks to those in 5.5.5 apply to algorithm 5.6.6 as well.

5.7. Finding the p -component of an Abelian Group is in ZNC

One major application of the coherent basis algorithm is finding p -components of finitely generated abelian groups.

In Section 4.1.1, we considered a ring R , an R -module M and a prime ideal $I \subseteq R$. We defined the localization M_I of the R -module M at the ideal I . Now we recall that a $\mathbf{Z}$ -module is the same as an Abelian group. This is due to the fact that scalar multiplication by elements of $\mathbf{Z}$ in an Abelian group A may be defined by iterated addition: for $a \in A$, $k \in \mathbf{N}$, we define

$$\begin{aligned}
 ka &= a + \cdots + a, \text{ (} k \text{ times)} \\
 0a &= 0 \\
 (*) \quad (-1)a &= -a \\
 (-k)a &= k(-a).
 \end{aligned}$$

Thus, we may talk about the localization $A_{(p)}$ of an Abelian group A (a $\mathbf{Z}$ -module) at the prime ideal $(p) \subseteq \mathbf{Z}$. The Abelian group $A_{(p)}$ is called the p -component of the Abelian group A .

We now turn to finitely generated Abelian groups. An Abelian group A is called *finitely generated* if there is a finite set

$$\{z_1, \cdots, z_n\} \in A$$

such that every element $x \in A$ can be expressed in the form

$$x = \sum_{i=1}^n k_i z_i, \quad k_i \in \mathbf{Z}.$$

The elements z_i are called the *generators* of A .

5.7.1. Lemma: For every finitely generated Abelian group A with generators $z_1, \dots, z_n$ there is an integral matrix $\bar{A}=(a_{ij})$ such that

$$a_{11}z_1 + \dots + a_{1n}z_n = 0$$

.....

$$a_{m1}z_1 + \dots + a_{mn}z_n = 0$$

and, moreover, for any integral row vector $b=(b_1, \dots, b_n)$ such that

$$\sum_{i=1}^n b_i z_i = 0,$$

b is an integral linear combination of the rows of $\bar{A}$. We call $\bar{A}$ the matrix of A with respect to the generators $z_1, \dots, z_n$.

Proof: see [9]. $\square$

Now let $z=(z_1, \dots, z_n)^T$ be " n -dimensional column vector over A " formed by generators z_i . Based on the definitions (*), it is allowed to multiply z by integral matrices of the suitable size. For example, Lemma 5.7.1 states that

$$\bar{A} \cdot z = 0.$$

Now it is well known that for every integral matrix $\bar{A}$ there are invertible integral matrices B, C such that

$$(5.7.1.1.) B\bar{A}C = \begin{pmatrix} M & O_{12} \\ O_{21} & O_{22} \end{pmatrix}$$

where M is a $k \times k$ diagonal matrix with positive integral diagonal entries such that $m_1 | m_2 | \dots | m_k$ and O_{12}, O_{21}, O_{22} are $k \times (n-k), (n-k) \times k, (n-k) \times (n-k)$ zero matrices respectively ([9], Chapter XI).

We now see that $B\bar{A}$ (as well as $\bar{A}$) is a matrix of A with respect to the generators z , because B is invertible over $\mathbf{Z}$. Thus, $B\bar{A}C$ is the matrix of A with respect to the generators in the " n -dimensional column A -vector" $C^{-1}z$. We conclude that

$$A \cong \mathbf{Z}_{m_1} \times \cdots \times \mathbf{Z}_{m_k} \times \mathbf{Z} \times \cdots \times \mathbf{Z}.$$

where $\mathbf{Z}$ occurs $n-k$ times in the above expression and $\cong$ denotes isomorphic. Also note that

$$(\mathbf{Z}_n)_{(p)} = \mathbf{Z}_{p^{v(n)}}, \text{ for } n \geq 1$$

$$(\mathbf{Z})_{(p)} = \mathbf{Z}_{(p)}$$

where $v(n)$ is the valuation of n with respect to $\mathbf{Z}_{(p)}$. Also, localization commutes with (finite) products (see [6]). We have thus proved the following theorem (for details, see [9]).

5.7.3. Theorem: Every finitely generated Abelian group A can be expressed in the form

$$A \cong \mathbf{Z}_{m_1} \times \cdots \times \mathbf{Z}_{m_k} \times \mathbf{Z} \times \cdots \times \mathbf{Z}.$$

($\mathbf{Z}$ occurs $n-k$ times in the above expression) where $m_1 | \cdots | m_k$ are positive integers. Moreover,

$$A_{(p)} \cong \mathbf{Z}_{p^{v(m_1)}} \times \cdots \times \mathbf{Z}_{p^{v(m_k)}} \times \mathbf{Z}_{(p)} \times \cdots \times \mathbf{Z}_{(p)};$$

(there are $n-k$ $\mathbf{Z}_{(p)}$'s in the above expression).

Proof: see [9]. $\square$

There are known polynomial algorithms for finding the numbers $m_1, \dots, m_k, k$ (based on finding the expression (5.5.5.1.), see [9], Chapter XI). It is unknown whether finding these numbers is in NC or RNC or ZNC. Our progress in this matter

is the following theorem.

5.7.3. Theorem: Finding the numbers $v(m_1), \dots, v(m_k), k$ is in ZNC.

Proof. Similarly as above, one can show that whenever O, F are invertible matrices over $\mathbf{Z}_{(p)}$ such that

$$(5.7.3.1.) \quad O\bar{A}F = \begin{bmatrix} W & O_{12} \\ O_{21} & O_{22} \end{bmatrix}$$

where W is a $l \times l$ diagonal matrix with diagonal entries: $p^{w_1}, \dots, p^{w_l}$, $0 \leq w_1 \leq \dots \leq w_l$ and O_{21}, O_{12}, O_{22} are zero matrices, then $k=l$ and $w_i = v(m_i)$. (Denote the matrix on the right hand side of (5.7.3.1) by P .) It therefore suffices to present a ZNC algorithm for constructing the expression (5.7.3.1). We exploit Algorithm 5.6.6. which allows us to find a coherent basis $\{b_1, \dots, b_m\}$ of the free $\mathbf{Z}_{(p)}$ -module M of all $\mathbf{Z}_{(p)}$ -columns of dimension m with respect to the submodule $\text{col}(\bar{A})$. Let B be the $m \times m$ matrix formed by the columns $b_1, \dots, b_m$. By the assumption that $\{b_1, \dots, b_m\}$ is a basis,

(5.7.3.2.) B is invertible over $\mathbf{Z}_{(p)}$.

Now by the definition of a coherent basis, there is an $l \leq m$ and numbers $0 \leq w_1 \leq \dots \leq w_l$ such that

(5.7.3.3.) $\{p^{w_1}b_1, \dots, p^{w_l}b_l\}$ is a basis of $\text{col}(\bar{A})$.

Moreover, the numbers $l, w_1, \dots, w_l$ are found by Algorithm 5.6.6 (see Lemma 5.6.2).

There is thus an invertible $n \times n$ matrix F over $\mathbf{Z}_{(p)}$ such that

$$\bar{A}F = (p^{w_1}b_1, \dots, p^{w_l}b_l, 0, \dots, 0) = BP. \quad (n-l \text{ 0's})$$

We conclude that

$$B^{-1}\bar{A}F=P$$

so that (5.7.3.1) is satisfied with $O=B^{-1}$ and therefore $l=k, w_i=v(m_i)$. $\square$

Chapter 6

OPEN PROBLEMS AND FUTURE RESEARCH

There are several questions and conjectures resulting from this research which are not dealt with in this thesis.

As mentioned several times in this introduction, the central open question of the area remains *whether or not GCD is in NC*. While we still have no idea of the answer to this question, this thesis gives a possible method for tackling a related question as to whether 2-ILP is in NC, or P-complete.

In effect, we show that this is equivalent to asking the complexity of the MOD^* function with arguments $x, a_1, \dots, a_n$, where $a_1, \dots, a_n$ arise in the intermediate stages of Euclidean algorithm. Let us express this last condition on $a_1, \dots, a_n$, for the moment, by saying that $a_1, \dots, a_n$ satisfy the Euclidean constraint.

We know that if $a_1, \dots, a_n$ are not required to satisfy any constraint then $MOD^*(x, a_1, \dots, a_n)$ is P-complete (we show this by a simulation argument). Is it possible to adapt the simulation argument in such a way that it works for sequences satisfying the Euclidean constraint? Then 2-ILP is P-complete. Even if this question is too hard, there is still room for research: one may consider constraints which are weaker than (or otherwise related to) the Euclidean constraint. To mention a concrete question: Is, for a given constant $C > 0$, $MOD^*(x, a_1, \dots, a_n)$ P-complete for sequences $a_1, \dots, a_n$ satisfying the constraint

$$a_i > Ca_{i+1}, i=1, \dots, n-1?$$

We conjecture that this is true.

As to linear Diophantine equations, we find ZNC algorithms for what number theorists call *local questions*. A typical *global question* is the following: Is the problem of solving systems of linear Diophantine equations NC-reducible to GCD (or, at least, k-ILP)?

Another open question stems from the material of Section 5.3: we show that finding a basis of minimum weight is in NC for a linear matroid. This was previously known for graphic matroids. Is an analogous result true for general matroids?

APPENDIX

1. Let Q be a field, v is a valuation defined on Q . R is the discrete valuation ring with respect to Q, v . We know that Q is the fraction field of R . Let m be the maximal ideal of R . Choose an element $p \in m - m^2$, we can easily see that $v(p)=1$. Now we

introduce a new ring $R[p^{\frac{1}{t}}]$, where t is an integer, and

$$R[p^{\frac{1}{t}}] = \{r_0 + r_1 p^{\frac{1}{t}} + r_2 p^{\frac{2}{t}} + \cdots + r_{t-1} p^{\frac{t-1}{t}} \mid r_i \in R\}.$$

2. Define the arithmetic operations on $R[p^{\frac{1}{t}}]$ in the most obvious way: $\forall$

$a, b \in R[p^{\frac{1}{t}}]$, let

$$a = a_0 + a_1 p^{\frac{1}{t}} + a_2 p^{\frac{2}{t}} + \cdots + a_{t-1} p^{\frac{t-1}{t}}, \quad b = b_0 + b_1 p^{\frac{1}{t}} + b_2 p^{\frac{2}{t}} + \cdots + b_{t-1} p^{\frac{t-1}{t}}$$

$$(1) \quad a \pm b = (a_0 \pm b_0) + (a_1 \pm b_1) p^{\frac{1}{t}} + (a_2 \pm b_2) p^{\frac{2}{t}} + \cdots + (a_{t-1} \pm b_{t-1}) p^{\frac{t-1}{t}}$$

$$(2) \quad a \cdot b = \left(\sum_{i=0}^{t-1} a_i p^{\frac{i}{t}} \right) \cdot \left(\sum_{i=0}^{t-1} b_i p^{\frac{i}{t}} \right) = \left(\sum_{i=0}^{t-1} c_i p^{\frac{i}{t}} \right)$$

where $c_k = \sum_{0 \leq i+j=k < t} a_i b_j + \sum_{0 \leq i+j-t=k \leq t-2} p a_i b_j$ and $0 \leq i, j \leq t-1$.

3. Theorem:

(1) The fraction field of $R[p^{\frac{1}{t}}]$ is

$$Q' = \{q_0 + q_1 p^{\frac{1}{t}} + q_2 p^{\frac{2}{t}} + \cdots + q_{t-1} p^{\frac{t-1}{t}} \mid q_i \in Q\}$$

(2) There is a valuation w on Q' , given by

$$w(\bar{q} = q_0 + q_1 p^{\frac{1}{t}} + q_2 p^{\frac{2}{t}} + \cdots + q_{t-1} p^{\frac{t-1}{t}}) = \min\{tv(q_i) + i \mid 0 \leq i \leq t-1\}$$

$$(3) R[p^{\frac{1}{t}}] = \{\bar{q} \in Q' \mid w(\bar{q}) \geq 0\}$$

(The proof of this theorem is below.)

4. Definition: Let F be a field and $P(x)$ a polynomial over F . $P(x)$ is called *irreducible* if there exist no polynomials $Q(x)$, $R(x)$ over F of degree ≥ 1 , such that $Q(x)R(x) = P(x)$.

5. Theorem: If a polynomial $P(x)$ ($\deg(P(x)) = t$) is irreducible over a field F , then the ring $F(\theta) = F[\theta]/(P(\theta))$ is a field.

Proof: see [6]. $\square$

(Here $F[\theta]$ is the ring of polynomials with one indeterminate θ over F , $(P(\theta))$ is the principal ideal of $F[\theta]$ generated by $P(\theta)$. The proof of Theorem 5 relies on the fact that $(P(\theta))$ is a maximal ideal if $P(\theta)$ is irreducible, see 4.1.5.)

To prove Theorem 3, we choose to consider the case $P(x) = x^t - p$.

6. Definition: A polynomial $a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$ over R is called *primitive* if $\min\{v(a_i) \mid 0 \leq i \leq n\} = 0$.

7. Lemma (Gauss): A polynomial in R is irreducible over Q if and only if it is irreducible over R .

Proof. Suppose $P(x)$ is reducible over Q and irreducible over R . Let $P(x)=P_1(x)P_2(x)$, where $P_1(x), P_2(x)$ are polynomials over Q , and $\deg(P_1(x)) \cdot \deg(P_2(x)) > 0$. We consider the maximum i_1, i_2 such that

$$P_1(x)p^{i_1} \in R[x], P_2(x)p^{i_2} \in R[x] \text{ and both } P_1(x)p^{i_1}, P_2(x)p^{i_2} \text{ are primitive.}$$

i.e. the i_1 -th coefficient a_{i_1} of $P_1(x)p^{i_1}$, and i_2 -th coefficient b_{i_2} of $P_2(x)p^{i_2}$ are such that

$$v(a_{i_1})=v(b_{i_2})=0 \text{ and } \max\{i_1, i_2\} > 0, \text{ hence } i_1+i_2 > 0.$$

Consider the equality

$$p^{i_1+i_2}P(x)=(P_1(x)p^{i_1})(P_2(x)p^{i_2})$$

Since $v(a_{i_1})=v(b_{i_2})=0$, then the (i_1+i_2) -th coefficient of $p^{i_1+i_2}P(x)$, $c_{i_1+i_2}$, has

$$v(c_{i_1+i_2})=0$$

But $i_1+i_2 > 0$, because $P(x)$ is assumed irreducible over R , and $v(c_{i_1+i_2}) \geq i_1+i_2$. A contradiction. $\square$

The above lemma reduces our problem to showing that $x^t - p$ is irreducible over R .

8. Lemma (Eisenstein): Let $P(x)=a_0+a_1x+a_2x^2+\cdots+a_nx^n$ be a polynomial over R . Let further $v(a_n)=0$, $v(a_i)>0$, for all $0 < i < n$, $v(a_0)=1$. Then $P(x)$ is irreducible over R .

Proof. Suppose

$$P(x)=(\sum_{i=0}^k b_i x^i)(\sum_{j=0}^m c_j x^j),$$

where $b_i, c_j \in R$, $k+m=n$. Since

$$v(a_n) = v(b_k \cdot c_m) = v(b_k) + v(c_m) = 0,$$

we have

$$v(b_k) = v(c_m) = 0$$

In addition, $v(a_0) = v(b_0 c_0) = v(b_0) + v(c_0) = 1$. Without loss of generality, assume $v(b_0) = 0$ and $v(c_0) = 1$. Let $j \leq m$ be the smallest with

$$v(c_j) = 0 \text{ (there is at least one such } j, \text{ namely } j = m).$$

But this leads to

$$v(a_j) = v(b_0 c_j + b_1 c_{j-1} + \cdots + b_j c_0) = 0.$$

since we have $v(b_0 c_j) = 0$, $v(b_i c_{j-i}) > 0$ for all $i > 0$. A contradiction. $\square$

9. Corollary: For any $t > 0$, the polynomial $x^t - p$ is irreducible over $\mathcal{Q}$.

Proof. By Lemmas 7, 8, and

$$P(x) = x^t - p = -p + 0x + \cdots + 0x^{t-1} + 1x^t.$$

$$v(a_0) = v(-p) = 1, v(a_n) = v(1) = 0, v(a_i) = v(0) = +\infty > 0. \quad \square$$

10. Proof of Theorem 3: Only the part (1) deserves a proof. $P(x) = x^t - p$ is irreducible over $\mathcal{Q}$, $\deg(P(x)) = t$. By Theorem 5, the ring $\mathcal{Q}(x) = \mathcal{Q}[x]/(P(x))$ is a field. Elements of

$$\mathcal{Q}(p^{\frac{1}{t}}) = \mathcal{Q}[p^{\frac{1}{t}}]/(P(p^{\frac{1}{t}}))$$

may be expressed in the form

$$q_0 + q_1 p^{\frac{1}{t}} + q_2 p^{\frac{2}{t}} + \cdots + q_{t-1} p^{\frac{t-1}{t}}, q_i \in \mathcal{Q}$$

and the arithmetic operations coincide. Hence $\mathcal{Q}'$ is a field containing $R[p^{\frac{1}{t}}]$, and is

obviously contained in the fraction field Q'' of $R[p^{\frac{1}{t}}]$. We conclude that $Q' = Q''$. $\square$

BIBLIOGRAPHY

- [1] Aggarwal, A., Chazelle, B., Guibas, L., O'Dunlaing, C., Yap, C., "Parallel computational geometry", *26th IEEE Symp. on the Foundations of Computer Science*, 1985, pp. 486-474.
- [2] Aho, A.V., Hopcroft, J.E., Ullman, J.D., *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, 1974.
- [3] Akl, S.G., *Parallel Sorting Algorithms*, Academic Press, Orlando, 1985.
- [4] Alon, N., Meggido, N., "Parallel linear programming in fixed dimension almost surely in constant time", *Proc. 32nd Annual IEEE Symp. on Foundations of Comp. Sci.*, 1990, pp. 574-582.
- [5] Anderson, R.J., Mayr, E.W., "Parallel and greedy algorithm", *Advances in Computing Research, 4 (Parallel and Distributed Computing)*, ed. F. P. Preparata, 1987, pp. 17-38.
- [6] Atiyah, M.F., MacDonald, I.G., *Introduction to Commutative Algebra*, Addison-Wesley, Reading, 1969.
- [7] Babai, L., "A Las Vegas NC algorithm for isomorphism of graphs with bounded multiplicity of eigenvalues", *Proc. 27th Annual IEEE Symp. on Foundations of Comp. Sci.*, 1986, pp. 303-312.
- [8] Berkowitz, S.J., "On computing the determinant in small parallel time using a small number of processors", *Info. Proc. Letters*, vol. 18, 1984, pp. 147-150.
- [9] Birkhoff, G., MacLane, S., *Algebra*, 3rd edition, Chelsea, New York, 1988.
- [10] Borevich, Z.I., Shafarevich, I.R., *Number Theory*, Academic Press, Orlando, 1966.
- [11] Borodin, A., Cook, S.A., Pippenger, N., "Parallel computation for well-endowed rings and space bounded probabilistic machines", *Inform. and Control.*, vol. 58, 1983, pp. 113-136.
- [12] Borodin, A., von zur Gathen, J., Hopcroft, J.E., "Fast parallel matrix and GCD computations", *Proc. 23rd Annual IEEE Symp. on Foundations of Comp. Sci.*, 1982, pp. 65-71.
- [13] Bourbaki, N., *Elements of Mathematics (Algebra I)*, Springer Verlag, Berlin, 1989.
- [14] Burton, D.M., *Elementary Number Theory*, Allyn and Bacon, New York, 1977.

- [15] Cassels, J.W.S., *An Introduction to Diophantine Approximations*, Cambridge University Press, Cambridge, 1957.
- [16] Chistov, A.L., "Fast parallel calculation of the rank of matrices over a field of arbitrary characteristic", *Proc. Intl. Conf. Foundations of Computing Theory*, Springer Lecture Notes in Computer Science, vol. 199, Berlin, 1985, pp. 63-79.
- [17] Chvatal, V., *Linear Programming*, W.H. Freeman and Company, New York, 1983.
- [18] Cook, S.A., "Towards a complexity theory of synchronous parallel computation", *Enseign. Math.*, vol. 27, 1981, pp. 99-124.
- [19] Cook, S.A., "A taxonomy of problems with fast parallel algorithms", *Inform. and Control*, vol. 64, 1985, pp. 2-22.
- [20] Csanky, L., "Fast parallel matrix inversion algorithms", *SIAM J. Computing* 5, 1976, pp. 618-623.
- [21] Deng, X., "Mathematical Programming: Complexity and Application", Ph.D. Dissertation, Stanford University, 1989.
- [22] Deng, X., "On parallel complexity of integer linear programming," *Proc. ACM Symp. on Parallel Algorithms and Architectures*, 1989, pp. 110-116.
- [23] Eberly, W., "Very fast parallel matrix and polynomial arithmetic", *Proc. 25th Annual IEEE Symp. on Foundations of Comp. Sci.*, 1984, pp. 21-30.
- [24] Epstein, D., Galil, Z., "Parallel algorithmic techniques for combinatorial computation", *Technical Report*, Computer Science Department, Columbia University, New York, 1988.
- [25] Ficken, F.A., *Linear Transformations and Matrices*, Prentice-Hall, Englewood Cliffs, 1984.
- [26] Garey, M.R., Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, 1979.
- [27] von zur Gathen, J., "Parallel algorithms for algebraic problems", *SIAM J. Comput.*, vol. 13, 1984, pp. 802-824.
- [28] von zur Gathen, J., "Parallel Powering", *Proc. 25th Annual IEEE Symp. on Foundations of Comp. Sci.*, 1984, pp. 31-36.
- [29] von zur Gathen, J. "Parallel arithmetic computations: a survey", *Proc. 12th Intl.*

Symp. on Math. Foundations of Comp. Sci., Springer Lecture Notes in Computer Science, vol. 233, Berlin, 1986, pp. 93-112.

[30] Goldberg, M., Spencer, T., "A new parallel algorithm for the maximal independent set problem", *Proc. 28th Annual IEEE Symp. on Foundations of Comp. Sci.*, 1987, pp. 161-165.

[31] Grigoriev, D.Y., Karpinski, M., "The matching problem for bipartite graphs with polynomially bounded permanents is in NC", *Proc. 28th Annual IEEE Symp. on Foundations of Comp. Sci.*, 1987, pp. 166-172.

[32] Hirschberg, D.S., Wong, C.K., "A polynomial-time algorithm for the Knapsack problem with two variables", *J. ACM*, vol. 23, 1976, pp. 147-154.

[33] Hopcroft, J.E., Ullman, J.D., *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, 1979.

[34] Ibarra, D.H., Moran, S., Roiser, L.E., "A note on the parallel complexity of computing the rank of order n matrices", *Info. Proc. Letters*, vol. 11, 1980, p. 162.

[35] Israeli, A., Itai, A., "A fast and simple randomized parallel algorithm for maximal matching", *Info. Proc. Letters*, vol. 22, 1986, pp. 77-80.

[36] Israeli, A., Shiloach, Y., "An improved parallel algorithm for maximal matching", *Info. Proc. Letters*, vol. 22, 1986, pp. 57-60.

[37] Kannan, R., "A polynomial algorithm for the two-variable integer linear programming problem", *J. of ACM*, vol. 27, 1980, pp. 118-122.

[38] Kannan, R., Miller, G.L., Rudolph, L., "Sublinear parallel algorithm for computing the greatest common divisor of two integers", *Proc. 25th Annual IEEE Symp. on Foundations of Comp. Sci.*, 1984, pp. 7-11.

[39] Karloff, H.J., "A Las Vegas RNC algorithm for maximum matching", *Combinatorica*, vol. 6, 1986, pp. 387-392.

[40] Karp, R.M., Ramachandran, V., "A survey of parallel algorithms for shared-memory machines", *Handbook of Theoretical Computer Science*, North-Holland, Amsterdam, 1990, pp. 869-941.

[41] Karp, R.M., Upfal, E., Wigderson, A., "Constructing a perfect matching is in random NC", *Combinatorica*, vol. 6, 1986, pp. 35-48.

[42] Karp, R.M., Wigderson, A., "A fast parallel algorithm for the maximal independent set problem", *J. ACM*, vol. 32, 1985, pp. 762-773.

- [43] Knuth, D.E., *The Art of Computer Programming: Seminumerical Algorithms*, vol. 2, Addison-Wesley, Reading, 1981.
- [44] Lenstra, H.W., Jr., "Integer linear programming with a fixed number of variables", *Mathematics of Operations Research* 8, 1983, pp. 538-548.
- [45] Lovasz, L., "On determinants, matchings and random algorithms", *Fundamentals of Computing Theory*, ed. L. Budach, Akademie-Verlag, Berlin, 1979.
- [46] Lovasz, L., *An Algorithmic Theory of Numbers, Graphs and Convexity*, CBMS-NSF Regional Conference Series in Applied Mathematics 50, SIAM, Philadelphia, 1986.
- [47] Luby, M., "A simple parallel algorithm for the maximal independent set problem", *Proc. 17th Annual ACM Symp. on Theory of Computing*, 1985, pp. 1-10.
- [48] McDonald, B.R., *Linear Algebra over Commutative Rings*, Prentice-Hall, Englewood Cliffs, 1967.
- [49] Mulmuley, K., "A fast parallel algorithm to compute the rank of a matrix over an arbitrary field", *Combinatorica*, vol. 7, 1987, pp. 101-104.
- [50] Mulmuley, K., Vazirani, U.V., Vazirani, V.V., "Matching is as easy as matrix inversion", *Proc. 19th Annual ACM Symp. on Theory of Computing*, 1987, pp. 355-364.
- [51] Pan, V., "Fast and efficient algorithms for the exact inversion of integer matrices", *Fifth Annual Foundations of Software Technology and Theoretical Computer Science Conference*, Springer Lecture Notes in Computer Science, vol. 241, Berlin, 1985, pp. 504-521.
- [52] Pan, V., Galil, Z., "Improved processor bounds for algebraic and combinatorial problems in RNC", *Proc. 26th Annual IEEE Symp. on Foundations of Comp. Sci.*, 1985, pp. 490-495.
- [53] Pan, V., Reif, J., "Efficient parallel solution of linear systems", *Prod. 17th Annual ACM Symp. on Theory of Computing*, 1985, pp. 143-152.
- [54] Papadimitriou, C.H., Steiglitz, K., *Combinational Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, 1982.
- [55] Pratt, V.R., Stockmeyer, L.J., "A characterization of the power of vector machines", *J. Comput. System Sci.*, vol. 12, 1975, pp. 198-221.
- [56] Shannon, C.E., "A universal Turing machine with two internal states", *Automata*

Studies, Princeton University Press, Princeton, 1956, pp. 157-166.

[57] Valiant, L.G., "Parallel computation", *Proc. Seventh IBM Symp. on Mathematical Foundations of Computer Science*, 1982, pp. 173-189.

[58] Zachos, S., "A decisive characterization of BPP", *Inform. and Control*, vol. 69, 1986, pp. 125-135.