Jus Measuring: Algorithm and Complexity

Min-Zheng Shieh

August 25, 2004

ABSTRACT

We study the water jug problem and obtain new lower and upper bounds on the minimum number of measuring steps. These bounds are tight and significantly improve previous results. We prove that to compute the crucial number $\mu_{\mathbf{c}}(x)$ (i.e., $\min\limits_{x=\mathbf{x}\cdot\mathbf{c}} ||\mathbf{x}||_1$, where $\mathbf{c} \in \mathbf{N}^n, x \in \mathbf{N}, ||\mathbf{x}||_1 = \sum_{i=1}^n |x_i|$) for estimating the minimum measuring steps is indeed a problem in P^{NP}. Moreover, we prove that testing whether $\mu_{\mathbf{c}}(x)$ is bounded by a fixed number is indeed NP-complete and thus the optimal jug measuring problem is NP-hard, which was also proved independently by [6]. Finally, we give a pseudo-polynomial time algorithm for computing $\mu_{\mathbf{c}}(x)$ and a polynomial time algorithm, which is based on *LLL* basis reduction algorithm, for approximating the minimum number of jug measuring steps efficiently.

Keywords: Jug problems, lower bound, upper bound, NP, LLL algorithm

CONTENTS

LIST OF FIGURES

1. INTRODUCTION

There is a scene in the movie *Die Hard: With a Vengeance*, where the actors need to defuse a bomb by measuring four gallons of water using two jugs of capacities three and five. This measuring problem is the so called water jug problem [3, 5, 2], which has been studied for a long time and is a popular problem for programming contests, a frequent heuristic search exercise in artificial intelligence and algorithms. Boldi et al. [3] generalized the jug problem by considering a set of jugs of fixed capacities and they found out which quantities are measurable and proved upper and lower bounds on the number of steps necessary for measuring a specified amount of water. More specifically, the general water jug problem is [3]:*given a set of jugs of fixed capacities, find out which quantities are measurable.* In this paper we also deal with the *optimal jug measuring problem*, which considers the minimum number of measuring steps. Suppose we are given n jugs with integer capacities c_i, $i \in [n]$, where $[n]$ denotes the set $\{1, \cdots, n\}$. WLOG, we assume $c_1 \leq c_2 \leq \cdots \leq c_n$. We can perform three types of elementary operations on the jugs with the following notations introduced by Boldi et al.[3]:

(1) $\downarrow i$: *fill* the ith jug (from the source) up to its capacity, and we call it the *fill* operation;

(2) $i \uparrow$: *empty* the ith jug (into the drain) completely, and we call it the *empty* operation;

(3) $i \to j$: *pour* the contents of the ith jug to the jth jug, $i \neq j$, and we call it the *pour* operation. Note that pour operation never changes the total sum of the contents, and at the end of this operation, the ith jug is empty or the jth jug is full.

By following Boldi et al. [3], we formally describe the operation as follows. Let O denotes the set of all possible elementary operations, that is, $O = \{\downarrow i | \forall i \in [n]\} \cup \{i \uparrow | \forall i \in [n]\} \cup \{i \to j | \forall i, j \in [n], i \neq j\}$. Let $\mathbf{N}$ be the set of non-negative integers. A state is a vector $\mathbf{s} \in \mathbf{N}^n$, where s_i denotes the amount contained in jug i. The next-state function $\delta : \mathbf{N}^n \times O \to \mathbf{N}^n$ is defined as following:

(1) $\delta(\mathbf{s}, \downarrow i) = (s_1, \ldots, s_{i-1}, c_i, s_{i+1}, \ldots, s_n)$;

(2) $\delta(\mathbf{s}, i \uparrow) = (s_1, \ldots, s_{i-1}, 0, s_{i+1}, \ldots, s_n)$;

(3) $\delta(\mathbf{s}, i \to j) = (t_1, \ldots, t_n)$, where $t_k = s_k$ for all $k \notin \{i, j\}$, $t_i = max\{0, s_i - (c_j - s_j)\}$, and $t_j = min\{c_j, s_i + s_j\}$.

We call a finite sequence of elementary operations an algorithm for jug measuring, where each operation is a feasible move. We say a state $\mathbf{s}$ is reachable if $\delta(0, \sigma) = \mathbf{s}$ for some algorithm $\sigma \in O^*$. It is clear that $\delta(\mathbf{s}, \epsilon) = \mathbf{s}$ and for any algorithm $\sigma \in O^*$ and $o \in O$ we have $\delta(\mathbf{s}, \sigma o) = \delta(\delta(\mathbf{s}, \sigma), o)$. A quantity $x \in \mathbf{N}$ is measurable via algorithm σ iff one of the components of $\delta(\mathbf{0}, \sigma)$ is equal to x. For convenience, let $\mathbf{c} = \{c_1, \cdots, c_n\}$ and $\gcd(\mathbf{c})$ denote the greatest common divisor of $c_1, \cdots, c_n$. The set of quantities that are measurable using the capacities in $\mathbf{c}$ is denoted by $\mathbf{M}(\mathbf{c})$. Boldi et al.[3] proved the following theorem, which shows that all of the measurable quantities are exactly the multiples of the greatest common divisor of the capacities.

Theorem 1: [3] $\mathbf{M}(\mathbf{c}) = \{m \cdot \gcd(\mathbf{c}) |$ for all non-negative integer $m \leq \frac{c_n}{\gcd(\mathbf{c})}\}$.

We extend the measurability by defining that a quantity $x \in \mathbf{N}$ is *additively measurable* via algorithm σ iff the sum of the contents in $\delta(\mathbf{0}, \sigma)$ is equal to x. The set of quantities that are *additively measurable* using the capacities in $\mathbf{c}$ is denoted by $\mathbf{M}^+(\mathbf{c})$. Obviously, this is more general than $\mathbf{M}(\mathbf{c})$ and can measure larger quantities up to $\sum_{i=1}^{n} c_i$. We prove that all of the additively measurable quantities again are exactly the multiples of the greatest common divisor of the capacities, that is: $\mathbf{M}^+(\mathbf{c}) = \{m \cdot \gcd(\mathbf{c}) |$ for all non-negative integer $m \leq \frac{\sum_{i=1}^{n} c_i}{\gcd(\mathbf{c})}\}$.

Each $x \in \mathbf{M}^+(\mathbf{c})$ has a (one or more) vector representation $\mathbf{x} = (x_1, \cdots, x_n) \in \mathbf{Z}^n$ with respect to $\mathbf{c}$, such that $x = \mathbf{x} \cdot \mathbf{c} = \sum_{i=1}^{n} x_i c_i$. It is not hard to see that such a representation $\mathbf{x}$ implies a sequence of operations achieving the quantity x, and vice versa. Define $\mu(x) = \min_{x = \mathbf{x} \cdot \mathbf{c}} ||\mathbf{x}||_1$, where $||\mathbf{x}||_1 = \sum_{i=1}^{n} |x_i|$. $\mu(x)$ plays an important role on estimating the upper and lower bounds of measuring steps. Boldi et al. proved the following bounds:

Theorem 2: [3] (1). Every $x \in M(\mathbf{c})$ can be measured in at most $\frac{5}{2}\mu(x)$ steps. (2). No algorithm can measure $x \in M(\mathbf{c})$ in less than $\frac{1}{2}\mu(x)$ steps.

With a deeper observation we improve the lower bound to $2\mu(x) - 1$, which is tight in many cases. We prove that: No algorithm can measure $(0, \cdots, 0, x)$ for all $x \in M(\mathbf{c})$ in less than $2\mu(x) - 1$ steps. Also all the measurable quantity can be measured in at most $2\mu(x)$ steps.

From the above, given $\mathbf{c}$ and x, $\mu(x)$ (for convenience, denoted as $\mu_{\mathbf{c}}(x)$) provides a pretty good estimation on the number of measuring steps. However it is not clear how to compute $\mu(x)$ efficiently. We prove that computing $\mu_{\mathbf{c}}(x)$ is in P^{NP}. Formal definition of the notation P^{NP} can be found in several standard textbooks as in the references [4, 10, 9]. Moreover, we prove that testing whether $\mu_{\mathbf{c}}(x)$ is bounded by a fixed number is actually NP-complete. The consequence is that the optimal jug measuring problem is

NP-hard.

Since computing $\mu_{\mathbf{c}}(x)$ is indeed in P^{NP}, we propose a polynoimial time approximation algorithm which is based on the famous *LLL* (Lenstra-Lenstra-Lovasz) basis reduction algorithm. In our experiments, it gives much better results than its theorectical bounds. We also present some inapproximable results provided by Havas and Seifert [6] and conclude that the problem of computing $\mu_{\mathbf{c}}(x)$ is inapproximable in polynomial time within a factor of k, where $k \geq 1$ is an arbitrary constant.

The rest of the paper is organized as follows. In chapter 2, we characterize additive measurability and prove new lower and upper bounds for the number of minimum measuring steps. In chapter 3, we prove that computing $\mu_{\mathbf{c}}(x)$ is in P^{NP}, its bounded version is NP-complete, and the optimal jug measuring problem is *NP-hard*. A pseudo-polynomial time for computing $\mu_{\mathbf{c}}(x)$ is also given. In chapter 4, we give a polynomial time approximation algorithm for computing $\mu_{\mathbf{c}}(x)$. Inapproximable results are also given. Chapter 5 concludes the paper.

2. MEASURABILITY, LOWER AND UPPER BOUNDS

2.1 Measurability

First we prove that the additively measurable quantities are exactly the multiples of the gcd of all of the jug capacities.

Theorem 3: Given $\mathbf{c}$, $\mathbf{M}^+(\mathbf{c}) = \{m \cdot \gcd(\mathbf{c}) |$ for all non-negative integer $m \leq \frac{\sum_{i=1}^n c_i}{\gcd(\mathbf{c})}\}$

Proof. There are two parts to be proved. First we need to prove that for any non-negative multiple of $\gcd(\mathbf{c})$ (bounded by $\sum_{i=1}^n c_i$), it is additively measurable. Secondly, we need to show that any additively measurable quantity is a multiple of $\gcd(\mathbf{c})$. We prove both parts by induction on n, the number of jugs. It is trivial for both parts when $n = 1$. Assume that the theorem holds up to $n - 1$.

1. Assume $x = m \cdot \gcd(\mathbf{c})$ and $x \leq \sum_{i=1}^n c_i$, for some $m \in \mathbf{N}$. It is clear that $c_n \geq \gcd(c_1, c_2, c_3 \ldots, c_{n-1})$, since $c_1 \leq c_2 \leq c_3 \cdots \leq c_n$. If $x \leq \sum_{i=1}^{n-1} c_i$, then let $y \equiv x (\bmod \gcd(c_1, c_2, c_3 \ldots, c_{n-1}))$. We know that $x - y$ is a multiple of $\gcd(c_1, c_2, c_3 \ldots, c_{n-1})$ and thus a multiple of $\gcd(\mathbf{c})$. We already know x is a multiple of $\gcd(\mathbf{c})$ by assumption, and then so is y. By theorem 1, we have $y \in \mathbf{M}(\mathbf{c})$. This implies that we can reach $(0, 0, 0, \ldots, y)$ first. By induction hypothesis, $x - y \in \mathbf{M}^+(c_1, c_2, \ldots, c_{n-1})$. So we can reach a state $\mathbf{s}$ by using the first

$n-1$ jugs, where the sum of its contents is equal to $x-y$. Together with the quantity y in jug n, we can achieve the total sum x.

If $x > \sum_{i=1}^{n-1} c_i$, then let $y = x - \sum_{i=1}^{n-1} c_i \leq c_n$. We know that y is a multiple of $\gcd(\mathbf{c})$, since x and $\sum_{i=1}^{n-1} c_i$ are both multiples of $\gcd(\mathbf{c})$ and thus by theorem 1 we have $y \in \mathbf{M}(\mathbf{c})$. This implies that we can reach $(0, 0, 0, \ldots, x - \sum_{i=1}^{n-1} c_i)$ first, and then we can reach a state $\mathbf{s}$ with the sum of the full content in each jug equal to x, by filling all of the jugs other than jug n.

2. Assume x is additively measurable with $\mathbf{c}$. We want to show that x is a multiple of $\gcd(\mathbf{c})$. Let $(s_1, \cdots, s_n)$ be a reachable state with $x = \sum_{i=1}^{n} s_i$. It is obvious that each s_i is measurable. Again by theorem 1, we know each s_i is a multiple of $\gcd(\mathbf{c})$ and thus x is multiple of $\gcd(\mathbf{c})$.

This completes the proof. $\square$

2.2 Lower bound

In this section, we prove the following lower bound, which improves previous result, $\frac{1}{2}\mu(x)$, by Boldi et al.[3].

Theorem 4: No algorithm can measure $(0, \cdots, 0, x)$, for all $x \in \mathbf{M}(\mathbf{c})$, in less than $2\mu(x) - 1$ steps.

Actually we prove a stronger result:

Theorem 5: Let $\mathbf{s} = (s_1, \cdots, s_n)$ be a reachable state, $x = \sum_{i=1}^{n} s_i$ and n_{ne} be the number of non-zero entries of $\mathbf{s}$, then no algorithm can reach $\mathbf{s}$ in less than $2\mu(x) - n_{ne}$ steps.

It is clear that Theorem 4 is a special case of Theorem 5. We need the following lemma before proving the theorem.

Lemma 6: Let $\sigma = o_1 o_2 \cdots o_m \in O^*$ be an arbitrary sequence of m legal operations such that $\delta(0,\sigma) = (t_1, t_2, \cdots, t_n)$. Then for any $i \in [n]$, there exists another sequence of m operations ρ such that $\delta(0,\rho) = (t_1, \cdots, t_{i-1}, t'_i, t_{i+1}, \cdots, t_n)$, where $t'_i = 0$ or c_i.

Proof. For any $i \in [n]$, let $\omega = o_1 \cdots o_k$ be the maximum prefix of σ such that $\delta(0,\omega) = (u_1, \cdots, u_n)$, where $u_i = 0$ or c_i, i.e. after the $(k+1)$-st operation the i-th jug is neither full nor empty and this status (non-full and non-empty) of jug i is kept throughout the rest of operations in σ. We can assume such maximum prefix of σ always exists, otherwise the lemma is already true, since jug i will be either empty or full during the operations.

Now we want to construct a sequence of operations $\rho = \omega o'_{k+1} \cdots o'_m$, where for each $\ell \in \{k+1, \cdots, m\}$ define o'_ℓ to be $j \uparrow$ if $o_\ell = j \to i$ for some j, $\downarrow j$ if $o_\ell = i \to j$ for some j, or o_ℓ, otherwise. Note that we simply copy the operation(s) not related to jug i from σ.

We prove by induction on the number of operations after o_k in σ relating to jug i for the correctness of the above construction. If there is no operation relating to jug i after o_k, then it is clear that $\sigma = \omega = \rho$ and the claim is trivially true. If there is exactly one operation after o_k relating to jug i, then it must be o_{k+1} (otherwise we would be able to find a longer prefix ω), and o_{k+1} can only be $j \to i$ or $i \to j$ for some j. (1) If jug i is empty right before o_{k+1}, then o_{k+1} must be $j \to i$ and jug j must become empty after o_{k+1}, otherwise we won't know the amount poured from jug j to jug i. In this case we let $o'_{k+1} = j \uparrow$. Thus we have ρ as σ except with the $(k+1)$-st operation different. And it affects only the i-th jug, which is empty instead of being filled with the amount t_i. (2) If jug i is full right before o_{k+1}, then o_{k+1} must be $i \to j$ and jug j must become full after o_{k+1}, otherwise, with the same reasoning, we won't know the amount poured from jug i to jug j, since jug i will be partially filled. In this case we let $o'_{k+1} = \downarrow j$. The new sequence ρ will keep jug i full and won't affect the rest.

Assume that for up to d jug i related operations to the right of ω in σ, we can always successfully construct ρ as claimed. Now consider the case of $(d+1)$ jug i related operations after ω in σ. Let $r > k$ and o_r be the *last* such operation after o_k. Since o_r is after o_k, we know o_r cannot be $i \uparrow$ or $\downarrow i$. There are two possibilities for o_r, i.e. $j \to i$ or $i \to j$ for some j. In both cases, after the operation of o_r jug i cannot be empty or full by the choice of ω.

- **Case $o_r = j \to i$:** Right before o_r, jug i is neither full nor empty and after o_r jug i will maintain the same status but with extra water poured from jug j. Jug j must become empty after o_r, since jug i is partially filled. In this case we replace o_r with $j \uparrow$ and obtain a new sequence σ', which has d operations related to jug i. Then by induction hypothesis, we can construct ρ from σ' as required.

- **Case $o_r = i \to j$:** As above right before and after o_r, jug i must be partially filled. In this case, jug j must be full after o_r. We replace o_r with $\downarrow j$ and obtain a new sequence σ', which has d operations related to jug i. Then by induction hypothesis, we obtain ρ from σ' as required.

$\square$

We say a state $\mathbf{s} = (s_1, \cdots, s_n)$ is *reachable* if there exists a sequence of operations $\sigma \in O^*$ such that $\delta(0, \sigma) = \mathbf{s}$. We say that a reachable state $\mathbf{s}$ is additively measurable if $\sum_{i=1}^{n} s_i$ is additively measurable. Furthermore, σ is called *optimal*, if it is the shortest sequence of operations that reaches $\mathbf{s}$.

The following lemma states that when we empty a jug, it can be full right before emptying and when we fill water into a jug from the source it can be empty right before pouring.

Lemma 7: For any reachable state $\mathbf{s}$, there exists an optimal sequence of operations $\sigma' = o'_1 \cdots o'_m$ with $\delta(0, \sigma') = \mathbf{s}$, such that for any $k \in [m]$, if o'_k is $i \uparrow$ or $\downarrow i$ for some $i \in [n]$, then $\delta(0, o'_1 \cdots o'_{k-1})$ will reach a state with jug i full or empty, respectively.

Proof. Suppose there is an optimal sequence $\rho = o_1 \cdots o_m$, such that $\delta(0, \rho) = \mathbf{s}$ and there exist $k \in [m]$ and $i \in [n]$ with $o_k \in \{i \uparrow, \downarrow i\}$ and $\delta(0, o_1 \cdots o_{k-1}) = (t_1, \cdots, t_n)$ but $t_i \neq 0$ and c_i. Without loss of generality, let o_k be the rightmost of such operations in ρ that disagrees with the lemma. Then by lemma 6, we can find a sequence of operations $\omega = o'_1 \cdots o'_{k-1}$ such that $\delta(0, \omega) = (t_1, \cdots, t_{i-1}, t'_i, t_{i+1}, \cdots, t_n)$, where $t'_i = 0$ or c_i. Since $o_k \in \{i \uparrow, \downarrow i\}$, we have $\delta(0, o_1 \cdots o_k) = \delta(0, \omega o_k)$ and so $\delta(0, \rho) = \delta(0, \omega o_k \cdots o_m)$. By repeating this, we will eliminate all such disagreeing operations and obtain an optimal sequence of operations as claimed. $\square$

Lemma 8: For any reachable state $\mathbf{s}$ with $x = \sum_{i=1}^{n} s_i$, there exists an optimal sequence of operations $\sigma = o_1 \cdots o_m$ with $\delta(0, \sigma) = \mathbf{s}$, such that the number of fill and empty operations in σ is at least $\mu(x)$.

Proof. By Lemma 7, there exists an optimal sequence of operations σ such that the total amount of water from each jug is increased or decreased by the amount of c_i after $\downarrow i$ or $i \uparrow$ operation for each $i \in [n]$. Note that operations $i \to j$ and $j \to i$ do not change the total sum. Suppose for each

$i \in [n]$, there are f_i $(\downarrow i)$-operations and e_i $(i \uparrow)$-operations in σ. It is clear $x = \sum_{i=1}^{n}(f_i - e_i)c_i$ and thus $\sum_{i=1}^{n}|f_i - e_i| \geq \mu(x)$. From the above we have $\sum_{i=1}^{n}(f_i + e_i) \geq \sum_{i=1}^{n}|f_i - e_i| \geq \mu(x)$. $\square$

We prove the lower bound on the number of pour operations by inspecting a property of an optimal sequence of operation via a graph representation as following.

Lemma 9: Let $\sigma = o_1 \cdots o_m$ be an optimal sequence for a reachable state $\mathbf{s} = \delta(0, \sigma)$ that satisfies lemma 7. Let n_{ne} be the number of non-zero entries of $\mathbf{s}$, then the number of pour operations in σ is at least $\mu(x) - n_{ne}$.

Proof. We prove the lower bound by constructing a graph $G(V, E)$ from σ. For jug i, $i \in [n]$, σ can be partitioned into disjoint subsequences of operations, say $\sigma = \sigma_{i,1} \cdots \sigma_{i,g_i}$, where after each subsequence of operations jug i becomes empty and the first operation of the next subsequence will change the status of jug i into non-empty. Note that each subsequence of operations will make jug i empty once, except the last subsequence for which jug i may end up being non-empty. If the state of jug i changed from empty to non-empty g_i times with respect to σ, then σ will be partitioned into g_i subsequences. For each jug, there will be a unique partition.

For jug i, suppose we partition σ into g_i subsequences $\sigma_{i,1} \cdots \sigma_{i,g_i}$. We define the vertex set $V = \{v_{i,j} | i \in [n], j \in [g_i]\}$, where $v_{i,j}$ corresponds to $\sigma_{i,j}$. For a pour operation $o = i \to j$, if $o \in \sigma_{i,a}$ and $o \in \sigma_{j,b}$, where $a \in [g_i]$ and $b \in [g_j]$, then we define an edge $(v_{i,a}, v_{j,b})$ for operation o. For each pour operation we define an edge. Thus we define the edge set E to be the collection of all such edges. So $G(V, E)$ is well defined from σ. It is clear that the number of pour operations in σ is $|E|$.

For example, consider an instance with $\mathbf{c} = \{14, 28, 31\}$ and $x = 20$. Let $\sigma' = o_1 o_2 \cdots o_{14} = \downarrow 1 \circ \downarrow 3 \circ 1 \rightarrow 2 \circ \downarrow 1 \circ 1 \rightarrow 2 \circ 2 \uparrow \circ \downarrow 1 \circ 1 \rightarrow 2 \circ 3 \rightarrow 2 \circ 2 \uparrow \circ 3 \rightarrow 2 \circ \downarrow 3 \circ 3 \rightarrow 2 \circ 2 \uparrow$. It is clear that $\delta(\mathbf{0}, \sigma') = (0, 0, 20)$, but σ' is not an optimal sequence of operations. We construct a graph as in figure 2.1, where each block (or subsequence) represents a vertex and $g_1 = g_2 = 3, g_3 = 2$.

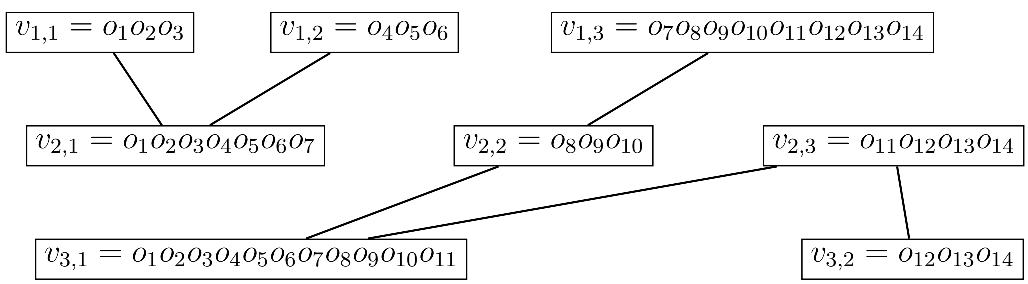

Fig. 2.1: Graph representation from the sequence σ'.

The number of fill operations associated with a vertex is at most 1, because o_k is $\downarrow i$ if and only if $\delta(\mathbf{0}, o_1 o_2 \cdots o_{k-1}) = (t_1, t_2, \ldots, t_{i-1}, 0, t_{i+1} \ldots, t_n)$. Also the number of empty operations associated with a vertex is at most 1. For each i, let e_i be the number of $(i \uparrow)$-operations and f_i be the number of $(\downarrow i)$-operations in σ. $|V| = \Sigma_{i=1}^n g_i \geq \Sigma_{i=1}^n max\{e_i, f_i\} \geq \Sigma_{i=1}^n |f_i - e_i| \geq \mu(x)$, since $\Sigma_{i=1}^n (f_i - e_i) c_i = x$. Since σ is optimal, $G(V, E)$ has at most n_{ne} connected components, where each component corresponds to at least one non-empty jug, in other words, each component contains at least one vertex v_{i,g_i} with jug i non-empty. A crucial observation is: if there are more than n_{ne} connected components in $G(V, E)$, then there must exist one connected component whose corresponding operations do not contribute in the measuring and can be removed without changing the final outcome, since these operations are redundant. For instance, as in figure 2.1, the connected component of $\{v_{1,1}, v_{1,2}, v_{2,1}\}$ does not connect to any v_{i,g_i} and thus all the operations in $v_{1,1}, v_{1,2}$, and $v_{2,1}$ related to jug 1 and 2 can be removed without changing the final jug status. While for any optimal sequence, this cannot happen.

Since there are at most n_{ne} connected components, there are at least $|V| - n_{ne}$ edges in $G(V, E)$. Thus $|E| \geq |V| - n_{ne}$. Since each edge stands for a pour operation and $|V| \geq \mu(x)$, there are at least $\mu(x) - n_{ne}$ pour operations in σ. $\square$

By Lemma 8 and Lemma 9, we have Theorem 5 as an immediate consequence. Note that this lower bound is tight for many cases, for example for all $x \in M(\mathbf{c})$.

2.3 Upper bound

Suppose we are allowed to use an extra jug with infinite capacity and $x = \sum_{i=1}^{n} c_i x_i$. Then the algorithm is simply: (1) for each $x_i > 0$ repeat $\{\downarrow i; i \to (n+1)\}$ for x_i times; (2) for each $x_i < 0$ repeat $\{(n+1) \to i; \uparrow i\}$ for $|x_i|$ times. It is clear that the total number of measuring steps is $2\sum_{i=1}^{n} |x_i|$ steps. With the above observation, given the optimal representation of x, we obtain an algorithm as in Figure 2.2, which measures x in $2\mu(x) + l - 1$ steps, where l is the minimum number of jugs needed to hold the quantity x. The key idea is simply simulate the imaginary jug of infinite capacity with the n jugs. And hence the upper bound won't be exactly $2\mu(x)$. Note that the upper and lower bounds are tight when we consider the case that the quantity x must fit into a jug in the last step. In other words, for $x \in M(\mathbf{c})$, given the optimal representation of x, our algorithm achieves the best possibility. However, it is not clear how to compute $\mu_{\mathbf{c}}(x)$ and the optimal representation efficiently.

Theorem 10: For all $x \in M^+(\mathbf{c})$, if we need at least l jugs to hold the quantity x, then the algorithm Measure additively measures x in $2\mu(x)+l-1$ steps.

We prove the correctness and analyze the algorithm with the following two lemmas.

Lemma 11: The algorithm Measure outputs a sequence σ of operations such that $\delta^*(\mathbf{0}, \sigma) = \mathbf{s}$, $\sum_{i=1}^{n} |s_i| = \mathbf{xc} = x$.

Proof. Initially, let $F = \{i | x_i > 0\}$ and $E = \{i | x_i < 0\}$. The post condition of the third loop (in line 8) is that for all $i \in F$ and $j \in E$, there are x_i fill operations on empty jug i and x_j empty operations on full jug j in σ. Note that after each fill operation some v_i with $i \in F$ will decrease by 1, and after each empty operation some v_j with $j \in E$ will increase by 1. We will show that the quantity of water in the jugs is x when the algorithm terminates. We also need to show that the loop invariants hold to ensure the progress of the algorithm.

The post condition of the first loop (in line 2) is trivial that for all $i \in F$, $s_i = c_i$. This makes the loop invariant of the second loop hold initially.

Next we will show that after an iteration of the second loop (in lines 3-7), if there still exists an $v_i < 0$, then we can always find j in line 4. There are two possibilities after $pour(j, i)$ is executed in line 5, i.e., jug j can become non-empty or empty. *Case 1:* (Jug j is still non-empty.) The loop invariant holds, since $s_j > 0$ and $v_j \geq 0$. *Case 2:*(Jug j becomes empty.) If $v_j > 0$, then the loop invariant trivially holds, since jug j will be refilled immediately. If $v_j = 0$, assume that the loop invariant did not hold, i.e., line 4 failed to find the j in the following iteration and it implied that for all $k \in F$ with $v_k \geq 0$, we had $s_k = 0$. While with line 6, we know that for each $k \in F$, v_k cannot be

Algorithm Measure($\mathbf{c}$, x, $\mathbf{x}$)

Input: $\mathbf{c} = (c_1, \cdots, c_n)$, the capacity of jugs.
x, the quantity to be measured.
$\mathbf{x} = (x_1, \cdots, x_n)$, the optimal representation of x that achieves $\mu_{\mathbf{c}}(x)$.

Output: the operation sequence σ, such that $\delta^*(\mathbf{0}, \sigma)$ achieves the quantity x.

Variable: $\mathbf{s}$, the state of jugs, which is initialized to be zero state.
$\mathbf{v} = (v_1, \cdots, v_n)$, initialized to be $\mathbf{x}$.

begin

1. $\sigma := \epsilon$;
2. **for all** i **if** $(s_i = 0$ and $v_i > 0)$ **do** *fill*(i);
3. **while**$(\exists i$ s.t. $v_i < 0)$ **do**
4. Find j s.t. $s_j > 0$ and $v_j \geq 0$;
5. *pour*(j, i);
6. **if** $(s_j = 0$ and $v_j > 0)$ **then** *fill*(j);
7. **if** $(s_i = c_i)$ **then** *empty*(i);
8. **while** $(\exists v_i > 0)$ **do**
9. Find $j > n - l$ with $v_j = 0$ and $s_j \neq c_j$;
10. *pour*(i,j);
11. **if** $s_i = 0$ **then** *fill*(i);

end

Procedure fill(i)

begin $\sigma := \sigma \circ (\downarrow i)$; $s_i := c_i$; $v_i := v_i - 1$; **end**

Procedure empty(i)

begin $\sigma := \sigma \circ (i \uparrow)$; $s_i := 0$; $v_i := v_i + 1$; **end**

Procedure pour(i, j)

begin

1. $\sigma := \sigma \circ (i \rightarrow j)$;
2. **if** $(s_i + s_j > c_j)$ **then** $\{s_i := s_i + s_j - c_j;\ s_j := c_j;\}$
3. **else** $\{s_j := s_i + s_j;\ s_i := 0;\}$

end

Fig. 2.2: Measuring algorithm given $\mu_{\mathbf{c}}(x)$.

greater than 0 (otherwise it would be refilled right the way), and thus all v_k must become 0. Line 7 shows that for all $i \in E$, if $v_i < 0$ then $s_i < c_i$ and thus the amount of water in the jugs is less than $\sum_{i\in E; v_i<0} c_i$. Since we have done $\sum_{i\in F} x_i$ fill operations and $\sum_{i\in E}(v_i - x_i)$ empty operations, the quantity of water left in the jugs is exactly $\sum_{i\in F} c_i x_i + \sum_{i\in E} c_i(x_i - v_i)$=$\mathbf{x} \cdot \mathbf{c} - \sum_{i\in E} c_i v_i$, which is greater than $\sum_{i\in E; v_i<0} c_i$– a contradiction! Thus we have that as long as the loop condition in line 3 holds, line 4 can always find a jug to perform the pour operation.

The post condition of the second loop (lines 3-7) is clear that for all $i \in E$, $v_i = 0$, $s_i = 0$ and for all $i \in F$ with $v_i > 0$, we have $s_i > 0$ by line 6. Note that the quantity of water left in the jugs is $\sum_{i\in F}(x_i - v_i)c_i - \sum_{j\in E}(v_j - x_j)c_j$=$x - \sum_{i\in F} c_i v_i > 0$. If for all $i \in F$, $v_i = 0$, then we are done.

We prove the invariant of the third loop (lines 8-11) as following. The largest l jugs are sufficient to contain the quantity x by the assumption in the theorem. Note that if $\sum_{j>n-l} c_j = x$, then the optimal measuring will be by filling the jugs with indices greater than $n-l$. So without loss of generality, we assume $\sum_{j>n-l} c_j > x$. It is clear that we cannot have for all $i > n-l$, $v_i > 0$ (actually $= 1$), otherwise $x - \sum_{i\in F} c_i v_i < 0$. Thus there exists some $i > n-l$, $v_i = 0$ and from which some s_i must be less than c_i. Suppose for all $j > n-l$, either $v_j > 0$ or ($v_j = 0$ and $s_j = c_j$). Then $\sum_{j>n-l; v_j>0} c_j v_j + \sum_{j>n-l; v_j=0} c_j$ $= \sum_{j>n-l} c_j > x$. But $\sum_{j>n-l; v_j>0} c_j v_j < \sum_{i\in F} c_i v_i$ and $\sum_{j>n-l; v_j=0} c_j <$ $(x - \sum_{i\in F} c_i v_i)$, which is the current total quantity in the jugs. The sum of the latter two inequalities leads to a contradiction. Thus, whenever there exists $x_i > 0$, line 9 can always find a suitable jug for pouring.

Finally, when the algorithm terminates, it actually performed $|F|$ fill operations and $|E|$ empty operations and the net quantity is $\sum_{i\in F} c_i x_i + \sum_{j\in E} c_j x_j$=$x$.$\square$

Lemma 12: For all $x \in M^+(\mathbf{c})$, there exists $\mathbf{x}$ such that $\mathbf{x} \cdot \mathbf{c} = x$ and $\mu_{\mathbf{c}}(x) = \sum_{i=1}^{n} |x_i|$, and the algorithm MEASURE outputs a sequence σ of operations such that $|\sigma| \leq 2\mu_{\mathbf{c}}(x) + l - 1$.

Proof. If $x = c_{n-l+1} + \cdots + c_n$, then it is clear that $\mathbf{x} = (0, \ldots, 0, \overbrace{1, \ldots, 1}^{l})$ satisfies all requirements in this lemma, moreover, $\mu_{\mathbf{c}}(x) = l$. Since after the first loop, all $v_i = 0$, we know that $|\sigma| = l \leq 2\mu_{\mathbf{c}}(x)+l-1$. Thus without loss of generality, we assume that $x < c_{n-l+1} + \cdots + c_n$. The fact that $x \in M^+(\mathbf{c})$ ensures the existence of such $\mathbf{x}$.

It is clear that after performing MEASURE($\mathbf{c}$, x, $\mathbf{x}$), there are $\mu_{\mathbf{c}}(x)$ fill and empty operations executed. For the rest, we need to estimate the number of pour operations executed. Now we try to associate each pour operation with a fill or empty operations as following.

For any pour operation $pour(i, j)$: *Case 1: (Jug i becomes empty.) Then we associate this $pour(i, j)$ with the closest prior $fill(i)$ operation. Case 2: (Jug j becomes full.) If there is no $empty(j)$ operation after it, then we associate it with jug j, else associate it with the next $empty(j)$ operation after it.*

Let $F = \{i | x_i > 0\}$ and $E = \{i | x_i < 0\}$. Note that the algorithm MEASURE starts by filling all jugs with indices in F. Every fill operation associates with at most one pour operation, since by following the algorithm for any $i \in F$ after a pour operation that empties jug i, there is either an immediate fill(i) operation or no more pour operation about jug i. Every empty operation also associates with at most one pour operation, since for any $i \in E$, the only possible operation between a pour operation that fills jug i fully and the next empty(i) is a fill operation, not a pour or an empty operation. These two facts imply there are at most $\mu(x)$ pour operations associated with fill and empty operations.

The pour operations associated with jugs are always executed in the third loop (lines 8-11), and there are at most $l-1$ pour operations associated with jugs since when the algorithm terminates, there are at most $l-1$ fully filled jugs among jugs from jug $n-l+1$ to jug n. Since each pour operation is associated with a fill operation, an empty operation or a jug, we conclude that the number of pour operations is less than $\mu_{\mathbf{c}}(x)+l-1$ and thus $|\sigma| \leq 2\mu_{\mathbf{c}}(x)+l-1$. $\square$

From lemma 12, it clear that when $l = 1$, i.e., the water is eventually poured into a single jug, the bound is very close to the lower bound. While there still exists a gap when considering the quantities that exceed the largest capacity.

3. THE COMPLEXITY OF JUG MEASURING PROBLEM

3.1 NP-hardness of computing $\mu(x)$

We have used $\mu(x)$ to bound the number of steps on jug measuring. In this chapter, we investigate the difficulty of computing $\mu(x)$. Given $\mathbf{c} = (c_1, \cdots, c_n) \in \mathbf{N}^n$ and $x \in M(\mathbf{c})$, we define $\mu_{\mathbf{c}}(x) = \min_{\mathbf{x} \in \mathbf{Z}^n, \mathbf{x} \cdot \mathbf{c} = x} \sum_{i=1}^{n} |x_i|$, where $\mathbf{x} = (x_1, \cdots, x_n)$. To study the complexity of computing $\mu_{\mathbf{c}}(x)$, we investigate a bounded version of it. Define $L = \{< \mathbf{c}, x, u > | \mu_{\mathbf{c}}(x) \leq u\}$. Given $\mathbf{c}, x$ and u, we want to determine if $\mu_{\mathbf{c}}(x) \leq u$, in other words, we want to determine if $< \mathbf{c}, x, u > \in L$. We prove it indeed NP-complete by reducing the 3-Dimensional Matching to it. 3-Dimensional Matching problem [8, 9, 4] is a well known NP-complete problem, which is defined as: Given three sets $P = Q = R = [n]$, and a subset $T \subseteq P \times Q \times R$, is there a subset S of T with $|S| = n$ such that whenever (p, q, r) and (p', q', r') are distinct triples in S, $p \neq p'$, $q \neq q'$ and $r \neq r'$? We call such an S a match for the 3-Dimensional Matching problem.

Theorem 13: The membership problem of L is NP-complete.

Proof. For any instance of the form $< \mathbf{c}, x, u >$, we can nondeterministically guess an $\mathbf{x}$ and verify if $\mathbf{x} \cdot \mathbf{c} = x$ and $\sum_{i=1}^{n} |x_i| \leq u$. Since all the verification can be done in polynomial time in terms of the input size, we know L is in

NP.

Next, we give a polynomial time reduction from 3-Dimensional Matching to L. For convenience let $|T| = t$. To have a match, we need $t \geq n$. Given an instance of 3-Dimensional Matching as above, for each $(p_i, q_i, r_i) \in T$, we construct a positive number $c_i = 2^{3t+3n} + 2^{2t+2n+p_i-1} + 2^{t+n+q_i-1} + 2^{r_i-1}$, which has $3t + 3n + 1$ bits in binary representation. Construct $x = n \times 2^{3t+3n} + (2^n - 1)(2^{2t+2n} + 2^{t+n} + 1)$ and $u = n$. If the instance has a match S, then we set $x_i = 1$ when $(p_i, q_i, r_i) \in S$; 0 otherwise. Since $|S| = n$, it is not hard to see $\sum_{i=1}^{t} x_i c_i = x$ and $\sum_{i=1}^{t} |x_i| = n \leq u$.

On the other hand, suppose we have x_i's such that $\sum_{i=1}^{t} x_i c_i = x$ and $\sum_{i=1}^{t} |x_i| \leq u$. Then it is clear that there are at most n non-zero x_i's and $\sum_{i=1}^{t} x_i = n$, since because of the construction of c_i's and x, even we add up the largest c_i for n times the terms with lower power of 2 won't carry over to 2^{3t+3n}. We claim that there are exactly n non-zero x_i's and each of them is 1. We can check the sum $\sum_{i=1}^{t} x_i 2^{r_i-1}$, which can be simplified to $2^n - 1$, if all the nonzero x_i is 1 and the corresponding r_i's range over every number from 1 to n. If some x_i were greater than 1, then $\sum_{i=1}^{t} x_i 2^{r_i-1}$ could be merged into the sum of less than n numbers and each of them would have a distinct power of 2. But this cannot add up to $2^n - 1$. Similarly, we can argue for the other terms. Hence each non-zero x_i is 1 and $\sum_{i=1}^{t} x_i c_i = x$ implies a match $S = \{(p_i, q_i, r_i) : \text{for all } x_i = 1\}$. It is clear the reduction can be done in polynomial time. Therefore, L is NP-complete. $\square$

The above result implies that computing $\mu_{\mathbf{c}}(x)$ is NP-hard. Moreover, we prove it is in P^{NP} with the following fact by Boldi et al. [3].

Fact 1: [3] Let $x \in M(\mathbf{c})$, then $\mu_{\mathbf{c}}(x) < \max\{2c_n, c_n + x\}/\gcd(\mathbf{c})$.

We use L as an oracle and the above bound to compute $\mu_{\mathbf{c}}(x)$ and prove the following result.

Theorem 14: The problem of computing $\mu_{\mathbf{c}}(x)$ is in P^{NP}.

Proof. Let μ be the upper bound from fact 1, which is bounded by a polynomial in terms of the input size. We use L as an oracle. Then we can apply binary search to find the minimum value of $\mu_{\mathbf{c}}(x)$ by repeatedly querying L. The algorithm is described as below:

1. Let $\ell = 0$, $u = \mu$. If x is not a multiple of $\gcd(c_1, \cdots, c_n)$, then output "No solution". Query if $< \mathbf{c}, x, \ell > \in L$. If YES, then output 0 and EXIT.

2. If $u = \ell + 1$, then output u and EXIT.

3. Let m $= \lceil \frac{u+\ell}{2} \rceil$. Query if $< \mathbf{c}, x, m > \in L$. If YES, then let $u = m$, else let $\ell = m$. Go to 2.

This algorithm is a typical binary search with an oracle L and runs in $O(\log(\mu))$ time, which a polynomial with respect to the input size. Since $L \in NP$, we know that computing $\mu_{\mathbf{c}}(x)$ belongs to the class P^{NP}. $\square$

Corollary 15: The optimal jug measuring problem is NP-hard.

Proof. Consider an instance $\mathbf{c}$ and $x \in M(\mathbf{c})$. Suppose K is the minimum number of measuring steps for x. From theorem 4 and 10, we have $\mu_{\mathbf{c}}(x) = \lceil K/2 \rceil$. Thus it is at least as hard as computing $\mu_{\mathbf{c}}(x)$. $\square$

3.2 A pseudo-polynomial time algorithm for computing $\mu(x)$

For $x = c_i$ or 0, $\mu_{\mathbf{c}}(x)$ is obviously 1 and 0, respectively.

Lemma 16: For any $x \in M^+(\mathbf{c})$, if $\mu_{\mathbf{c}}(x) > 1$, then there exists $y \in M^+(\mathbf{c})$ with $|x - y| = c_i$ for some $i \in [n]$, such that $\mu_{\mathbf{c}}(y) + 1 = \mu_{\mathbf{c}}(x)$.

Proof. Let $\mathbf{c} \cdot \mathbf{x} = x$, $E = \{i | x_i < 0\}$, $F = \{i | x_i > 0\}$, and $Y = \{(x + c_i) \in M^+(\mathbf{c}) | i \in E\} \cup \{(x - c_j) \in M^+(\mathbf{c}) | j \in F\}$. By the assumption that $\mu_{\mathbf{c}}(x) > 1$, it is clear $Y \neq \emptyset$. We claim that $\exists y \in Y$, $\mu_{\mathbf{c}}(x) = \mu_{\mathbf{c}}(y) + 1$. It is clear to see that if for all $y \in Y$, $\mu_{\mathbf{c}}(y) < \mu_{\mathbf{c}}(x) - 1$ then we can find an even better representation for x. Hence $\mu_{\mathbf{c}}(y) \geq \mu_{\mathbf{c}}(x) - 1$. But $|x - y| = c_i$ for some i, thus the claim must hold and $\mu_{\mathbf{c}}(x) = 1 + \min_{y \in Y} \mu_{\mathbf{c}}(y)$. □

From the proof, we derive a pseudo-polynomial time algorithm to compute $\mu_{\mathbf{c}}(x)$ for $x \in M^+(\mathbf{c})$ and the optimal representation of x. The algorithm is shown in figure 3.1.

Theorem 17: The Search algorithm outputs $\mu_{\mathbf{c}}(x)$ and the optimal $\mathbf{x}$ in $O(n \cdot |M^+(\mathbf{c})|)$ time.

Proof. First, we consider that the search is done on a graph starting from 0 until reaching x, where the vertex set is $M^+(\mathbf{c})$ and for $x, y \in M^+(\mathbf{c})$, (x, y) is an edge iff $|x - y| = c_i$ for some c_i. Thus there are at most $O(n \cdot |M^+(\mathbf{c})|)$ edges. It is clear that what Search does is a typical Breadth-First-Search(BFS) and thus the time complexity is $O(n \cdot |M^+(\mathbf{c})|)$. Since the input size is $\sum_i^n \log |c_i| + \log |x|$ and $|M^+(\mathbf{c})| = \lfloor \frac{\sum_{i=1}^n c_i}{\gcd(\mathbf{c})} \rfloor$ can be exponential in terms of the input size. Therefore Search$(\mathbf{c}, x)$ is a pseudo-polynomial time algorithm. □

Algorithm $\textsc{Search}(\mathbf{c}, x)$

Inputs: $\mathbf{c}$, the capacities of jugs; $x \in M^+(\mathbf{c})$.

Outputs: μ=$\mu_{\mathbf{c}}(x)$ and $\mathbf{x} = (x_1, \cdots, x_n)$, where $||\mathbf{x}||_1 = \mu_{\mathbf{c}}(x)$.

Variables: $m[0..|M^+(\mathbf{c})|]$, temporal storages for $\mu_{\mathbf{c}}$'s and each is initialized to be ∞.

$prev[0..|M^+(\mathbf{c})|]$, $prev[y]$ stores the value visited right before y.

begin

$y := 0$; $m[0] := 0$;

while(x is not reached) **do**

    for $i = 1$ to n **do**

        if ($y + c_i \in M^+(\mathbf{c})$ and $m[y + c_i] > m[y] + 1$) **then**

            $m[y + c_i] := m[y] + 1$; $prev[y + c_i] := y$; $\textsc{Enqueue}(y + c_i)$;

        if ($y - c_i \in M^+(\mathbf{c})$ and $m[y - c_i] > m[y] + 1$) **then**

            $m[y - c_i] := m[y] + 1$; $prev[y - c_i] := y$; $\textsc{Enqueue}(y - c_i)$;

    $y :=\textsc{Dequeue}()$;

$\mu := m[x]$; $y := x$; $\mathbf{x} = (0, \cdots, 0)$;

while($y \neq 0$) **do**

    $d := y - prev[y]$;

    Find the index i with $c_i = |d|$;

    if ($d > 0$) **then** $x_i = x_i + 1$ **else** $x_i = x_i - 1$;

    $y := prev[y]$;

end

Fig. 3.1: A pseudo-polynomial time algorithm for computing $\mu(x)$ and $\mathbf{x}$.

4. APPROXIMATING THE JUG MEASURING PROBLEM

4.1 Convert computing $\mu_{\mathbf{c}}(x)$ to CVP

We have shown that computing $\mu_{\mathbf{c}}(x)$ is indeed NP-hard. In this section we propose a polynomial reduction from the problem of computing $\mu_{\mathbf{c}}(x)$ to *CVP*, and an *LLL*-based approximation algorithm for computing $\mu_{\mathbf{c}}(x)$. Our approximation algorithm is based on the fact: computing $\mu_{\mathbf{c}}(x)$ can be polynomially reduced to the *closest lattice vector* problem (*CVP*). First, we introduce lattice and the closest lattice problem briefly as follows:

Definition 1: A lattice in $\mathbf{R}^n$ is the set all integer linear combination m independent vectors $b_1, b_2, \cdots, b_m$. The lattice generated by $b_1, b_2, \cdots, b_m$ denoted $L(b_1, b_2, \cdots, b_m)$ is the set $\{\sum_{i=1}^{m} \lambda_i b_i | \forall i \in [m], \lambda_i \in \mathbf{Z}\}$. The independent vectors $b_1, b_2, \cdots, b_m$ are called a basis of the lattice.

Definition 2: The closest lattice vector problem is: suppose we are given a basis $b_1, b_2, \cdots, b_m$, a vector $v \in \mathbf{R}^n$ and an integer p, to find the lattice point $u \in L(b_1, b_2, \cdots, b_m)$ which is closest to v under l_p-*norm*.

In order to complete the reduction from computing $\mu_{\mathbf{c}}(x)$ to *CVP*, we introduce the *Hermite normal form*:

Definition 3: a matrix A is said to be in Hermite normal form if it has the form $[B \;\; 0]$ where the matrix B is a nonsingular, lower triangle, nonnegative matrix, in which each row has a unique maximum entry, which is located on the main diagonal of B.

We will give an example after we introduce the *unimodular* matrices, the key tool for computing the Hermite normal form. The following operations on a matrix are called *elementary (unimodular)* column operations:

1. exchange two columns;

2. multiply a column by -1;

3. adding an integral multiple of one column to another column

Thus a nonsingular matrix U is called unimodular matrix if U is integral and has determinant 1 or -1.

Theorem 18: [11] Each rational matrix of full row rank can be brought into Hermite normal form by a series of elementary column operations.

Corollary 19: [11] For each rational matrix A of full row rank, there is a unimodular matrix U such that AU is the Hermite normal form of A.

For example:

$$A = \begin{bmatrix} 1 & 5 & 4 & 7 \\ 0 & 3 & 6 & 3 \\ 0 & 0 & 5 & 7 \end{bmatrix} \text{ and its Hermite normal form } AU = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

and the unimodular matrix:

$$U = \begin{bmatrix} 1 & -5 & 22 & -52 \\ 0 & 1 & -4 & 9 \\ 0 & 0 & 3 & -7 \\ 0 & 0 & -2 & 5 \end{bmatrix}$$

Lemma 20: [11] Given a feasible system $Ax = b$ of rational linear diophantine equations, we can find in polynomial time integral vectors $x_0, x_1, x_2, \cdots, x_t$ such that $\{x|Ax = b; x \text{ is integral}\} = \{x_0 + \lambda_1 x_1 + \cdots + \lambda_t x_t | \lambda_1, \cdots, \lambda_t \in \mathbf{Z}\}$ with $x_1, x_2, \cdots, x_t$ linearly independent. Moreover, $[x_0\ x_1\ x_2\ \cdots\ x_t] = U \begin{bmatrix} B^{-1}b & 0 \\ 0 & I \end{bmatrix}$, where $AU = [B\ \ 0]$ is the Hermite normal form of A.

By the following theorem and lemma, we can find a reduction from computing $\mu_{\mathbf{c}}(x)$ to *CVP*. And we can complete the reduction:

Corollary 21: The problem of computing $\mu_{\mathbf{c}}(x)$ can be polynomially reduced to *CVP*.

Proof. Assume $< (c_1, c_2, \cdots, c_n), x >$ is an instance of the problem of $\mu_{\mathbf{c}}(x)$. Let the matrix $C = [c_1\ c_2\ \cdots\ c_n]$. By corollary 19, there exists unimodular matrix U such that $CU = [B\ 0]$ is the Hermite normal form of C. Let $[v_0\ v_1\ v_2\ \cdots\ v_{n-1}] = U \begin{bmatrix} B^{-1}x & 0 \\ 0 & I \end{bmatrix}$. By lemma 20, we know $v_1, v_2 \cdots, v_{n-1}$ are linearly independent vectors and form a basis of a lattice L. Hence $\mu_{\mathbf{c}}(x) = \min_{\mathbf{v} \in \{\mathbf{x}|\mathbf{c}\cdot\mathbf{x}=x\}} ||\mathbf{v}||_1 = \min_{\lambda_1, \cdots, \lambda_{n-1} \in \mathbf{Z}} ||v_0 + \lambda_1 v_1 + \cdots + \lambda_{n-1} v_{n-1}||_1 = \min_{\mathbf{w} \in L} ||\mathbf{w} - (-v_0)||_1$. Thus $\mu_{\mathbf{c}}(x)$ is the l_1-*norm* of the vector $\mathbf{v} \in L$ which is closest to $-v_0$. It is clear that all computation can be done in polynomial time. □

After we introduce the closest lattice vector problem and the Hermite normal form, we can sketch the approximation algorithm for computing $\mu_{\mathbf{c}}(x)$:

Step 1. Transform the instance $< (c_1, c_2, \cdots, c_n), x >$ of the problem of computing $\mu_{\mathbf{c}}(x)$ into an instance of *CVP* by computing the unimodular matrix U such that $[c_1, c_2, \cdots, c_n]U$ is in the Hermite normal form.

Step 2. Approximate the closest vector by *LLL*-based algorithm. $\mu_{\mathbf{c}}(x)$ is equivalent to the ℓ_1-norm of the difference of the target vector v and the lattice vector closest v.

The complexity of the approximation algorithm depends on the implementation of these two steps. As we will mention later, the complexity of the *LLL* basis reduction algorithm is dependent to the number of basis vectors and the maximum ℓ_2-norm of the basis vectors, thus for the computing the unimodular matrix U such that $[c_1, c_2, \cdots, c_n]U$ is in Hermite normal form, an algorithm giving U with smaller entries usually reduces the running time of whole approximation algorithm.

For the problem of computing the unimodular matrix U such that CU is in the Hermite normal form, where $C = [c_1 \ c_2 \ \cdots \ c_n]$, we propose an algorithm which is simpler than the algorithm in [11]. But it works only on the special case $[c_1 \ c_2 \ \cdots \ c_n]U$, it can't compute the unimodular matrix U for any other n by m matrix, where $n > 1$. The algorithm is shown in figure 4.1, and it is based on the Euclidean algorithm. It compute the greatest common divisor of the first and the ith entries by applying Euclidean algorithm with unimodular operations. Each iteration terminates when the greatest common divisor is written back to the first entry and 0 is written to the ith entry, thus when the algorithm terminates, $gcd(c_1, c_2, \cdots, c_n)$ will be written to the first entry and 0 will be written to all the others. This implementation runs in $O(n^2 \log c_n)$ since each iteration of the Euclidean algorithm takes $O(n)$ time for there are n entries in a column of U and the Euclidean algorithm runs in $O(\log c_i)$ iterations for every i. And each time we execute line 4 in figure 4.1, the bit-length of entries of u_i increases at most $O(\log q)$ and $q = \lfloor \frac{c_i}{c_1} \rfloor$; this fact imply after the ith iteration of the for-loop, the maximum bit-length of the absolute value of the entries of U increases at most $O(\log c_i)$. Thus the maximum bit-length of the absolute value of the entries of U won't exceed $O(n \log c_n)$.

4.2 Approximating CVP

Before we introduce the approximation algorithm for *CVP*, we introduce the famous *LLL* basis reduction algorithm. This algorithm is based on Gram-Schmidt orthogonalization. Let $b_1, b_2, \cdots, b_n$ be a basis of lattice L and $b_1^*, b_2^*, \cdots, b_n^*$ be the orthogonal basis of $span(b_1, b_2, \cdots, b_n)$ obtained by the Gram-Schmidt algorithm:

$$b_1^* = b_1,$$
$$b_i^* = b_i - \sum_{j=1}^{i-1} \frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle} b_j^* \quad (i = 2, \cdots, n)$$

We can write the recurrence in the following form:

$$b_i = \sum_{j=1}^{i} \mu_{i,j} b_j^* \quad \text{where } \mu_{i,j} = \begin{cases} 1 & , i = j \\ \frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle} & , i < j \end{cases} \quad (i = 1, \cdots, n)$$

We say a basis $b_1, b_2, \cdots, b_n$ is *weakly reduced* if its Gram-Schmidt orthogonal basis $b_1^*, b_2^*, \cdots, b_n^*$ satisfies the property:

$$\text{If } b_i = \sum_{j=1}^{i} \mu_{i,j} b_j^* \text{ then } |\mu_{i,j}| \leq \frac{1}{2} \text{ for } 1 \leq j < i \leq n$$

Moreover, we say a basis $b_1, b_2, \cdots, b_n$ is *reduced* or *LLL reduced* if it is *weakly reduced* and satisfies the following inequality:

$$||b_i^*||_2^2 \leq \frac{4}{3} ||b_{i+1}^* + \mu_{i+1,i} b_i^*||_2^2$$

Let $\lambda(L)$ be the length of the shortest nonzero vector in the lattice L. Let $d(L)$ be the volume of the *fundamental parallelepiped* $P = \{\sum_{i=1}^{n} r_i b_i | \forall r_i \in [0, 1)\}$. A. K. Lenstra, H. W. Lenstra and L. Lovasz proved the following two theorems in 1982:

Algorithm Hermite Normalize(C)

Inputs: $C = [c_1 c_2 \cdots c_n]$, the capacities of jugs.

Outputs: $U = [u_1 u_2 \cdots u_n]$ such that $[c_1 c_2 \cdots c_n]U$ is in the Hermite normal form.

Variables: q, temporal storages $\lfloor \frac{c_i}{c_1} \rfloor$

begin

1. $U := I$;

2. **for** $i = 2$ to n **do**

3.   **while** $(true)$ **do**

4. $q := \lfloor \frac{c_i}{c_1} \rfloor$; $c_i := c_i - qc_1$; $u_i := u_i - qu_1$;

5.   **if** $(c_i = 0)$ **then** break;

6.   Swap(c_i, c_1); Swap(u_i, u_1);

7.   **loop**

8.**next** i

end

Fig. 4.1: A simple algorithm to compute U.

Lemma 22: [7] Let $b_1, b_2, \cdots, b_n$ be a basis of a lattice L and let $b_1^*, b_2^*, \cdots, b_n^*$ be its Gram-Schmidt orthogonalization. Then $\lambda(L) \geq \min_i ||b_i^*||_2$.

Proof. Let $b \in L$ and $b \neq 0$. Then we can write $b = \sum_{i=1}^{n} \lambda_i b_i$ for λ_i are integers. Since $b \neq 0$, there exists $k = \max_{\lambda_i \neq 0} i$. So we can rewrite $b = \sum_{i=1}^{k} \lambda_i' b_i^*$. By the property of Gram-Schmidt orthogonalization, $\lambda_k' = \lambda_k \neq 0$ is a non-zero integer. Hence $||b||_2^2 = \sum_{i=1}^{k} |\lambda_i'|^2 ||b_i^*||_2^2 \geq |\lambda_k'|^2 ||b_k^*||_2^2 \geq |b_k^*||_2^2 \geq \min_i |b_i^*||_2^2$. This proves the lemma. □

Theorem 23: [7] Let $b_1, b_2, \cdots, b_n$ be a reduced basis of the lattice L. Then:

(1) $||b_1||_2 \leq 2^{(n-1)/2} \lambda(L)$;

(2) $||b_1||_2 \leq 2^{(n-1)/4} \sqrt[n]{d(L)}$;

(3) $\prod_{i=1}^{n} ||b_i||_2 \leq 2^{\frac{1}{2}\binom{n}{2}} d(L)$;

Proof. Since $b_1, b_2, \cdots, b_n$ is reduced, $||b_i^*||_2^2 \leq \frac{4}{3} ||b_{i+1}^* + \mu_{i+1,i} b_i^*||_2^2$ and $|\mu_{i+1,i}| \leq \frac{1}{2}$. It implies $||b_i^*||_2^2 \leq \frac{4}{3} ||b_{i+1}^*||_2^2 + \frac{1}{3} ||b_i^*||_2^2$, thus $||b_i^*||_2^2 \leq 2 ||b_{i+1}^*||_2^2$. By induction, we have $||b_1^*||_2^2 \leq 2^{i-1} ||b_i^*||_2^2$. By lemma 22 and $||b_1||_2 = ||b_1^*||_2$, $||b_1||_2^2 \leq \min_i 2^{i-1} ||b_i^*||_2^2 \leq 2^{n-1} \min_i ||b_i^*||_2^2 \leq 2^{n-1} \lambda(L)$, this proves property (1). $||b_1||_2^{2n} \leq \prod_{i=1}^{n} 2^{i-1} ||b_i^*||_2^2 = 2^{n(n-1)/2} d(L)$, thus we have property (2). And $||b_i||_2^2 = \sum_{j=1}^{i} \mu_{i,j}^2 ||b_j^*||_2^2 \leq \sum_{j=1}^{i-1} \frac{1}{4} ||b_j^*||_2^2 + ||b_i^*||_2^2 \leq (1 + (2 + 2^2 + \cdots + 2^{i-1})/4) ||b_i^*||_2^2 \leq 2^{i-1} ||b_i^*||_2^2$, hence $\prod_{i=1}^{n} ||b_i||_2^2 \leq 2^{\binom{n}{2}} \prod_{i=1}^{n} ||b_i^*||_2^2 = 2^{\binom{n}{2}} d(L)$, proving property (3). □

In general, the *LLL* algorithm can be decribed as follow:

- Step 1. Make the basis weakly reduced.

- Step 2. Check if the basis is reduced. If the basis is reduced, it is done.
- Step 3. Exchange b_i and b_{i+1} with $||b_i^*||_2^2 > \frac{4}{3}||b_{i+1}^* + \mu_{i+1,i}b_i^*||_2^2$, then go to Step 1.

Theorem 24: [7] Given a rational basis $b_1, b_2, \cdots, b_n$ of the lattice L, a reduced basis $b_1', b_2', \cdots, b_n'$ in L can be found in polynomial time.

Proof. W.L.O.G., we can convert rational basis into integral basis. For any lattice L formed by integral basis, $d(L) \geq 1$. We define $D(b_1, \cdots, b_n) = \prod_{i=1}^{n} ||b_i^*||_2^{n-i} = \prod_{i=1}^{n-1} d(L(b_1, \cdots, b_i)) \geq 1$. It is clear that only Step 3 will change $D(b_1, \cdots, b_n)$. Each execution of Step 3 reduces $D(b_1, \cdots, b_n)$ by a factor of $\frac{2}{\sqrt{3}}$ or more, since exchanging b_i and b_{i+1} will also exchange b_i^* and $b_{i+1}^* + \mu_{i+1,i}b_i^*$. And initially, $D(b_1, \cdots, b_n) \leq \prod_{i=1}^{n} ||b_i||_2^{n-i}$, which is a polynomial size of input. We can conclude that the algorithm will terminate within polynomial rounds. And Step 1 and Step 2 runs in polynomial time, thus we prove the theorem. □

Theorem 23 describes the properties of the LLL reduced bases, and Theorem 24 implies the LLL basis reduction algorithm runs in polynomial time.

But there are some drawbacks of LLL, and one of them is computing the Gram-Schmidt orthogalization would involving division and the precision of floating point numbers isn't enough. For our application to compute the $\mu_{\mathbf{c}}(x)$, we are dealing with integers and the precision is one of our demands, so we choose De Weger's implementation of the LLL algorithm in [12], p73-75. This implementation runs in $O(n^4 \log a)$, where a is the maximum of the ℓ_2-norm of the basis vectors. Substitute a with $\sqrt{n} \cdot 2^{O(n \log c_n)}$, and we know the time consumed by this implementation is $O(n^5 \log c_n)$.

L. Babai [1] provided two polynomial-time approximation algorithms for *CVP*. Both algorithms are based on *LLL* basis reduction algorithm. Assume we are given an LLL reduced basis $\{b_1, b_2, \cdots, b_n\}$, a vector $\mathbf{x} = \sum_{i=1}^{n} \alpha_i b_i$ and we are to find a vector $\mathbf{w} \in L(b_1, b_2, \cdots, b_n)$ close to $\mathbf{x}$. The first algorithm is called rounding off heuristic algorithm. It just output $\mathbf{w} = \sum_{i=1}^{n} \beta_i b_i$ where β_i is the closest integer to α_i. The second algorithm is called nearest plane heuristic algorithm. It is a recusive algorithm. Let $U = span(b_1, b_2, \cdots, b_{n-1})$, and find $\mathbf{v} \in L(b_1, b_2, \cdots, b_n)$ such that the distance between $U + \mathbf{v}$ and $\mathbf{x}$ is minimal. Let $\mathbf{x}'$ be the orthogonal projection of $\mathbf{x}$ on $U + \mathbf{v}$. Then find $\mathbf{y} \in L(b_1, b_2, \cdots, b_{n-1})$ close to $\mathbf{x}' - \mathbf{v}$, and output $\mathbf{w} = \mathbf{y} + \mathbf{v}$. Both algorithms guarantee $\mathbf{w}$ is close to $\mathbf{x}$.

Theorem 25: [1] The rounding off heuristic algorithm find a vector $\mathbf{w}$ such that $||\mathbf{x} - \mathbf{w}||_2 \leq (1 + 2n(\frac{9}{2})^{\frac{n}{2}}) \min_{\mathbf{v} \in L(b_1, b_2, \cdots, b_n)} ||\mathbf{x} - \mathbf{v}||_2$.

Proof. see Babai [1].

Theorem 26: [1] The nearest plane algorithm heuristic algorithm find a vector $\mathbf{w}$ such that $||\mathbf{x} - \mathbf{w}||_2 \leq 2^{\frac{n}{2}} \min_{\mathbf{v} \in L(b_1, b_2, \cdots, b_n)} ||\mathbf{x} - \mathbf{v}||_2$. Moreover, $||\mathbf{x} - \mathbf{w}||_2 < 2^{\frac{n}{2} - 1} ||b_n^*||_2$.

Proof. Babai [1] showed this by induction on n. Let $\mathbf{u}$ be the closest lattice point to $\mathbf{x}$. For $n = 1$, we find the nearest lattice point. For $n > 1$, since $\mathbf{x}'$ is the orthogonal projection of $\mathbf{x}$, $||\mathbf{x} - \mathbf{x}'||_2 \leq ||b_n^*||_2/2$ and $||\mathbf{x} - \mathbf{x}'||_2 \leq ||\mathbf{x} - \mathbf{u}||_2$. From $||\mathbf{x} - \mathbf{x}'||_2 \leq ||b_n^*||_2/2$, we obtain by induction that $||\mathbf{x} - \mathbf{w}||_2^2 \leq (||b_1^*||_2^2 + \cdots + ||b_n^*||_2^2)/4$. And by the property of LLL-reduced basis, we have $||\mathbf{x} - \mathbf{w}||_2^2 \leq (2^{n-1} + 2^{n-2} + \cdots + 1)||b_n^*||_2^2/4 = (2^n - 1)||b_n^*||_2^2/4 < 2^{n-2}||b_n^*||_2^2$, hence $||\mathbf{x} - \mathbf{w}||_2 \leq 2^{\frac{n}{2} - 1}||b_n^*||_2$.

We have to consider two cases:

- $\mathbf{u} \in U + \mathbf{v}$: Clearly, $\mathbf{u} - \mathbf{v}$ is the closest lattice point to $\mathbf{x}' - \mathbf{v}$ in $L(b_1, b_2, \cdots, b_n)$ and therefore $||\mathbf{x}' - \mathbf{w}||_2 = ||\mathbf{x}' - \mathbf{v} - \mathbf{y}||_2 \leq 2^{\frac{n-1}{2}} ||\mathbf{x}' - \mathbf{u}||_2 \leq 2^{\frac{n-1}{2}} ||\mathbf{x} - \mathbf{u}||_2$. Since $||\mathbf{x} - \mathbf{x}'||_2 \leq ||\mathbf{x} - \mathbf{u}||_2$, we have $||\mathbf{x} - \mathbf{w}||_2 = \sqrt{||\mathbf{x} - \mathbf{x}'||_2^2| + ||\mathbf{x}' - \mathbf{w}||_2^2} \leq \sqrt{1 + 2^{n-1}} ||\mathbf{x} - \mathbf{u}||_2 < 2^{\frac{n}{2}} ||\mathbf{x} - \mathbf{u}||_2$

- $\mathbf{u} \notin U + \mathbf{v}$: Clearly, $||\mathbf{x} - \mathbf{u}||_2 \geq \frac{1}{2} ||b_n^*||_2 > \frac{1}{2} \cdot 2^{1 - \frac{n}{2}} ||\mathbf{x} - \mathbf{w}||_2$, hence we have $||\mathbf{x} - \mathbf{w}||_2 < 2^{\frac{n}{2}} ||\mathbf{x} - \mathbf{u}||_2$. □

Corollary 27: There exists a polynomial-time algorithm find a vector $\mathbf{x}$ such that $\mathbf{c} \cdot \mathbf{x} = x$ and $||\mathbf{x}||_1 \leq \sqrt{n} \cdot 2^{\frac{n-1}{2}} \mu_{\mathbf{c}}(x)$.

Proof. By Theorem 21 and Theorem 26, we have an algorithm outputs a vector $\mathbf{x}$ such that $||\mathbf{x}||_1 \leq \sqrt{n} ||\mathbf{x}||_2 \leq \sqrt{n} 2^{\frac{n-1}{2}} \min_{\mathbf{v} \in \{\mathbf{x} | \mathbf{c} \cdot \mathbf{x} = x\}} ||\mathbf{v}||_2 \leq \sqrt{n} 2^{\frac{n-1}{2}} \mu_{\mathbf{c}}(x)$, since for any vector $\mathbf{v} = (v_1, v_2, \cdots, v_n)$, $||\mathbf{v}||_2 \leq ||\mathbf{v}||_1 \leq \sqrt{n} ||\mathbf{v}||_2$.

We provide a non-recursive implementation for the nearest plane heuristic, see figure 4.2. This implementation provides the same output as Babai's nearest plane algorithm. It runs iteratively and computes w with x' which is different from orthogonal projection of x, but the lattice point closest to x' is also closest to x. Clearly, it runs in $O(n^2)$ with the Gram-Schmidt orthogonalization, the byproduct of *LLL* basis reduction, as its input. The whole approximation algorithm runs in $O(n^5 \log c_n)$, either an algorithm which finds a shorter basis or a better implementation of *LLL* algorithm will enhance the performance.

Algorithm Non-recursive Nearest Plane Heuristic

Inputs: $(b_1, b_2, \cdots, b_n)$, the LLL-reduced basis.

x, the target vectors.

$(b_1^*, b_2^*, \cdots, b_n^*)$, the Gram-Schmidt orthogalization of $(b_1, b_2, \cdots, b_n)$

Outputs: w, the vector close to x.

Variables: x', temporal storages for the modified objective vector.

y, temporal storages for the b_i's component of x'.

begin

1. $w := 0$; $x' := x$;

2. **for** $i = n$ to 1 **do**

3. $\quad y := \frac{\langle x', b_i^* \rangle}{\langle b_i^*, b_i^* \rangle} b_i$;

4. $\quad w := w + \lfloor \frac{\langle w, b_i^* \rangle}{\langle b_i^*, b_i^* \rangle} \rceil b_i$; //where $\lfloor x \rceil$ is the integer closest to x

5. $\quad x' := x' - y$;

6.**next** i

end

Fig. 4.2: A non-recursive implementation of the nearest plane heuristic algorithm.

4.3 The complexity of Approximating $\mu_{\mathbf{c}}(x)$

The algorithm above can only approximate within a very large factor, say $\sqrt{n}2^{\frac{n-1}{2}}$, in polynomial time. But it is still far from the inapproximable result provided by G. Havas and J.-P. Seifert [6]:

Theorem 28: [6] Unless $NP \subseteq P$, there exists no polynomial-time algorithm which approximate the shortest GCD multiplier problem in l_p-norm within a factor of k, where $k \geq 1$ is an arbitrary constant.

Theorem 29: [6] Unless $NP \subseteq DTIME(n^{poly(\log n)})$, there exists no polynomial-time algorithm which approximate the shortest GCD multiplier problem in l_p-norm within a factor of $n^{1/(p \log^{\gamma} n)}$, where γ is an arbitrary small positive constant.

The shortest GCD multiplier problem is: suppose we are given $c_1, c_2, \cdots, c_n$ and to find $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ such that $\sum_{i=1}^{n} x_i c_i = gcd(c_1, c_2, \cdots, c_n)$ and $||\mathbf{x}||_p$ is minimal. It is clear that this problem is a special case of the problem computing $\mu_{\mathbf{c}}(x)$ when $p = 1$. Moreover, the problem of computing $\mu_{\mathbf{c}}(x)$ has the same result if we follow ideas in [6].

Corollary 30: Unless $NP \subseteq P$, there exists no polynomial-time algorithm which approximate the computing $\mu_{\mathbf{c}}(x)$ problem within a factor of k, where $k \geq 1$ is an arbitrary constant.

Corollary 31: Unless $NP \subseteq DTIME(n^{poly(\log n)})$, there exists no polynomial-time algorithm which approximate the computing $\mu_{\mathbf{c}}(x)$ problem within a factor of $n^{1/\log^{\gamma} n}$, where γ is an arbitrary small positive constant.

4.4 Experiments and results

We implement the pseudo-polynomial time search algorithm and the LLL-based approximation algorithms, both of the rounding off heuristic and nearest plane heuristic algorithms, for computing $\mu_{\mathbf{c}}(x)$. Surprisingly, the rounding off algorithm exactly outputs the value of $\mu_{\mathbf{c}}(x)$ for some inputs, such as $\mathbf{c} = (15, 21, 35)$ in [3] and some other $\mathbf{c}$. Moreover, the rounding off heuristic algorithm outperforms the nearest plane heuristic algorithm in most cases despite the higher theoretical bound of approximation factor.

Let $\nu_{\mathbf{c}}(x)$ be the output of the nearest plane heuristic algorithm and $\xi_{\mathbf{c}}(x)$ be the output of the rounding off heuristic algorithm with $\mathbf{c}$ and x as input.

For little n and capacities c_i's, the approximation algorithms give quite good estimation for $\mu_{\mathbf{c}}(x)$. For example, the figure 4.3 and 4.4 show that the rounding off heuristic algorithm gives optimal solution and the nearest plane heuristic algorithm gives good approximation in most cases.

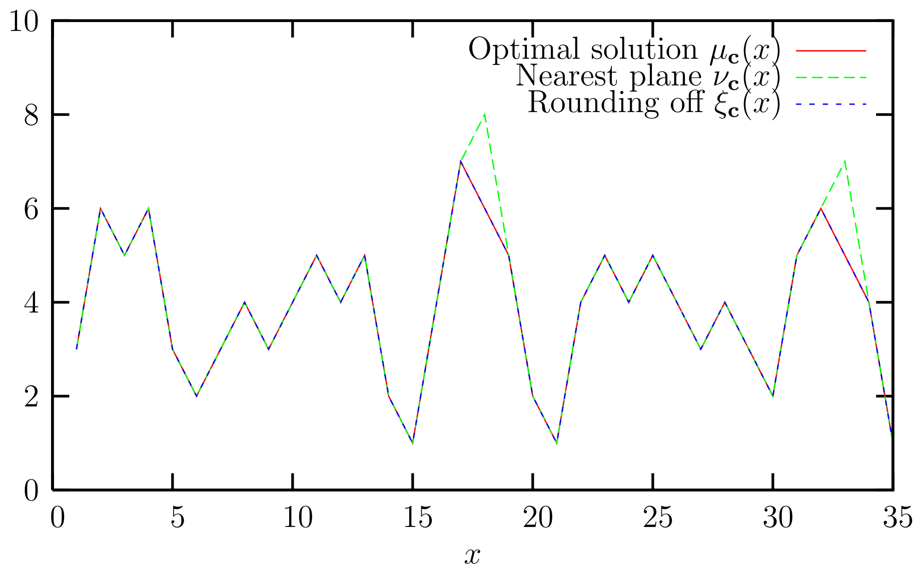

Fig. 4.3: Three jugs. $\mathbf{c} = (15, 21, 35)$

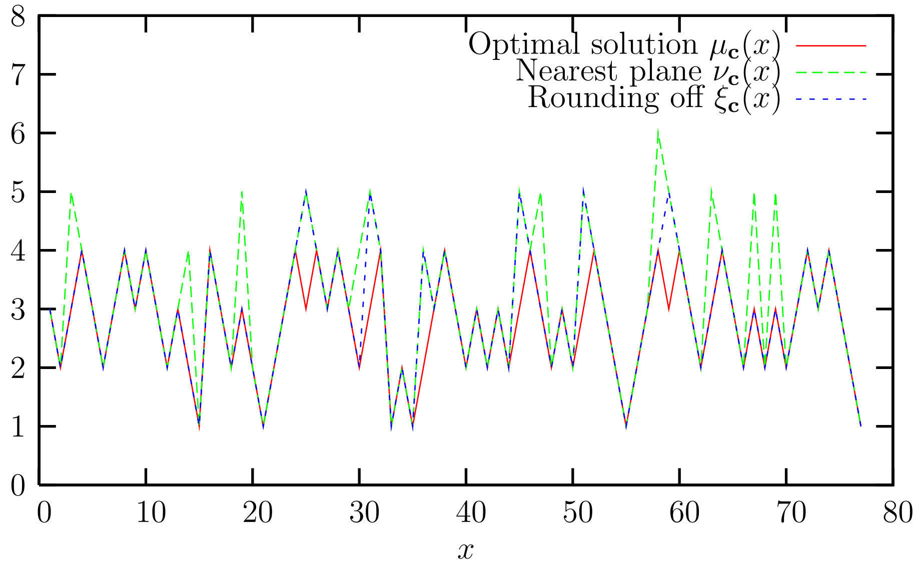

Fig. 4.4: Six jugs. $\mathbf{c} = (15, 21, 33, 35, 55, 77)$

For large n or capacities c_i's, it is hard to generate a fair evaluation of approximation performance for some n since the sets of capacities dominate the approximation performance. Currently it is not clear how to find worst cases and average cases for the *LLL*-based approximation algorithms. Thus we pick some special capacities, such as arithmetic sequence, geometry sequence, Fibonacci numbers and randomly generated capacities, to be experimented. The randomly generated capacities $\mathbf{c}$ with respect ro m is generated by uniformly randomly pick n elements from the set $[1, m]$.

We will present the approximation performance for large c_i's by showing the distribution of $\mu_{\mathbf{c}}(x)$, $\nu_{\mathbf{c}}(x)$ and $\xi_{\mathbf{c}}(x)$, since it is hard to plot curves which have more than thousands points. The approximation performance is better if the distribution of the approximation is closer to the distribution of the optimal solution. Now we illustrate some distribution figures as follows. Currently we can point out that when c_i is a geometry sequence, the performance of the approximation algorithms are good, see figure 4.5, 4.6, and 4.7. And figure and show that the rounding off heuristic algorithm seems to give optimal solution when $c_i = f_{i+1}$, where f_i is the i-th Fibonacci number, see figure 4.8 and 4.9. The approximation factor of approximation algorithms increase when n becomes larger, but for the randomly generated capacities, it seems the factor will not grow as fast as its theoretical upper bounds. Thus the upper bounds are possible loose, see figure 4.10, 4.11, and 4.12. The distribution of $\mu_{\mathbf{c}}(x)$ for the capacities consisting of arithmetic sequence is interesting for most k, $|\{x|\mu_{\mathbf{c}}(x) = k\}| = n - 1$, see figure 4.13 and 4.14.

LLL-based approximation can handle much larger $\mathbf{c}$ and n than the pseudo-polynomial time search algorithm which require exponential time and space. By current test results, it seems the *LLL*-based approximation is capable of approximating the jugs measuring problem in good factors for random inputs. But there might be also some good cases for the approximation algorithm, such as geometry sequences and Fibonacci numbers, we conjecture the shape of the fundamental parallelepiped of the reduced basis is heavily related to the approximation performance.

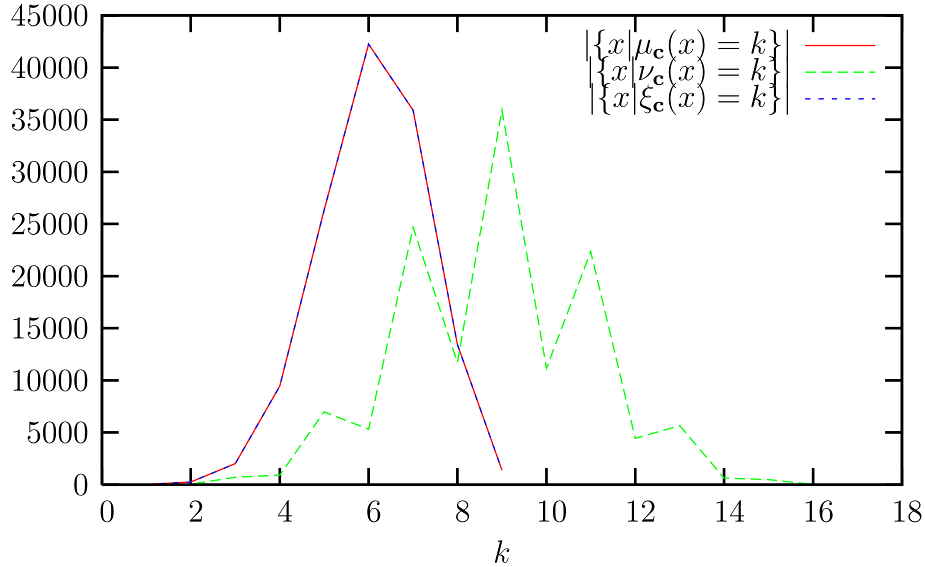

Fig. 4.5: Geometric series: $n = 8, c_i = 5^{i-1}$

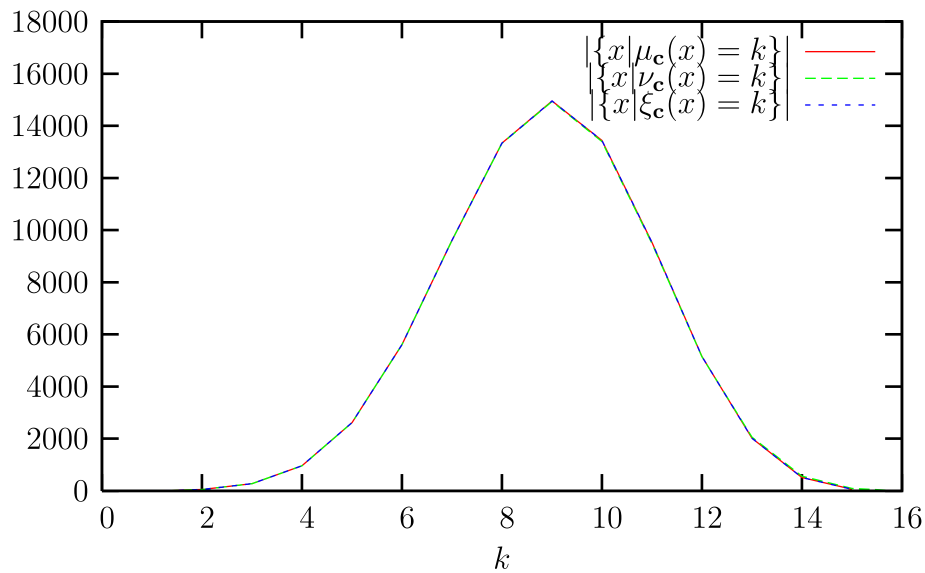

Fig. 4.6: Geometric series: $n = 8, c_i = 5^{i-1}$

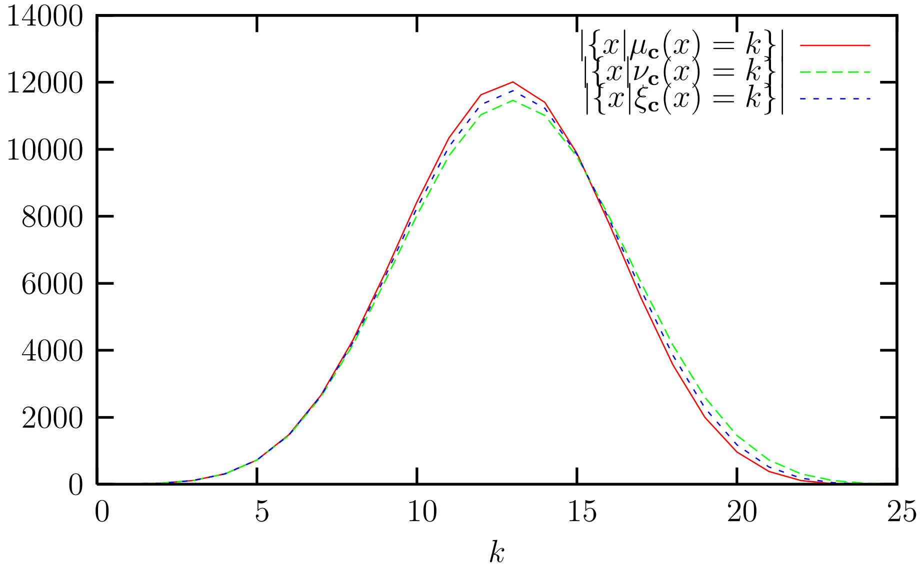

Fig. 4.7: Geometric series: $n = 6, c_i = 10^{i-1}$

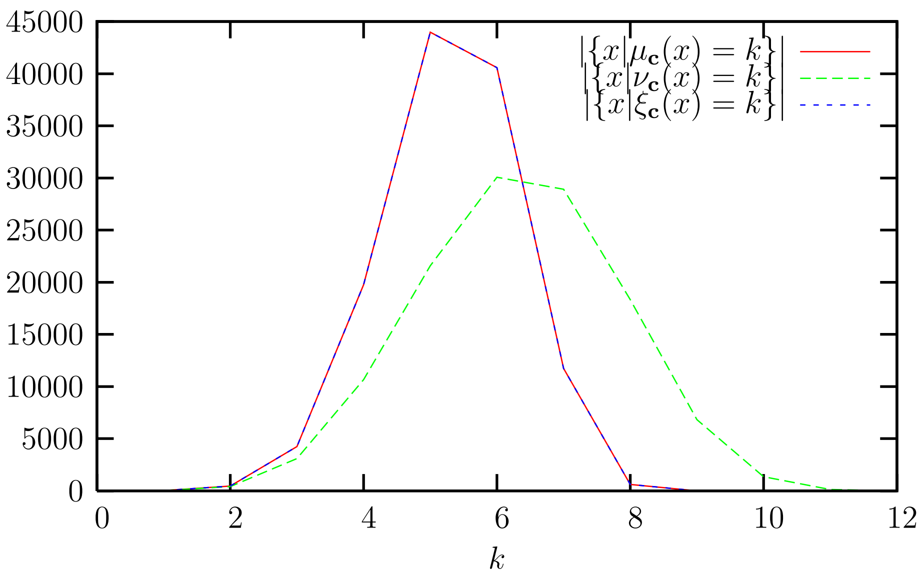

Fig. 4.8: Fibonacci series: $n = 25$

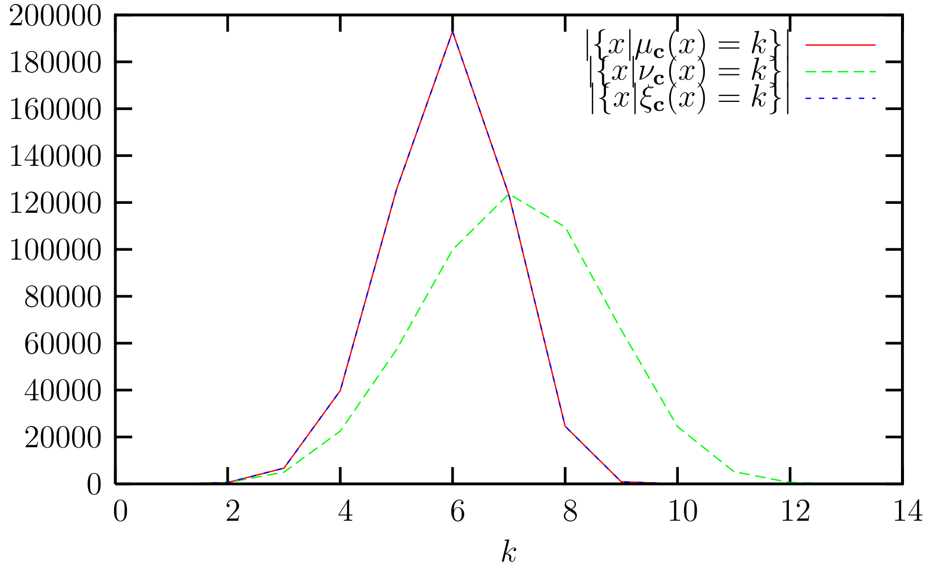

Fig. 4.9: Fibonacci series: $n = 28$

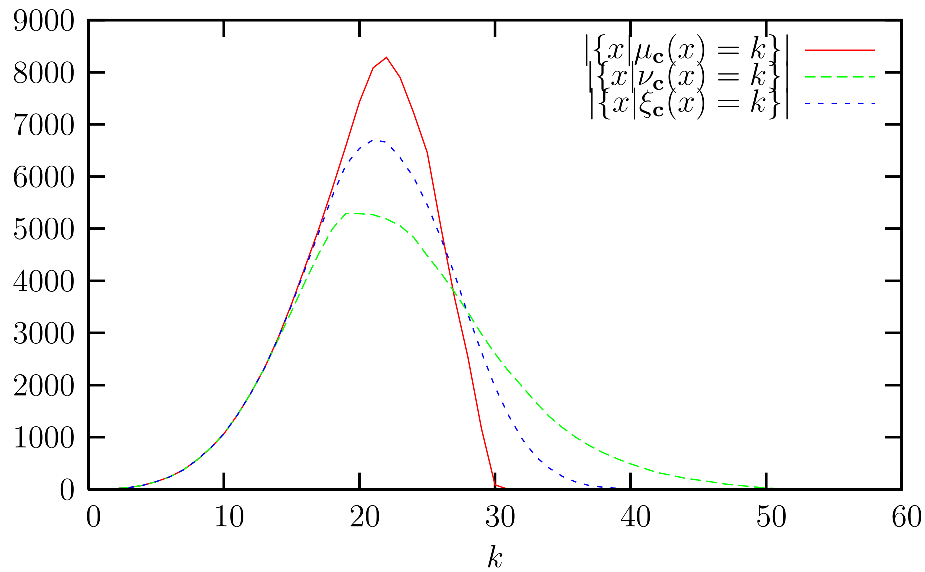

Fig. 4.10: Random case: $n = 5, c_i \in [100000]$

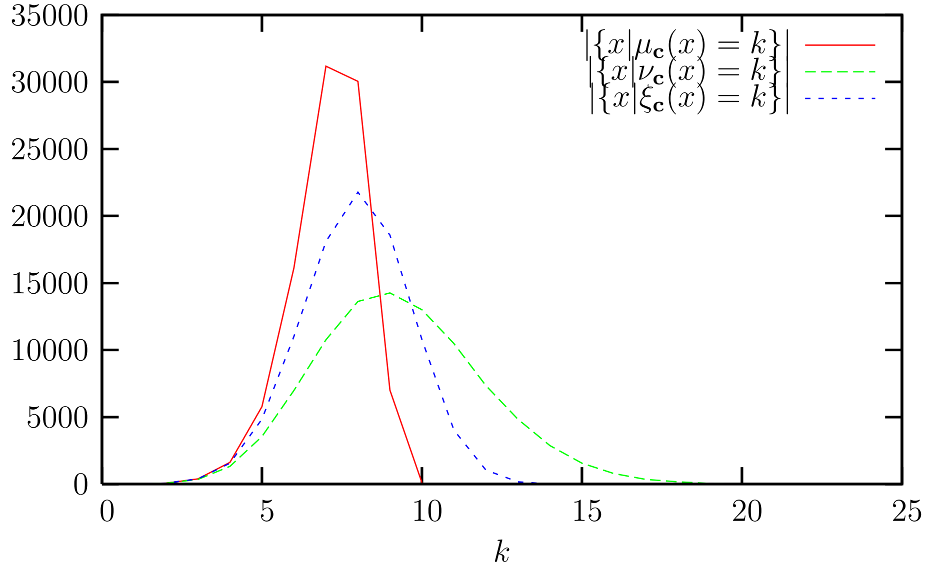

Fig. 4.11: Random case: $n = 10, c_i \in [100000]$

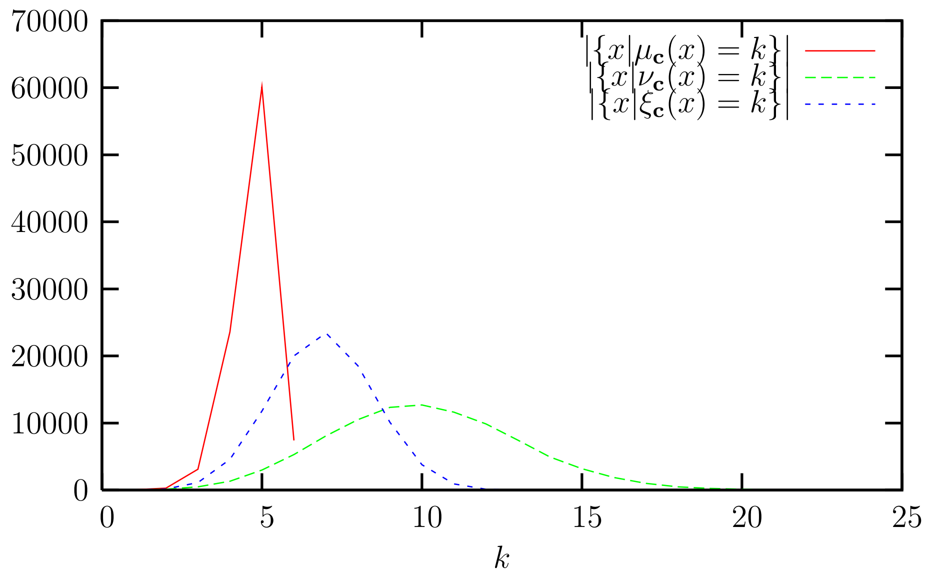

Fig. 4.12: Random case: $n = 20, c_i \in [100000]$

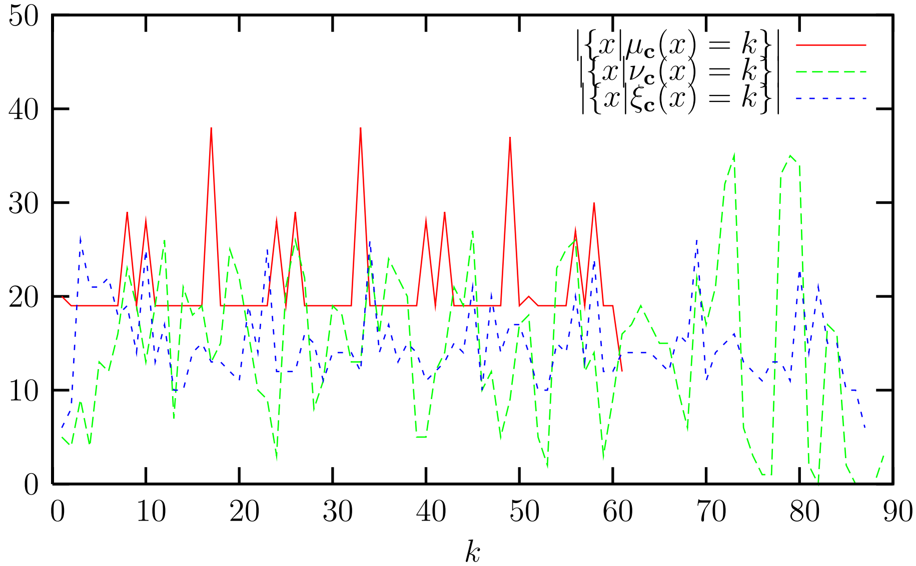

Fig. 4.13: Arithmetic series: $n = 20, c_i = 1001 + 15(i - 1)$

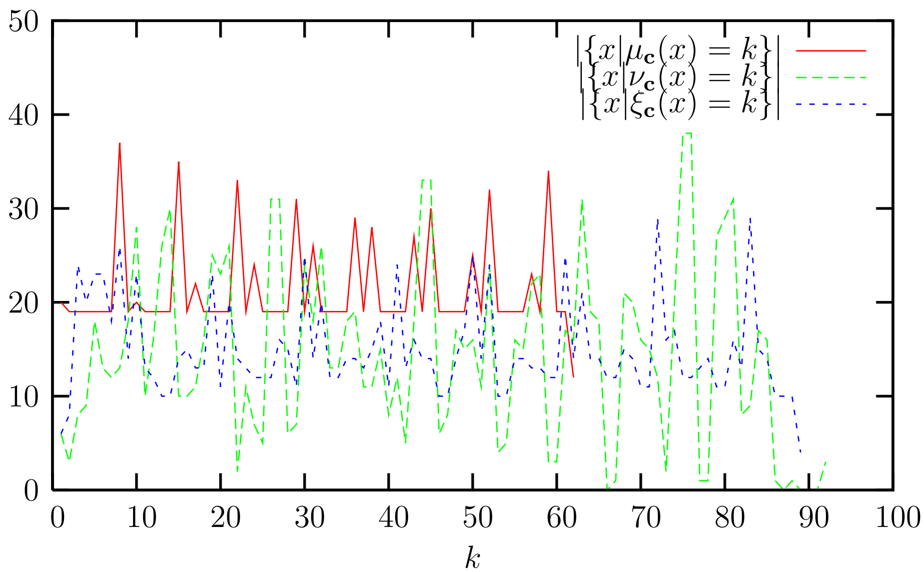

Fig. 4.14: Arithmetic series: $n = 20, c_i = 1001 + 17(i - 1)$

5. CONCLUSION AND REMARKS

We have characterized the additively measurable quantities, and proved new lower and upper bounds for the minimum number of measuring steps. We prove that the problem of computing $\mu_{\mathbf{c}}(x)$ is in the class P^{NP} and is indeed NP-hard, since the bounded version is proved to be NP-complete. It concludes that the optimal jug measuring problem is NP-hard.

BIBLIOGRAPHY

[1] L. Babai, On Lovasz' Lattice Reduction and the Nearest Lattice Point Problem, Combinatorica 6:1-13, 1986.

[2] The American Mathematical Monthly, Volume 109 (1), 2002, page 77.

[3] P. Boldi, M. Santini ans S. Vigna, Measuring with jugs, Theoretical Computer Science, 282 (2002) 259–270.

[4] D.-Z. Du and Ker-I Ko, Theory of Computational Compexity, John Wiley & Sons Inc., 2000.

[5] C. McDiarmid and J. Alfonsin, Sharing jugs of wine, Discrete Math, 125 (1994) 279–287.

[6] G. Havas and J.-P. Seifert, The Complexity of the Extended GCD Problem, Springer LNCS vol.1672, 1999.

[7] L. Lovasz, An Algorithmic Theory of Numbers, Graphs and Convexity, Philadelphia, Pennsylvania, SIAM, 1986.

[8] C. Papadimitriou and K. Steiglitz, Combinatorial Optimization, algorithms and complexity, Prentice Hall, Inc., 1982.

[9] C. Papadimitriou, Computational Complexity, Addison-Wesley Publishing Co, 1995.

[10] M. Sipser, Introduction to the Theory Computation, PWS Publishing Company, 1997.

[11] A. Schrijver, Theory of Linear and Integer Programming, John Wiley & Sons Inc., 1986.

[12] N. P. Smart The Algorithmic Resolution of Diophantine Equations, Cambridge, 1998.