Jug measuring: algorithms and complexity (Extended abstract)

Min-Zheng Shieh and Shi-Chun Tsai
Department of Computer Science
National Chiao Tung University, Hsinchu, Taiwan
{mzhsieh,sctsai}@csie.nctu.edu.tw

Abstract

We study the hardness of the optimal jug measuring problem. By proving tight lower and upper bounds on the minimum number of measuring steps required, we are able to reduce an inapproximable NP-hard problem (i.e., the shortest GCD multiplier problem [6]) to it. It follows that the optimal jug measuring problem is NP-hard and so is the problem of approximating the minimum number of measuring steps within any constant factor. Along the way, we give a polynomial time approximation algorithm with an exponential error based on the well-known LLL basis reduction algorithm for it.

1 Introduction

Let α and β be two positive integers. Given an α-liter jug, a β-liter jug, an unlimited source of water and a drain. You can *fill* a jug full of water from the source, *empty* water in a jug into the drain, or *pour* the water from one jug to another. How do you measure x liter of water? This measuring problem is the so called *water jug problem*, 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, etc. The water jug problem has several variants, such as the problem of sharing jugs of wine [5], i.e., *given an x-liter jug full of wine and two empty jugs of capacity α and β where $x = \alpha + \beta$, what is the quickest way to divide the wine equally by pouring the wine among the jugs?* This problem can be reduced to the water jug problem, since we can replace "pouring from the x-liter jug to any other jug" by the filling operation, and "pouring from any jug to the x-liter jug" by the emptying operation. In [3], a problem proposed by Ehrlich asks: *for two relatively prime integers α and β and given an integer m with $1 \leq m \leq \beta$, is it possible to measure m liters?*

Boldi et al. [4] generalized the water 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. Here a quantity x is called *measurable* if after several steps one of the jugs holds the quantity x. In other words, the generalized water jug problem is: *given a set of jugs of fixed capacities, find out which quantities are measurable.* More specifically, suppose we are given n jugs with positive integer capacities c_i, $i \in [n]$, where $[n]$ denotes the set $\{1, \cdots, n\}$. Without loss of generality, we assume $c_1 \leq c_2 \leq \cdots \leq c_n$. Let x be a positive integer $\leq c_n$. Boldi et al. proved that x is measurable if and only if it is a multiple of the greatest common divisor of c_i's. Define $\mu_{\boldsymbol{c}}(x) = \min_{x=\boldsymbol{x}\cdot\boldsymbol{c}} \|\boldsymbol{x}\|_1$, where $\boldsymbol{x} = (x_1, \cdots, x_n) \in \boldsymbol{Z}^n$ and $\|\boldsymbol{x}\|_1 = \sum_{i=1}^{n} |x_i|$. They also proved that, for every measurable x, (1) x can be measured in at most $\frac{5}{2}\mu_{\boldsymbol{c}}(x)$ steps. (2) No algorithm can measure x in less than $\frac{1}{2}\mu_{\boldsymbol{c}}(x)$ steps.

In this paper we deal with the *optimal jug measuring problem*, which considers the *minimum* number of measuring steps. We generalize the measurability by defining that a quantity x is *additively measurable* if there is a sequence of operations such that the sum of the final contents in the jugs is equal to x. We prove that x is additively measurable if and only if it is a multiple of the greatest common divisor of c_i's. For every additively measurable integer x, we obtain a lower bound $2\mu_{\boldsymbol{c}}(x) - n_e$, where n_e is the number of non-empty jugs. Consequently, we have that: a measurable integer x cannot be measured in less than $2\mu_{\boldsymbol{c}}(x) - n$ steps. Furthermore, all the measurable quantity can be measured in at most $2\mu_{\boldsymbol{c}}(x)$ steps such that the largest jug contains the quan-

tity x and the others are empty. With the tight lower and upper bounds, we can reduce the problem of computing $\mu_{\boldsymbol{c}}(x)$ to the optimal jug measuring problem. Meanwhile, Havas and Seifert [6] proved that a special case of computing $\mu_{\boldsymbol{c}}(x)$ is inapproximable in polynomial time within a factor of k, where $k \geq 1$ is an arbitrary constant. This implies that the optimal jug measuring problem is inapproximable in polynomial time within a constant factor. Moreover, we can reduce the problem of computing $\mu_{\boldsymbol{c}}(x)$ to the closest lattice vector problem. Finally, we propose a polynomial time approximation algorithm with exponential errors for computing $\mu_{\boldsymbol{c}}(x)$ based on the famous *LLL* (Lenstra-Lenstra-Lovasz) basis reduction algorithm.

The rest of the paper is organized as follows. In section 2, we give some notation and definitions. In section 3, we characterize additive measurability and prove new lower and upper bounds for the number of minimum measuring steps. In section 4, we show that computing $\mu_{\boldsymbol{c}}(x)$ is inapproximable and give a polynomial time approximation algorithm with exponential errors for computing $\mu_{\boldsymbol{c}}(x)$. Inapproximable results on the jug measuring problem are also addressed. Section 5 concludes the paper.

2 Notation and definitions

By following Boldi et al. [4], we define three types of elementary operations on the jugs.

Definition 1 *Elementary jug operations:*

(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.*

Let O denotes the set of all possible elementary operations, that is, $O = \{\downarrow i | \forall i \in [n]\} \cup \{i \uparrow | \forall i \in [n]\} \cup \{i \to j | \forall i, j \in [n], i \neq j\}$. We say $\sigma \in O^*$ is a sequence of operations, and use $|\sigma|$ to denote the length of σ (number of operations in σ). We use ϵ to denote the empty sequence, i.e., $|\epsilon| = 0$. Let $\boldsymbol{N}$ be the set of non-negative integers. A *state* is a vector $\boldsymbol{s} \in \boldsymbol{N}^n$, where s_i denotes the amount contained in jug i. We define the state-transition function as follows.

Definition 2 *A function $\delta : \boldsymbol{N}^n \times O^* \to \boldsymbol{N}^n$ is a state transition function if:*

(1) $\delta(\boldsymbol{s}, \epsilon) = \boldsymbol{s}$*;*

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

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

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

(5) for $|\sigma| \geq 1$ *and* $o \in O$*,* $\delta(\boldsymbol{s}, \sigma o) = \delta(\delta(\boldsymbol{s}, \sigma), o)$*.*

We say a state $\boldsymbol{s}$ is *reachable* if $\delta(\boldsymbol{0}, \sigma) = \boldsymbol{s}$ by some sequence $\sigma \in O^*$. Furthermore, we say σ is *optimal* if it is the shortest one that reaches $\boldsymbol{s}$. $\delta_i(\boldsymbol{s}, \sigma)$ denotes the i-th entry of $\delta(\boldsymbol{s}, \sigma)$. $e_i(\sigma)$ denotes the number of $i \uparrow$ operation in σ and $e(\sigma) = \sum_{i \in [n]} e_i(\sigma)$. $f_i(\sigma)$ denotes the number of $\downarrow i$ operation in σ and $f(\sigma) = \sum_{i \in [n]} f_i(\sigma)$. $p_i(\sigma)$ denotes the number of pour operations applied to jug i and $p(\sigma) = \frac{1}{2} \sum_{i \in [n]} p_i(\sigma)$. A quantity $x \in \boldsymbol{N}$ is measurable via sequence σ iff there exists $i \in [n]$ such that $\delta_i(\boldsymbol{0}, \sigma) = x$. For convenience, let $\boldsymbol{c} = \{c_1, \cdots, c_n\}$ and $\gcd(\boldsymbol{c})$ denote the greatest common divisor of $c_1, \cdots, c_n$. The set of quantities that are measurable using the capacities in $\boldsymbol{c}$ is denoted as $\boldsymbol{M}(\boldsymbol{c})$. Boldi et al.[4] proved that all of the measurable quantities are multiples of the greatest common divisor of the capacities as stated in the following.

Prop 1 *[4]* $\boldsymbol{M}(\boldsymbol{c}) = \{ m \cdot \gcd(\boldsymbol{c}) \mid$ *for all non-negative integer* $m \leq c_n / \gcd(\boldsymbol{c})\}$*.*

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

Each $x \in \boldsymbol{M}^+(\boldsymbol{c})$ has one (or more) vector representation $\boldsymbol{x} = (x_1, \cdots, x_n) \in \boldsymbol{Z}^n$ with respect to $\boldsymbol{c}$, such that $x = \boldsymbol{x} \cdot \boldsymbol{c} = \sum_{i=1}^n x_i c_i$. Moreover, we say a vector representation $\boldsymbol{x}$ is minimum if $\|\boldsymbol{x}\|_1$ is minimum. We denote $\mu_{\boldsymbol{c}}(x) = \min_{x = \boldsymbol{x} \cdot \boldsymbol{c}} \|\boldsymbol{x}\|_1$. We will use a representation $\boldsymbol{x}$ to construct a se-

quence of operations achieving the quantity x, and vice versa.

3 Measurability, Lower and Upper bounds

3.1 Measurability

First we show that the additively measurable quantities are multiples of $\gcd(\boldsymbol{c})$, which form a subgroup of $(Z_q, +)$ with $\gcd(\boldsymbol{c})$ as a generator, where $q = \sum_{i=1}^{n} c_i$. It can be proved by induction on the number of jugs.

Theorem 1 *Given $\boldsymbol{c}$, $\boldsymbol{M}^+(\boldsymbol{c}) = \{m \cdot \gcd(\boldsymbol{c}) |$ for all non-negative integer $m \leq \sum_{i=1}^{n} c_i / \gcd(\boldsymbol{c})\}$.*

Proof. There are two parts to be proved. First we show that any additively measurable quantity is a multiple of $\gcd(\boldsymbol{c})$. Secondly, we prove that every non-negative multiple of $\gcd(\boldsymbol{c})$, bounded by $\sum_{i=1}^{n} c_i$, is additively measurable.

The first part is simple. Assume x is additively measurable with $\boldsymbol{c}$. 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. By theorem 1, we know each s_i is a multiple of $\gcd(\boldsymbol{c})$ and thus x is a multiple of $\gcd(\boldsymbol{c})$.

We prove the second part by induction on n, the number of jugs. It is trivial when $n = 1$. Assume that the theorem holds up to $n-1$. Assume $x = m \cdot \gcd(\boldsymbol{c})$ and $x \leq \sum_{i=1}^{n} c_i$, for some $m \in \boldsymbol{N}$. 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 $y = x \bmod \gcd(c_1, c_2, c_3 \ldots, c_{n-1})$. 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 $\gcd(\boldsymbol{c})$. We already know x is a multiple of $\gcd(\boldsymbol{c})$ by assumption, and then so is y. By theorem 1, we have $y \in \boldsymbol{M}(\boldsymbol{c})$. This implies that we can reach $(0, 0, 0, \ldots, y)$ first. By induction hypothesis, $x - y \in \boldsymbol{M}^+(c_1, c_2, \ldots, c_{n-1})$. So we can reach a state $\boldsymbol{s}$ 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 $\gcd(\boldsymbol{c})$, since x and $\sum_{i=1}^{n-1} c_i$ are both multiples of $\gcd(\boldsymbol{c})$, and thus by Theorem 1 we have $y \in \boldsymbol{M}(\boldsymbol{c})$. 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 $\boldsymbol{s}$ with the sum of the quantity in each jug equal to x, by filling all of the jugs other than jug n. From the above, we know that $\boldsymbol{M}^+(\boldsymbol{c}) = [0, \sum_{i=1}^{n} c_i] \cap \{m \cdot \gcd(\boldsymbol{c}) | \forall m \in \boldsymbol{N}\}$. □

3.2 Lower bound

In this subsection, we prove the following lower bound, which improves the previous bound $\frac{1}{2}\mu_{\boldsymbol{c}}(x)$ by Boldi et al.[4].

Theorem 2 *No sequence of operations can measure x, for all $x \in \boldsymbol{M}(\boldsymbol{c})$, in less than $2\mu_{\boldsymbol{c}}(x) - n$ steps.*

Actually we prove a stronger result:

Theorem 3 *Let $\boldsymbol{s} = (s_1, \cdots, s_n)$ be a reachable state, $x = \sum_{i=1}^{n} s_i$ and n_{ne} be the number of non-zero entries of $\boldsymbol{s}$, then no sequence of operations can reach $\boldsymbol{s}$ in less than $2\mu_{\boldsymbol{c}}(x) - n_{ne}$ steps.*

It is clear that if there is a state $\boldsymbol{s}^*$ with sum equal to x can be reached in less than $2\mu_{\boldsymbol{c}}(x) - n$ steps, then $(0, \cdots, 0, x)$ is reachable in less than $2\mu_{\boldsymbol{c}}(x) - 1$ steps by applying at most $n - 1$ extra pour operations.

Before the proof of Theorem 3, we sketch the idea. We show that for a reachable state $\boldsymbol{s}$ with $\sum_{i=1}^{n} s_i = x$, there is an optimal sequence containing at least $\mu_{\boldsymbol{c}}(x)$ non-pour operations and at least $\mu_{\boldsymbol{c}}(x) - n_{ne}$ pour operations.

To prove the bound for non-pour operations, we transform a sequence of operations into another sequence with a property, i.e., *fill* operations are always applied to empty jugs, and *empty* operations to full jugs. A sequence of operations with this property is called a *standard sequence*. Formally, a standard sequence $\sigma = o_1 \cdots o_m$ satisfies:

$$\forall o_l \in \{\downarrow i, i \uparrow\}, \delta_i(\mathbf{0}, o_1 \cdots o_{l-1}) = \begin{cases} 0 & , \text{ if } o_l = \downarrow i \\ c_i & , \text{ if } o_l = i \uparrow \end{cases}$$

Then it suffices to show that for every optimal standard sequence, it has at least $\mu_{\boldsymbol{c}}(x)$ non-pour operations. To prove the bound for pour operations, we start with an optimal standard sequence σ. For σ, we construct a graph $G_\sigma = \langle V_\sigma, E_\sigma \rangle$, where $|V_\sigma| \geq \mu_{\boldsymbol{c}}(x)$ and $|E_\sigma| = p(\sigma)$ (i.e., the number of pour operations). We will show that G_σ must contain at most n_{ne} connected components. Hence $|E_\sigma| \geq |V_\sigma| - n_{ne} \geq \mu_{\boldsymbol{c}}(x) - n_{ne}$.

Lemma 4 *Let $\sigma = o_1 o_2 \cdots o_m \in O^*$ be an arbitrary sequence of m operations such that $\delta(\mathbf{0}, \sigma) = \boldsymbol{t} = (t_1, t_2, \cdots, t_n)$. Then for every $i \in [n]$, there exists a sequence of m operations ρ such that $\delta(\mathbf{0}, \rho) \neq (t_1, \cdots, t_{i-1}, 0, t_{i+1}, \cdots, t_n)$ implies $\delta(\mathbf{0}, \rho) = (t_1, \cdots, t_{i-1}, c_i, t_{i+1}, \cdots, t_n)$.*

Proof. Let $i \in [n]$. We prove the lemma by induction on m. The base case $m = 0$ is trivial. Assume the lemma holds for $m < k$ and let o_l be the last operation involving jug i. Let $\boldsymbol{t}_l = (t_{l_1}, \cdots, t_{l_n}) = \delta(\mathbf{0}, o_1 \cdots o_l)$. If $t_{l_i} = 0$ or $t_{l_i} = c_i$ then the lemma obviously holds. Otherwise, o_l must be $i \to j$ or $j \to i$ for some j. Moreover, after applying $o_1 \cdots o_l$, $t_{l_j} = c_j$ if $o_l = i \to j$, else 0. Now let

$$q_l = \begin{cases} \downarrow j & , \text{ if } o_l = i \to j \\ j \uparrow & , \text{ if } o_l = j \to i \end{cases}$$

Let $\boldsymbol{u} = (u_1, \cdots, u_n) = \delta(\mathbf{0}, o_1 \cdots o_{l-1})$. By the induction hypothesis, there exists a sequence $q_1 \ldots q_{l-1}$ such that if $\delta(\mathbf{0}, q_1 \cdots q_{l-1}) \neq (u_1, \cdots, u_{i-1}, 0, u_{i+1}, \cdots, u_n)$ then $\delta(\mathbf{0}, q_1 \cdots q_{l-1}) = (u_1, \cdots, u_{i-1}, c_i, u_{i+1}, \cdots, u_n)$. Observe that o_l only affects jug i and j. Therefore $u_r = t_{l_r}$ for every $r \in [n] - \{i, j\}$. We have that if $\delta(\mathbf{0}, q_1 \cdots q_l) \neq (t_{l_1}, \cdots, t_{l_{i-1}}, 0, t_{l_{i+1}}, \cdots, t_{l_n})$ then $\delta(\mathbf{0}, q_1 \cdots q_l) = (t_{l_1}, \cdots, t_{l_{i-1}}, c_i, t_{l_{i+1}}, \cdots, t_{l_n})$. Since $o_{l+1}, \cdots, o_k$ do not involve jug i, we have that if $\delta(\mathbf{0}, q_1 \cdots q_l o_{l+1} \cdots o_k) \neq (t_1, \cdots, t_{i-1}, 0, t_{i+1}, \cdots, t_n)$ then $\delta(\mathbf{0}, q_1 \cdots q_l o_{l+1} \cdots o_k) = (t_1, \cdots, t_{i-1}, c_i, t_{i+1}, \cdots, t_n)$. □

The following lemma states that for every sequence σ', we can construct a standard sequence σ which reaches the same state as σ' does without length overhead. Therefore, we can focus on standard sequences, where fill operations are always applied to empty jugs and empty operations to full jugs.

Lemma 5 *For any reachable state $\boldsymbol{s} = (s_1, \cdots, s_n)$, if $\delta(\mathbf{0}, \sigma') = \delta(\mathbf{0}, o'_1 \cdots o'_m) = \boldsymbol{s}$, then there exists a standard sequence $\sigma = o_1 \cdots o_k$ with $k \leq m$ and $\delta(\mathbf{0}, \sigma) = \boldsymbol{s}$.*

Proof. We prove the lemma by induction on $|\sigma'|$. The base case $|\sigma'| = 0$ is trivial. Assume the lemma is true for $|\sigma'| < r$. Consider $\sigma' = o'_1 \cdots o'_r$. If o'_r is a pour operation, then by the induction hypothesis, we are done. Thus we can assume o'_r is a non-pour operation. Assume o'_r is applied to jug i and $\boldsymbol{t} = (t_1, \cdots, t_n) = \delta(\mathbf{0}, o'_1 \cdots o'_r)$. By lemma 4, there is a sequence $\rho' = p'_1 \cdots p'_{r-1}$ such that $\delta(\mathbf{0}, \rho') = (t_1, \cdots, t_{i-1}, t'_i, t_{i+1}, \cdots, t_n)$ where $t'_i \neq 0$ implies $t'_i = c_i$. By the induction hypothesis, there exists a standard sequence ρ, where $|\rho| \leq r - 1$ and $\delta(\mathbf{0}, \rho) = (t_1, \cdots, t_{i-1}, t'_i, t_{i+1}, \cdots, t_n)$.

For the case $o'_r = \downarrow i$: if $s_i = 0$ then $\rho o'_r$ is a standard sequence and $\delta(\mathbf{0}, \rho o'_r) = \boldsymbol{s}$; else if $s_i \neq 0$ then $s_i = c_i$ and it means o'_r is a redundant operation and hence ρ satisfies the lemma. Similarly, for the case $o'_r = i \uparrow$: if $s_i = 0$ then ρ satisfies the lemma; otherwise $\rho o'_r$ does. □

Lemma 5 implies that for every reachable state $\boldsymbol{s}$, there exists an optimal standard sequence of operations to reach it.

Lemma 6 *For any reachable state $\boldsymbol{s}$ with $x = \sum_{i=1}^{n} s_i$, there exists an optimal sequence of operations $\sigma = o_1 \cdots o_m$ with $\delta(\mathbf{0}, \sigma) = \boldsymbol{s}$, such that the number of non-pour operations in σ is at least $\mu_{\boldsymbol{c}}(x)$.*

Proof. By Lemma 5, there exists an optimal standard sequence of operations σ reaching $\boldsymbol{s}$. Since σ is standard, the total amount of water in the jugs is increased or decreased by the amount of c_i after $\downarrow i$ or $i \uparrow$ operation for each $i \in [n]$. Note that pour operations do not change the sum. There are $f_i(\sigma)$ ($\downarrow i$)-operations and $e_i(\sigma)$ ($i \uparrow$)-operations. It is clear $x = \sum_{i=1}^{n} (f_i(\sigma) - e_i(\sigma)) c_i$ and thus $\sum_{i=1}^{n} |f_i(\sigma) - e_i(\sigma)| \geq \mu_{\boldsymbol{c}}(x)$. From the above we have $\sum_{i=1}^{n} (f_i(\sigma) + e_i(\sigma)) \geq \sum_{i=1}^{n} |f_i(\sigma) - e_i(\sigma)| \geq \mu_{\boldsymbol{c}}(x)$. □

We prove the lower bound for pour operations by inspecting a graph G_σ corresponding to an optimal standard sequence of operations $\sigma = o_1 \cdots o_m$. Recall that the quantity in jug i after applying a sequence of operation ρ to state $\boldsymbol{s}$ is denoted as $\delta_i(\boldsymbol{s}, \rho)$. We denote the number of operations making jug i empty in σ as $z_i(\sigma)$, i.e., $z_i(\sigma) = |\{k : k > 0, \delta_i(\mathbf{0}, o_1 \cdots o_k) = 0, \text{ and } \delta_i(\mathbf{0}, o_1 \cdots o_{k-1}) > 0\}|$. Note that $z_i(\sigma)$ can be larger than $e_i(\sigma)$.

For jug i, we define vertices $v_{(i,j)}$, where $i \in [n]$ and $j = 0, \cdots, z_i(\sigma)$ if $\delta_i(\mathbf{0}, \sigma) \neq 0$, else $j = 0, \cdots, (z_i(\sigma) - 1)$. Let V_σ be the set of all $v_{(i,j)}$'s. For each vertex $v_{(i,j)} \in V_\sigma$, we associate it with every operation o_k satisfying: *(1) o_k is applied to jug i. (2) There are exactly j operations making jug i empty before o_k.* More precisely, we associate $v_{(i,j)}$ with the following set of operations: $\{o_k : o_k \text{ is applied to jug } i \text{ and } |\{o_{k'} : k' < k, \delta_i(\mathbf{0}, o_1 \cdots o_{k'}) = 0, \delta_i(\mathbf{0}, o_1 \cdots o_{k'-1}) > 0\}| = j\}$.

Note that in a standard sequence each $v_{(i,j)}$ associates with at most one empty and one fill operation. Now we define the corresponding graph $G_\sigma = \langle V_\sigma, E_\sigma \rangle$ of σ, where V_σ is the set of all $v_{(i,j)}$'s and $\{v_{(i,j)}, v_{(i',j')}\} \in E_\sigma$ if and only if both $v_{(i,j)}$ and $v_{(i',j')}$ associate with a common operation o_l. It is clear that $|E_\sigma|$ is the number of pour operations. We color a vertex $v_{(i,j)}$ white if it as-

sociates with an operation making jug i empty, otherwise we color it gray.

There are some interesting properties about the first and last operations associated with a vertex. Let o_s and o_e be the first and last operations associated with vertex $v_{(i,j)} \in V_\sigma$. We have $\delta_i(\mathbf{0}, o_1 \cdots o_{s-1}) = 0$, since if there exists an operation applied to jug i before o_s, then it must make jug i empty (note that if o_s is the first operation applied to jug i then prior to o_s jug i is empty). Also $\delta_i(\mathbf{0}, o_1 \cdots o_e) = 0$ if and only if $v_{(i,j)}$ is white (note that a gray vertex cannot be associated with an operation making it empty).

For example, we consider an instance with $\boldsymbol{c} = (14, 28, 31)$ and $x = 20$. Let $\sigma' = o_1 o_2 \cdots o_{14} = \downarrow 1 \circ \downarrow 3 \circ 1 \to 2 \circ \downarrow 1 \circ 1 \to 2 \circ 2 \uparrow \circ \downarrow 1 \circ 1 \to 2 \circ 3 \to 2 \circ 2 \uparrow \circ 3 \to 2 \circ \downarrow 3 \circ 3 \to 2 \circ 2 \uparrow$. It is clear that $\delta(\mathbf{0}, \sigma') = (0, 0, 20)$, but σ' is not an optimal sequence of operations. We construct the corresponding graph in figure 1, where each vertex v associates with a set of operations O_v (denoted as $v \leftarrow O_v$) and $z_1(\sigma') = z_2(\sigma') = 3, z_3(\sigma') = 1$.

We now prove the following crucial lemma, which is a key tool to prove the lower bound of pour operations.

Lemma 7 *Let $G_\sigma = \langle V_\sigma, E_\sigma \rangle$ be the corresponding graph of $\sigma = o_1 \cdots o_m$. If a connected component $G'_\sigma = \langle V'_\sigma, E'_\sigma \rangle \subseteq G_\sigma$ contains no gray vertex, then $\delta(\mathbf{0}, \sigma) = \delta(\mathbf{0}, \sigma')$, where σ' is obtained from σ by removing all operations associated with the vertices in V'_σ.*

Proof. Suppose $\sigma' = o'_1 \cdots o'_{m'}$. Since σ' is obtained by removing some operations from σ, we define a 1-1 mapping $g : \{1, \cdots, m'\} \to \{1, \cdots, m\}$ such that o'_i in σ' corresponds to $o_{g(i)}$ in σ in order. In other words, σ' is simply a projection of σ. We claim that: *for every $k \in \{1, \cdots, m'\}$, if o'_k is applied to jug i, then $\delta_i(\mathbf{0}, o'_1 \cdots o'_k) = \delta_i(\mathbf{0}, o_1 \cdots o_{g(k)})$.*

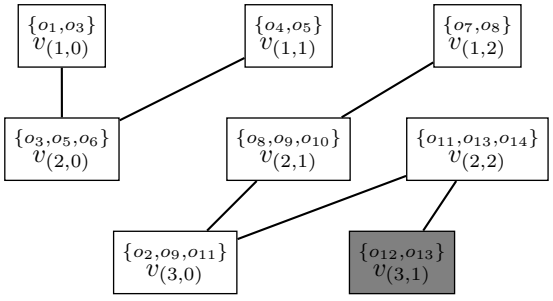

Figure 1: $G_{\sigma'}$: the corresponding graph of σ'.

We show that the lemma follows from the claim. First consider every i with $\delta_i(\mathbf{0}, \sigma') \neq 0$. Let o'_l be the last operation applied to jug i in σ'. We have that $\delta_i(\mathbf{0}, \sigma') = \delta_i(\mathbf{0}, o'_1 \cdots o'_l) = \delta_i(\mathbf{0}, o_1 \cdots o_{g(l)}) = \delta_i(\mathbf{0}, \sigma)$, since there is no operation $o_{l'}$ applied to jug i in σ with $l' > g(l)$. Next for those i with $\delta_i(\mathbf{0}, \sigma) = 0$, there are two possibilities: *(1) All operations on jug i are removed. Therefore it is clear that $\delta_i(\mathbf{0}, \sigma') = \delta_i(\mathbf{0}, \epsilon) = 0 = \delta_i(\mathbf{0}, \sigma)$; (2) There are still some operations on jug i in σ'. Note that $\delta_i(\mathbf{0}, \sigma) = 0$ implies all operations on jug i in σ are associated with white vertex. Let the last operation applied to jug i in σ' be o'_l. Then $\delta_i(\mathbf{0}, \sigma') = \delta_i(\mathbf{0}, o'_1 \cdots o'_l) = \delta_i(\mathbf{0}, o_1 \cdots o_{g(l)}) = 0 = \delta_i(\mathbf{0}, \sigma)$, since $o_{g(l)}$ is the last operation associated with some white vertex.* We conclude that for every $i \in [n]$, $\delta_i(\mathbf{0}, \sigma') = \delta_i(\mathbf{0}, \sigma)$ implies $\delta(\mathbf{0}, \sigma') = \delta(\mathbf{0}, \sigma)$.

Now we prove the claim by way of contradiction. Let k be the smallest index such that when o'_k is applied to some jug π, $\delta_\pi(\mathbf{0}, o'_1 \cdots o'_k) \neq \delta_\pi(\mathbf{0}, o_1 \cdots o_{g(k)})$. It is clear that o'_k must be a pour operation, thus we assume $o'_k = i \to j$. Since o'_k is applied to both jug i and jug j, one of the following two inequalities must hold: $\delta_i(\mathbf{0}, o'_1 \cdots o'_{k-1}) \neq \delta_i(\mathbf{0}, o_1 \cdots o_{g(k)-1})$ or $\delta_j(\mathbf{0}, o'_1 \cdots o'_{k-1}) \neq \delta_j(\mathbf{0}, o_1 \cdots o_{g(k)-1})$. Therefore, if we can show that $\delta_i(\mathbf{0}, o'_1 \cdots o'_{k-1}) = \delta_i(\mathbf{0}, o_1 \cdots o_{g(k)-1})$ and $\delta_j(\mathbf{0}, o'_1 \cdots o'_{k-1}) = \delta_j(\mathbf{0}, o_1 \cdots o_{g(k)-1})$, then the assumption leads to a contradiction.

We show that $\delta_i(\mathbf{0}, o'_1 \cdots o'_{k-1}) = \delta_i(\mathbf{0}, o_1 \cdots o_{g(k)-1})$, and the proof for $\delta_j(\mathbf{0}, o'_1 \cdots o'_{k-1}) = \delta_j(\mathbf{0}, o_1 \cdots o_{g(k)-1})$ is similar. If $o_{g(k)}$ is the first operation applied to jug i in σ, then $\delta_i(\mathbf{0}, o_1 \cdots o_{g(k)-1}) = \delta_i(\mathbf{0}, \epsilon) = 0 = \delta_i(\mathbf{0}, o'_1 \cdots o'_{k-1})$. Assume $o_{g(k)}$ is not the first operation applied to jug i. Let o_l be an operation applied to jug i prior to $o_{g(k)}$. We have two possible cases:

- (Case 1: o_l is not removed.) Then we have $\delta_i(\mathbf{0}, o_1 \cdots o_{g(k)-1}) = \delta_i(\mathbf{0}, o_1 \cdots o_l) = \delta_i(\mathbf{0}, o'_1 \cdots o'_{g^{-1}(l)}) = \delta_i(\mathbf{0}, o'_1 \cdots o'_{k-1})$, since by assumption no integer $k' < k$ such that $o_{k'}$ is applied to jug i and $\delta_i(\mathbf{0}, o'_1 \cdots o'_{k'}) \neq \delta_i(\mathbf{0}, o_1 \cdots o_{g(k')})$.
- (Case 2: o_l is removed.) Then o_l must make jug i empty, otherwise $o_{g(k)}$ and o_l would be associated with the same removed vertex. From this point of view, if there are still some operations applied to jug i in σ, then the last of them must empty jug i. Thus $\delta_i(\mathbf{0}, o_1 \cdots o_{g(k)-1}) = 0 = \delta_i(\mathbf{0}, o'_1 \cdots o'_{k-1})$.

Thus $\delta_i(\mathbf{0}, o'_1 \cdots o'_{k-1}) = \delta_i(\mathbf{0}, o_1 \cdots o_{g(k)-1})$ and similarly $\delta_j(\mathbf{0}, o'_1 \cdots o'_{k-1}) = \delta_j(\mathbf{0}, o_1 \cdots o_{g(k)-1})$. Therefore the claim is true and the lemma is proved. □

Now we can give the lower bound on the number of pour operations in an optimal standard sequence.

Lemma 8 *Let $\sigma = o_1 \cdots o_m$ be an optimal standard sequence for a reachable state $\boldsymbol{s} = \delta(\mathbf{0}, \sigma)$. Let n_{ne} be the number of non-zero entries of $\boldsymbol{s}$, then the number of pour operations in σ is at least $\mu_{\boldsymbol{c}}(x) - n_{ne}$.*

Proof. By lemma 7, G_σ does not contain any connected component consisting of white vertex. Therefore there are at most n_{ne} connected components, i.e., there are at least $|V_\sigma| - n_{ne}$ pour operations. Since σ is standard, for every vertex $v \in V_\sigma$, v associates with at most one fill operation and at most one empty operation. We have $|V_\sigma| \geq \sum_{i=1}^{n} \max(e_i(\sigma), f_i(\sigma)) \geq \sum_{i=1}^{n} |e_i(\sigma) - f_i(\sigma)| \geq \mu_{\boldsymbol{c}}(x)$; hence the number of pour operation in σ is at least $\mu_{\boldsymbol{c}}(x) - n_{ne}$. □

By Lemma 6 and Lemma 8, we have Theorem 3 as an immediate consequence. Note that this lower bound is tight empirically for many cases, for example, for measuring $(0, \cdots, 0, x)$ with $x \in \boldsymbol{M}(\boldsymbol{c})$.

3.3 Upper bound

Assume that there is an extra jug with infinite capacity and $x = \sum_{i=1}^{n} c_i x_i$. Then the following operations can measure x: (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, which additively measures x in $2\mu_{\boldsymbol{c}}(x) + l - 1$ 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. Hence the upper bound won't be exactly $2\mu_{\boldsymbol{c}}(x)$. 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 $x \in \boldsymbol{M}(\boldsymbol{c})$, given the optimal representation of x, our algorithm achieves the best possibility. However, it is not clear how to compute $\mu_{\boldsymbol{c}}(x)$ and the minimum representation efficiently.

Theorem 9 *For all $x \in M^+(\boldsymbol{c})$, if we can use the largest l jugs to hold the quantity x, then the algorithm* MEASURE *additively measures x in $2\mu_{\boldsymbol{c}}(x) + l - 1$ steps.*

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

Lemma 10 *Let $\boldsymbol{x}$ be an optimal representation of x with capacity $\boldsymbol{c}$. The algorithm* MEASURE *outputs a sequence of operations σ such that $\delta(\mathbf{0}, \sigma) = \boldsymbol{s}$ and $\sum_{i=1}^{n} |s_i| = x$.*

Proof. Let $F = \{i | x_i > 0\}$ and $E = \{i | x_i < 0\}$. The integer variables v_i's are used to track the number of empty and fill operations performed. Initially, $v_i = x_i$ for $i \in [n]$. 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. Observe that during the execution the number of fill and empty operations performed on jug i is $(x_i - v_i)$ for $i \in F$ and $(v_i - x_i)$ for $i \in E$, respectively. Thus the total quantity in the jugs is $\sum_{i \in F}(x_i - v_i)c_i - \sum_{i \in E}(v_i - x_i)c_i$ during the operations.

After the first loop (in line 2), $s_i = c_i$ for all $i \in F$. Next we show that after an iteration of the second loop (in lines 3-7), if there still exists a $v_i < 0$, 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.) If the loop condition holds, we can always find j in line 4, since $s_j > 0$ and $v_j \geq 0$.
- (Case 2: Jug j becomes empty.) If $v_j > 0$, then jug j will be refilled immediately and $s_j > 0$. If $v_j = 0$, suppose that line 4 fails to find a j in the following iteration and it implies that for all $k \in F$ with $v_k \geq 0$, we have $s_k = 0$. While with line 6, we know that for each $k \in F$, v_k cannot be 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 $\sum_{i \in F} c_i x_i + \sum_{i \in E} c_i(x_i - v_i) = \boldsymbol{x} \cdot \boldsymbol{c} - \sum_{i \in E} c_i v_i$, 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.

Algorithm MEASURE($\boldsymbol{c}$, x, $\boldsymbol{x}$)
Input: $\boldsymbol{c} = (c_1, \cdots, c_n)$, the capacity of jugs.
x, the quantity to be measured.
$\boldsymbol{x} = (x_1, \cdots, x_n)$, the optimal representation of x that achieves $\mu_{\boldsymbol{c}}(x)$.
Output: the sequence σ, such that $\delta^*(\mathbf{0}, \sigma)$ achieves the quantity x.
Variable: $\boldsymbol{s}$, the state of jugs, which is initialized to be zero state.
$\boldsymbol{v} = (v_1, \cdots, v_n)$, initialized to be $\boldsymbol{x}$.
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** ($s_i + s_j > c_j$)
3. **then** $\{s_i := s_i + s_j - c_j$; $s_j := c_j;\}$
4. **else** $\{s_j := s_i + s_j$; $s_i := 0;\}$
end

Figure 2: Measuring algorithm given $\mu_{\boldsymbol{c}}(x)$.

After the second loop (lines 3-7), no more empty operation will be performed. Also it is clear that for all $i \in E$, $v_i = 0$ 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.

By the assumption, the largest l jugs are sufficient to contain the quantity x and thus $\sum_{j > n-l} c_j \geq x$. After the second loop and by the fact that we always fill jug i at line 11 when $v_i > 0$ and $s_i = 0$, we have that for every $k \in [n]$ if $v_k > 0$ then $s_k > 0$. This implies there must be some jug $j > n - l$ with $s_j \neq c_j$ and $v_j = 0$. Otherwise $\forall j > n - l$ we have $s_j = c_j$ or $v_j > 0$, which implies total quantity in the jugs will be more than $\sum_{j > n-l} c_j \geq x$, a contradiction. Thus, whenever the loop condition at line 8 holds (i.e., $v_i > 0$), line 9 can always find a suitable jug for pouring.

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

Lemma 11 *Let $\boldsymbol{x}$ be an optimal representation of x with capacity $\boldsymbol{c}$. The algorithm* MEASURE *outputs a sequence of operations σ such that $|\sigma| \leq 2\mu_{\boldsymbol{c}}(x) + l - 1$.*

Proof. Our strategy is to relate each pour operation to a fill or empty operation. For every pour operation *pour*(i, j), there are two possible cases:

- (Case 1: Jug i becomes empty.) Then it is the last operation associated with vertex $v_{(i,k)}$ for some k. We relate it to the closest prior $fill(i)$ operation, which is associated with a vertex $v_{(i,k')}$ for some $k' \leq k$.
- (Case 2: Jug i is not empty and jug j becomes full.) Then it is associated with a vertex $v_{(j,k)}$ for some k and may be followed by an *empty*(j) operation which is also associated with a vertex $v_{(j,k)}$. If there is no *empty*(j) operation after it, then we relate it to jug j, else to the following *empty*(j) operation.

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 can be related to at most one pour operation, since by following the algorithm for every $i \in F$ after a pour operation that empties jug i, there is either an

immediate fill(i) operation or no more pour operation from jug i. Also every empty operation can be related to at most one pour operation, since for every $j \in E$, the only possible operation between a pour operation that fills jug j fully and the next empty(j) is a fill operation, not a pour or an empty operation. These two facts imply the pour operations relate to at most $\mu_{\boldsymbol{c}}(x)$ fill and empty operations together.

The pour operations related to jugs are always executed in the third loop (lines 8-11), and there are at most $l-1$ pour operations related to jugs, otherwise $x > \sum_{i=n-l+1}^{n} c_i$. We conclude that the number of pour operations is no more than $\mu_{\boldsymbol{c}}(x)+l-1$ and thus $|\sigma| \leq 2\mu_{\boldsymbol{c}}(x)+l-1$. □

Theorem 9 follows by lemma 10 and lemma 11.

Corollary 12 *For all $x \in M(\boldsymbol{c})$, there exists a sequence of operations ρ such that $\delta(\mathbf{0},\rho) = (0,\cdots,0,x)$ and $|\rho| \leq 2\mu_{\boldsymbol{c}}(x)$.*

Proof. Let σ be the sequence output by MEASURE. With a closer observation, there is no pour operation related to jug in σ when $l = 1$. Let $F^* = \{i : v_i > 0$, right after the second loop (lines 3-7)$\}$. Note that for every $i \in F^*$, the last $fill(i)$ won't be followed by a pour operation applied to jug i, since v_i becomes 0 and the loop condition won't hold. Thus when the algorithm terminates, there are at least $|F^*|$ full jugs and we have $|\sigma| \leq 2\mu_{\boldsymbol{c}}(x) - |F^*|$, since there are at least $|F^*|$ fill operations not related to any pour operation. Let $\rho = \sigma \circ i_1 \rightarrow n \circ \cdots \circ i_{|F^*|} \rightarrow n$ where $F^* = \{i_1,\cdots,i_{|F^*|}\}$. It is clear that $\delta(\mathbf{0},\rho) = (0,\cdots,0,x)$ and $|\rho| \leq 2\mu_{\boldsymbol{c}}(x)$. □

From corollary 12, it is clear that when measuring $(0,\cdots,0,x)$ with $x \in \boldsymbol{M}(\boldsymbol{c})$, the bound is very close to the lower bound. While there still exists a gap when considering larger quantities that exceed the largest capacity.

4 The complexity of computing $\mu_{\boldsymbol{c}}(x)$ and its approximation

4.1 Hardness of jug measuring

Recall that we define the optimal jug measuring problem as: given n jugs with capacity $c_1,\cdots,c_n$, and a state $\boldsymbol{s}$, to find the length of shortest sequence σ such that $\delta(\mathbf{0},\sigma) = \boldsymbol{s}$.

We have used $\mu_{\boldsymbol{c}}(x)$ to bound the number of steps on jug measuring. In this section, we investigate the hardness of computing $\mu_{\boldsymbol{c}}(x)$ and the optimal jug measuring problem. It can be shown directly that computing $\mu_{\boldsymbol{c}}(x)$ is indeed NP-hard. For the notations of computational complexity we refer to standard textbooks such as by Sipser [10] and Papadimitriou [9]. However, we have stronger results, i.e., it is hard even to approximate it.

To study the complexity of computing $\mu_{\boldsymbol{c}}(x)$, we investigate the shortest GCD multiplier problem [6], which is: given $c_1, c_2, \cdots, c_n$, we want to find $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$ such that $\sum_{i=1}^{n} x_i c_i = gcd(c_1, c_2, \cdots, c_n)$ and $\|\boldsymbol{x}\|_p$ is minimal. It is clear that this problem is a special case of the problem of computing $\mu_{\boldsymbol{c}}(x)$ when $p = 1$. Havas and Seifert [6] proved that: (1) Unless $NP \subseteq P$, there exists no polynomial-time algorithm which approximates the shortest GCD multiplier problem in l_p-norm within a factor of k, where $k \geq 1$ is an arbitrary constant. (2) Unless $NP \subseteq DTIME(n^{poly(\log n)})$, there exists no polynomial-time algorithm which approximates 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.

Let σ be an optimal sequence to measure $x \in \boldsymbol{M}(\boldsymbol{c})$. By the lower and upper bounds of the previous section, we have $\mu_{\boldsymbol{c}}(x) = \lceil \frac{|\sigma|}{2} \rceil$ where $\delta(\mathbf{0},\sigma) = (0,\cdots,0,x)$. In other words, if we know how to solve the optimal jug measuring problem, then we know the value of $\mu_{\boldsymbol{c}}(x)$. By selecting $x = \gcd(\boldsymbol{c})$, for the optimal jug measuring problem we have analogous results:

Theorem 13 *Unless $NP \subseteq P$, there exists no polynomial-time algorithm which approximates the optimal jug measuring problem within a factor of k, where $k \geq 1$ is an arbitrary constant.*

Theorem 14 *Unless $NP \subseteq DTIME(n^{poly(\log n)})$, there exists no polynomial-time algorithm which approximates the optimal jug measuring problem within a factor of $n^{1/\log^{\gamma} n}$, where γ is an arbitrary small positive constant.*

4.2 Reduction from computing $\mu_{\boldsymbol{c}}(x)$ to *CVP*

In this section we give a polynomial time reduction from the problem of computing $\mu_{\boldsymbol{c}}(x)$ to *CVP* and an *LLL*-based approximation algorithm for computing $\mu_{\boldsymbol{c}}(x)$ with *exponential* errors (approximation ratio). Our approximation algorithm is based on the fact: computing $\mu_{\boldsymbol{c}}(x)$ can be polynomially reduced to the *closest lattice vector* problem (*CVP*). This is an extension of the approach

given by Havas et al.[7] for approximating the extended GCD problem.

We introduce lattice and the closest lattice problem briefly as follows. A *lattice* in $\boldsymbol{R}^n$ is the set all integer linear combination of m independent column vectors $\boldsymbol{b}_1, \boldsymbol{b}_2, \cdots, \boldsymbol{b}_m$. The lattice generated by $\boldsymbol{b}_1, \boldsymbol{b}_2, \cdots, \boldsymbol{b}_m$, denoted as $L(\boldsymbol{b}_1, \boldsymbol{b}_2, \cdots, \boldsymbol{b}_m)$, is the set $\{\sum_{i=1}^{m} \lambda_i \boldsymbol{b}_i | \forall i \in [m], \lambda_i \in \boldsymbol{Z}\}$. The independent vectors $\boldsymbol{b}_1, \boldsymbol{b}_2, \cdots, \boldsymbol{b}_m$ are called a basis of the lattice. The *closest lattice vector problem* is: given a basis $\boldsymbol{b}_1, \boldsymbol{b}_2, \cdots, \boldsymbol{b}_m$, a vector $v \in \boldsymbol{R}^n$ and an integer p, we want to find the lattice point $u \in L(\boldsymbol{b}_1, \boldsymbol{b}_2, \cdots, \boldsymbol{b}_m)$ which is closest to v under l_p-*norm*.

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

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.

A nonsingular matrix U is a unimodular matrix if U is integral and has determinant 1 or -1. Unimodular matrix can be obtained by applying some unimodular operations to I. For example, $\begin{bmatrix} 3 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ is a unimodular matrix which is obtained by: exchanging column 1 and 2, multiplying column 1 by -1, then adding 3 times column 2 to column 1. There are several known facts about Hermite normal form:

Theorem 15 *[11] (1) Each rational matrix of full row rank can be brought into Hermite normal form by a series of elementary column operations. (2) 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. (3) Given a feasible system* $A\boldsymbol{y} = \boldsymbol{b}$ *of rational linear diophantine equations, we can find in polynomial time integral vectors* $\boldsymbol{y}_0, \boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_t$ *such that* $\{\boldsymbol{y} | A\boldsymbol{y} = \boldsymbol{b}; \boldsymbol{y}$ *is integral*$\} = \{\boldsymbol{y}_0 + \lambda_1 \boldsymbol{y}_1 + \cdots + \lambda_t \boldsymbol{y}_t | \lambda_1, \cdots, \lambda_t \in \boldsymbol{Z}\}$ *with* $\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_t$ *linearly independent. Moreover,* $[\boldsymbol{y}_0\ \boldsymbol{y}_1\ \boldsymbol{y}_2\ \cdots\ \boldsymbol{y}_t] = U \begin{bmatrix} B^{-1}\boldsymbol{b} & 0 \\ 0 & I \end{bmatrix}$, *where* $AU = [B\ \ 0]$ *is the Hermite normal form of A.*

From the above we have the following result, with which we find a reduction from computing $\mu_{\boldsymbol{c}}(x)$ to *CVP*.

Corollary 16 *The problem of computing* $\mu_{\boldsymbol{c}}(x)$ *can be polynomially reduced to CVP.*

Proof. Assume $\langle (c_1, c_2, \cdots, c_n), x \rangle$ is an instance for computing $\mu_{\boldsymbol{c}}(x)$. Let 1-by-n-matrix $C = [c_1\ c_2\ \cdots\ c_n]$. By theorem 15-(2), there exists a unimodular n-by-n-matrix U such that $CU = [b\ 0\ \cdots\ 0]$ is the Hermite normal form of C. Let $[\boldsymbol{v}_0\ \boldsymbol{v}_1\ \boldsymbol{v}_2\ \cdots\ \boldsymbol{v}_{n-1}] = U \begin{bmatrix} \frac{x}{b} & 0 \\ 0 & I \end{bmatrix}$. By theorem 15-(3), we know $\boldsymbol{v}_1, \boldsymbol{v}_2 \cdots, \boldsymbol{v}_{n-1}$ are linearly independent column vectors and form a basis of a lattice L. Hence $\mu_{\boldsymbol{c}}(x) = \min_{\boldsymbol{v} \in \{\boldsymbol{x} | \boldsymbol{c} \cdot \boldsymbol{x} = x\}} \|\boldsymbol{v}\|_1 = \min_{\lambda_1, \cdots, \lambda_{n-1} \in \boldsymbol{Z}} \|\boldsymbol{v}_0 + \lambda_1 \boldsymbol{v}_1 + \cdots + \lambda_{n-1} \boldsymbol{v}_{n-1}\|_1 = \min_{\boldsymbol{w} \in L} \|\boldsymbol{w} - (-\boldsymbol{v}_0)\|_1$. Thus $\mu_{\boldsymbol{c}}(x)$ is the l_1-*norm* of the vector $\boldsymbol{v} \in L$ which is closest to $-\boldsymbol{v}_0$. It is clear that all computation can be done in polynomial time. □

For the problem of computing the unimodular matrix U such that $[c_1\ c_2\ \cdots\ c_n]U$ is in the Hermite normal form, we propose an algorithm which is simpler than the general algorithm in [11] and the algorithms mentioned in [7]. However, the algorithms in [7] could outperform our algorithm. Our algorithm works for our application on the special case $[c_1\ c_2\ \cdots\ c_n]U$, but it can't compute the unimodular matrix U for any other n by m matrix, where $n > 1$. The algorithm is shown in figure 3, and it is based on the Euclidean algorithm. It computes 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 it runs in $O(n^2 \log c_n)$ time. The maximum of the absolute value of the entries of U won't exceed $O(n \log c_n)$.

Babai [2] provided two polynomial-time approximation algorithms for *CVP*. Both algorithms are based on *LLL* basis reduction algorithm proposed by A. K. Lenstra, H. W. Lenstra and L. Lovasz [8]. Assume we are given an *LLL* reduced basis $\{\boldsymbol{b}_1, \boldsymbol{b}_2, \cdots, \boldsymbol{b}_n\}$ where $\|\boldsymbol{b}_1\|_2 \leq 2^{\frac{n-1}{2}} \min_{\boldsymbol{v} \in L(\boldsymbol{b}_1, \cdots, \boldsymbol{b}_n) - \{\boldsymbol{0}\}} \|\boldsymbol{v}\|_2$, a vector $\boldsymbol{x} = \sum_{i=1}^{n} \alpha_i \boldsymbol{b}_i$ and we are to find a vector $\boldsymbol{w} \in L(\boldsymbol{b}_1, \boldsymbol{b}_2, \cdots, \boldsymbol{b}_n)$ that is close to $\boldsymbol{x}$. The first algorithm is called the rounding off heuristic algorithm. It simply outputs $\boldsymbol{w} = \sum_{i=1}^{n} \beta_i b_i$ where β_i is the closest integer to α_i.

Algorithm HERMITE NORMALIZE(C)
Inputs: $C = [c_1 c_2 \cdots c_n]$, the capacities of jugs.
Outputs: $U = [\boldsymbol{u}_1 \cdots \boldsymbol{u}_n]$ such that $[c_1 c_2 \cdots c_n]U$ is in the Hermite normal form.
Variables: q, temporal storages for $\lfloor \frac{c_i}{c_1} \rfloor$

begin
1. $U := I$;
2. **for** $i = 2$ to n **do**
3. **while** (*true*) **do**
4. $q := \lfloor \frac{c_i}{c_1} \rfloor$; $c_i := c_i - qc_1$;
5. $\boldsymbol{u}_i := \boldsymbol{u}_i - q\boldsymbol{u}_1$;
6. **if** ($c_i = 0$) **then** break;
7. Swap(c_i, c_1); Swap($\boldsymbol{u}_i, \boldsymbol{u}_1$);
8. **loop**
9. **next** i
end

Figure 3: A simple algorithm to compute U.

The second algorithm is called the nearest plane heuristic algorithm. It is a recursive algorithm. Let $V = span(\boldsymbol{b}_1, \boldsymbol{b}_2, \cdots, \boldsymbol{b}_{n-1})$, and find $\boldsymbol{v} \in L(\boldsymbol{b}_1, \boldsymbol{b}_2, \cdots, \boldsymbol{b}_n)$ such that the distance between $V + \boldsymbol{v}$ and $\boldsymbol{x}$ is minimal. Let $\boldsymbol{x}'$ be the orthogonal projection of $\boldsymbol{x}$ on $V + \boldsymbol{v}$. Then find $\boldsymbol{y} \in L(\boldsymbol{b}_1, \boldsymbol{b}_2, \cdots, \boldsymbol{b}_{n-1})$ close to $\boldsymbol{x}' - \boldsymbol{v}$, and output $\boldsymbol{w} = \boldsymbol{y} + \boldsymbol{v}$. Both algorithms guarantee that $\boldsymbol{w}$ is close to $\boldsymbol{x}$.

Theorem 17 *[2] The rounding off heuristic algorithm find a vector $\boldsymbol{w}$ such that $\|\boldsymbol{x} - \boldsymbol{w}\|_2 \leq (1 + 2n(\frac{9}{2})^{\frac{n}{2}}) \min_{\boldsymbol{v} \in L(\boldsymbol{b}_1, \boldsymbol{b}_2, \cdots, \boldsymbol{b}_n)} \|\boldsymbol{x} - \boldsymbol{v}\|_2$.*

Theorem 18 *[2] The nearest plane algorithm heuristic algorithm find a vector $\boldsymbol{w}$ such that $\|\boldsymbol{x} - \boldsymbol{w}\|_2 \leq 2^{\frac{n}{2}} \min_{\boldsymbol{v} \in L(\boldsymbol{b}_1, \boldsymbol{b}_2, \cdots, \boldsymbol{b}_n)} \|\boldsymbol{x} - \boldsymbol{v}\|_2$.*

Using the nearest plane heuristic algorithm, we can approximate $\mu_{\boldsymbol{c}}(x)$ in polynomial time but with an exponential error (approximation ratio) .

Corollary 19 *There exists a polynomial-time algorithm to find a vector $\boldsymbol{x}$ such that $\boldsymbol{c} \cdot \boldsymbol{x} = x$ and $\|\boldsymbol{x}\|_1 \leq \sqrt{n} \cdot 2^{\frac{n-1}{2}} \mu_{\boldsymbol{c}}(x)$.*

The *LLL* algorithm can reduce a basis of some lattice L to a shorter one. If we replace the *LLL* reduced basis with another shorter basis in the nearest plane heuristic algorithm, it will have a better performance. Using the basis reduction algorithm in [1], we can have a smaller approximation ratio $2^{O(n \log \log n / \log n)}$.

5 Conclusion and remarks

We have characterized the additively measurable quantities, and proved new lower and upper bounds on the minimum number of measuring steps. We prove that the optimal jug measuring problem is hard to approximate within constant factor. Finally, based on *LLL*-algorithm we give a polynomial time approximation algorithm with exponential errors for it.

References

[1] Miklos Ajtai, Ravi Kumar, and D. Sivakumar, A sieve algorithm for the shortest lattice vector problem. In Proceedings of the 33rd ACM Symposium on Theory of Computing, pages 601–610, 2001.

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

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

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

[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] G. Havas, B. S. Majewski, and K. R. Matthews, Extended GCD and Hermite Normal Form Algorithm via Lattice Basis Reduction, Experimental Mathematics 7, 1998.

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

[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.