When Input Integers are Given in the Unary Numeral Representation

Tomoyuki Yamakami[1,*,†]

[1]Faculty of Engineering, University of Fukui, 3-9-1 Bunkyo, Fukui 910-8507, Japan

Abstract

Many NP-complete problems take integers as part of their input instances. These input integers are generally binarized, that is, provided in the form of the "binary" numeral representation, and the lengths of such binary forms are used as a basis unit to measure the computational complexity of the problems. In sharp contrast, the "unarization" (or the "unary" numeral representation) of numbers has been known to bring a remarkably different effect onto the computational complexity of the problems. When no computational-complexity difference is observed between binarization and unarization of instances, on the contrary, the problems are said to be strong NP-complete. This work attempts to spotlight an issue of how the unarization of instances affects the computational complexity of various combinatorial problems. We present numerous NP-complete (or even NP-hard) problems, which turn out to be easily solvable when input integers are represented in unary. We then discuss the computational complexities of such problems when taking unary-form integer inputs. We hope that a list of such problems signifies the structural differences between strong NP-completeness and non-strong NP-completeness.

Keywords

unary numeral representation, unarization of integers, logarithmic-space computation, pushdown automata, log-space reduction

1. Background and Quick Overview

1.1. Unary Representations of Integer Inputs

The *theory of NP-completeness* has made great success in providing a plausible evidence to the hardness of target computational problems when one tries to solve them in feasible time. The proof of NP-completeness of those problems therefore makes us turn away from solving them in a practical way but it rather guides us to look into the development of approximation or randomized algorithms.

In computational complexity theory, we often attempt to determine the minimum amount of computational resources necessary to solve target combinatorial problems. Such computational resources are measured in terms of the sizes of input instances given to the problems. Many NP-complete problems, such as the *knapsack problem*, the *subset sum problem*, and the *integer linear programming problem*, concern the values of integers and require various integer manipulations. When instances contain integers, these integers are usually expressed in the form of "binary" representation. Thus, the computational complexities, such as running

The 24th Italian Conference on Theoretical Computer Science (ICTCS 2023), Palermo, Italy, September 13–15, 2023

*Corresponding author.

TomoyukiYamakami@gmail.com (T. Yamakami)

time or memory space, are measured with respect to the total number of bits used for this representation.

This fact naturally brings us a question of what consequences are drawn when input integers are all provided in "unary" using the unary numeral system instead. How does the unary representation affects the computational complexity of the problems? In comparison to the binary numeral system, the unary numeral system is so distinctive, that we need to heed a special attention in our study on the computational complexity of algorithms.

When input integers given to combinatorial problems are expressed in unary, a simple transformation of the unary expression of input integers to their binary representations makes the original input lengths look exponentially larger than their binary lengths. Thus, any algorithm working with the unarized integer inputs seem to consume exponentially less time than the same algorithm with binarized integer inputs. This turns out to be a quite short-sighted analysis.

We often observe that the use of the unary representation significantly alters the computational complexity of combinatorial problems. However, a number of problems are known to remain NP-complete even after we switch binarized integer inputs to their corresponding unarized ones (see [1, Section 4.2]). Those problems are said to be *strong NP-complete*[1] and this notion has been used to support a certain aspect of the robustness of NP-completeness notion. Non-strong NP-complete problems are, by definition, quite susceptible to the change of numeral representations of their input integers between the binary representation and the unary representation. It is therefore imperative to study a structural-complexity aspect of those non-strong NP-complete problems when input integers are provided in the form of the unary numeral representation. In this work, we wish to look into the features of such non-strong NP-complete problems.

Earlier, Cook [2] discussed the computational complexity of a unary-representation analogue of the knapsack problem, called the *unary 0-1 knapsack problem* (UK), which asks whether or not there is a subset of given positive integers, represented in unary, whose sum matches a given target positive integer. This problem UK naturally falls in NL (nondeterministic logarithmic space) but he conjectured that UK may not be NL-complete. This conjecture is supported by the fact that even an appropriately designed one-way one-turn nondeterministic counter automaton can recognize UK (e.g., [3]). Jenner [4] introduced a variant of UK whose input integers are given in a "shift-unary" representation, where a *shift-unary representation* $[1^a, 1^b]$ represents the number $a \cdot 2^b$. She then demonstrated that this variant is actually NL-complete. We further expand such a shift-unary representation to a *multiple shift-unary representation* by allowing a finite series of positive integers and to a *general unary (numeral) representation* by taking all possible integers, including zero and negative ones. See Section 2.1 for their precise definitions.

Driven by our great interest in the effect of the unarization of inputs, throughout this work, we wish to study the computational complexity of combinatorial problems whose input integers are in part unarized by the unary representation.

[1] Originally, the strong NP-completeness has been studied in the case where all input integers are *polynomially bounded*. This case is essentially equivalent to the case where all input integers are given in unary. See a discussion in, e.g., [1, Section 4.2].

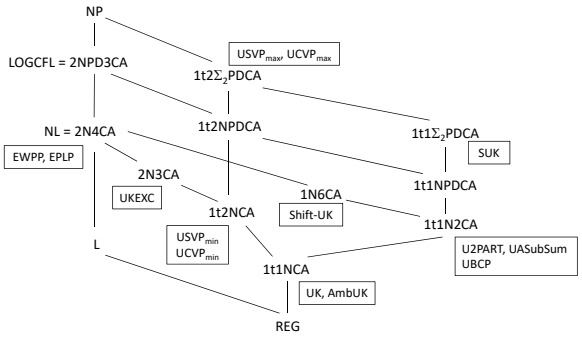

Figure 1: Inclusion relationships among complexity classes with solid lines and membership relations of numerous decision problems listed in small boxes to specific complexity classes.

For the succinctness of further descriptions, we hereafter refer to input integers expressed in the unary numeral system as *unary-form integers* (or more succinctly, *unarized integers*) in comparison to *binary-form integers* (or *binarized integers*).

1.2. New Challenges and Main Contributions

A simple pre-processing of converting a given unary-form input integer into its binary representation provides obvious complexity upper bounds for target combinatorial problems but it does not seem to be sufficient to determine their precise computational complexity.

Our goal is to explore this line of study in order to clarify the roles of the binary-to-unary transformation in the theory of NP-completeness (and beyond it) and cultivate a vast research area incurred by the use of unary-form integers. In particular, we plan to focus on several non-strong NP-complete (or even NP-hard) problems and study how their computational complexities can change when we replace the binary-form input integers to the unary-form ones. Among various types of combinatorial problems taking integer inputs, we look into number problems, graph-related problems, and lattice-related problems. It is desirable for us to determine the precise complexity of each combinatorial problem whose instances are unarized.

A brief summary of our results are illustrated in Figure 1. The detailed explanation of the combinatorial problems and complexity classes listed in this figure will be provided in Sections 2–5. We remark that the underlying machines used to define those complexity classes are all limited to run in polynomial time.

2. Basic Notions and Notation

2.1. Numbers, Sets, Languages

We assume that all *polynomials* have nonnegative integer coefficients. All *logarithms* are always taken to the base 2. The notations $\mathbb{Z}$ and $\mathbb{N}$ denote respectively the set of all integers and that of all nonnegative integers. We further define $\mathbb{N}^+$ to be $\mathbb{N} - \{0\}$. As a succinct notation, we use $[m, n]_{\mathbb{Z}}$ to denote the *integer interval* $\{m, m+1, \ldots, n\}$ for two integers m and n with $m \leq n$. In particular, $[1, n]_{\mathbb{Z}}$ is abbreviated as $[n]$ whenever $n \geq 1$. Given $n \in \mathbb{N}^+$, we define the linear

order $<_{[n]}$ on the set $[n]\times[n]$ as: $(i_1,i_2) <_{[n]} (i_2,j_2)$ iff either $i_1 < i_2$ or $i_1 = i_2 \wedge j_1 < j_2$. Moreover, $\mathbb{R}$ denotes the set of all real numbers. Given a vector $x = (x_1, x_2, \ldots, x_n)$ in $\mathbb{R}^n$, the *Euclidean norm* $\|x\|_2$ of x is given by $(\sum_{i\in[n]} x_i^2)^{1/2}$ and the *max norm* $\|x\|_\infty$ is done by $\max\{|x_i| : i \in [n]\}$, where $|\cdot|$ indicates the absolute value. Given a set A, $P(A)$ denotes the *power set* of A.

Conventionally, we freely identify decision problems with their associated languages. We write 1^* (as a regular expression) for the set of strings of the form 1^n for any $n \in \mathbb{N}$. For convenience, we define 1^0 to be the *empty string* ε. Let 1^+ denote the set $1^* - \{\varepsilon\}$. Similarly, we use the notation 0^* for $\{0^n \mid n \in \mathbb{N}\}$ with $0^0 = \varepsilon$.

Given a positive integer a, the *unary representation* of a is of the form 1^a (viewed as a string) compared to its binary representation. Notice that the length of 1^a is exactly a rather than $O(\log(a+1))$, which is the length of the binary representation of a. A finite series $(a_1, a_2, \ldots, a_n)$ of positive integers is expressed by the *multiple unary representation* of the form $(1^{a_1}, 1^{a_2}, \ldots, 1^{a_n})$. When such an instance $x = (1^{a_1}, 1^{a_2}, \ldots, 1^{a_n})$ is given to a machine, we explicitly assume that x has the form of $1^{a_1}\#1^{a_2}\#\cdots\#1^{a_n}$ with a designated separator symbol #. For a series of such instances $x = (x_1, x_2, \ldots, x_m)$ with $x_i = (1^{a_{i1}}, 1^{a_{i2}}, \ldots, 1^{a_{in_i}})$ for $i \in [m]$ and $n_i \in \mathbb{N}^+$, we further assume that x takes the form of $1^{a_{11}}\#1^{a_{12}}\#\cdots\#1^{a_{1n_1}}\#\#1^{a_{21}}\#1^{a_{22}}\#\cdots\#1^{a_{2n_2}}\#\#\cdots\#\#1^{a_{m1}}\#1^{a_{m2}}\#\cdots\#1^{a_{mn_m}}$. Moreover, for any positive integer of the form $a = p\cdot 2^t$ for nonnegative integers p and t, a *shift-unary representation*[2] of a is a pair $[1^p, 1^t]$, which is different from the unary representation 1^a of a. The length of $[1^p, 1^t]$ is $O(p+t)$ but not a. We also use a *multiple shift-unary representation*, which has the form $[[1^{a_1}, 1^{b_1}], [1^{a_2}, 1^{b_2}], \ldots, [1^{a_n}, 1^{b_n}]]$ with the condition that $2^{b_{i+1}} > a_i 2^{b_i}$ for all $i \in [n-1]$. This form represents the number $\sum_{i=1}^n a_i \cdot 2^{b_i}$. We intend to call an input integer by the name of its representation. For this purpose, we call the expression 1^a and $[1^p, 1^t]$ the *unary-form (positive) integer* (or more succinctly, the *unarized (positive) integer*) and the *shift-unary-form (positive) integer*, respectively.

To deal with "general" integers, including zero and negative integers, however, we intend to express such an integer a as a unary string by applying the following special encoding function $\langle\cdot\rangle$. Let $\langle a\rangle = 1$ if $a = 0$; $\langle a\rangle = 2a$ if $a > 0$; and $\langle a\rangle = 2|a|+1$ if $a < 0$. We define the *general unary (numeral) representation* of a as $1^{\langle a\rangle}$. In a similar way, a *general shift-unary representation* of $-a$ for $a > 0$ is defined as a pair of the form $[1^{\langle -p\rangle}, 1^{\langle t\rangle}]$, where p and t in $\mathbb{N}$ must satisfy $a = p\cdot 2^t$.

2.2. Turing Machines and Log-Space Reductions

Our interest mostly lies on space-bounded computation. As a basic machine model, we thus use deterministic/nondeterministic Turing machines (or DTMs/NTMs, for short), each of which is equipped with a read-only input tape, a rewritable work tape, and (possibly) a write-once[3] output tape.

The notation L (resp., NL) denotes the collection of all decision problems solvable on DTMs (resp., NTMs) in polynomial time using logarithmic space (or log space, for short). A function

[2]Unlike the unary and binary representations, a positive integer in general has more than one shift-unary representation. In most applications, the choice of such representations does not significantly affect the computational resources of running machines.

[3]A tape is called *write-once* if its tape head never moves to the left and, whenever it writes a non-blank symbol, it must move to the next blank cell.

f from Σ^* to Γ^* for two alphabets Σ and Γ is *computable in log space* if there is a DTM such that, on input x, it produces $f(x)$ on a write-once output tape in $|x|^{O(1)}$ time and $O(\log |x|)$ space. The notation FL refers to the class of all such functions.

Let L_1 and L_2 denote two arbitrary languages. We say that L_1 is L-*m-reducible to* L_2 (denoted $L_1 \leq_m^{\mathrm{L}} L_2$) if there exists a function f (called a reduction function) in FL such that, for any x, $x \in L_1$ iff $f(x) \in L_2$. Given a language family F, the notation LOG(F) denotes the family of all languages that are L-m-reducible to appropriately chosen languages in F. We further say that L_1 is L-*tt-reducible to* L_2 (denoted $L_1 \leq_{tt}^{\mathrm{L}} L_2$) if there are a reduction function $f \in$ FL and a truth-table $E : \{0,1\}^* \to \{0,1\}$ in FL such that, for any string x, (i) $x \in L_1$ iff $f(x) = (y_1, y_2, \ldots, y_m)$ with $y_i \in \Sigma^*$ for any index $i \in [m]$ and (ii) $E(L_2(y_1), L_2(y_2), \ldots, L_2(y_m)) = 1$, where $L_2(y) = 1$ if $y \in L_2$ and $L_2(y) = 0$ otherwise.

Before solving a given problem on an input $(1^{a_1}, 1^{a_2}, \ldots, 1^{a_n})$ of unary-form numbers, it is often useful to sort all entries $(a_1, a_2, \ldots, a_n)$ of this input. Let us define the function f_{order} making the following behavior: on input of the form $(1^{a_1}, 1^{a_2}, \ldots, 1^{a_n})$ with $a_1, a_2, \ldots, a_n \in \mathbb{N}^+$, f_{order} produces a tuple $(1^{a_{i_1}}, 1^{a_{i_2}}, \ldots, 1^{a_{i_n}})$ such that (1) $a_{i_1} \geq a_{i_2} \geq \cdots \geq a_{i_n}$ and (2) if $a_{i_j} = a_{i_k}$ with $i_j \neq i_k$, then $i_j < i_k$ holds. Condition (2) is a useful property for one-way finite automata.

There is an occasion where, for a set of shift-unary-form integers $[1^{p_1}, 1^{t_1}], [1^{p_2}, 1^{t_2}], \ldots, [1^{p_n}, 1^{t_n}]$, we wish to compute the sum $s = \sum_{i=1}^{n} p_1 \cdot 2^{t_i}$ and output the binary representation of s in the reverse order. The notation f_{sum} denotes the function that computes this value s. The following lemma is useful.

Lemma 2.1 *The functions f_{order} and f_{sum} are both in* FL.

Those functions will be implicitly used for free when solving combinatorial problems in the subsequent sections.

2.3. Multi-Counter Pushdown Automata

A *one-way determinsitic/nondeterminitic pushdown automaton* (or a 1dpda/1npda, for short) is another computational model with a read-once input tape and a standard (pushdown) stack whose operations are restricted to the topmost cell. A *counter* is a FILO (first in, last out) memory device that behaves like a stack but its alphabet consists only of a "single" symbol, say, 1 except for the bottom marker $\perp$. A *one-way nondeterministic k-counter automaton* (or a k-counter 1ncta) is a one-way nondeterministic finite automaton (1nfa) equipped with k counters. We write 1NkCA to denote the family of all languages recognized by appropriate k-counter 1ncta's running in polynomial time. We further expand 1NkCA to 1NPDkCA by supplementing k counters to 1npda's. These specific machines are called *one-way nondeterministic k-counter pushdown automata* (or 1npdcta's). When tape heads of multi-counter automata and pushdown automata are allowed to move in all directions, we call such polynomial-time machines by 2ncta's and 2npdcta's, respectively. With the use of these 2ncta's and 2npdcta's, we respectively obtain 2NkCA and 2NPDkCA.

An *alternating machine* must use two groups of inner states: existential states and universal states. We are concerned with the number of times that such a machine switches between existential and universal inner states. When this number is upper-bounded by k ($k \geq 0$) along all computation paths of M on any input x, the machine is said to have at most $k+1$ *alternations*.

For any $k \geq 1$, the complexity class $1\Sigma_k$PDCA (resp., $2\Sigma_k$PDCA) is composed of all languages recognized by *one-way* (resp., *two-way*) *alternating 1-counter pushdown automata* running in polynomial time with at most k alternations starting with existential inner states. Note that $1\Sigma_1$PDCA coincides with 1NPDCA.

The notion of *turns* was studied initially by Ginsburg and Spanier [5]. Turn-restricted counter automata were called *reversal bounded* in the past literature. A 1ncta is said to *make a turn* along a certain accepting computation path if the stack height (i.e., the size of stack's content) changes from nondecreasing to decreasing exactly once. A *1-turn 1ncta* is a 1ncta that makes at most one turn during each computation. We add the prefix "1t" to express the restriction that every underlying machine makes at most one turn. For example, we write 1t1NCA when all underlying 1ncta's in the definition of 1NCA are 1-turn 1ncta's. Similarly, we define, e.g., 1t2NPDCA and 1t2Σ_kPDCA by requiring all stacks and counters to make at most one turn. It follows that REG $\subseteq$ 1t1NCA $\subseteq$ 1NCA $\subseteq$ CFL, where REG (resp., CFL) denotes the class of all regular (resp., context-free) languages. Notice that CFL = 1NPD. It also follows that L $\subseteq$ LOG(1t1NCA) $\subseteq$ LOG(1NCA) = NL. Conventionally, we write LOGCFL instead of LOG(CFL).

Lemma 2.2 *For any $k \geq 1$,* 2NkCA $\subseteq$ NL, 2NPDkCA $\subseteq$ LOGCFL, *and* 2Σ_2PDkCA $\subseteq$ NP.

Proof Sketch. For any index $j \in \mathbb{N}^+$, we define $2\Sigma_j$AuxPDATI,SP$(t(n), s(n))$ to be the collection of all decision problems solvable by *two-way alternating auxiliary pushdown automata* running within time $t(n)$ using space at most $s(n)$ with at most j alternations starting with existential inner states. When $j = 1$, such machines are succinctly called *2aux-npda's*, which will be used in the proof of Proposition 2.3. It is known that $2\Sigma_1$AuxPDATI,SP$(n^{O(1)}, O(\log n))$ coincides with LOGCFL [6] and $2\Sigma_2$AuxPDATI,SP$(n^{O(1)}, O(\log n))$ coincides with NP [7]. Moreover, it is possible to simulate the polynomial time-bounded behaviors of k counters using an $O(\log n)$-space bounded auxiliary work tape. From these facts, the lemma follows immediately. □

Proposition *2.3* NL = 2N4CA *and* LOGCFL = 2NPD3CA.

Proof Sketch. Following an argument of Minsky [8], given a log-space NTM (or a log-space 2aux-npda), we first simulate the behavior of an $O(\log n)$-space auxiliary work tape by two stacks whose alphabets are of the form $\{0, 1, \bot\}$. Since the heights of such stacks are upper-bounded by $O(\log n)$, the stacks can be further simulated by multiple counters whose heights are all $n^{O(1)}$-bounded. Let M denote the obtained 2ncta (or 2npdcta) running in polynomial time.

We remark that Minsky's method of reducing the number of counters down to two does not work for machines with few computational resources. Thus, we need to seek other methods. For this purpose, we review the result of [9], which makes it possible to simulate two counters by one counter with the heavy use of an appropriately defined "pairing" function.

Claim 1 *[9] There exists a fixed deterministic procedure by which any single move of push/pop operations on two counters of M can be simulated by a series of $n^{O(1)}$ push/pop operations on one counter with the help of 3 extra counters. These 3 extra counters are emptied after each simulation*

and thus they are reusable for any other purposes. If a stack is further available, then one extra counter can be simulated by this stack.

The recursive applications of this simulation procedure eventually reduce the number of counters down to four. If we freely use a stack during the procedure, then we can further reduce the number of counters down to three.

The first part of the proposition then follows by combining the above claim with Lemma 2.2. If we use one counter as a stack, then we can further reduce 4 counters to 3 counters. This proves the second part of the proposition. □

3. Combinatorial Number Problems

We study the computational complexity of decision problems whose input instances are composed of unarized positive integers.

3.1. Variations of the Unary 0-1 Knapsack Problem

The starting point of our study on the computational analyses of decision problems with unarized integer inputs is the *unary 0-1 knapsack problem*, which was introduced in 1985 by Cook [2] as a unary analogue of the *knapsack problem*.

Unary 0-1 Knapsack Problem (UK):

- Instance: $(1^b, 1^{a_1}, 1^{a_2}, \ldots, 1^{a_n})$, where $n \in \mathbb{N}^+$ and $b, a_1, a_2, \ldots, a_n$ are all positive integers.
- Question: is there a subset S of $[n]$ satisfying $\sum_{i \in S} a_i = b$?

It seems more natural to view UK as a unary analogue of the *subset sum problem*, which is closely related to the *2-partition problem*. Let us consider a unary analogue of this 2-partition problem.

Unary 2 Partition Problem (U2PART):

- Instance: $(1^{a_1}, 1^{a_2}, \ldots, 1^{a_n})$, where $n \in \mathbb{N}^+$ and $a_1, a_2, \ldots, a_n$ are all positive integers.
- Question: is there a subset S of $[n]$ such that $\sum_{i \in S} a_i = \sum_{i \in \overline{S}} a_i$, where $\overline{S} = [n] - S$?

The original knapsack problem and the subset sum problem are both proven in 1972 by Karp [10] to be NP-complete. The problems UK and U2PART, in contrast, situated within NL.

Lemma 3.1 *(1)* UK *is in* 1t1NCA. *(2)* U2PART *is in both* 1NCA *and* 1t1N2CA.

We further introduce two variants of UK and U2PART, called AmbUK and UASubSum as follows.

Ambiguous UK Problem (AmbUK):

- Instance: $((1^{b_1}, 1^{b_2}, \ldots, 1^{b_m}), (1^{a_1}, 1^{a_2}, \ldots, 1^{a_n}))$, where $n, m \in \mathbb{N}^+$ and $b_1, b_2, \ldots, b_m, a_1, a_2, \ldots, a_n$ are all positive integers.
- Question: are there an index $j \in [m]$ and a subset S of $[n]$ satisfying $b_j = \sum_{i \in S} a_i$?

Unary Approximate Subset Sum Problem (UASubSum):

- Instance: $((1^{b_1}, 1^{b_2}), (1^{a_1}, 1^{a_2}, \ldots, 1^{a_n}))$, where $n \in \mathbb{N}^+$ and $b_1, b_2, a_1, a_2, \ldots, a_n$ are positive integers.
- Question: is there a subset S of $[n]$ such that $b_1 \leq \sum_{i \in S} a_i \leq b_2$?

Lemma 3.2 *(1)* AmbUK *is in* 1t1NCA. *(2)* UASubSum *is in* 1t1N2CA.

Under two different types of log-space reductions, the computational complexities of U2PART, AmbUK, and UASubSum are all equal to that of UK.

Proposition *3.3* *(1)* UK $\equiv^{\mathrm{L}}_{m}$ U2PART. *(2)* UK $\equiv^{\mathrm{L}}_{tt}$ AmbUK. *(3)* UK $\equiv^{\mathrm{L}}_{m}$ UASubSum.

Proof Sketch. (1) We begin with showing that UK $\equiv^{\mathrm{L}}_{m}$ U2PART. Our first goal is to prove that UK $\leq^{\mathrm{L}}_{m}$ U2PART by constructing an L-m-reduction function f. Let $x = (1^b, 1^{a_1}, \ldots, 1^{a_n})$ and let $c = \sum_{i \in [n]} a_i$. (a) If $b = c/2$, then we set $f(x) = x$. (b) If $b < c/2$, then we add a new entry $a_{n+1} = c - 2b$ and set $f(x) = (1^{a_1}, \ldots, 1^{a_n}, 1^{a_{n+1}})$. (c) If $b > c/2$, then we exchange between b and $c - b$ and apply (b). Our second goal is to show that U2PART $\leq^{\mathrm{L}}_{m}$ UK. Letting $c = \sum_i a_i$, if c is odd, then we set $f(x)$ to be any fixed rejected input. Otherwise, we set $b = c/2$ and define $f(x) = (1^b, 1^{a_1}, \ldots, 1^{a_n})$.

(2) It is obvious that UK $\leq^{\mathrm{L}}_{m}$ AmbUK by setting $m = 1$ in the definition of AmbUK. Next, we show that AmbUK $\leq^{\mathrm{L}}_{tt}$ UK. Let $w = ((1^{b_1}, 1^{b_2}, \ldots, 1^{b_m}), (1^{a_1}, 1^{a_2}, \ldots, 1^{a_n}))$ be any instance of AmbUK. We define an L-tt-reduction function f as $f(w) = (f_1(w), f_2(w), \ldots, f_m(w))$, where $f_i(w) = (1^{b_i}, 1^{a_1}, \ldots, 1^{a_n})$. Clearly, f belongs to FL. We also define a truth table E as $E(e_1, e_2, \ldots, e_m) = \bigvee_{i=1}^{m} e_i$. It then follows that $w \in$ AmbUK iff there exists an index $i \in [m]$ for which $f_i(w) \in$ UK iff $E(f_1(w), f_2(w), \ldots, f_m(w)) = 1$. Therefore, AmbUK $\leq^{\mathrm{L}}_{tt}$ UK follows.

(3) To show that UK $\leq^{\mathrm{L}}_{m}$ UASubSum, it suffices to set $b_1 = b_2 = b$ and define the desired reduction function f to map $(1^b, 1^{a_1}, \ldots, 1^{a_n})$ to $((1^{b_1}, 1^{b_2}), (1^{a_1}, \ldots, 1^{a_n}))$. On the contrary, we wish to verify that UASubSum $\leq^{\mathrm{L}}_{m}$ UK. Let $w = ((1^{b_1}, 1^{b_2}), (1^{a_1}, \ldots, 1^{a_n}))$ be any input to UASubSum. If $b_1 = b_2$, then the reduction is trivial. In the case of $b_1 > b_2$, it suffices to define $f(w) = (1, 1^2)$ since $w \notin$ UASubSum. In what follows, we assume that $b_1 < b_2$. For any $S \subseteq [n]$, we define $a_S = \sum_{i \in S} a_i$. If $a_{[n]} < b_1$, then we construct $f(w) = (1, 1^2)$, which is obviously not in UASubSum. If $b_1 \leq a_{[n]} \leq b_2$, then we define $x = (1, 1)$. Next, we assume that $b_2 < a_{[n]}$. Let us define $a_{n+i} = a_{[n]} + i - 1$ for any $i \in [b_2 - b_1 + 1]$ and let $b_{max} = b_2 + a_{[n]}$. We then set $f(w) = (1^{b_{max}}, 1^{a_1}, 1^{a_2}, \ldots, 1^{a_n}, 1^{a_{n+1}}, 1^{a_{n+2}}, \ldots, 1^{a_{n+b_2-b_1+1}})$. This concludes that $w \in$ UASubSum iff $f(w) \in$ UK. □

Jenner [4] studied a variant of UK, which we intend to call by the *shift-unary 0-1 knapsack problem* (shift-UK) in order to signify the use of the shift-unary representation. She proved that this problem is actually NL-complete.

Shift-Unary 0-1 Knapsack Problem (shift-UK):

- Instance: $[1^q, 1^s]$ and a series $([1^{p_1}, 1^{t_1}], [1^{p_2}, 1^{t_2}], \ldots, [1^{p_n}, 1^{t_n}])$ of nonnegative integers represented in shift-unary, where $q, p_1, p_2, \ldots, p_n$ are all positive integers and $s, t_1, t_2, \ldots, t_n$ are all nonnegative integers.
- Question: is there a subset S of $[n]$ such that $\sum_{i \in S} p_i 2^{t_i} = q2^s$?

In a similar way, we can define the shift-unary representation versions of AmbUK, UASubSum, and U2PART, denoted by shift-AmbUK, shift-UASubSum, and shift-U2PART, respectively.

Lemma 3.4 *The problem* shift-UK *is in* 1N6CA.

Proposition *3.5* shift-UK $\equiv_m^{\mathrm{L}}$ shift-U2PART $\equiv_m^{\mathrm{L}}$ shift-AmbUK $\equiv_m^{\mathrm{L}}$ shift-UASubSum.

Proof Sketch. We abbreviate the set {shift-U2PART, shift-AmbUK, shift-UASubSum} as S. It is easy to obtain, by modifying the proof of Proposition 3.3, that shift-UK $\leq_m^{\mathrm{L}} C$ for any $C \in S$. To show that $C \leq_m^{\mathrm{L}}$ shift-UK for any $C \in S$, it suffices to show that C belongs to NL because the L-m-completeness of shift-UK [4] guarantees that $C \leq_m^{\mathrm{L}}$ shift-UK. In the case of $C =$ shift-UASubSum, we consider the following algorithm. On input of the form $(([1^{q_1}, 1^{s_1}], [1^{q_2}, 1^{s_2}]), ([1^{p_1}, 1^{t_1}], \ldots, [1^{p_n}, 1^{t_n}]))$, we nondeterministically choose a number $k \in [t]$ and indices $i_1, i_2, \ldots, i_k \in [n]$ and check that $q_1 2^{s_1} \leq \sum_{i \in S} p_i 2^{t_i} \leq q_2 2^{s_2}$. Note that, by Lemma 2.1, the sum $\sum_{i \in S} p_i 2^{t_i}$ can be computed using only log space. Hence, C is in NL. The other cases can be similarly handled. □

From Proposition 3.5, since shift-UK is NL-complete under L-m-reductions [4], we immediately obtain the following corollary.

Corollary 3.6 *The problems* shift-U2PART, shift-AmbUK, *and* shift-UASubSum *are all* NL*-complete.*

For later references, we introduce another variant of UK. This variant requires a prohibition of certain successive choices of unarized integer inputs.

UK with Exception (UKEXC):

- Instance: $(1^b, 1^{a_1}, 1^{a_2}, \ldots, 1^{a_n})$ and $EXC \subseteq \{(i, j) \mid i, j \in [n], i < j\}$ given as a set of pairs $(1^i, 1^j)$, where $n \in \mathbb{N}^+$ and $b, a_1, a_2, \ldots, a_n$ are in $\mathbb{N}^+$.
- Question: is there a subset $S = \{i_1, i_2, \ldots, i_k\}$ of $[n]$ with $k \in \mathbb{N}^+$ and $i_1 < i_2 < \cdots < i_k$ such that (i) $\sum_{i \in S} a_i = b$ and (ii) no $j \in [k-1]$ satisfies $(i_j, i_{j+1}) \in EXC$?

Note that, when $EXC = \emptyset$, UKEXC is equivalent to UK.

Lemma 3.7 UKEXC *is in* 2N3CA.

Proof Sketch. We nondeterministically choose 1^{a_i} by reading an input from left to right. We use two counters to remember the index i (in the form of 1^i) for checking that the next possible choice, say, 1^{a_j} satisfies $(i, j) \notin EXC$. The third counter is used to store 1^b and then sequentially pop 1^{a_i} for the chosen indices i. □

We also introduce another variant of UK, which concerns simultaneous handling of input integers.

Simultaneous Unary 0-1 Knapsack Problem (SUK):

- Instance: $(1^{b_1}, 1^{a_{11}}, 1^{a_{12}}, \ldots, 1^{a_{1n}}), \ldots, (1^{b_m}, 1^{a_{m1}}, 1^{a_{m2}}, \ldots, 1^{a_{mn}})$, where $m, n \in \mathbb{N}^+$ and b_i and a_{ij} ($i \in [n], j \in [m]$) are all positive integers.

- QUESTION: is there a subset $S \subseteq [n]$ such that $\sum_{j \in S} a_{ij} = b_i$ for all indices $i \in [m]$?

The complexity class 1t1Σ_2PDCA is the one-way restriction of 1t2Σ_2PDCA.

Lemma 3.8 *The problem* SUK *is in* 1t1Σ_2PDCA.

Proof Sketch. To recognize SUK, let us consider a *one-way alternating counter pushdown automaton* (or a 1apdcta) that behaves as follows. Given an input x of the form $(1^{b_1}, 1^{a_{11}}, 1^{a_{12}}, \dots, 1^{a_{1n}}), \dots, (1^{b_m}, 1^{a_{m1}}, 1^{a_{m2}}, \dots, 1^{a_{mn}})$, we call each segment $(1^{b_i}, 1^{a_{i1}}, 1^{a_{i2}}, \dots, 1^{a_{in}})$ of x by the *ith block* of x.

In existential inner states, we first choose a string $w \in \{0,1\}^*$ and push it into a stack, where $w = e_1 e_2 \cdots e_n$ indicates that, for any $j \in [n]$, we select the jth entry from every block exactly when $e_j = 1$. Let $A_w = \{j \in [n] \mid e_j = 1\}$. In universal inner states, we then check whether $b_i = \sum_{j \in A_w} a_{ij}$ holds for all $i \in [m]$. This last procedure is achieved by first storing 1^{b_i} into a counter. As popping the values e_j one by one from the stack, if $j \in A_w$, then we decrease the counter by a_{ij}. Otherwise, we do nothing. When either the stack gets empty or the assigned block $(1^{a_{i1}}, \dots, 1^{a_{in}})$ of x is finished, if the stack is empty and the counter becomes 0, then we accept x; otherwise, we reject x. Note that the stack and the counter make only 1-turns and the input-tape head moves only in one direction. □

3.2. Unary Bounded Correspondence Problem

We turn our attention to the *bounded Post correspondence problem* (BPCP), which is a well-known problem of determining whether, given a set $\{(a_i, b_i)\}_{i \in [n]}$ of binary string pairs and a number $k \geq 1$, a certain sequence $(i_1, i_2, \dots, i_t)$ of elements in $[n]$ with $t \leq k$ satisfies $a_{i_1} a_{i_2} \cdots a_{i_t} = b_{i_1} b_{i_2} \cdots b_{i_t}$. This problem is known to be NP-complete [11]. When we replace binary strings a_i and b_i by unary strings, we obtain the following "unary" variant of PCP.

UNARY BOUNDED CORRESPONDENCE PROBLEM (UBCP):

- INSTANCE: $((a_1, b_1), (a_2, b_2), \dots, (a_n, b_n))$ for unary strings $a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_n \in 1^+$ and 1^k for $k \in \mathbb{N}^+$.
- QUESTION: is there a sequence $(i_1, i_2, \dots, i_t)$ of elements in $[n]$ with $t \in [k]$ satisfying $a_{i_1} a_{i_2} \cdots a_{i_t} = b_{i_1} b_{i_2} \cdots b_{i_t}$?

Since a_i and b_i are unary strings, the above requirement $a_{i_1} \cdots a_{i_t} = b_{i_1} \cdots b_{i_t}$ is equivalent to $\sum_{j \in S} |a_j| = \sum_{j \in S} |b_j|$, where $|a_j|$ and $|b_j|$ mean the lengths of strings a_j and b_j, respectively.

Lemma 3.9 UBCP *belongs to* 1t1N2CA.

Proposition *3.10* UK $\leq_m^{\mathrm{L}}$ UBCP $\leq_m^{\mathrm{L}}$ AmbUK.

Proof Sketch. Here, we only show the last reduction UBCP $\leq_m^{\mathrm{L}}$ AmbUK. Let $w = ((a_1, b_1), \dots, (a_n, b_n))$ be any instance to UBCP. Assume that, for any $i \in [n]$, $a_i = 1^{k_i}$ and $b_i = 1^{l_i}$ for certain numbers $k_i, l_i \in \mathbb{N}^+$. Let $m = \max\{\sum_{i \in [n]} k_i, \sum_{i \in [n]} l_i\}$. Note that $|S| \leq \sum_{i \in S} k_i \leq m$ and $|S| \leq \sum_{i \in S} l_i \leq m$ for any $S \subseteq [n]$. We then set $c_i = m - (k_i - l_i)$ for each $i \in [n]$. We remark that $\sum_{i \in S} c_i = m|S| - (\sum_{i \in S} k_i - \sum_{i \in S} l_i) \geq m|S| - (m - |S|) = m(|S| - 1) + |S| \geq 0$ for any nonempty

subset S of $[n]$. If $w \in$ UBCP, then there is a nonempty set $S \subseteq [n]$ such that $\sum_{i \in S} k_i = \sum_{i \in S} l_i$, that is, $\sum_{i \in S}(k_i - l_i) = 0$. It then follows that $\sum_{i \in S} c_i = m|S|$.

We also check if $\sum_{i \in [n]} k_i = \sum_{i \in [n]} l_i$ or if $k_{i_0} = l_{i_0}$ for a certain index $i_0 \in [n]$. If this is true, then we know that $S = [n]$ or $\{i_0\}$. Otherwise, we set $d_i = i \cdot m$ for each $i \in [2, n-1]_{\mathbb{Z}}$ and then define $u = ((1^{d_2}, 1^{d_3}, \ldots, 1^{d_{n-1}}), (1^{c_1}, 1^{c_2}, \ldots, 1^{c_n}))$. It follows that $w \in$ UBCP iff $u \in$ AmbUK. □

Corollary 3.11 UK $\equiv_{tt}^{\mathrm{L}}$ UBCP.

4. Graph Problems

We look into decision problems that are related to graphs, in particular, weighted graphs in which either vertices or edges are labeled with "weights". Here, we study the computational complexity of these specific problems.

We begin with edge-weighted path problems, in which we search for a simple path whose weight matches a target unarized number, which is given in unary. We consider, in particular, *directed acyclic graphs* (or dags) whose edges are further labeled with weights, which are given in unary.

For the purpose of this work, a dag $G = (V, E)$ given as part of inputs to an underlying machine is assumed to satisfy that V is a subset of $\mathbb{N}^+$, e.g., $V = \{i_1, i_2, \ldots, i_n\}$ for $i_1, i_2, \ldots, i_n \in \mathbb{N}^+$. We also use the following specific encoding of G. Given a vertex i, we write $E[i]$ for the set of all adjacent vertices entering from i, namely, $\{j \in V \mid (i, j) \in E\}$. When $E[i]$ equals $\{j_1, j_2, \ldots, j_n\}$ with $j_1 < j_2 < \cdots < j_n$, we express it in the "multi-unary" form of $(1^i, 1^{j_1}, 1^{j_2}, \ldots, 1^{j_n})$. The *unary adjacency list representation* UAL_G of G is of the form $((1^{i_1}, 1^{j_{i_1,1}}, 1^{j_{i_1,2}}, \ldots, 1^{j_{i_1,k_1}}), \ldots, (1^{i_n}, 1^{j_{i_n,1}}, 1^{j_{i_n,2}}, \ldots, 1^{j_{i_n,k_n}}))$ with $k_i \in \mathbb{N}$ if $V = \{i_1, i_2, \ldots, i_n\}$ with $i_1 < i_2 < \cdots < i_n$ and $E[i_s] = \{j_{i_s,1}, j_{i_s,2}, \ldots, j_{i_s,k_s}\}$ with $j_{i_s,1} < j_{i_s,2} < \cdots < j_{i_s,k_s}$ for any $s \in [n]$.

Edge-Weighted Path Problem (EWPP)

- Instance: a dag $G = (V, E)$ with $V \subseteq \mathbb{N}^+$ given in the form of UAL_G, a vertex $s \in V$ given as 1^s, edge weights $w(i, j) \in \mathbb{N}^+$ given as $1^{w(i,j)}$ for all edges $(i, j) \in E$, and 1^c with $c \in \mathbb{N}^+$, provided that edges are enumerated according to the linear order $<_{[n]}$.
- Question: is there a vertex $v \in V$ such that the total edge weight of a path from s to v equals c?

In comparison, the *graph connectivity problem* for directed "unweighted" graphs (DSTCON) is known to be NL-complete [12].

Lemma 4.1 EWPP *is in* NL.

When each edge weight is 1, the total weight of a path is the same as the length of the path. This fact makes us introduce another decision problem. A *sink* of a directed graph is a vertex of outdegree 0 in the graph.

Exact Path Length Problem (EPLP)

- Instance: a dag $G = (V, E)$ with $V \subseteq \mathbb{N}^+$ given as UAL_G, a vertex $s \in V$ given as 1^s, and 1^c with $c \in \mathbb{N}^+$.

◦ QUESTION: is there a sink v of G such that a path from s to v has length exactly c?

Proposition 4.2 EWPP $\equiv_m^{\mathrm{L}}$ EPLP.

Proof Sketch. Firstly, we show that EPLP $\leq_m^{\mathrm{L}}$ EWPP. Given an instance $x = (G, s, 1^c)$ of EPLP with $G = (V, E)$, we define another instance, say, y of EWPP as follows. For every leaf v of G, we add a new vertex $\bar{v}$ and a new edge $(v, \bar{v})$ to G and obtain the new vertex set V' and the new edge set E'. Consider the resulting graph $G' = (V', E')$. Let $n = |V'|$. We define $w(u, v) = 1$ for all pairs $(u, v) \in E$ and additionally we set $w(v, \bar{v}) = n + 1$ for any old leaf $v \in V$. Let $c' = c + n + 1$. We define $y = (G', \{w(e)\}_{e \in E'}, 1^{c'})$. We want to verify that (*) $x \in$ EPLP iff $y \in$ EWPP.

To prove (*), let us assume that $x \in$ EPLP. We then take a length-c path $\gamma = (v_0, v_1, \ldots, v_c)$ of G with $v_0 = s$ and v_c is a leaf. Consider y and $\gamma^{(+)} = (v_0, v_1, \ldots, v_c, \bar{v}_c)$. The total weight of $\gamma^{(+)}$ in G' is $c + w(v_c, \bar{v}_c) = c + n + 1 = c'$. Hence, y is in EWPP. Conversely, assume that $y \in$ EWPP and consider a path $\gamma = (v_0, v_1, \ldots, v_k)$ of G' with $v_0 = s$ satisfying $\sum_{i=0}^{k} w(v_i, v_{i+1}) = c'$. Since $n = |V'|$, γ must include a leaf. Hence, v_k must be $\bar{v}_{k-1}$. Moreover, $k - 1 = c$ follows. The path $\gamma^{(-)} = (v_0, v_1, \ldots, v_{k-1})$ is a path from s to a leaf of G and its length is exactly c. This implies that $x \in$ EPLP.

Secondly, we show that EWPP $\leq_m^{\mathrm{L}}$ EPLP. Let $x = (G, s, \{w(e)\}_{e \in E}, 1^c)$ be any instance of EWPP. We construct an instance of EPLP as follows. For each $(u, v) \in E$ with $w(u, v) = 1^k$, we introduce $k + 1$ extra vertices $\{\bar{v}, u'_1, u'_2, \ldots, u'_k\}$ and extra edges $\{(u, u'_1), (u'_1, u'_2), \ldots, (u'_i, u'_{i+1}), (u'_k, v), (u'_k, \bar{v}) \mid i \in [k-2]\}$. Those edges form two paths from u to v and u to $\bar{v}$ of length exactly k. Notice that $\bar{v}$ becomes a new leaf. Let V' and E' denote the extended sets of V and E, respectively, and set $G' = (V', E')$. For the instance $y = (G', s, 1^c)$, it is possible to prove that $x \in$ EWPP iff $y \in$ EPLP. □

We then obtain the following NL-completeness result.

Proposition 4.3 EWPP *and* EPLP *are both* L*-m-complete for* NL.

Proof Sketch. Here, we wish to prove (*) $L \leq_m^{\mathrm{L}}$ EPLP for any language L in 1NCA. If this is true, then all languages in NL are L-m-reducible to EPLP since LOG(1NCA) = NL. Moreover, since EWPP belongs to NL by Lemma 4.1, EPLP is also in NL. Therefore, EPLP is L-m-complete for NL. Since EWPP $\equiv_m^{\mathrm{L}}$ EPLP by Proposition 4.2, we also obtain the NL-completeness of EWPP.

To show the statement (*), let us take any 1ncta M and consider surface configurations of M on input x. Note that, since surface configurations have size $O(\log |x|)$, we can express them as unary-form positive integers. Between two surface configurations, we define a single-step transition relation $\vdash_M$. We then define a computation graph $G_x = (V_x, E_x)$ of M on the input x. The vertex set V_x is composed of all possible surface configurations of M on x. We further define E_x to be the set of all pairs (v_1, v_2) of surface configurations satisfying $v_1 \vdash_M v_2$. The root s of G_x is set to be the initial surface configuration of M. It then follows that $x \in L(M)$ iff $(UAL_{G_x}, 1^s, 1^{|x|+2}) \in$ EPLP. □

Let us argue how EWPP is related to UKEXC and UK. To see the desired relationships, we introduce two restricted variants of EWPP. We say that a dag $G = (V, E)$ with $V \subseteq \mathbb{N}^+$ is

topologically ordered if, for any two vertices $i, j \in V$, $(i, j) \in E$ implies $i < j$. We define TO-EWPP as the restriction of EWPP onto instances that are topologically ordered. A dag $G = (V, E)$ is *edge-closed* if, for any three vertices $u, v, w \in V$, (1) $(u, v) \in E$ and $(v, w) \in E$ imply $(u, w) \in E$ and (2) $(u, w) \in E$ and $(v, w) \in E$ imply either $(u, v) \in E$ or $(v, u) \in E$. We write EC-EWPP for TO-EWPP whose instance graphs are all edge-closed.

Proposition 4.4 *(1)* TO-EWPP $\equiv_m^{\mathrm{L}}$ UKEXC. *(2)* EC-EWPP $\equiv_m^{\mathrm{L}}$ UK.

Proof Sketch. In what follows, we focus only on (1). We first show that UKEXC $\leq_m^{\mathrm{L}}$ TO-EWPP. Given an instance $x = ((1^b, 1^{a_1}, \ldots, 1^{a_n}), EXC)$ of UKEXC, we construct an associated instance, say, y of TO-EWPP as follows. We then define $V = \{s, v_1, v_2, \ldots, v_n\}$ and $E = \{(v_i, v_j) \mid i, j \in [n], i < j, (i, j) \notin EXC\} \cup \{(s, v_j) \mid j \in [n]\}$. It follows that $G = (V, E)$ is directed, acyclic, and topologically ordered. Furthermore, we define edge weights as $w(v_i, v_j) = a_i$ and $w(s, v_j) = 1$ for any $(v_i, v_j), (s, v_j) \in E$. Finally, we set $c = b + 1$. Let us consider $y = (G, \{1^{w(e)}\}_{e \in E}, 1^c)$. It is possible for us to verify that $x \in$ UKEXC iff $y \in$ TO-EWPP. This implies UKEXC $\leq_m^{\mathrm{L}}$ TO-EWPP.

Conversely, we intend to verify that TO-EWPP $\leq_m^{\mathrm{L}}$ UKEXC. Assume that $x = (G, s, \{1^{w(e)}\}_{e \in E}, 1^c)$ with $G = (V, E)$ is any instance of TO-EWPP. An associated instance y of UKEXC is constructed as follows. Firstly, we assume that G is acyclic because, otherwise, we convert G to an acyclic graph $G' = (V', E')$ by setting $V' = \{(i, v) \mid i \in [|V|], v \in V\}$ and $E' = \{((i, u), (i + 1, v)) \mid i \in [|V| - 1], (u, v) \in E\}$. For simplicity, we further assume that $V = [2, n]_{\mathbb{Z}}$ and s is vertex 2. We define an encoding $\langle i, j \rangle$ of two distinct vertices $i, j \in [n]$ as $\langle i, j \rangle = (i - 1)n + j$. We consider only a unique connected component including s because any other connected component is irrelevant. Hereafter, we assume that this connected component contains all vertices in V. We expand G to $G' = (V', E')$ by setting $V' = V \cup \{1\}$ and $E' = E \cup \{(1, j) \mid j \in [2, n]_{\mathbb{Z}}\}$. As for new weights, we set $w'(i, j) = w(i, j)$ for any $(i, j) \in E$ and $w'(1, j) = w_* + 1$ for any $j \in [2, n]_{\mathbb{Z}}$, where $w_* = \sum_{(i,j) \in E} w(i, j)$.

We further set $a_{\langle i,j \rangle} = w(i, j)$ if $(i, j) \in E$. Moreover, let $a_{\langle 1,j \rangle} = w_* + 1$ for any $j \in [2, n]_{\mathbb{Z}}$. For any other pair (i, j), the value $a_{\langle i,j \rangle}$ is not defined. Let A denote the set of all $a_{\langle i,j \rangle}$'s that are defined. Assume that A has the form $\{a_{k_1}, a_{k_2}, \ldots, a_{k_m}\}$ with $m \leq n^2$, where $k_1 < k_2 < \cdots < k_m$. We write K for the set $\{k_1, k_2, \ldots, k_m\}$. The set EXC is defined to be the union of the following sets: $\{(\langle i, j \rangle, \langle i', j' \rangle) \mid i, j, i', j' \in [n], (\langle i, j \rangle \not< \langle i', j' \rangle \vee j \neq i' \vee i = j'), (i, v) \in E, (i', j') \in E\}$, $\{(\langle i, j \rangle, \langle i', j' \rangle) \mid i, i', j, j' \in [n], \langle i, j \rangle < \langle i', j' \rangle, ((i, j) \notin E \vee (i', j') \notin E)\}$, and $\{(\langle i, j \rangle, \langle 1, j' \rangle) \mid i, j, j' \in [2, n]_{\mathbb{Z}}\}$. Let $b = c + w_* + 1$.

We define the desired instance y as $y = ((1^b, 1^{a_{k_1}}, 1^{a_{k_2}}, \ldots, 1^{a_{k_m}}), EXC)$. For this instance y, we can prove that $x \in$ TO-EWPP iff $y \in$ UKEXC. □

5. Lattice Problems

Let us discuss lattice problems. A *lattice* is a set of integer linear combinations of basis vectors. Here, we deal only with the case where bases are taken from $\mathbb{Z}^n$ for a certain $n \in \mathbb{N}^+$. The decision version of the *closest vector problem* (CVP) is known to be NP-complete [13]. We then consider a variant of CVP. To fit into our setting of unary-form integers, a simple norm

notion of the *max norm* $\|\cdot\|_\infty$ from Section 2.1. In a syntactic similarity, we further introduce the notation $\|v\|_{\min}$ to express $\min\{|v_i| \,:\, i \in [n]\}$ for any real-valued vector $v = (v_1, v_2, \ldots, v_n)$. Notice that $\|v\|_{\min}$ does not serve as a true "distance". For convenience, $L(v_1, v_2, \ldots, v_m)$ denotes the lattice spanned by a given set $\{v_1, v_2, \ldots, v_m\}$ of basis vectors.

Unary Max-Norm Closest Vector Problem ($\mathrm{UCVP}_{\max}$):

- Instance: 1^b for a positive integer b, a tuple $(1^{\langle v_1[1]\rangle}, 1^{\langle v_1[2]\rangle}, \ldots, 1^{\langle v_1[n]\rangle})$ for a set $\{v_1, v_2, \ldots, v_m\}$ of lattice bases with $v_i = (v_i[1], v_i[2], \ldots, v_i[n]) \in \mathbb{Z}^n$, and a tuple $(1^{\langle x_0[1]\rangle}, 1^{\langle x_0[2]\rangle}, \ldots, 1^{\langle x_0[n]\rangle})$ for a target vector $x_0 = (x_0[1], x_0[2], \ldots, x_0[n]) \in \mathbb{Z}^n$.
- Question: is there a lattice vector w in $L(v_1, v_2, \ldots, v_m)$ such that the max norm $\|w - x_0\|_\infty$ is at most b?

In the above definition of $\mathrm{UCVP}_{\max}$, we replace $\|\cdot\|_\infty$ by $\|\cdot\|_{min}$ and then obtain $\mathrm{UCVP}_{\min}$.

For simplicity, in what follows, we intend to write $\bar{v}$ for $(1^{\langle v[1]\rangle}, 1^{\langle v[2]\rangle}, \ldots, 1^{\langle v[n]\rangle})$ if $v = (v[1], v[2], \ldots, v[n])$.

Proposition *5.1* $\mathrm{SUK} \leq^{\mathrm{L}}_m \mathrm{UCVP}_{\max}$.

Proof Sketch. In a similar way to obtaining SUK from UK, we can introduce a variant of U2PART, called the *simultaneous unary 2 partition problem* (SU2PART). It is possible to prove that $\mathrm{SUK} \leq^{\mathrm{L}}_m \mathrm{SU2PART}$. It therefore suffices to verify that $\mathrm{SU2PART} \leq^{\mathrm{L}}_m \mathrm{UCVP}_{\max}$.

Let $x = (a_1, a_2, \ldots, a_m)$ with $a_j = (1^{a_{j1}}, 1^{a_{j2}}, \ldots, 1^{a_{jn}})$ for any $j \in [m]$ be any instance of SU2PART. Let $d_j = \sum_{i\in[n]} a_{ji}$ for each $j \in [m]$ and set $d_{max} = \max_{j\in[m]}\{d_j\}$. Given any $j \in [m]$ and $i \in [n]$, we further set $a'_{ji} = d_{max}a_{ji}$ and $d'_j = d_{max}d_j$. Let us define n vectors $v_1, v_2, \ldots, v_n$ as $v_1 = (a'_{11}, a'_{21}, \ldots, a'_{m1}, 2, 0, 0, \ldots, 0)$, $v_2 = (a'_{21}, a'_{22}, \ldots, a'_{2n}, 0, 2, 0, \ldots, 0)$, ..., $v_n = (a'_{n1}, a'_{n2}, \ldots, a'_{nm}, 0, 0, 0, \ldots, 0, 2)$. Moreover, we set $x_0 = (\lfloor d'_1/2\rfloor, \lfloor d'_2/2\rfloor, \ldots, \lfloor d'_m/2\rfloor, 1, 1, \ldots, 1)$ and $b = 1$. We then define y to be $(1^b, \bar{v}_1, \bar{v}_2, \ldots, \bar{v}_n, \bar{x}_0)$. Clearly, y is an instance of $\mathrm{UCSP}_{\max}$. It then follows that $x \in \mathrm{U2PART}$ iff $y \in \mathrm{UCVC}_{\max}$. □

Lemma 5.2 *(1)* $\mathrm{UCVP}_{\max} \in \mathrm{1t2}\Sigma_2\mathrm{PDCA}$. *(2)* $\mathrm{UCVP}_{\min} \in \mathrm{1t2NCA}$.

It is not clear that $\mathrm{UCVP}_{\max}$ belongs to 1t2NPDCA or even P.

Next, we look into another relevant problem, known as the *shortest vector problem*. We consider its variant.

Unary Max-Norm Shortest Vector Problem ($\mathrm{USVP}_{\max}$):

- Instance: 1^b with $b \in \mathbb{N}^+$ and a tuple $(1^{\langle v_1[1]\rangle}, 1^{\langle v_1[2]\rangle}, \ldots, 1^{\langle v_1[n]\rangle})$ for a set $\{v_1, v_2, \ldots, v_m\}$ of lattice bases with $v_i = (v_i[1], v_i[2], \ldots, v_i[n]) \in \mathbb{Z}^n$.
- Question: is there a "non-zero" lattice vector w in $L(v_1, v_2, \ldots, v_m)$ such that the max norm $\|w\|_\infty$ is at most b?

Similarly, we can define $\mathrm{USVP}_{\min}$ by replacing $\|\cdot\|_\infty$ with $\|\cdot\|_{\min}$. In a way similar to prove Lemma 5.2, we can prove that $\mathrm{USVP}_{\max} \in \mathrm{1t2}\Sigma_2\mathrm{PDCA}$. Moreover, the following relations hold.

Lemma 5.3 $\mathrm{USVP}_{\min} \leq^{\mathrm{L}}_{tt} \mathrm{UCVP}_{\min}$ *and* $\mathrm{USVP}_{\max} \leq^{\mathrm{L}}_{tt} \mathrm{UCVP}_{\max}$.

We do not know if $\mathrm{USVP}_{\min} \equiv^{\mathrm{L}}_{tt} \mathrm{UCVP}_{\min}$ and $\mathrm{USVP}_{\max} \equiv^{\mathrm{L}}_{tt} \mathrm{UCVP}_{\max}$.

References

[1] M. R. Garey, D. S. Johnson, Computers and Interactability: A Guide to the Theory of NP-Completeness, W. H. Freeman and Company, New York, NY, 1979.

[2] S. A. Cook, A taxonomy of problems with fast parallel algorithms, Information and Control 64 (1985) 2–22.

[3] S. Cho, D. T. Huynh, On a complexity hierarchy between L and NL, Information Processing Letters 29 (1988) 177–182.

[4] B. Jenner, Knapsack problems for NL, Information Processing Letters 54 (1995) 169–174.

[5] G. Ginsburg, E. H. Spanier, Finite-turn pushdown automata, SIAM Journal on Control 4 (1966) 429–453.

[6] I. H. Sudborough, On the tape complexity of determinsitic context-free languages, Journal of ACM 25 (1978) 405–414.

[7] B. Jenner, B. Kirsg, Characterizing the polynomial hierarchy by alternating pushdown automata, RAIRO–Theoretical Informatics and Applications 23 (1989) 87–99.

[8] M. L. Minsky, Computation: Finite and Infinite Machines, Prentice-Hall, Englewood Cliffs, NJ, 1967.

[9] T. Yamakami, Strengths and weaknesses of nonuniform families of multi-counter pushdown automata of polynomial state-stack complexity, Manuscript, 2023. Submitted to an international conference.

[10] R. M. Karp, Reducibility among combinatorial problems, in: R. E. Miller, J. W. T. (eds.) (Eds.), Complexity of Computer Computations, Plenum Press, New York, NY, 1972, pp. 85–103.

[11] R. L. Constable, H. B. H. III, S. Sahni, On the computational complexity of scheme equivalence, Technical Report No. 74-201, Depaerment of Computer Science, Cornell University, 1974.

[12] N. D. Jones, Y. E. Lien, W. T. Laaser, New problems complete for nondeterministic log space, Mathematical Systems Theory 10 (1976) 1–17.

[13] P. van Emde Boas, Another NP-complete partition problem and the complexity of computing short vectors in a lattice, Technical Report No. 81-04, Department of Mathematice, University of Amsterdam, 1981.