When Input Integers are Given in the Unary Numeral Representation

Tomoyuki Yamakami

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 in general provided in the form of the "binary" numeral representation and the lengths of such binary forms are used as a basis unit of measuring the computational complexity of the problems. In sharp contrast, the "unary" numeral representation has been known to bring a remarkably different effect onto the computational complexity of the problems. When no computational-complexity difference is observed between these two representations, on the contrary, the problems are called strong NP-complete. This work attempts to spotlight an issue of how the unary representation affects the computational complexity of various combinatorial problems. We present numerous NP-complete problems, most of which turn out to be easily solvable when input integers are represented in unary. We hope that a list of such problems signifies the structural differences between strong NP-completeness and non-strong NP-completeness.

1 Background and 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 if one tries to solve them in feasible time. The proof of NP-completeness of the problems therefore makes us turn away from solving them exactly in an efficient way but rather guide us to the development of approximation or randomized algorithms.

In computational complexity theory, we 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 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 how different consequences can be drawn when input integers are all provided by the "unary" numeral system

of describing these integers. The unary numeral system is so distinctive, in comparison to the binary numeral system, that we need to heed a special attention in our analyses of algorithms.

When input integers given to combinatorial problems are expressed in unary, how does these unary forms affect the computational complexity of the problems? A simple transformation of input integers expressed in unary to their binary representations makes the original input lengths look exponentially larger than their binary lengths. Thus, any algorithm working with the unary-represented input integers seems to be exponentially more time consuming than the same algorithm with binary-represented input integers. 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 binary-expressed input integers to their corresponding unary-expressed ones (see [5, Section 4.2]). Those problems are known as *strong NP-complete.*[1] The notion of strong NP-completeness of combinatorial problems 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 from the binary representation to the unary one. It is therefore imperative (and also quite intriguing) to study a computational 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 [4] discussed 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 1-turn nondeterministic counter automaton can recognize UK (e.g., [1]). As for a variant of UK, Jenner [7] further considered the case where 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 can expand such a shift-unary representation to a *multiple shift-unary representation* for a series of positive integers and to a *general unary (numeral) representation* for all integers, including zero and negative ones. See Section 2.1 for their precise definitions.

Driven by our great interest in the effect of the unary numeral system, we wish to study the computational complexity of combinatorial problems whose input integers are in part represented by the unary representation. For the succinctness of further descriptions, we hereafter refer to input integers expressed in the unary numeral system as *unary-form integers* in comparison to *binary-form integers.*

[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., [5, Section 4.2].

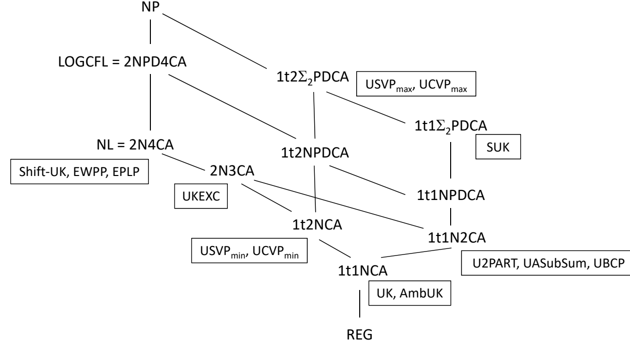

Fig. 1. Inclusion relationships among complexity classes with solid lines and membership relations of numerous decision problems listed in small boxes to specific complexity classes.

1.2 New Challenges

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 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 problems and study how their computational complexities can change when we switch the binary representation of input integers to the unary one. Among various types of combinatorial problems, we look into number problems, graph-related problems, and lattice-related problems, which deal with input integers.

A brief summary of our results are illustrated in Fig. 1. The detailed explanation of the combinatorial problems and complexity classes listed in the figure will be provided in Sections 2–5.

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 sets of all integers and 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 $n \geq 1$. 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 $\max\{|x_i| : i \in [n]\}$,

where $|\cdot|$ indicates the absolute value. Given a set A, $\mathcal{P}(A)$ 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. 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 (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 $x = (1^b, 1^{a_1}, 1^{a_2}, \ldots, 1^{a_n})$ is given to a machine, we explicit assume that x has the form of $1^b\#1^{a_1}\#1^{a_2}\#\cdots\#1^{a_n}$ with a designated separator symbol #. 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* and the *shift-unary-form (positive) integer*, respectively.

To deal with "general" integers, including zero and negative integers, we further 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 $1^{\langle a \rangle}$. A *general shift-unary representation* of $-a$ for $a > 0$ is 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

Since our interest mostly lies on space-bounded computation, as a basic machine model, we 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 f from Σ^* to Γ^* for alphabets Σ and Γ is *computable in log space* if there is a DTM equipped with a write-once output tape such that, on input x, it produces $f(x)$ in $|x|^{O(1)}$ time and $O(\log|x|)$ space. The notation FL refers to the class of all such functions.

[2] Unlike the unary and binary representations, a positive integer in general has more than one shift-unary representation.

[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 right.

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

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.

Given 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 want 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.

Lemma 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 1nca) 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 1nca's running in polynomial time. We further expand 1NkCA to 1NPDkCA by supplementing k counters to 1npda's. These specific machines are called *k-counter pushdown automata*. When a tape head of a multi-counter automaton is allowed to move in all directions, we call such a polynomial-time machine a 2nca. With the use of 2nca's in place of 1nca's, we obtain 2NPDkCA from 1NPDkCA.

An *alternating machine* must use two groups of inner states: existential states and universal states. We are concerned with the number of times that an alternating 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 nondeterministic inner states. Note that $1\Sigma_1$PDCA coincides with 1NPDCA.

The notion of *turns* was discussed by Ginsburg and Spanier [6]. Turn-restricted counter automata are called *reversal bounded* in the past literature. A 1nca 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 1nca* is a 1nca that makes at most one turn during each computation. We add the prefix "1t" to express the restriction of the maximum number of turns of any underlying machine to be 1. For example, we write 1t1NCA when we restrict all underlying 1nca's in the definition of 1NCA to 1-turn 1nca's. Similarly, we define, e.g., 1t2NPDCA and $1\text{t}2\Sigma_k$PDCA. Note 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 for LOG(CFL).

Lemma 2. *For any* $k \geq 1$, 2NkCA $\subseteq$ NL, 2NPDkCA $\subseteq$ LOGCFL, *and* $2\Sigma_2$PDkCA $\subseteq$ NP.

Proof Sketch. For any $j \in \mathbb{N}^+$, 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. It is known that $2\Sigma_1$AuxPDATI,SP$(n^{O(1)}, O(\log n))$ coincides with LOGCFL [15] and $2\Sigma_2$AuxPDATI,SP$(n^{O(1)}, O(\log n))$ coincides with NP [8]. 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 3. NL = 2N4CA *and* LOGCFL = 2NPD4CA.

Proof Sketch. Following an argument of Minsky [11], we first simulate the behavior of an $O(\log n)$-space work tape by two stacks whose alphabet is of the form $\{0, 1, \perp\}$. Such a stack can be further simulated by two counters whose heights are $n^{O(1)}$-bounded. The proposition then follows from Lemma 2. □

3 Combinatorial Number Problems

We study the computational complexity of decision problems whose input instances are composed of unary-form (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 input integers expressed in unary is the *unary 0-1 knapsack problem*,

which was introduced in 1985 by Cook [4] 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 $b, a_1, a_2, \ldots, a_n$ are positive integers.
- Question: is there a subset S of $[n]$ satisfying $\sum_{i \in S} a_i = b$?

The problem UK seems to be more natural to be viewed 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 the 2-partition problem.

Unary 2 Partition Problem (U2PART):

- Instance: $(1^{a_1}, 1^{a_2}, \ldots, 1^{a_n})$, where $a_1, a_2, \ldots, a_n$ are 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 by Karp [10] to be NP-complete in 1972. The problem UK, by contrast, situated in between 1t1DCA and 1t1NCA in the following sense, where 1t1DCA is the deterministic version of 1t1NCA.

Lemma 4. *(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 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 5. *(1)* AmbUK *is in* 1t1NCA. *(2)* UASubSum *is in* 1t1N2CA.

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

Proposition 6. *(1)* UK $\equiv^{\mathrm{L}}_{m}$ U2PART. *(2)* UK $\equiv^{\mathrm{L}}_{tt}$ AmbUK. *(3)* UK $\equiv^{\mathrm{L}}_{m}$ UASubSum.

Jenner [7] studied a variant of UK, which we intend to call the *shift-unary 0-1 knapsack problem* (shift-UK) because of 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 shift-AmbUK, shift-UASubSum, and shift-U2PART, respectively.

Lemma 7. *(cf. [7]) The problem* shift-UK *is in* 1N6CA.

Proposition 8. shift-UK $\equiv^{\mathrm{L}}_m$ shift-U2PART $\equiv^{\mathrm{L}}_m$ shift-AmbUK $\equiv^{\mathrm{L}}_m$ shift-UASubSum.

Proof Sketch. Let $\mathcal{S}$ denote the set $\{$shift-U2PART, shift-AmbUK, shift-UASubSum$\}$. It is easy to obtain, by modifying the proof of Proposition 6, that shift-UK $\leq^{\mathrm{L}}_m \mathcal{C}$ for any $\mathcal{C} \in \mathcal{S}$. To show that $\mathcal{C} \leq^{\mathrm{L}}_m$ shift-UK for any $\mathcal{C} \in \mathcal{S}$, it suffices to show that $\mathcal{C}$ belongs to NL because the L-m-completeness of shift-UK guarantees that $\mathcal{C} \leq^{\mathrm{L}}_m$ shift-UK. Consider the case of $\mathcal{C}$ = shift-UASubSum. Let us 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 1, the sum $\sum_{i\in S} p_i 2^{t_i}$ can be computed using only log space. Hence, C is in NL. □

Since shift-UK is NL-complete under L-m-reductions [7], we immediately obtain the following corollary.

Corollary 9. *The following problems are all* NL*-complete:* shift-U2PART*,* shift-AmbUK*, and* shift-UASubSum*.*

For later references, we introduce another variant of UK. This variant concerns certain successive choices of unary-form integers.

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 $i_1 < i_2 < \cdots < i_k$ 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 $EXC = \varnothing$, UKEXC is equivalent to UK.

Lemma 10. 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 firstly and then sequentially pop 1^{a_i} for the chosen indices i. $\square$

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 any index $i \in [m]$?

The complexity class $1\mathrm{t}1\Sigma_2\mathrm{PDCA}$ is the one-way restriction of $1\mathrm{t}2\Sigma_2\mathrm{PDCA}$.

Lemma 11. *The problem* SUK *is in* $1\mathrm{t}1\Sigma_2\mathrm{PDCA}$.

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

In nondeterministic 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 we select the jth entry of each block exactly when $e_j = 1$. Let $A_w = \{j \in [n] \mid e_j = 1\}$. In universal inner states, we then check whether $b_i = \sum_{j \in A_w} a_{ij}$. This 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 $(1^{a_{i1}}, \ldots, 1^{a_{in}})$ of x is over, 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. $\square$

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, given a set $\{(a_i, b_i)\}_{i \in [n]}$ of binary string pairs and a number $k \geq 1$, a sequence $(i_1, i_2, \ldots, i_t)$ 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 [2]. 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: $((a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n))$ for unary strings $a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_n \in \{1\}^*$ and 1^k for $k \in \mathbb{N}^+$.
- Question: is there a sequence $(i_1, i_2, \ldots, i_t)$ of elements in $[n]$ with $t \leq k$ 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 requirement $a_{i_1} \cdots a_{i_t} = b_{i_1} \cdots b_{i_t}$ of UBCP is equivalent to $\sum_{j \in S} |a_j| = \sum_{j \in S} |b_j|$, where $|a_j|$ and $|b_j|$ are the lengths of strings a_j and b_j, respectively.

Lemma 12. UBCP *belongs to* 1t1N2CA.

Proposition 13. UK $\leq_m^{\mathrm{L}}$ UBCP $\leq_m^{\mathrm{L}}$ AmbUK.

Corollary 14. UK $\equiv_{tt}^{\mathrm{L}}$ UBCP.

4 Graph-Related 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". We study the computational complexity of these problems.

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

For the purpose of this work, a dag $G = (V, E)$ given as part of inputs to an underlying machine is assumed to satisfy the following specific conditions. , The vertex set 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 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 "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 $((1^{i_1} : 1^{j_{i_1,1}}, 1^{j_{i_1,2}}, \ldots, 1^{j_{i_1,k_1}}), \ldots, (1^{i_n} : 1^{j_{i_n,1}}, 1^{j_{i_n,2}}, \ldots, 1^{j_{i_n,k_n}}))$ 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 as 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 $c \in \mathbb{N}^+$.
- 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 [9].

Lemma 15. EWPP *is in* NL.

When each edge weight is 1, the total weight of a path is the same as the length of a 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 16. EWPP $\equiv_m^{\mathrm{L}}$ EPLP.

We then obtain the following NL-completeness result.

Proposition 17. EWPP *and* EPLP *are both* L*-m-complete for* NL.

Proof Sketch. We prove that (*) for any language L in 1NCA, $L \leq_m^{\mathrm{L}}$ EPLP. 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 15, EPLP is also in NL. Therefore, EPLP is L-m-complete for NL. Since EWPP $\equiv_m^{\mathrm{L}}$ EPLP by Proposition 16, we also obtain the NL-completeness of EWPP.

To show the statement (*), let us take any 1nca 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 $\vdash$. 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 $v_1 \vdash v_2$. 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. □

We discuss how EWPP is connected to UKEXC and UK. To see the desired connections, we introduce two restricted variants of EWPP. We say that a dag $G = (V, E)$ with $V \subseteq \mathbb{N}^+$ is *topologically sorted* if, for any two vertices $i, j \in V$, $(i, j) \in E$ implies $i < j$. We define TS-EWPP as the restriction of EWPP onto instances that are topologically sorted. 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 TS-EWPP whose instance graphs are all edge-closed.

Proposition 18. *(1)* TS-EWPP $\equiv_m^{\mathrm{L}}$ UKEXC. *(2)* EC-EWPP $\equiv_m^{\mathrm{L}}$ UK.

5 Lattice-Related Problems

Let us discuss lattice-related problems. The decision version of the *closest vector problem* (CVP) is known to be NP-complete [16]. We then consider a variant of CVP. To fit into our handling of unary representations, here we deal only with lattices over $\mathbb{Z}^n$ and a simple norm notion. Recall from Section 2.1 the *max norm* $\|v\|_\infty$ of a vector v. By contrast, we define $\|v\|_{\min}$ to be $\min\{|v_i| : i \in [n]\}$ for any real-valued vector $v = (v_1, v_2, \ldots, v_n)$. Notice that $\|v\|_{\min}$ does not serve as a true "distance".

The notation $\mathcal{L}(v_1, v_2, \ldots, v_m)$ denotes the lattice spanned by a given set $\{v_1, v_2, \ldots, v_m\}$ of basis vectors.

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

- Instance: 1^b for a positive integer b, a tuple $(1^{\langle v_i[1]\rangle}, 1^{\langle v_i[2]\rangle}, \ldots, 1^{\langle v_i[n]\rangle})$ 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 $x_0 = (x_0[1], x_0[2], \ldots, x_0[n])$.

- QUESTION: is there a lattice vector w in $\mathcal{L}(v_1, v_2, \ldots, v_m)$ such that the max norm $\|w - x_0\|_\infty$ is at most b?

We further define UCVP$_{\min}$ by replacing $\|\cdot\|_\infty$ in the above definition of UCVP$_{\max}$ with $\|\cdot\|_{\min}$.

For simplicity, in what follows, we 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 19. SUK $\leq^{\mathrm{L}}_m$ UCVP$_{\max}$.

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

Here, we show that SU2PART $\leq^{\mathrm{L}}_m$ UCVP$_{\min}$. 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 $d_{max} = \max_{j\in[m]}\{d_j\}$. For any $j \in [m]$ and $i \in [n]$, we set $a'_{ji} = d_{max}a_{ji}$ and $d'_j = d_{max}d_j$.

We define n vectors $v_1, v_2, \ldots, v_n$ as $v_1 = (a'_{11}, a'_{21}, \ldots, a'_{m1}, 2, 0, 0, \ldots, 0)$, $v_2 = (a'_{21}, a'_{22}, \ldots, a'_{2n}, 0, 2, 0, \ldots, 0)$, $\ldots$, $v_n = (a'_{n1}, a'_{n2}, \ldots, a'_{nm}, 0, 0, 0, \ldots, 0, 2)$. Moreover, we set $x_0 = (\lfloor d'_1/2\rfloor, \lfloor d'_2/2\rfloor, \ldots, \lfloor d'_m/2\rfloor, 1, 1, \ldots, 1)$. We also set $b = 1$. Recall the notation $\bar{v}$ for a vector v. We 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 UCSP$_{\max}$. It then follows that $x \in$ U2PART iff $y \in$ UCVC$_{\max}$. □

Lemma 20. *(1)* UCVP$_{\max} \in$ 1t2Σ_2PDCA. *(2)* UCVP$_{\min} \in$ 1t2NCA.

It is not clear that UCVP$_{\max}$ belongs to P.

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

UNARY MAX-NORM SHORTEST VECTOR PROBLEM (USVP$_{\max}$):

- INSTANCE: 1^b with $b \in \mathbb{N}^+$ and a tuple $(1^{\langle v_i[1]\rangle}, 1^{\langle v_i[2]\rangle}, \ldots, 1^{\langle v_i[n]\rangle})$ 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 $\mathcal{L}(v_1, v_2, \ldots, v_m)$ such that the max norm $\|w\|_\infty$ is at most b?

Similarly, we can define USVP$_{\min}$ by replacing $\|\cdot\|_\infty$ with $\|\cdot\|_{\min}$. In a way similar to prove Lemma 20, we can prove that USVP$_{\max} \in$ 1t2Σ_2PDCA.

Lemma 21. USVP$_{\min} \leq^{\mathrm{L}}_{tt}$ UCVP$_{\min}$ *and* USVP$_{\max} \leq^{\mathrm{L}}_{tt}$ UCVP$_{\max}$.

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

In this work, we have studied the significant roles of unary representations of integers when they are given as part of input instances. We have observed through this work that many NP-complete (or even NP-hard) problems fall into lower complexity classes when input integers are expressed in unary and therefore they are far away from strong NP-completeness. We expect that a further study will reveal underlying features that make non-strong NP-completeness differ from strong NP-completeness.

References

1. Cho, S., Huynh, D.T.: On a complexity hierarchy between L and NL. Inform. Process. Lett. 29, 177–182 (1988)
2. Constable, R.L., Hunt III, H.B., Sahni, S.: On the computational complexity of scheme equivalence. Report No. 74-201, Dept. of Computer Science, Cornell University (1974)
3. Cook, S.A.: Deterministic CFLs are accepted simultaneously in polynomial time and log squared space. In the Proceedings of the 1lth Annual ACM Symposium on Theory of Computing, pp. 338–345 (1979)
4. Cook, S.A.: A taxonomy of problems with fast parallel algorithms. Inform. Control 64 (1985) 2–22.
5. Garey, M.R., Johnson, D.S.: Computers and Interactability: A Guide to the Theory of NP-Completeness, W. H. Freeman and Company, New York (1979)
6. Ginsburg, G., E. H. Spanier, E.H.: Finite-turn pushdown automata. SIAM J. Control 4, 429–453 (1966)
7. Jenner, B.: Knapsack problems for NL. Inform. Process. Lett. 54, 169–174 (1995)
8. Jenner, B., Kirsg, B.: Characterizing the polynomial hierarchy by alternating pushdown automata. RAIRO–Theoret. Inform. App. 23, 87–99 (1989)
9. Jones, N.D., Lien, Y.E., Laaser, W.T.: New problems complete for nondeterministic log space. Math. Systems Theory 10, 1–17 (1976)
10. Karp, R.M.: Reducibility among combinatorial problems. In: R. E. Miller and J. W. Thatcher (eds.), Complexity of Computer Computations. Plenum Press, New York, pp. 85–103 (1972)
11. Minsky, M.L.: Computation: Finite and Infinite Machines. Prentice-Hall, Englewood Cliffs, NJ (1967)
12. Monien, B., Sudborough, I.H.: Foraml language theory. In: Formal Language Theory, R. V. Book (ed), Academic Press, New York (1980).
13. Reingold, O.: Undirected connectivity in log-space. J. ACM 55, article 17 (2008)
14. Stockmeyer, L.J., Meyer, A.R.: Word problems requiring exponential time. In the Proc. of the 5th Ann. ACM Symp. on Theory of Computing, pp. 1–9 (1973)
15. Sudborough, I.H.: On the tape complexity of determinsitic context-free languages. J. ACM 25, 405–414 (1978)
16. van Emde Boas, P.: Another NP-complete partition problem and the complexity of computing short vectors in a lattice. Report No. 81-04, Department of Mathematice, University of Amsterdam (1981)