数理解析研究所講究録
第 754 巻 1991 年 196-205

On the Complexity of Computing Optimal Solutions

Zhi-Zhong CHEN and Seinosuke TODA
Dept. Comput. Sci. & Infor. Math.
Univ. Electro-Communications
Chofu-shi, Tokyo 182, Japan

Abstract

We study the computational complexity of computing optimal solutions *but not just giving optimal costs* for NP optimization problems where the costs of feasible solutions are bounded above by a polynomial in the length of their instances (we call such an NP optimization problem an NP *combinatorial* optimization problem, or simply, an *NPCOP*). It is of particular interest to find a computational structure (or equivalently, a complexity class) which captures that complexity, if we consider the problems of computing optimal solutions for *NPCOP*'s as a class of *functions* giving those optimal solutions. In this paper, we will observe that $\mathrm{PF}_{tt}^{\mathrm{NP}}$, the class of functions computable in polynomial-time with one free evaluation of unbounded parallel queries to NP oracle sets, captures that complexity. We first show that for any *NPCOP* Π, there exists a polynomial-time bounded randomized algorithm which, given an instance of Π, uses one free evaluation of parallel queries to an NP oracle set and outputs some optimal solution of the instance with very high probability. We then show that for several natural *NPCOP*'s, any function giving those optimal solutions is at least as computationally hard as all functions in $\mathrm{PF}_{tt}^{\mathrm{NP}}$. To show the hardness results, we introduce a property of *NPCOP*'s, called *linear paddability*, and we show a general result that if Π is a linearly paddable *NPCOP* and its associated decision problem is NP-hard, then all functions in $\mathrm{PF}_{tt}^{\mathrm{NP}}$ are computable in polynomial-time with one free evaluation of an arbitrary function giving optimal solutions for instances of Π. The hardness results are applications of this general result. Among the *NPCOP*'s, we include MAXIMUM CLIQUE, MINIMUM COLORING, LONGEST PATH, 0-1 TRAVELING SALESPERSON, and 0-1 INTEGER PROGRAMMING.

1 Introduction

Many papers have been devoted to the study of the complexity of NP optimization problems (*NPOP* for short) from different points of view. One approach to *NPOP*'s has been to study the complexity of their *associated decision problems* [5]. For example, instead of studying the complexity of finding a maximum clique of a given graph, one may study the complexity of deciding whether, given a graph G and a positive integer k, G has a clique of size at least k. Since NP-complete problems are widely believed to be intractable and solving an *NPOP* is at least as computationally hard as its associated decision problem, an *NPOP* appears to be hard to solve if its associated decision problem is proved to be NP-complete.

Another approach to *NPOP*'s has been developed by Krentel [6]. In [6], he considered the complexity of *NPOP*'s by investigating the computational complexity of computing optimal costs for *NPOP*'s. He defined two classes of functions to capture that complexity. One is $\mathrm{PF}^{\mathrm{NP}}$, the class of functions computable in polynomial-time with polynomial number of queries to NP oracle sets. The other is $\mathrm{PF}^{\mathrm{NP}[\log]}$, the class of functions computable in polynomial-time with logarithmic number of queries to NP oracle sets. Krentel showed that computing optimal costs for several well-known *NPOP*'s falls into $\mathrm{PF}^{\mathrm{NP}}$ and is as computationally hard as all functions in $\mathrm{PF}^{\mathrm{NP}}$, while computing optimal costs for several other well-known $NPOP$'s falls into $\mathrm{PF}^{\mathrm{NP[log]}}$ and is as computationally hard as all functions in $\mathrm{PF}^{\mathrm{NP[log]}}$ [6]. Since $\mathrm{PF}^{\mathrm{NP}} = \mathrm{PF}^{\mathrm{NP[log]}}$ implies P=NP [6], Krentel concluded that not all $NPOP$'s appear to have the same computational complexity when considering the computational complexity of computing the optimal cost for them. For example, computing the length of shortest tours is strictly harder than computing the size of maximum cliques unless P=NP. Krentel's results make finer distinctions on the complexity of $NPOP$'s than previously known in the theory of NP-completeness.

In this paper, we study the computational complexity of computing optimal solutions *but not just giving optimal costs* for NP optimization problems where the costs of feasible solutions are bounded above by a polynomial in the length of their instances (we call such an NP optimization problem an NP *combinatorial* optimization problem, or simply, an $NPCOP$). It is of particular interest to find a computational structure (or equivalently, a complexity class) which captures that complexity, if we consider the problems of computing optimal solutions for $NPCOP$'s as a class of *functions* giving those optimal solutions. This question was posed by Gasarch and has been stated by Krentel as a further step of his research on the computational complexity of optimization problems [6]. However, to our knowledge, no progress along these lines has been known. We think that the difficulty in investigating the computational complexity of computing optimal solutions for $NPCOP$'s is finding a suitable approach. In order to investigate the complexity of computing the optimal cost for a given $NPOP$ Π, the approach was to investigate the complexity of the function which computes the optimal cost of x for any given instance x of Π [6]. Nevertheless, since an arbitrary instance of $NPCOP$'s may have two or more optimal solutions, we can not make the same approach when we investigate the complexity of computing optimal solutions. As introduced in this paper, we overcome this difficulty by considering the problem of computing optimal solutions for a given $NPCOP$ Π as a class of functions giving an optimal solution for any given instance of Π (we call such a function *an optimal-solution function* of Π). When discussing the upper bound of the complexity of computing optimal solutions for a given $NPCOP$ Π, we may only require an algorithm solving Π to compute *some* optimal-solution function of Π. On the other hand, when discussing the lower bound of the complexity of computing optimal solutions for an $NPCOP$ Π, we examine the relative complexity between *all* optimal-solution functions of Π and some complexity classes of functions. Our this approach enables us to observe that $\mathrm{PF}^{\mathrm{NP}}_{\mathrm{tt}}$, the class of functions computable in polynomial-time with one free evaluation of unbounded parallel queries to NP oracle sets, captures the computational complexity of computing optimal solutions for $NPCOP$'s.

In Section 3, we decribe a polynomial-time bounded randomized algorithm for any $NPCOP$ Π which given an instance x of Π, uses one free evaluation of parallel queries to a set in NP and outputs *some* optimal solution for x with very high probability. This general result shows that computing optimal solutions is as easy as evaluating parallel queries to NP sets, giving an upper bound of computing optimal solutions for $NPCOP$'s.

In Section 4, we show that for several natural $NPCOP$'s Π, any optimal-solution function of Π is at least as computationally hard as *all* functions in $\mathrm{PF}^{\mathrm{NP}}_{\mathrm{tt}}$. To show the hardness results, we introduce a notion called *linear paddability*. Intuitively speaking, we say that an $NPCOP$ Π is linearly paddable if we can efficiently pad any two instances of Π into one instance of Π whose length is linear in the sum of the lengths of the original two instances, and we can efficiently compute optimal solutions for the original two instances from any optimal solution for the padded instance. In addition, with each $NPCOP$ Π, we associate a decision problem of deciding whether the optimal cost of x is at least (or at most if Π is a minimization problem) k for a given pair $\langle x, k\rangle$ of an instance and an integer. Then we show a general result that if Π is a linearly paddable $NPCOP$ and its associated decision problem is NP-hard, then all functions in $\mathrm{PF}^{\mathrm{NP}}_{\mathrm{tt}}$ are computable in polynomial-time with one free evaluation of any optimal-solution function of Π. This general result provides a *uniform* way to show the hardness of computing optimal solutions for $NPCOP$'s whose associated decision problems are known to be NP-hard. In fact, our hardness results are applications of this general result. Among the natural $NPCOP$'s, we include MAXIMUM CLIQUE, MINIMUM COLORING, LONGEST PATH, 0-1 TRAVELING SALESPERSON, 0-1 INTEGER PROGRAMMING, and MAXIMUM TWO SATISFIABILITY.

2 Preliminaries

We assume that the reader is familiar with the basic concepts from the theories of optimization problems and computational complexity. We use $\Sigma = \{0,1\}$ as our alphabet. By a *language* or a *set*, we mean a subset of Σ^*. We denote by $|x|$ the length of a string x. The empty string is denoted by λ. For any finite set A, $||A||$ denotes the number of elements of A and χ_A denotes the characteristic function of A. Let $A^{\leq n}$ and $A^{=n}$ denote the sets $\{x \in A \ : \ |x| \leq n\}$ and $\{x \in A \ : \ |x| = n\}$, respectively. For any sets A and B, $A \oplus B$ denotes the marked union of A and B; that is, $A \oplus B = \{0x \ : \ x \in A\} \cup \{1y \ : \ y \in B\}$. The symbol $\oplus$ is also used to denote the exclusive-or operation of Boolean values. We write N for the set of non-negative integers. Let $bin(n)$ be a standard binary representation of non-negative integer n over Σ, and write $\log(n)$ to mean the base 2 logarithm of n for $n > 0$. We assume a standard one-to-one pairing function from $\Sigma^* \times \Sigma^*$ to Σ^* that is polynomial-time computable and polynomial-time invertible. For strings x and y, we denote the output of the pairing function by $\langle x, y\rangle$; this notation is extended to denote any k-tuples for $k > 2$ in a usual manner.

An *optimization problem* Π is a quintuple (op, D, S, R, c), where

(1) $op \in \{max, min\}$ is the *underlying operation* (i.e., maximization or minimization),
(2) D is the set of *instances*,
(3) S is the set of *feasible solutions* ,
(4) $R \subseteq D \times S$ is the *instance-solution relation*, and
(5) $c: \ D \times S \rightarrow$N is the *solution cost function* (for simplicity, we consider only the case that all costs are non-negative integers).

We call Π a *maximization problem* if $op = max$ and call it a *minimization problem* otherwise. To each instance $x \in D$, we associate a finite subset $S(x) = \{y \in S \ : \ R(x,y) \text{ holds true }\}$, and call it the *solution space* of x. The *optimal cost function* $c^*: \ D \rightarrow$N of Π is defined by

$$c^*(x) = op\{c(x,y) \ : \ y \in S(x)\}$$

and the set of optimal solutions for an instance $x \in D$, $\mathrm{optsol}_\Pi(x)$, is defined by

$$\mathrm{optsol}_\Pi(x) = \{y \in S(x) \ : \ c(x,y) = c^*(x)\}.$$

The objective in solving a given optimization problem Π is to compute an optimal solution for any instance of Π. We particularly note that examining the complexity of computing optimal solutions but not just giving optimal costs is the essence of this paper.

A *combinatorial optimization problem* is an optimization problem $\Pi = (op, D, S, R, c)$ for which there exists a polynomial p such that for all $x \in D$ and all $y \in S(x)$, $c(x,y) \leq p(|x|)$. We call a combinatorial optimization problem $\Pi = (op, D, S, R, c)$ an *NP combinatorial optimization problem* (*NPCOP* for short) if it satisfies the following additional conditions:

(1) D, S, and R are polynomial-time decidable,
(2) c is polynomial-time computable, and
(3) for some polynomial p, all $x \in D$, and all $y \in S$, $y \in S(x)$ implies $|y| \leq p(|x|)$.

When Π is a maximization (resp., minimization) problem, we see from these conditions that the problem of deciding whether, given an instance $x \in D$ and a natural number k, the optimal cost $c^*(x)$ is at least (resp., at most) k is decidable in NP. This is the reason why Π is called an NP combinatorial optimization problem. Throughout this paper, we will deal with only NP combinatorial optimization problems.

When discussing the upper bound of the complexity of computing optimal solutions for a given *NPCOP* Π, we may only require an algorithm solving Π to compute *some* optimal solution for any given instance of Π. In particular, when we consider a randomized algorithm solving the *NPCOP*, the algorithm may produce some different optimal solutions depending on random bits used; we will never require the algorithm to compute a single optimal solution independent of random bits used. Only a requirement to randomized algorithms is that for each instance, they must compute some optimal solution *with very high probability*.

On the other hand, when discussing the lower bound of the complexity of computing optimal solutions for a given *NPCOP*, we will examine the relative complexity between solving the *NPCOP* and the other classes of problems, as in the theory of NP-completeness. Intuitively speaking, we want to show that solving the *NPCOP* is at least as hard as a problem whose computational complexity appears to be settled. As mentioned in the introduction, we regard the problem of solving an *NPCOP* as a class of functions giving an optimal solution for any given instance of the *NPCOP*, and the opponents compared with those functions are functions in the class $\mathrm{PF}^{\mathrm{NP}}_{tt}$. Thus, with each *NPCOP* $\Pi = (op, D, S, R, c)$, we associate a class OPTSOL_Π of functions defined by

$$\mathrm{OPTSOL}_\Pi = \{F : D \to S \ : \ (\forall x \in D)[F(x) \in \mathrm{optsol}_\Pi(x)] \ \}.$$

Then, the reducibility notion below will capture the above purpose.

Definition 2.1 Let F and H be two classes of functions, and let H be a function. Then, H is *uniformly polynomial-time 1-Turing reducible to* F, in symbols $H \leq^{\mathrm{uniform-PF}}_{\mathrm{1-T}} \mathrm{F}$, if there exist polynomial-time computable functions f and g such that for every function $F \in$F and every $x \in \Sigma^*$, $H(x) = g(x, F(f(x)))$. We call the pair $\langle f, g\rangle$ a $\leq^{\mathrm{uniform-PF}}_{\mathrm{1-T}}$*-reduction* of H to F (note that the reduction $\langle f, g\rangle$ of H to F must be the same for all functions chosen from F). F is $\leq^{\mathrm{uniform-PF}}_{\mathrm{1-T}}$*-hard* for H if every function in H is $\leq^{\mathrm{uniform-PF}}_{\mathrm{1-T}}$-reducible to F.

If we can show that some OPTSOL_Π is $\leq^{\mathrm{uniform-PF}}_{\mathrm{1-T}}$-hard for $\mathrm{PF}^{\mathrm{NP}}_{tt}$, then we see that computing optimal solutions for the *NPCOP* Π is at least as computationally hard as computing the hardest functions in $\mathrm{PF}^{\mathrm{NP}}_{tt}$.

We finally mention some complexity classes that we deal with in the present paper. Throughout this paper, we mean by an NTM a nondeterministic Turing machine. Let NP be the class of sets accepted by polynomial-time bounded NTM's. A language A is NP-*hard* if for any language B in NP, there exists a polynomial-time computable function f such that for every $x \in \Sigma^*$, $x \in A$ if and only if $f(x) \in B$. $\mathrm{PF}^{\mathrm{NP}}_{tt}$ is the class of all functions F for which there exist a set A in NP and two polynomial-time computable functions g, e such that for all strings x, $F(x) = e(x, \chi_A(y_1), \cdots, \chi_A(y_m))$, where $g(x) = \langle y_1, \cdots, y_m\rangle$. More intuitively, a function F is in $\mathrm{PF}^{\mathrm{NP}}_{tt}$ if there exist a polynomial-time bounded deterministic oracle transducer (DOTM for short) N and a set $A \in$NP such that N^A computes F and N^A on all inputs prepares all query strings before asking them to the oracle set A. In Section 3, we will use this intuitive definition of $\mathrm{PF}^{\mathrm{NP}}_{tt}$, to describe an algorithm computing a function in $\mathrm{PF}^{\mathrm{NP}}_{tt}$.

3 An upper bound of computing optimal solutions

In this section, we first show an upper bound on the complexity of computing optimal solutions. Intuitively speaking, we describe a polynomial-time bounded randomized algorithm for any *NPCOP*

Π which, given an instance x of Π, uses one free evaluation of parallel queries to a set in NP and outputs some optimal solution for x with very high probability. Thus, computing optimal solutions is as easy as evaluating parallel queries to NP sets. To show this, we use a technique developed by Valiant and Vazirani [8]. They developed a randomization technique, with using a small amount of random bits, by which we can reduce any given finite subset of $\{0,1\}^n$ (for $n \geq 1$) to a set of exactly one element with high probability.

For any strings x and y in $\{0,1\}^n$, let $x \cdot y$ denotes $(x_1 \wedge y_1) \oplus \cdots \oplus (x_n \wedge y_n)$, where x_i (resp., y_i) denotes the i-th bit of x (resp., y) and $\oplus$ denotes the exclusive-or. Let X be a finite set of strings and Q be a predicate over strings. We denote by $Prob\{w_1, \cdots, w_k \in X : Q(w_1, \cdots, w_k)\}$ the probability that $Q(w_1, \cdots, w_k)$ holds for strings $w_1, \cdots, w_k$ chosen randomly from X under uniform distribution. Then, Valiant and Vazirani showed the following result:

Theorem 3.1 [8] Let n be a positive integer. Then, for any subset S of $\{0,1\}^n$,

$$Prob\{\ w_1, \cdots, w_n \in \{0,1\}^n : (\exists j, 0 \leq j \leq n)[\ ||\{\ y \in S : (\forall i, 1 \leq i \leq j)[y \cdot w_i = 0]\ \}|| = 1\ ]\ \} \ \geq\ \tfrac{1}{4}.$$

Theorem 3.2 Let $\Pi = (op, D, S, R, c)$ be an $NPCOP$ and let e be any polynomial. Then, there exist a function $G \in \mathrm{PF}_{tt}^{\mathrm{NP}}$ and a polynomial r such that for all $x \in D$ with $|x| = n$,

$$Prob\{w \in \{0,1\}^{r(n)} : G(x, w) \in \mathrm{optsol}_{\Pi}(x)\} \geq 1 - 2^{-e(n)}.$$

Proof We consider only the case that Π is a maximization problem. The other case is quite similar. For simplicity, we assume that there exists a polynomial q such that for all $x \in D$ and all $y \in S$, $y \in S(x)$ implies $|y| = q(|x|)$. We lose no generality under this assumption. Let p be a polynomial such that for all $x \in D$ and all $y \in S(x)$, $c(x,y) \leq p(|x|)$. Then we first define two sets A and B as follows:

$A = \{\langle x, i\rangle : i \geq 0,\ x$ has a solution with cost $i\}$

$B = \{\langle x, i, j\rangle : x \in D,\ 0 \leq i \leq p(|x|),\ 1 \leq j \leq q(|x|)$, and there exists a $y \in S(x)$ with cost i such that the j-th symbol of y is $1\}$
$\cup\{\langle x, i, j, w_1, \cdots, w_k\rangle : x \in D,\ 0 \leq i \leq p(|x|),\ 1 \leq j \leq q(|x|),\ 1 \leq k \leq q(|x|)$, $w_1, \cdots, w_k \in \{0,1\}^{q(|x|)}$, and there exists a $y \in S(x)$ with cost i such that the j-th symbol of y is 1 and $y \cdot w_1 = \cdots = y \cdot w_k = 0\}$.

Obviously, A and B are in NP. Thus, we have $A \oplus B \in$NP. We also define the polynomial r by $r(n) = 3 \cdot e(n) \cdot q(n)^2$.

Below, we define a deterministic oracle transducer N which uses $A \oplus B$ as an oracle set. Given an instance $x \in D$ with $|x| = n$ and a string $w \in \{0,1\}^{r(n)}$, N operates as follows:

Step 1. N computes the optimal cost $c^*(x)$. This is done by asking the queries $\langle x, 0\rangle$, $\langle x, 1\rangle$, $\cdots$, $\langle x, p(n)\rangle$ to the oracle set A and computing the largest integer k $(= c^*(x))$ such that $\langle x, k\rangle$ is in A.

Step 2. Let w_1, w_2, $\cdots$, $w_{3e(n)}$ be the strings in $\{0,1\}^{q(n)^2}$ such that $w_1 w_2 \cdots w_{3e(n)} = w$, and for every l, $1 \leq l \leq 3e(n)$, let $w_{l,1}, \cdots, w_{l,q(n)}$ be the strings in $\{0,1\}^{q(n)}$ such that $w_{l,1} \cdots w_{l,q(n)} = w_l$. Then, by asking queries to the oracle set B, N computes strings $s_{i,l,m}$ as follows: for all i, l, m such that $0 \leq i \leq p(n)$, $1 \leq l \leq 3e(n)$, and $0 \leq m \leq q(n)$,

(a) $s_{i,l,0} = \chi_B(\langle x, i, 1\rangle)\chi_B(\langle x, i, 2\rangle) \cdots \chi_B(\langle x, i, q(n)\rangle)$ and

(b) $s_{i,l,m} = \chi_B(\langle x, i, 1, w_{l,1}, \cdots, w_{l,m}\rangle) \cdots \chi_B(\langle x, i, q(n), w_{l,1}, \cdots, w_{l,m}\rangle)$ for $1 \leq m \leq q(n)$.

Step 3. Let $k = c^*(x)$. If $s_{k,l,m} \in S(x)$ for some l and m, then N outputs the string $s_{k,l,m}$; otherwise, N outputs nothing (in this case, the function computed here is supposed to be "undefined" on x and w).

Let G denote the function computed by $\mathrm{N}^{A\oplus B}$. We can easily see that N is polynomial-time bounded and each query string is prepared independently of the other query strings; hence, the query strings made by N on input $\langle x, w\rangle$ can be realized as parallel queries to the oracle set $A \oplus B$. Thus, G is in $\mathrm{PF}^{\mathrm{NP}}_{\mathrm{tt}}$. To show that G satisfies the theorem, we first show the following claim:

Claim Suppose that for some l and m, $\|\{y \in \mathrm{optsol}_\Pi(x) \ : \ y\cdot w_{l,1} = y\cdot w_{l,2} = \cdots = w_{l,m} = 0\}\| = 1$. Then $s_{k,l,m}$ is an optimal solution of x, where $k = c^*(x)$.

Proof of Claim. Let y be the unique optimal solution of x in the set $\{y \in \mathrm{optsol}_\Pi(x) \ : \ y \cdot w_{l,1} = \cdots = y \cdot w_{l,m} = 0\}$. Then we see, from the definition of B, that for all $1 \le j \le q(|x|)$, $\chi_B(\langle x, k, j, w_{l,1}, \cdots, w_{l,m}\rangle) = 1$ iff the j-th bit of y is 1. Thus, we have $s_{k,l,m} = y$. ∎

From this claim and Theorem 3.1, we have that for all $x \in D$ with $|x| = n$,

$$\begin{aligned}
&Prob\{w \in \{0,1\}^{r(n)} \ : \ G(x,w) \in \mathrm{optsol}_\Pi(x)\}\\
&= Prob\{w \in \{0,1\}^{r(n)} \ : \ (\exists l, m, 1 \le l \le 3e(n), 0 \le m \le q(n))[s_{c^*(x),l,m} \in \mathrm{optsol}_\Pi(x)]\}\\
&\ge Prob\{w \in \{0,1\}^{r(n)} \ : \ (\exists l, m, 1 \le l \le 3e(n), 0 \le m \le q(n))\\
&\qquad\qquad [\ \|\{y \in \mathrm{optsol}_\Pi(x) \ : \ y \cdot w_{l,1} = \cdots = y \cdot w_{l,m} = 0\}\| = 1]\}\\
&\ge 1 - \textstyle\prod_{l=1}^{3e(n)} Prob\{w_{l,1}, \cdots, w_{l,q(n)} \in \{0,1\}^{q(n)} \ : \ (\forall m, 0 \le m \le q(n))\\
&\qquad\qquad [\ \|\{y \in \mathrm{optsol}_\Pi(x) \ : \ y \cdot w_{l,1} = \cdots y \cdot w_{l,m} = 0\}\| \ne 1\ ]\}\\
&\ge 1 - (\tfrac{3}{4})^{3e(n)} \ = \ 1 - (\tfrac{27}{64})^{e(n)} \ \ge \ 1 - 2^{-e(n)}.
\end{aligned}$$

Thus, we have this theorem. ∎

4 Hardness of computing optimal solutions

In this section, we first give a sufficient condition for showing that for a given $NPCOP$ Π, OPTSOL_Π is $\le^{\mathrm{uniform-PF}}_{\mathrm{1-T}}$-hard for $\mathrm{PF}^{\mathrm{NP}}_{\mathrm{tt}}$. We use the following notions in the general result.

Definition 4.1 Let $\Pi = (op, D, S, R, c)$ be an optimization problem. Then, we define the *decision problem* L_Π *associated with* Π as follows:

$$L_\Pi = \{\langle x, k\rangle \ : \ x \in D, \ k \text{ is a non-negative integer, and } c^*(x) \ \theta \ k\},$$

where θ is $\le$ (less than or equal to) if $op = min$, and θ is $\ge$ (greater than or equal to) otherwise. Π is said to be *linearly paddable* if there exist two polynomial-time computable functions $f_1 : D \times D \to D$ and $f_2 : D \times D \times S \to S \times S$ such that

(a) for all x_1, $x_2 \in D$, $|f_1(x_1, x_2)| = O(|x_1| + |x_2|)$, and

(b) for all x_1, x_2, $x \in D$, and all $H \in \mathrm{OPTSOL}_\Pi$, if $x = f_1(x_1, x_2)$ and $f_2(x_1, x_2, H(x)) = \langle y_1, y_2\rangle$, then $y_1 \in \mathrm{optsol}_\Pi(x_1)$ and $y_2 \in \mathrm{optsol}_\Pi(x_2)$.

Intuitively speaking, we say that Π is linearly paddable if we can efficiently pad any two instances into one instance (by using the function f_1), and we can efficiently compute optimal solutions of the original two instances from any optimal solution of the padded instance (by using the function f_2). The condition (a) above guarantees that after we pad two instances into a single instance, the length of the padded instance is linear in the sum of the lengths of the original instances.

Theorem 4.1 Let $\Pi = (op, D, S, R, c)$ be a linearly paddable $NPCOP$ whose associated decision problem L_Π is NP-hard. Then, OPTSOL_Π is $\le^{\mathrm{uniform-PF}}_{\mathrm{1-T}}$-hard for $\mathrm{PF}^{\mathrm{NP}}_{\mathrm{tt}}$.

We next show that several well-known $NPCOP$'s are linearly paddable. For the graph-theoretic problems below, we suppose that each graph is encoded by its adjacency matrix, an n by n (0, 1)-matrix whose $(i,\ j)$-component is 1 if and only if the i-th vertex is connected to the j-th vertex in the graph, where n is the number of vertices in the graph and all vertices are assumed to be indexed by 1 through n.

Theorem 4.2 The following $NPCOP$'s are linearly paddable:

- MAXIMUM TWO SATISFIABILITY (MAX2SAT for short):
  - **Instance:** A CNF Boolean formula ϕ such that each clause of ϕ contains at most two literals.
  - **Output:** A truth assignment to the variables under which the maximum number of clauses become true.
- MAXIMUM CLIQUE (MAXCLIQUE for short):
  - **Instance:** An undirected graph G.
  - **Output:** A maximum clique of G.
- MINIMUM COLORING (MINCOLOR for short):
  - **Instance:** An undirected graph $G = (V, E)$.
  - **Output:** A partition $\langle U_1, U_2, \cdots, U_k\rangle$ of V such that k is the chromatic number of G and no two vertices u, v of G belong to the same U_i whenever $\{u,v\} \in E$. (Below, we simply call such a partition of V a *k-coloring* of G.)
- LONGEST PATH (LONGPATH for short):
  - **Instance:** An undirected graph G.
  - **Output:** A longest simple path in G.
- 0-1 INTEGER PROGRAMMING (01IP for short):
  - **Instance:** A 3-tuple $\langle A, B, C\rangle$ of a $(0,1)$-matrix and two $(0,1)$-vectors.
  - **Output:** A $(0,1)$-vector X maximizing $C^T X$ subject to $AX \leq B$.
- 0-1 TRAVELING SALESPERSON (01TSP for short):
  - **Instance:** An undirected complete graph G with weights 0 or 1 on the edges.
  - **Output:** A shortest traveling salesperson tour in G.

Proof. **(01TSP)** Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two complete graphs with weights 0 or 1 on their edges. We first construct a copy G_{c1} of G_1 and a copy G_{c2} of G_2. That is, there exists a one-to-one correspondence between vertices of G_1 (resp., G_2) and vertices of G_{c1} (resp., G_{c2}), and for any two vertices u and v in G_1 (resp., G_2), the weight on the edge $\{u,v\}$ in G_1 (resp., G_2) is the same as the weight on the edge $\{u_c, v_c\}$ in G_{c1} (resp., G_{c2}) if u_c and v_c are the vertices in G_{c1} (resp., G_{c2}) corresponding to u and v, respectively. Without loss of generality, we assume that V_1, V_2, V_{c1}, and V_{c2} are pairwise disjoint. Fix two vertices $v_0^{(c1)} \in V_{c1}$ and $v_0^{(c2)} \in V_{c2}$. We now introduce six new vertices $s^{(1)}$, $t_1^{(1)}$, $t_2^{(1)}$, $s^{(2)}$, $t_1^{(2)}$, $t_2^{(2)}$, and define $f_1(G_1, G_2)$ to be the graph $G = (V, E)$ such that

(1) $V = V_1 \cup V_2 \cup V_{c1} \cup V_{c2} \cup \{s^{(1)}, t_1^{(1)}, t_2^{(1)}, s^{(2)}, t_1^{(2)}, t_2^{(2)}\}$,

(2) $E = \{\{u,v\}\ : u$ and v are two distinct vertices in $V\}$, and

(3) olny the following edges are of weight 0 in G: $\{s^{(1)}, v_0^{(c1)}\}$, $\{t_1^{(1)}, t_2^{(1)}\}$, $\{s^{(2)}, v_0^{(c2)}\}$, $\{t_1^{(2)}, t_2^{(2)}\}$, all the edges of weight 0 in G_1, G_2, G_{c1}, or G_{c2}, all edges $\{u, t_1^{(1)}\}$ such that $u \in V_{c1}$ and the weight on $\{v_0^{(c1)}, u\}$ is 0, and all edges $\{u, t_1^{(2)}\}$ such that $u \in V_{c2}$ and the weight on $\{v_0^{(c2)}, u\}$ is 0.

Then, f_1 satisfies the condition (a) in the definition of linear paddability. Intuitively speaking, G contains four subgraphs G_1, G_2, G_{c1}, and G_{c2}, and each of G_1, G_2, G_{c1}, and G_{c2} will play a special role: G_{c1} (resp., G_{c2}) will be used to find a shortest traveling salesperson tour of cost 0 in G_1 (resp., G_2), if G_1 (resp., G_2) contains such a traveling salesperson tour; on the other hand, G_1 (resp., G_2) will be used to find a shortest traveling salesperson tour in G_1 (resp., G_2) when the cost of shortest traveling salesperson tours in G_1 (resp., G_2) is greater than 0. These will become clear in the following.

Let H be an arbitrary function in $\text{OPTSOL}_{01\text{TSP}}$. That is, $H(G)$ is a shortest traveling salesperson tour in G. Below, we show that we can compute a shortest traveling salesperson tour in G_1 from G_1, G_2, and $H(G)$ in time polynomial in $|G_1| + |G_2| + |H(G)|$. We first need to show two claims. For convenience, we define the *cost of a path or a cycle* P to be the sum of weights on the edges in P, and denote it by $c(P)$.

Claim 1 The cost of shortest traveling salesperson tours in G_{c1} is 0 if and only if $H(G)$ contains a path of cost 0 which starts at $s^{(1)}$, passes through exactly all vertices of G_{c1} and $t_1^{(1)}$, and terminates at $t_2^{(1)}$.

Proof of Claim 1 ($\Leftarrow$) Suppose $H(G)$ contains a path P of cost 0 which starts at $s^{(1)}$, passes through exactly all vertices of G_{c1} and $t_1^{(1)}$, and terminates at $t_2^{(1)}$. We first note that all edges in P have weight 0. Let P' be the path of cost 0 obtained by removing the start vertex $s^{(1)}$ and the end vertex $t_2^{(1)}$ from P. Since $\{s^{(1)}, v_0^{(c1)}\}$ (resp., $\{t_1^{(1)}, t_2^{(1)}\}$) is the unique edge of weight 0 adjacent to $s^{(1)}$ (resp., $t_2^{(1)}$) in G, the start vertex and the end vertex of P' must be $v_0^{(c1)}$ and $t_1^{(1)}$, respectively. Let u be the neighbor of $t_1^{(1)}$ in P'. Then, the weight on $\{u, v_0^{(c1)}\}$ is 0 in G_{c1}, because the weight on $\{u, v_0^{(c1)}\}$ is the same as the weight on $\{u, t_1^{(1)}\}$ which is 0 by our construction of G. Thus, if we remove $t_1^{(1)}$ from P' and add the edge $\{u, v_0^{(c1)}\}$ to P', then we obtain a traveling salesperson tour of cost 0 in G_{c1}. Furthermore, note that this traveling salesperson tour can be obtained in time polynomial in $|G_1| + |G_2| + |H(G)|$.

($\Rightarrow$) Assume, on the contrary, that the cost of shortest traveling salesperson tours in G_{c1} is 0 but $H(G)$ does not contain any path of cost 0 which starts at $s^{(1)}$, passes through exactly all vertices of G_{c1} and $t_1^{(1)}$, and terminates at $t_2^{(1)}$. Then, for some $l \geq 2$, $H(G)$ must be of the form: T_1, e_1, P_1, e_2, T_2, e_3, $\cdots$, P_l, e_{2l}, where each T_j is a path of $H(G)$ not including any vertex of $V_{c1} \cup \{s^{(1)}, t_1^{(1)}, t_2^{(1)}\}$, each P_j is a path of $H(G)$ including only vertices of $V_{c1} \cup \{s^{(1)}, t_1^{(1)}, t_2^{(1)}\}$, each e_{2j-1} is the edge in $H(G)$ connecting T_j and P_j, and each e_{2j} is the edge in $H(G)$ connecting P_j and T_{j+1} (let $T_{l+1} = T_1$). Since the cost of shortest traveling salesperson tours in G_{c1} is 0, there exists a traveling salesperson tour $T_{G_{c1}}$ of cost 0 in G_{c1}. Let u be one of the two neighbors of $v_0^{(c1)}$ in $T_{G_{c1}}$. We construct a path T_p from $T_{G_{c1}}$ by deleting $\{v_0^{(c1)}, u\}$ from $T_{G_{c1}}$ and adding three edges $\{s^{(1)}, v_0^{(c1)}\}$, $\{u, t_1^{(1)}\}$, and $\{t_1^{(1)}, t_2^{(1)}\}$ to $T_{G_{c1}}$. Then, it is easy to see that the cost of T_p is 0 and T_p passes through exactly all vertices in $V_{c1} \cup \{s^{(1)}, t_1^{(1)}, t_2^{(1)}\}$. From $H(G)$, we now construct another traveling salesperson tour T_G in G by first removing e_j's from $H(G)$, secondly using T_p to replace P_j's, and finally using $l+1$ edges (of weight ≤ 1) to connect T_j's and T_p so that they become a cycle. Then, we have that $c(T_G) \leq \sum_{j=1}^{l} c(T_j) + (l+1) + c(T_p) = \sum_{j=1}^{l} c(T_j) + (l+1)$. On the other hand, since the weight on each of e_1, $\cdots$, e_{2l} must be 1, we have that $c(H(G)) = \sum_{j=1}^{l} c(T_j) + 2l + \sum_{j=1}^{l} c(P_j)$. Thus, $c(H(G)) - c(T_G) \geq l - 1 \geq 1$. However, this contradicts the fact that $H(G)$ is a shortest traveling salesperson tour in G. **(End of Claim 1)**

Claim 2 If the cost of shortest traveling salesperson tours in G_1 is greater than 0, then we can compute a shortest traveling salesperson tour in G_1 from G_1, G_2, and $H(G)$ in time polynomial in $|G_1| + |G_2| + |H(G)|$.

Proof of Claim 2 Obviously, for some $l \geq 1$, $H(G)$ must be of the form: T_1, e_1, P_1, e_2, T_2, e_3, $\cdots$, P_l, e_{2l}, where each T_j is a path of $H(G)$ not including any vertex of G_1, each P_j is a path of $H(G)$ including only vertices of G_1, each e_{2j-1} is the edge in $H(G)$ connecting T_j and P_j, and each e_{2j} is the edge in $H(G)$ connecting P_j and T_{j+1} (let $T_{l+1} = T_1$). We now construct a traveling salesperson tour T_{G_1} in G_1 by picking P_j's out of $H(G)$ and using l edges (of weight ≤ 1) to connect these P_j's so that they consist of a cycle. Note that T_{G_1} can be obtained in time polynomial in $|G_1| + |G_2| + |H(G)|$. Below, we show that T_{G_1} is a shortest traveling salesperson tour in G_1.

Assume, on the contrary, that T_{G_1} is not a shortest traveling salesperson tour in G_1. Then, there must exist a traveling salesperson tour T'_{G_1} in G_1 shorter than T_{G_1}. Since the cost of shortest traveling salesperson tours in G_1 is greater than 0, there exists an edge e of weight 1 in T'_{G_1}. Let T''_{G_1} be the path obtained by removing the edge e from T'_{G_1}. We now construct a traveling salesperson tour T_G from $H(G)$ by first removing e_j's from $H(G)$, secondly using T''_{G_1} to replace P_j's, and finally using $l+1$ edges (of weight ≤ 1) to connect T_j's and T''_{G_1} so that they become a cycle. Then, we have that $c(T_G) \leq \sum_{j=1}^{l} c(T_j) + (l+1) + c(T''_{G_1}) = \sum_{j=1}^{l} c(T_j) + l + c(T'_{G_1}) < \sum_{j=1}^{l} c(T_j) + l + c(T_{G_1}) \leq \sum_{j=1}^{l} c(T_j) + l + (\sum_{j=1}^{l} c(P_j) + l) = \sum_{j=1}^{l} c(T_j) + \sum_{j=1}^{l} c(P_j) + 2l$. On the other hand, since the weight on each of $e_1, \cdots, e_{2l}$ must be 1, we have that $c(H(G)) = \sum_{j=1}^{l} c(T_j) + 2l + \sum_{j=1}^{l} c(P_j)$. Thus, $c(H(G)) - c(T_G) > 0$. However, this contradicts the fact that $H(G)$ is a shortest traveling salesperson tour in G. **(End of Claim 2)**

We now use Claim 1 and Claim 2 to compute in polynomial-time a shortest traveling salesperson tour in G_1 from G_1, G_2, and $H(G)$ as follows: if $H(G)$ contains a path of cost 0 which starts at $s^{(1)}$, passes through exactly all vertices of G_{c1} and $t_1^{(1)}$, and terminates at $t_2^{(1)}$, then we compute a shortest traveling salesperson tour in G_{c1} (equivalently, in G_1) as shown in the proof of Claim 1; otherwise, we compute a shortest traveling salesperson tour in G_1 as shown in the proof of Claim 2.

Claim 1 and Claim 2 also hold even if we replace G_1 with G_2 and replace G_{c1} with G_{c2} at the same time. Since the proofs for these are very similar to the above two, we omit the details. Thus, we can also compute in polynomial-time a shortest traveling salesperson tour in G_2 from G_1, G_2, and $H(G)$. Therefore, from G_1, G_2, and $H(G)$, we can compute in polynomial-time some shortest traveling salesperson tours in G_1 and G_2, respectively, and thus we complete the proof of the linear paddability of 01TSP. ∎

All decision problems associated with the $NPCOP$'s in Theorem 4.2 are well known to be NP-complete [1, 2, 3, 5, 7] (see [2] for a comprehensive reference). Thus, we have the following corollary.

Corollary 4.3 For any $NPCOP$ Π in Theorem 4.2, OPTSOL_Π is $\leq_{1-T}^{\text{uniform-PF}}$-hard for $\mathrm{PF}_{tt}^{\mathrm{NP}}$.

5 Conclusion

We have developed an approach to study the complexity of computing optimal solutions for $NPCOP$'s. To summarize, we have first shown that for any $NPCOP$ Π, there exists a polynomial-time bounded randomized algorithm which, given an instance x of Π, uses one free evaluation of parallel queries to an NP oracle set and outputs some optimal solution for x with very high probability. This result shows that computing optimal solutions for $NPCOP$'s is as easy as evaluating parallel queries to NP oracle sets. We have then defined the notion of linear paddability and have shown that if Π is a linearly paddable $NPCOP$ and its associated decision problem is NP-hard, then all functions in $\mathrm{PF}_{tt}^{\mathrm{NP}}$ are computable in polynomial-time with one free evaluation of an arbitrary function giving an optimal solution for any given instance of Π. We have finally shown the hardness of computing optimal solutions for several natural $NPCOP$'s whose associated decision problems are known to be NP-hard, by showing their linear paddability. Thus, we consider the linear paddability as one of the interesting properties of $NPCOP$'s; this property tells us that unbounded parallel queries to NP-complete sets can be embedded into a single instance of NP-hard $NPCOP$'s.

Some interesting questions still remain open. A natural question closely related to this work is whether $\mathrm{PF}_{tt}^{\mathrm{NP}}$-hardness of $NPCOP$'s as in this paper implies the linear paddability of the $NPCOP$'s. A typical $NPCOP$ for this question is LONGEST CYCLE, the problem of finding a longest cycle of a given undirected graph. It is not so hard to show that LONGEST CYCLE is $\mathrm{PF}_{tt}^{\mathrm{NP}}$-hard in the sense of this paper, but it seems slightly hard to show that the problem is linearly paddable. A relaxed version of linear paddability can be considered. For instance, we can consider *polynomial paddability* which is defined by removing the condition (a) in the definition of linear paddability, In fact, it is not so hard to show that LONGEST CYCLE is polynomially paddable. However, we have not been able to show that the polynomial paddability of $NPCOP$'s implies $\mathrm{PF}_{tt}^{\mathrm{NP}}$-hardness of the $NPCOP$'s, as in Theorem 4.1. As mentioned by Krentel [6], there are some $NPCOP$'s which have not been classified yet. Some typical $NPCOP$'s are BIN PACKING and EDGE-COLORING [4].

References

[1] S. Even, *Graph Algorithms*, Computer Science Press, 1979.

[2] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, Freeman, San Francisco, 1979.

[3] M. Garey, D. Johnson, and L. Stockmeyer, *Some Simplified NP-Complete Graph Problems*, Theoret. Comput. Sci., 1 (1976), pp. 237-267.

[4] I. Holyer, *The NP-Completeness of Edge-Coloring*, SIAM J. Comput., 10 (1981), pp. 718-720.

[5] R. Karp, *Reducibility among combinatorial problems*, in R. E. Miller and J. W. Thetcher (eds.), Complexity of Computer Computations (1972), Plenum Press, New York, pp. 85-113.

[6] M. W. Krentel, *The Complexity of Optimization Problems*, J. Comput. System Sci., 36 (1988), pp. 490-509.

[7] C. Papadimitriou and K. Steiglize, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, N.J., 1982.

[8] L. Valiant and V. Vazirani, *NP Is as Easy as Detecting Unique Solutions*, Theoret. Comput. Sci. 47 (1986), pp. 85-93.

[9] O. Watanabe and S. Toda, *Structural Analysis on the Complexity of Inverting Functions*, SIGAL International Symposium on Algorithms, accepted for presentation, 1990.