<table>
<thead>
<tr>
<th>Title</th>
<th>On the Complexity of Computing Optimal Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author(s)</td>
<td>CHEN, Zhi-Zhong; TODA, Seinosuke</td>
</tr>
<tr>
<td>Citation</td>
<td>数理解析研究所講究録 (1991), 754: 196-205</td>
</tr>
<tr>
<td>Issue Date</td>
<td>1991-06</td>
</tr>
<tr>
<td>URL</td>
<td><a href="http://hdl.handle.net/2433/82102">http://hdl.handle.net/2433/82102</a></td>
</tr>
<tr>
<td>Type</td>
<td>Departmental Bulletin Paper</td>
</tr>
<tr>
<td>Textversion</td>
<td>publisher</td>
</tr>
</tbody>
</table>
On the Complexity of Computing Optimal Solutions

Zhi-Zhong CHEN and Seinosuke TODA
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 PF_{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 II, there exists a polynomial-time bounded randomized algorithm which, given an instance of II, 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 PF_{NP}. To show the hardness results, we introduce a property of NPCOP's, called linear paddability, and we show a general result that if II is a linearly paddable NPCOP and its associated decision problem is NP-hard, then all functions in PF_{NP} are computable in polynomial-time with one free evaluation of an arbitrary function giving optimal solutions for instances of II. 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 PF_{NP}, the class of functions computable in polynomial-time with polynomial number of queries to NP oracle sets. The other is PF_{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 PF_{NP} and is as computationally hard as all functions in PF_{NP}, while computing
optimal costs for several other well-known \(NPOP\)'s falls into \(PF^{N\Pi}_{NP} \log\) and is as computationally hard as all functions in \(PF^{N\Pi}_{NP} \log\) [6]. Since \(PF^{NP} = PF^{N\Pi}_{NP} \log\) 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 \textit{combinatorial} optimization problem, or simply, an \textit{NPOP}). 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 \(NPOP\)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 \(NPOP\)s is finding a suitable approach. In order to investigate the complexity of computing the optimal cost for a given \(NPOP\), the approach was to investigate the complexity of the function which computes the optimal cost of \(x\) for any given instance \(x\) of \(\Pi\) [6]. Nevertheless, since an arbitrary instance of \(NPOP\)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 investigating the problem of computing optimal solutions for a given \(NPOP\) as a class of functions giving an optimal solution for any given instance of \(\Pi\) (we call such a function an \textit{optimal-solution function} of \(\Pi\)). When discussing the upper bound of the complexity of computing optimal solutions for a given \(NPOP\), we may only require an algorithm solving \(\Pi\) to compute some optimal-solution function of \(\Pi\). On the other hand, when discussing the lower bound of the complexity of computing optimal solutions for an \(NPOP\), we examine the relative complexity between all optimal-solution functions of \(\Pi\) and some complexity classes of functions. Our this approach enables us to observe that \(PF^{NP}_{\Pi}\), 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 \(NPOP\)s.

In Section 3, we describe a polynomial-time bounded randomized algorithm for any \(NPOP\) which given an instance \(x\) of \(\Pi\), 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 \(NPOP\)s.

In Section 4, we show that for several natural \(NPOP\)'s, any optimal-solution function of \(\Pi\) is at least as computationally hard as all functions in \(PF^{NP}_{\Pi}\). To show the hardness results, we introduce a notion called \textit{linear paddability}. Intuitively speaking, we say that an \(NPOP\) \(\Pi\) is linearly paddable if we can efficiently pad any two instances of \(\Pi\) into one instance of \(\Pi\) 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 \(NPOP\) \(\Pi\), we associate a decision problem of deciding whether the optimal cost of \(x\) is at least (or at most if \(\Pi\) is a minimization problem) \(k\) for a given pair \((x,k)\) of an instance and an integer. Then we show a general result that if \(\Pi\) is a linearly paddable \(NPOP\) and its associated decision problem is NP-hard, then all functions in \(PF^{NP}_{\Pi}\) are computable in polynomial-time with one free evaluation of any optimal-solution function of \(\Pi\). This general result provides a \textit{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 $\Sigma^*$. We denote by $|z|$ the length of a string $z$. The empty string is denoted by $\lambda$. For any finite set $A$, $|A|$ denotes the number of elements of $A$ and $\chi_A$ denotes the characteristic function of $A$. Let $A^n$ and $A^\infty$ 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 $\text{bin}(n)$ be a standard binary representation of non-negative integer $n$ over $\Sigma$, 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 $\Sigma^*$ that is polynomial-time computable and polynomial-time invertible. For strings $x$ and $y$, we denote the output of the pairing function by $(x, y)$; this notation is extended to denote any $k$-tuples for $k > 2$ in a usual manner.

An optimization problem $\Pi$ is a quintuple $(\text{op}, D, S, R, c)$, where

1. $\text{op} \in \{\max, \min\}$ 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 $\Pi$ a maximization problem if $\text{op} = \max$ 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 $\Pi$ is defined by

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

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

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

The objective in solving a given optimization problem $\Pi$ is to compute an optimal solution for any instance of $\Pi$. 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 $\Pi = (\text{op}, D, S, R, c)$ 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 $\Pi = (\text{op}, D, S, R, c)$ 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 $\Pi$ 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 \( \Pi \) 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 \( \Pi \), we may only require an algorithm solving \( \Pi \) to compute some optimal solution for any given instance of \( \Pi \). 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 PF\(^{NP} \). Thus, with each NPCOP \( \Pi = (op, D, S, R, c) \), we associate a class OPTSOL\(_\Pi \) of functions defined by

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

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 \( H \leq_{\text{uniform-PF}}^{\Pi} F \), 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 \( (f, g) \) a \( \leq_{\text{uniform-PF}}^{\Pi} \)-reduction of \( H \) to \( F \) (note that the reduction \((f, g) \) of \( H \) to \( F \) must be the same for all functions chosen from \( F \)). \( F \) is \( \leq_{\text{uniform-PF}}^{\Pi} \)-hard for \( H \) if every function in \( H \) is \( \leq_{\text{uniform-PF}}^{\Pi} \)-reducible to \( F \).

If we can show that some \( \text{OPTSOL}_\Pi \) is \( \leq_{\text{uniform-PF}}^{\Pi} \)-hard for \( \text{PF}^{NP} \), then we see that computing optimal solutions for the NPCOP \( \Pi \) is at least as computationally hard as computing the hardest functions in \( \text{PF}^{NP} \).

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 \( f(x) \in B \). PF\(^{NP} \) 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 \( x \), \( F(x) = e(x, \chi_A(y_1), \ldots, \chi_A(y_m)) \), where \( g(x) = (y_1, \ldots, y_m) \). More intuitively, a function \( F \) is in \( \text{PF}^{NP} \) if there exist a polynomial-time bounded deterministic oracle transducer (DOTM for short) \( N \) and a set \( A \in \text{NP} \) such that \( N_A \) computes \( F \) and \( N \) 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 \( \text{PF}^{NP} \), to describe an algorithm computing a function in \( \text{PF}^{NP} \).

### 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 \( (x_1 \land y_1) \oplus \cdots \oplus (x_n \land y_n) \), 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 \( \{0,1\}^n \),
\[
Prob\{w_1, \cdots, w_n \in \{0,1\}^n : \exists j, 0 \leq j \leq n \} [\{ y \in S : (\forall i, 1 \leq i \leq j) [y \cdot w_i = 0] \} ] = 1 \} \geq \frac{1}{4}.
\]

**Theorem 3.2** Let \( \Pi = (op, D, S, R, c) \) be an NPCOP and let \( e \) be any polynomial. Then, there exist a function \( G \in PF_{tt}^{NP} \) and a polynomial \( r \) such that for all \( x \in D \) with \( |x| = n \),
\[
Prob\{w \in \{0,1\}^{r(n)} : G(x, w) \in \text{optsol}_\Pi(x)\} \geq 1 - 2^{-e(n)}.
\]

**Proof** We consider only the case that \( \Pi \) 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:

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

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 \((x, 0), (x, 1), \cdots, (x, p(n))\) to the oracle set \( A \) and computing the largest integer \( k = c^*(x) \) such that \((x, k)\) 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 \( w_1w_2 \cdots w_{3e(n)} = w \), and for every \( l, 1 \leq l \leq 3e(n) \), let \( w_{l1}, \cdots, w_{lq(n)} \) be the strings in \( \{0,1\}^{q(n)} \) such that \( w_{l1} \cdots w_{lq(n)} = w_l \). 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 \( 0 \leq m \leq q(n) \),
\( a ) s_{i,l,0} = \chi_B((x, i, 1)) \chi_B((x, i, 2)) \cdots \chi_B((x, i, q(n))) \) and
\( b ) s_{i,l,m} = \chi_B((x, i, l, w_{l1}, \cdots, w_{lm})) \cdots \chi_B((x, i, q(n), w_{l1}, \cdots, w_{lm})) \) 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 $N^{A\oplus B}$. 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 $(x, w)$ can be realized as parallel queries to the oracle set $A \oplus B$. Thus, $G$ is in $\text{PF}_{tt}^{NP}$. To show that $G$ satisfies the theorem, we first show the following claim:

Claim Suppose that for some $l$ and $m$, $\|\{y \in \text{optsol}_1(x) : y \cdot w_{l,1} = \cdots = y \cdot w_{l,m} = 0\}\| = 1$. 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 $\{y \in \text{optsol}_1(x) : y \cdot w_{l,1} = \cdots = y \cdot w_{l,m} = 0\}$. Then we see, from the definition of $B$, that for all $1 \leq j \leq q(|x|)$, $\chi_B((x, k, j, w_{l,1}, \cdots, w_{l,m})) = 1$ if 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 $|x| = n$,

$\text{Prob}\{w \in \{0, 1\}^{r(n)} : G(x, w) \in \text{optsol}_1(x)\}$

$= \text{Prob}\{w \in \{0, 1\}^{r(n)} : (3l, m, 1 \leq l \leq 3e(n), 0 \leq m \leq q(n))(s_{c^*(x),l,m} \in \text{optsol}_1(x))\}$

$\geq \text{Prob}\{w \in \{0, 1\}^{r(n)} : (3l, m, 1 \leq l \leq 3e(n), 0 \leq m \leq q(n))$

$\|\{y \in \text{optsol}_1(x) : y \cdot w_{l,1} = \cdots = y \cdot w_{l,m} = 0\}\| = 1\}$

$\geq 1 - \prod_{l=1}^{2e(n)} \text{Prob}\{w_{l,1}, \cdots, w_{l,m} \in \{0, 1\}^{r(n)} : (\forall m, 0 \leq m \leq q(n))$

$\|\{y \in \text{optsol}_1(x) : y \cdot w_{l,1} = \cdots = y \cdot w_{l,m} = 0\}\| \neq 1\}$

$\geq 1 - (\frac{1}{4})^{2e(n)} = 1 - (\frac{1}{4})^{3e(n)} \geq 1 - 2^{-e(n)}$.

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 $NP_{COP}$ problem, $\text{OPTSOL}_1$ is $\leq_{tt}^{\text{uniform-PF}}$-hard for $\text{PF}_{tt}^{NP}$. 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 $L_\Pi$ associated with $\Pi$ as follows:

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

where $\theta$ is $\leq$ (less than or equal to) if $op = \text{min}$, and $\theta$ is $\geq$ (greater than or equal to) otherwise. $\Pi$ is said to be linearly paddable if there exist two polynomial-time computable functions $f_1 : D \times D \rightarrow D$ and $f_2 : D \times D \times S \rightarrow S \times S$ 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 $H \in \text{OPTSOL}_1$, if $x = f_1(x_1, x_2)$ and $f_2(x_1, x_2, H(x))$

$= (y_1, y_2)$, then $y_1 \in \text{optsol}_1(x_1)$ and $y_2 \in \text{optsol}_1(x_2)$.

Intuitively speaking, we say that $\Pi$ 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 $NP_{COP}$ whose associated decision problem $L_\Pi$ is NP-hard. Then, $\text{OPTSOL}_1$ is $\leq_{tt}^{\text{uniform-PF}}$-hard for $\text{PF}_{tt}^{NP}$.

We next show that several well-known $NP_{COP}$'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
Theorem 4.2 The following NPCOP's are linearly paddable:

- MAXIMUM TWO SATISFIABILITY (MAX2SAT for short):
  
  **Instance:** A CNF Boolean formula $\phi$ such that each clause of $\phi$ 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 $\langle U_1, U_2, \ldots, U_k \rangle$ 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 $(A, B, C)$ 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 $\{u, v\}$ in $G_{c1}$ (resp., $G_{c2}$) if $u$ and $v$ 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 $v^{(c1)}_0 \in V_{c1}$ and $v^{(c2)}_0 \in V_{c2}$. We now introduce six new vertices $s^{(i)}, t_1^{(i)}, t_2^{(i)}, s^{(2)}, t_1^{(2)}, t_2^{(2)}$, 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. only the following edges are of weight 0 in $G$: $\{s^{(i)}, v^{(c1)}_0\}$, $\{t_1^{(i)}, t_2^{(i)}\}$, $\{s^{(2)}, v^{(c2)}_0\}$, $\{t_1^{(2)}, t_2^{(2)}\}$, all the edges of weight 0 in $G_{c1}$, $G_{c2}$, $G_{c1}$, or $G_{c2}$, all edges $\{u, t_1^{(1)}\}$ such that $u \in V_{c1}$ and the weight on $\{v^{(c1)}_0, u\}$ is 0, and all edges $\{u, t_1^{(2)}\}$ such that $u \in V_{c2}$ and the weight on $\{v^{(c2)}_0, u\}$ 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 OPTSO19TSP. 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 $G_1$, $G_2$, 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 $t^{(1)}_{2}$. 

Proof of Claim 1 ($\Rightarrow$) 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 $t^{(1)}_{2}$. 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 $t^{(1)}_{2}$ from $P$. Since $(s^{(1)}, v_{0}^{(c1)})$ (resp., $(t^{(1)}_{1}, t^{(1)}_{2})$) is the unique edge of weight 0 adjacent to $s^{(1)}$ (resp., $t^{(1)}_{2}$) 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 $(u, v^{(c1)})$ is 0 in $G_{c1}$, because the weight on $(u, v^{(c1)})$ is the same as the weight on $u, t^{(1)}_{1}$ which is 0 by our construction of $G$. Thus, if we remove $t^{(1)}_{2}$ from $P'$ and add the edge $(u, v^{(c1)})$ 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 $|G_1| + |G_2| + |H(G)|$.

($\Leftarrow$) 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 $t^{(1)}_{2}$. Then, for some $l \geq 2$, $H(G)$ must be of the form: $T_1, e_1, p_1, e_2, T_2, e_3, \ldots, p_1, e_{2l}$, where each $T_j$ is a path of $H(G)$ not including any vertex of $V_{c1} \cup \{s^{(1)}, t^{(1)}_{1}, t^{(1)}_{2}\}$, each $P_j$ is a path of $H(G)$ including only vertices of $V_{c1} \cup \{s^{(1)}, t^{(1)}_{1}, t^{(1)}_{2}\}$, 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 $v^{(c1)}_{0}$ in $T_{G_{c1}}$. We construct a path $T_P$ from $T_{G_{c1}}$ by deleting $\{v^{(c1)}_{0}, u\}$ from $T_{G_{c1}}$ and adding three edges $\{s^{(1)}, v^{(c1)}_{0}\}$, $\{u, t^{(1)}_{1}\}$, and $\{t^{(1)}_{1}, t^{(1)}_{2}\}$ 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 $V_{c1} \cup \{s^{(1)}, t^{(1)}_{1}, t^{(1)}_{2}\}$. From $H(G)$, we now construct another traveling salesperson tour $T_O$ in $G$ by first removing each $e_j$ from $H(G)$, secondly using $T_P$ to replace $P_j$'s, and finally using $l + 1$ edges (of weight $\leq 1$) to connect $T_j$'s and $T_P$ so that they become a cycle. Then, we have that $c(T_O) \leq \sum_{j=1}^{l} c(T_j) + (l + 1) + c(T_P) = \sum_{j=1}^{l} c(T_j) + (l + 1)$. On the other hand, since the weight on each of $e_1, \ldots, e_{2l}$ 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, $c(H(G)) - c(T_O) \geq l - 1 \geq 1$. 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: $T_1, e_1, p_1, e_2, T_2, e_3, \ldots, p_l, e_{2l}$, 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 $\leq 1$) 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 $G_1$. 

8
Assume, on the contrary, that $T_0$ is not a shortest traveling salesperson tour in $G_1$. Then, there must exist a traveling salesperson tour $T_0'$ in $G_1$ shorter than $T_0$. Since the cost of shortest traveling salesperson tours in $G_1$ is greater than 0, there exists an edge $e$ of weight 1 in $T_0'$. Let $T_0''$ be the path obtained by removing the edge $e$ from $T_0$. We now construct a traveling salesperson tour $T_G$ from $H(G)$ by first removing $e_j$'s from $H(G)$, secondly using $T_0''$, to replace $P_j$'s, and finally using $l + 1$ edges (of weight $\leq 1$) to connect $T_j$'s and $T_0''$, so that they become a cycle. Then, we have that $c(T_G) \leq \sum_{j=1}^{l} c(T_j) + (l + 1) + c(T_0'') = \sum_{j=1}^{l} c(T_j) + l + c(T_0') \leq \sum_{j=1}^{l} c(T_j) + l + (\sum_{j=1}^{l} c(P_j) + l) = \sum_{j=1}^{l} c(T_j) + \sum_{j=1}^{l} c(P_j) + 2l$. On the other hand, since the weight on each of $e_1, \ldots, e_l$ 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 $G_1$, and $t^{(1)}$, then we compute a shortest traveling salesperson tour in $G_1$ (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 $NPCOP$ II in Theorem 4.2, OPTSOL$_n$ is $\leq_{\Pi}^{\text{uniform-PF}}$-hard for PP$^{\text{NP}}$.

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 $NPCOP$ II, there exists a polynomial-time bounded randomized algorithm which, given an instance $x$ of II, 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 II is a linearly paddable $NPCOP$ and its associated decision problem is NP-hard, then all functions in PP$^{\text{NP}}$ are computable in polynomial-time with one free evaluation of an arbitrary function giving an optimal solution for any given instance of II. 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 PP$^{\text{NP}}$-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 $PF_{tt}^{NP}$-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 $PF_{tt}^{NP}$-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


