Using Integer Programming to Identify Languages in NP\P

Jian Yang
Department of Industrial and Manufacturing Engineering
New Jersey Institute of Technology
Newark, NJ 07102
Email: yang@adm.njit.edu

December 2006

Abstract

We propose the concept of sub-n languages based on the existing definition of languages. Just as a language is a subset of all finite-length strings, a sub-n language is a subset of the first n strings. We show that conditions expressible in terms of sub-n languages serve as necessary and sufficient conditions for the P-NP conjecture, whereas the conditions themselves can be checked by solving an integer programming problem. Solutions to a sequence of these integer programming problems will help us find the most difficult languages in NP\P.

Keywords: The P-NP Conjecture, Turing Machine, Integer Programming.

1 Introduction

The P-NP conjecture that P $\neq$ NP, i.e., the set P of languages deterministically recognizable by a Turing machine in polynomial time is merely a proper subset of the set NP of languages nondeterministically recognizable by a Turing machine in polynomial time, is the corner stone of many computer science/operations research results. Most attempts to disprove or prove it relies on showing that a known language in NP, such as an NP-complete language, can or cannot be deterministically recognized in polynomial time. The conjecture remains unresolved so far.

We make an attempt on the conjecture towards a different direction by trying to identify the hardest-to-recognize languages in NP. To bypass the difficulty of there being an infinite number of languages, we concentrate on what we shall call sub-n languages, concerning only the first n strings in a lexicographical order of the alphabet. We can formulate two conditions revolving around asymptotic properties of sub-n languages that turn out to be, respectively, the necessary and sufficient conditions for the P-NP conjecture.

We can dissolve both asymptotic conditions into questions of the following nature: for arbitrary positive integers r, n, t_N, and t_D, whether or not there are sub-n languages nondeterministically recognizable in t_N time but not deterministically recognizable in t_D time, by a machine employing a program that is no more than $r + 1$ in length. Following the tradition of Cook [5], we use binary variables to record the dynamics of a Turing machine. After introducing additional binary variables for describing, among other things, the nondeterministic and deterministic programs, we can devise an integer programming formulation for answering this type of questions.

Many attempts have been made to show the impossibility of recognizing (solving) certain NP-complete languages (problems) by certain broad class of algorithms in polynomial time. Chvatal [3] showed that it takes a certain class of algorithms on average an exponential time to solve the optimization version of the independent set problem when problem instances follow a certain distribution. Chvatal [4] and McDiarmid [10] achieved results in similar spirits for optimization versions of respectively, the knapsack and graph k-colorability problems.

There have also been efforts in building classes of languages that are contained by NP, in the hope that proving that any such class is different from NP or outside of P would be easier than directly proving P $\neq$ NP. Such classes include the class R proposed by Gill [7] and Adleman and Manders [1], and the class of γ-complete languages proposed by Adleman and Manders [1], containing languages that can be reduced from any language in NP using a nondeterministic machine.

The conjecture can be proved when any language in NP can be shown to require more than polynomial time to recognize by an apparatus more powerful than a Turing machine.

A Boolean circuit is such an apparatus. But so far, super-polynomial lower bounds were only achieved for restricted versions of the circuit. Results in this vein include Razborov [13] and Yao [15].

Many concepts related to the P-NP conjecture have been proposed. These include, for instance, the polynomial hierarchy (Meyer and Stockmeyer [11] and Stockmeyer [14]), classes PP and BPP that are connected to randomized algorithms (Gill [7]), the random NP-completeness (Levin [9]), the class D^{P} of all languages that can be expressed as differences of two languages in NP (Papadimitriou and Yannakakis [12]), the class IP of languages recognizable by interactive proof systems (Goldwasser, Micali, and Rackoff [8] and Babai and Moran [2]).

As we strive to identify the most difficult languages in NP\P, our approach is fairly different from the ones listed in the above. We organize the paper as follows: we introduce necessary background information in Section 2, lay out our main ideas and concepts in Section 3, present parameters needed for the formulation in Section 4, elaborate on the integer programming formulation in Section 5, and finally in Section 6, point out how the formulation should be used in conjunction with our earlier necessary and sufficient conditions.

2 Background

We introduce our notation while presenting known concepts as have been laid out in Garey and Johnson [6]. We fix our set of symbols at $\hat{\Gamma} = \{1, 2\}$, and let $\hat{\Gamma}^*$ be the set of finite-length strings made up of symbols from $\hat{\Gamma}$. We rank all finite strings in $\hat{\Gamma}^*$ in the order of

$$\begin{array}{l} \bar{\mathbf{x}}_1 = \text{blank}, \ \ \bar{\mathbf{x}}_2 = (1), \ \ \bar{\mathbf{x}}_3 = (2), \ \ \bar{\mathbf{x}}_4 = (11), \ \ \bar{\mathbf{x}}_5 = (12), \ \ \bar{\mathbf{x}}_6 = (21), \ \ \bar{\mathbf{x}}_7 = (22), \\ \bar{\mathbf{x}}_8 = (111), \ \ \bar{\mathbf{x}}_9 = (112), \ \ \cdots, \ \bar{\mathbf{x}}_{15} = (222), \ \ \bar{\mathbf{x}}_{16} = (1111), \ \ \cdots \ \ \cdots. \end{array} \tag{1}$$

Throughout this paper, we shall use $|\cdot|$ to stand for the length of a string and also the cardinality of a set. Note that, for $n = 1, 2, ...$, we have $|\bar{\mathbf{x}}_{2^n}| = |\bar{\mathbf{x}}_{2^n+1}| \cdots = |\bar{\mathbf{x}}_{2^{n+1}-1}| = n$. For $i = 1, 2, ...$ and $j = 1, ..., |\bar{\mathbf{x}}_i|$, we let $\bar{x}_i(j)$ stand for the jth symbol of the ith string. For instance, we have

$$|\bar{\mathbf{x}}_{17}| = 4, \ \ \text{and} \ \ \bar{\mathbf{x}}_{17} = (\bar{x}_{17}(1)\bar{x}_{17}(2)\bar{x}_{17}(3)\bar{x}_{17}(4)) = (1112). \tag{2}$$

We may let "0" stand for the blank symbol. For convenience, we let $\hat{\Gamma}^0 = \{0\} \cup \hat{\Gamma} = \{0, 1, 2\}$. Let $\mathcal{P}^* = \{0, 1\}^{\hat{\Gamma}^*}$ be the collection of all subsets of $\hat{\Gamma}^*$. A language π^* is identified with a subset of $\hat{\Gamma}^*$, i.e., a member of $\mathcal{P}^*$.

We can use a pair $(r, \mathbf{A})$ to describe a program employed by a Turing machine. Note that the latter is also in possession of a read-write head and a tape with cells $0, \pm 1, \pm 2, ...$

In the pair, r is an integer no less than 2, and we may suppose the set of program states to be $\hat{Q} = \{0, 1, ..., r\}$, with 2 standing for the *starting* state, 1 for the state of *halted at acceptance*, and 0 for the state of *halted at rejection*; while $\mathbf{A}$ is a mapping of the nature $\hat{Q} \times \hat{\Gamma}^0 \to \hat{Q} \times \hat{\Gamma}^0 \times \{-1, +1\}$. The mapping $\mathbf{A} = (A^q, A^s, A^\Delta)$ dictates that,

* when at time t, the program state is $q_t \in \hat{Q}$, read-write head position is $h_t = 0, \pm 1, \pm 2, ...$, and the tape content is $\mathbf{s}_t = (s_t(0), s_t(-1), s_t(+1), ...)$,

* then at time $t+1$, the program state will be $q_{t+1} = A^q(q_t, s_t(h_t))$, the tape content $\mathbf{s}_{t+1}$ will be such that $s_{t+1}(h_t) = A^s(q_t, s_t(h_t))$ while $s_{t+1}(p) = s_t(p)$ for $p \neq h_t$, and the read-write head position will be $h_{t+1} = h_t + A^\Delta(q_t, s_t(h_t))$.

We may denote the above state evolution by

$$(q_{t+1}, h_{t+1}, \mathbf{s}_{t+1}) = \mathbf{A}(q_t, h_t, \mathbf{s}_t). \tag{3}$$

So that the halting decisions will stay once reached, we require that $(A^q, A^s)(1, s) = (1, s)$ and $(A^q, A^s)(0, s) = (0, s)$ for any $s \in \hat{\Gamma}^0$.

For program $(r, \mathbf{A})$, string $\mathbf{x} \in \hat{\Gamma}^*$, language $\pi^* \in \mathcal{P}^*$, and integer $t =\mid \mathbf{x} \mid, \mid \mathbf{x} \mid +1, ...$, we say that a Turing machine employing program $(r, \mathbf{A})$ discerns string $\mathbf{x}$ for language π^* in t time, when starting with $q_0 = 2$, $h_0 = 1$, and $\mathbf{s}_0$ being such that $s_0(p) = x(p)$ for $p = 1, ..., \mid \mathbf{x} \mid$, and following the dynamics of (3), we will have $q_t = 1$ if and only if $\mathbf{x} \in \pi^*$. Given integer-valued function $f(n)$ with $f(n) = n, n+1, ...$, we say a machine employing $(r, \mathbf{A})$ recognizes language π^* in $f(n)$ time, when for any string $\mathbf{x} \in \hat{\Gamma}^*$, a machine employing $(r, \mathbf{A})$ will discern the string for π^* in $f(\mid \mathbf{x} \mid)$ time.

We consider the above procedure as deterministic when

d1) the remaining tape contents in the beginning must be string-independent, say uniformly empty: $s_0(p) = 0$ for $p = 0, -1, ..., -f(\mid \mathbf{x} \mid)$ and $p =\mid \mathbf{x} \mid +1, \mid \mathbf{x} \mid +2, ..., f(\mid \mathbf{x} \mid) + 1$; and,

d2) we can also tell if the string is unacceptable: $q_t = 0$ when $\mathbf{x} \in \hat{\Gamma}^* \setminus \pi^*$;

while we regard the procedure as nondeterministic when

n) the remaining tape contents in the beginning can be string-dependent: $s_0(p)$ can depend on $\mathbf{x}$ for $p = 0, -1, ..., -f(\mid \mathbf{x} \mid)$ and $p =\mid \mathbf{x} \mid +1, \mid \mathbf{x} \mid +2, ..., f(\mid \mathbf{x} \mid) + 1$.

We say a machine employing $(r, \mathbf{A})$ recognizes language π^* in polynomial time, when there exist positive integers a and n', so that for any $n = n', n'+1, ...$, we have $f(n) \leq n^a$ for the time $f(n)$ needed by a machine.

To be consistent with our own notational system, we use $\mathcal{P}^*_D$ to denote what is traditionally termed as P, the subset of $\mathcal{P}^*$ that contains languages deterministically recognizable in polynomial time by a machine employing some program, and use $\mathcal{P}^*_N$ to denote what is traditionally termed as NP, the subset of $\mathcal{P}^*$ that contains languages nondeterministically recognizable in polynomial time by a machine employing some program. It is obvious that

$\mathcal{P}_D^* \subset \mathcal{P}_N^*$. However, whether or not $\mathcal{P}_D^* = \mathcal{P}_N^*$ remains unresolved. Most evidences point to the negative answer, the celebrated P-NP conjecture, that

$$\mathcal{P}_N^* \setminus \mathcal{P}_D^* \neq \emptyset, \text{ or equivalently, } \mid \mathcal{P}_N^* \setminus \mathcal{P}_D^* \mid \geq 1. \tag{4}$$

3 Concepts

There are two potential approaches to prove (4). One is to show that a known languages π^* in $\mathcal{P}_N^*$ cannot be recognized deterministically in polynomial time; another is to construct a language π^* that is hard enough to belong to $\mathcal{P}_N^* \setminus \mathcal{P}_D^*$. Here we make an effort in the second direction.

First, we can say something about lengths of programs needed for our Turing machine. Given $r = 2, 3, ...$, we may use $\mathcal{P}_D^*(r)$ and $\mathcal{P}_N^*(r)$ to denote languages $\pi^* \in \mathcal{P}^*$ that are recognizable in polynomial time in, respectively, the deterministic and nondeterministic fashion, by a machine employing a program no more than $r + 1$ in length. Apparently, both $\{\mathcal{P}_D^*(r) \mid r = 1, 2, ...\}$ and $\{\mathcal{P}_N^*(r) \mid r = 1, 2, ...\}$ are ascending set sequences that converge to $\mathcal{P}_D^*$ and $\mathcal{P}_N^*$, respectively. We can arrive to the following conclusion, linking the P-NP conjecture to the $\mathcal{P}_D^*(r)$'s and $\mathcal{P}_N^*(r)$'s.

Theorem 1 *$\mathcal{P}_N^* \setminus \mathcal{P}_D^* \neq \emptyset$ if and only if there exist a positive integer r' and a language $\pi^* \in \mathcal{P}^*$, such that for any $r = r', r' + 1, ...,$ it is true that $\pi^* \in \mathcal{P}_N^*(r) \setminus \mathcal{P}_D^*(r)$.*

Proof: For the "only if" part, we need only to let π^* be a member of the supposedly nonempty $\mathcal{P}_N^* \setminus \mathcal{P}_D^*$, and let r' be the length of the shortest program that can be employed by a machine to recognize π^* nondeterministically in polynomial time. For the "if" part, we note that such a π^* in existence is definitely a member of $\mathcal{P}_N^* \setminus \mathcal{P}_D^*$, as it can be nondeterministically recognized in polynomial time by a machine employing a program no more than r' in length and yet cannot be recognized deterministically in polynomial time by a machine no matter what program has been employed. ∎

One obvious difficulty with our approach is that there are an infinite and indeed uncountable number of languages. To circumvent it, for $n = 1, 2, ...$, we may limit our consideration to the first n strings: $\bar{\mathbf{x}}_1, ..., \bar{\mathbf{x}}_n$, which we may term as sub-n strings. For any language $\pi^* \in \mathcal{P}^*$, we may define what we shall call a sub-n language (n, π^n), where $\pi^n = \pi^* \cap \{\bar{\mathbf{x}}_1, \cdots, \bar{\mathbf{x}}_n\}$. Note that the set $\mathcal{P}^n$ of all sub-n languages has 2^n members: $(n, \emptyset)$, $(n, \{\bar{\mathbf{x}}_1\})$, $\cdots$ $\cdots$, $(n, \{\bar{\mathbf{x}}_1, \cdots, \bar{\mathbf{x}}_n\})$. Therefore, for any $(n, \alpha^n) \in \mathcal{P}^n$ and any $\beta^* \in \mathcal{P}^*$, it will be natural for us to say that (n, α^n) is the sub-n representation of β^* when $\alpha^n \subset \beta^*$. In this regard, any

sub-n language is a representation of all languages that contain the same set of sub-n strings. When only sub-n strings are involved, two full-scale languages in $\mathcal{P}^*$ will be distinguishable only when their sub-n representations are different. For positive integers n and m satisfying $n \leq m$, we say that sub-n language $(n, \alpha^n) \in \mathcal{P}^n$ and sub-m language $(m, \beta^m) \in \mathcal{P}^m$ are consistent with each other if they are respectively, the sub-n and sub-m representations of one same full-scale language $\pi^* \in \mathcal{P}^*$, or equivalently, if $\alpha^n \subset \beta^m$.

Suppose $r = 2, 3, ...$ has been fixed. Now given positive integer t, we may use $\mathcal{P}^n_D(r,t)$ and $\mathcal{P}^n_N(r,t)$ to denote all sub-n languages $(n, \pi^n) \in \mathcal{P}^n$ that can be recognized by a machine employing a program no more than $r+1$ in length in t time in respectively, the deterministic and nondeterministic fashion. Suppose for any positive integers t_D and t_N, we can identify $\mathcal{P}^n_N(r,t_N)$ and $\mathcal{P}^n_N(r,t_N) \setminus \mathcal{P}^n_D(r,t_D)$ in finite time, then we will be able to draw helpful conclusions about $\mathcal{P}^*_N(r) \setminus \mathcal{P}^*_D(r)$.

First, whether or not $\mid \mathcal{P}^n_N(r,t_N) \setminus \mathcal{P}^n_D(r,t_D) \mid \geq 1$ contains valuable information. Indeed, note that, for $\pi^* \in \mathcal{P}^*$, we have $\pi^* \in \mathcal{P}^*_{D(N)}(r)$ if and only if, there exist positive integers a and n', so that for any $n = n', n'+1, ...,$ we have $(n, \pi^n) \in \mathcal{P}^n_{D(n)}(r, n^a)$. We have the following necessary condition for the nonemptyness of $\mathcal{P}^*_N(r) \setminus \mathcal{P}^*_D(r)$.

Theorem 2 *If* $\mid \mathcal{P}^*_N(r) \setminus \mathcal{P}^*_D(r) \mid \geq 1$*, then for positive integer* a_N *that is large enough, we will have* $\limsup_{n \to +\infty} \mid \mathcal{P}^n_N(r, n^{a_N}) \setminus \mathcal{P}^n_D(r, n^{a_D}) \mid \geq 1$ *for any positive integer* a_D*.*

Proof: According to the hypothesis, there is a full-scale language π^* that belongs to $\mathcal{P}^*_N(r) \setminus \mathcal{P}^*_D(r)$.

Since $\pi^* \in \mathcal{P}^*_N(r)$, there are positive integers $a(\pi^*)$ and $n'(\pi^*)$, so that for any $n = n'(\pi^*), n'(\pi^*)+1, , ...,$ the sub-n representation (n, π^n) can be nondeterministically recognized in n^a time by a machine employing a program no more than $r+1$ in length. Hence, for any $a_N = a(\pi^*), a(\pi^*)+1, ...$ and $n = n'(\pi^*), n'(\pi^*)+1, ...,$ it is true that (n, π^n) belongs to $\mathcal{P}^n_N(r, n^{a_N})$.

However, since $\pi^* \in \mathcal{P}^* \setminus \mathcal{P}^*_D(r)$, we know that for any positive integers a_D and $n'' = n'(\pi^*), n'(\pi^*)+1, ...,$ there exists some integer $n = n'', n''+1, ...,$ such that $(n, \pi^n) \in \mathcal{P}^n \setminus \mathcal{P}^n_D(r, n^{a_D})$.

From the above, we see that, for any integer a_N satisfying $a_N = a(\pi^*), a(\pi^*)+1, ...,$ and any integers a_D and n'', there will exist some integer $n = n'(\pi^*) \vee n'', n'(\pi^*) \vee n''+1, ...,$ such that $\mid \mathcal{P}^n_N(r, n^{a_N}) \setminus \mathcal{P}^n_D(r, n^{a_D}) \mid \geq 1$. ∎

Second, the identification of a sequence of hard sub-n languages that are consistent with each other will induce us to confirm the nonemptyness of $\mathcal{P}^*_N(r) \setminus \mathcal{P}^*_D(r)$.

Theorem 3 *Suppose there exist positive integers a_N and n', as well as a sequence $\{(n, \pi^n) \in \mathcal{P}_N^n(r, n^{a_N}) \mid n = n', n'+1, ...\}$, such that 1) for any $k, l = n', n'+1, ...$, sub-k language π^k and sub-l language π^l are consistent with each other, and 2) for any positive integer a_D and $n'' = n', n'+1, ...$, there exists some $n = n'', n''+1, ...$, such that the sub-n language (n, π^n) belongs to $\mathcal{P}_N^n(r, n^{a_N}) \setminus \mathcal{P}_D^n(r, n^{a_D})$. Then, $\mathcal{P}_N^*(r) \setminus \mathcal{P}_D^*(r) \neq \emptyset$.*

Proof: Due to 1), we can construct full-scale language $\pi^* \in \mathcal{P}^*$ by taking the limit of the ascending set sequence $\{\pi^n \mid n = n', n'+1, ...\}$: $\pi^* = \bigcup_{n=n'}^{+\infty} \pi^n$.

We have $\pi^* \in \mathcal{P}_N^*(r)$ from the fact that $(n, \pi^n) \in \mathcal{P}_N^n(r, n^{a_N})$ for $n = n', n'+1, ...$ On the other hand, since for any positive integer a_D and $n'' = n', n'+1, ...$, we can find $n = n'', n''+1, ...$, such that $(n, \pi^n) \in \mathcal{P}^n \setminus \mathcal{P}_D^n(r, n^{a_D})$, we have $\pi^* \in \mathcal{P}^* \setminus \mathcal{P}_D^*(r)$. Therefore, $\pi^* \in \mathcal{P}_D^*(r) \setminus \mathcal{P}_D^*(r)$, and hence the set cannot be empty. ∎

We can develop an integer programming problem, the solving of which will provide us not only constitutions of sets $\mathcal{P}_N^n(r, t_N)$ and $\mathcal{P}_N^n(r, t_N) \setminus \mathcal{P}_D^n(r, t_D)$, but also the numbers $z_N^n(r, t_N) = \mid \mathcal{P}_N^n(r, t_N) \mid$, $z_{ND}^n(r, t_N, t_D) = \mid \mathcal{P}_N^n(r, t_N) \cap \mathcal{P}_D^n(r, t_D) \mid$, and $\Delta z_{ND}^n(r, t_N, t_D) = z_N^n(r, t_N) - z_{ND}^n(r, t_N, t_D) = \mid \mathcal{P}_N^n(r, t_N) \setminus \mathcal{P}_D^n(r, t_D) \mid$.

4 Parameters

Suppose $n = 1, 2, ...$ has been given. We use $3n\lfloor \log_2 n \rfloor$ binary parameters $\bar{X}^n[i, p, k]$ for $i = 1, ..., n$, $p = 1, ..., \lfloor \log_2 n \rfloor$, and $k = 0, 1, 2$ to describe sub-n strings. For $p = 1, ..., \mid \bar{\mathbf{x}}_i \mid$, we ensure that $\bar{X}^n[i, p, \bar{x}_i(p)] = 1$, while for $p = \mid \bar{\mathbf{x}}_i \mid +1, ..., \lfloor \log_2 n \rfloor$, we ensure that $\bar{X}^n[i, p, 0] = 1$. So that the parameters will serve as indicators, we ensure that

$$\sum_{k=0,1,2} \bar{X}^n[i, p, k] = 1, \quad \forall i = 1, ..., n, \ p = 1, ..., n. \tag{5}$$

We use $n2^n$ binary parameters $\bar{P}^n[m, i]$ for $m = 1, ..., 2^n$ and $i = 1, ..., n$ to describe sub-n languages.

For example, when $n = 8$, the eight $\bar{X}^8[i]$'s are:

$$\bar{X}^8[1] = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \text{ standing for the blank string,} \tag{6}$$

$$\bar{X}^8[2] = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \text{ standing for string (1),} \tag{7}$$

$$\bar{X}^8[3] = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \text{ standing for string (2),} \tag{8}$$

$$\bar{X}^8[4] = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \text{ standing for string (11),} \tag{9}$$

$$\bar{X}^8[5] = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \text{ standing for string (12),} \tag{10}$$

$$\bar{X}^8[6] = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \text{ standing for string (21),} \tag{11}$$

$$\bar{X}^8[7] = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix}, \text{ standing for string (22),} \tag{12}$$

$$\bar{X}^8[8] = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \text{ standing for string (111);} \tag{13}$$

while the 256 $\bar{P}^8[m]$'s are:

$$\bar{P}^8[1] = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \text{ representing the empty language,} \tag{14}$$

$$\bar{P}^8[2] = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \text{ representing the language containing only string (111),} \tag{15}$$

$$\cdots \quad \cdots,$$

$$\bar{P}^8[86] = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}, \text{ representing the language containing strings (1), (11), (21), and (111),} \tag{16}$$

$$\cdots \quad \cdots,$$

$$\bar{P}^8[171] = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}, \text{ representing the language containing string (1), (11), (21), and (111),} \tag{17}$$

$$\cdots \quad \cdots,$$

$$\bar{P}^8[255] = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}, \text{ representing the language containing all strings but string (111),} \tag{18}$$

$$\bar{P}^8[256] = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}, \text{ representing the language of all strings.} \tag{19}$$

5 Formulation

Given r, we call a program no more than $r+1$ in length a feasible program. When n, t_N, and t_D are further given, the (nonlinear) binary integer programming problem, (IP), is nominally that of maximizing the sum of 1) the number of sub-n languages that can be nondeterministically recognized in t_N time by a machine employing a feasible program, and 2) the number of sub-n languages among the above that are further deterministically recognizable in t_D time by a machine employing a feasible program. The maximization will guarantee that a feasible program falling into either of the above categories will be identified as long as it exists. This will in turn ensure the accuracy of our counting.

We always let language index m range in $1, ..., 2^n$, string index l range in $1, ..., n$, program states j and j' range in $0, ..., r$, tape contents k and k' range in $0, 1, 2$, and head displacement d range in $-1, +1$. For our decision variables concerning the nondeterministic solving, we let time i range in $0, ..., t_N$ and head position p range in $-t_N, ..., 0, ..., t_N + 1$; while for our decision variables concerning the deterministic solving, we let time i range in $0, ..., t_D$ and head position p range in $-t_D, ..., 0, ..., t_D + 1$.

The following are our binary decision variables.

$A_{N(D)}[m, j, k, j', k', d]$, whose being 1 or 0 indicates: in the feasible program for recognizing the mth sub-n language nondeterministically (deterministically), whether or not program state j and tape content k will lead to program state j', tape content k', and head displacement d;

$B_{N(D)}[m]$, whose being 1 or 0 indicates: for the mth sub-n language, whether or not there is a feasible program that recognizes it nondeterministically (deterministically) in t_N time;

$Q_{N(D)}[m, l, i, j]$, whose being 1 or 0 indicates: for the mth sub-n language and the lth sub-n string, and under the feasible program used for recognizing the language nondeterministically (deterministically), whether the program state is j at time i;

$H_{N(D)}[m, l, i, p]$, whose being 1 or 0 indicates: for the mth sub-n language and the lth sub-n string, and under the feasible program used for recognizing the language nondeterministically (deterministically), whether the read-write head position is p at time i;

$S_{N(D)}[m, l, i, p, k]$, whose being 1 or 0 indicates: for the mth sub-n language and the lth sub-n string, and under the feasible program used for recognizing the language nondeterministically (deterministically), whether the tape content at position p is k at time i.

Note that our $Q_{N(D)}$, $H_{N(D)}$, and $S_{N(D)}$ are similar to variables used by Cook [5] to describe a Turing machine's state evolution over time.

To ensure that, for either the nondeterministic or the deterministic program, one and

only one action will be taken in each step, we impose

$$\begin{array}{l}\sum_{j'=0}^{r}\sum_{k'=0,1,2}\sum_{d=-1,+1}A_N[m,j,k,j',k',d]=1,\\ \quad\forall m=1,...,2^n,\ j=0,...,r,\ k=0,1,2;\end{array} \tag{20}$$

$$\begin{array}{l}\sum_{j'=0}^{r}\sum_{k'=0,1,2}\sum_{d=-1,+1}A_D[m,j,k,j',k',d]=1,\\ \quad\forall m=1,...,2^n,\ j=0,...,r,\ k=0,1,2.\end{array} \tag{21}$$

To make sure that both nondeterministic and deterministic programs will stay at halting states once reaching there, we impose

$$\begin{array}{l}A_N[m,0,k,0,k,+1]+A_N[m,0,k,0,k,-1]\\ \quad=A_N[m,1,k,1,k,+1]+A_N[m,1,k,1,k,-1]=1,\\ \quad\forall m=1,...,2^n,\ k=0,1,2;\end{array} \tag{22}$$

$$\begin{array}{l}A_D[m,0,k,0,k,+1]+A_D[m,0,k,0,k,-1]\\ \quad=A_D[m,1,k,1,k,+1]+A_D[m,1,k,1,k,-1]=1,\\ \quad\forall m=1,...,2^n,\ k=0,1,2.\end{array} \tag{23}$$

To guarantee that, for any string, under either the nondeterministic or the deterministic program, there will be one and only one program state at each step, we impose

$$\sum_{j=0}^{r}Q_N[m,l,i,j]=1,\quad \forall m=1,...,2^n,\ l=1,...,n,\ i=0,...,t_N; \tag{24}$$

$$\sum_{j=0}^{r}Q_D[m,l,i,j]=1,\quad \forall m=1,...,2^n,\ l=1,...,n,\ i=0,...,t_D. \tag{25}$$

To guarantee that, for any string, both nondeterministic and deterministic programs will start at state 2, we impose

$$Q_N[m,l,0,2]=1\quad \forall m=1,...,2^n,\ l=1,...,n; \tag{26}$$

$$Q_D[m,l,0,2]=1\quad \forall m=1,...,2^n,\ l=1,...,n. \tag{27}$$

To ensure that, for the mth sub-n language, the feasible nondeterministic program will accept those and only those strings belonging to the language whenever $B_N[m]=1$, and the feasible deterministic program will come to the right conclusions for all strings whenever $B_D[m]=1$, we impose

$$\begin{array}{l}1-Q_N[m,l,t_N,1]\geq B_N[m](1-\bar{P}^n[m,l])\text{ and }Q_N[m,l,t_N,1]\geq B_N[m]\bar{P}^n[m,l],\\ \quad\forall m=1,...,2^n,\ l=1,...,n;\end{array} \tag{28}$$

$$\begin{array}{l}Q_D[m,l,t_D,0]\geq B_D[m](1-\bar{P}^n[m,l])\text{ and }Q_D[m,l,t_D,1]\geq B_D[m]\bar{P}^n[m,l],\\ \quad\forall m=1,...,2^n,\ l=1,...,n.\end{array} \tag{29}$$

To guarantee that, for any string, under either the nondeterministic or the deterministic program, the read-write head will be at one and only one position at any time, we impose

$$\sum_{p=-t_N}^{t_N+1} H_N[m,l,i,p] = 1, \quad \forall m = 1,...,2^n,\ l = 1,...,n,\ i = 0,...,t_N; \tag{30}$$

$$\sum_{p=-t_D}^{t_D+1} H_D[m,l,i,p] = 1, \quad \forall m = 1,...,2^n,\ l = 1,...,n,\ i = 0,...,t_D. \tag{31}$$

To guarantee that, for any string, under either the nondeterministic or deterministic program, the read-write head will start at position 1, we impose

$$H_N[m,l,0,1] = 1, \quad \forall m = 1,...,2^n,\ l = 1,...,n; \tag{32}$$

$$H_D[m,l,0,1] = 1, \quad \forall m = 1,...,2^n,\ l = 1,...,n. \tag{33}$$

To guarantee that, for any string, under either the nondeterministic or the deterministic program, there will be one and only one tape content at any position at any time, we impose

$$\begin{aligned} &\textstyle\sum_{k=0,1,2} S_N[m,l,i,p,k] = 1, \\ &\quad \forall m = 1,...,2^n,\ l = 1,...,n,\ i = 0,...,t_N,\ p = -t_N,...,0,...,t_N+1; \end{aligned} \tag{34}$$

$$\begin{aligned} &\textstyle\sum_{k=0,1,2} S_D[m,l,i,p,k] = 1, \\ &\quad \forall m = 1,...,2^n,\ l = 1,...,n,\ i = 0,...,t_D,\ p = -t_D,...,0,...,t_D+1. \end{aligned} \tag{35}$$

To guarantee that, for any string, under the feasible nondeterministic program, the starting tape contents in positions 1 to n match the string; however, specification for tape contents elsewhere is conspicuously absent, and hence the contents can be string-dependent, we impose (36). To guarantee that, for any string, under the feasible deterministic program, it is not only that the starting tape contents in positions 1 to n match the string, but also that all other positions are filled with 0's, we impose (37) and (38).

$$\begin{aligned} &S_N[m,l,0,p,k] = \bar{X}^n[l,p,k], \\ &\quad \forall m = 1,...,2^n,\ l = 1,...,n,\ p = 1,...,n,\ k = 0,1,2; \end{aligned} \tag{36}$$

$$\begin{aligned} &S_D[m,l,0,p,k] = \bar{X}^n[l,p,k], \\ &\quad \forall m = 1,...,2^n,\ l = 1,...,n,\ p = 1,...,n,\ k = 0,1,2; \end{aligned} \tag{37}$$

$$\begin{aligned} &S_D[m,l,0,p,0] = 1, \\ &\quad \forall m = 1,...,2^n,\ l = 1,...,n,\ p = -t_D,...,0 \text{ and } p = n+1,...,t_D+1. \end{aligned} \tag{38}$$

To ensure that, for any string, under either the nondeterministic or the deterministic program, the program state evolution follows the program, we impose

$$\begin{aligned} Q_N[m,l,i+1,j'] = &\textstyle\sum_{j=0}^{r} Q_N[m,l,i,j] \sum_{p=-t_N}^{t_N+1} H_N[m,l,i,p] \sum_{k=0,1,2} S_N[m,l,i,p,k] \\ &\textstyle\sum_{k'=0,1,2} \sum_{d=\pm 1} A_N[m,j,k,j',k',d], \\ \forall m = 1,...,2^n,\ l = 1,...,n,\ i = 0,...,t_N-1,\ j' = 0,...,r; \end{aligned} \tag{39}$$

$$Q_D[m,l,i+1,j'] = \sum_{j=0}^{r} Q_D[m,l,i,j] \sum_{p=-t_D}^{t_D+1} H_D[m,l,i,p] \sum_{k=0,1,2} S_D[m,l,i,p,k]$$
$$\sum_{k'=0,1,2} \sum_{d=\pm 1} A_D[m,j,k,j',k',d], \tag{40}$$
$$\forall m = 1,...,2^n,\ l = 1,...,n,\ i = 0,...,t_D-1,\ j' = 0,...,r.$$

To ensure that, for any string, under either the nondeterministic or the deterministic program, the read-write head position evolution follows the program, we impose

$$H_N[m,l,i+1,p'] = \sum_{j=0}^{r} Q_N[m,l,i,j] \sum_{d=\pm 1 \text{ and } -t_N \le p'-d \le t_N+1} H_N[m,l,i,p'-d]$$
$$\sum_{k=0,1,2} S_N[m,l,i,p'-d,k] \sum_{j'=0}^{r} \sum_{k'=0,1,2} A_N[m,j,k,j',k',d], \tag{41}$$
$$\forall m = 1,...,2^n,\ l = 1,...,n,\ i = 0,...,t_N-1,\ p' = -t_N,...,0,...,t_N+1;$$

$$H_D[m,l,i+1,p'] = \sum_{j=0}^{r} Q_D[m,l,i,j] \sum_{d=\pm 1 \text{ and } -t_D \le p'-d \le t_D+1} H_D[m,l,i,p'-d]$$
$$\sum_{k=0,1,2} S_D[m,l,i,p'-d,k] \sum_{j'=0}^{r} \sum_{k'=0,1,2} A_D[m,j,k,j',k',d], \tag{42}$$
$$\forall m = 1,...,2^n,\ l = 1,...,n,\ i = 0,...,t_D-1,\ p' = -t_D,...,0,...,t_D+1.$$

To ensure that, for any string, under either the nondeterministic or the deterministic program, the evolution of tape contents, at both the read-write head position and all other positions, follows the program, we impose

$$S_N[m,l,i+1,p,k'] = \sum_{j=0}^{r} Q_N[m,l,i,j](\sum_{p'=-t_N,...,0,...,t_N+1 \text{ and } p' \neq p} H_N[m,l,i,p']$$
$$S_N[m,l,i,p,k'] + H_N[m,l,i,p] \sum_{k=0,1,2} S_N[m,l,i,p,k]$$
$$\sum_{j'=0}^{r} \sum_{d=\pm 1} A_N[m,l,j,k,j',k',d]), \tag{43}$$
$$\forall m = 1,...,2^n,\ l = 1,...,n,\ i = 0,...,t_N-1,$$
$$p = -t_N,...,0,...,t_N+1,\ k' = 0,1,2;$$

$$S_D[m,l,i+1,p,k'] = \sum_{j=0}^{r} Q_D[m,l,i,j](\sum_{p'=-t_D,...,0,...,t_D+1 \text{ and } p' \neq p} H_D[m,l,i,p']$$
$$S_D[m,l,i,p,k'] + H_D[m,l,i,p] \sum_{k=0,1,2} S_D[m,l,i,p,k]$$
$$\sum_{j'=0}^{r} \sum_{d=\pm 1} A_D[m,l,j,k,j',k',d]), \tag{44}$$
$$\forall m = 1,...,2^n,\ l = 1,...,n,\ i = 0,...,t_D-1,$$
$$p = -t_D,...,0,...,t_D+1,\ k' = 0,1,2.$$

To guarantee the proper ranges of decision variables, we impose

$$\text{All variables } A_N,\ A_D,\ B_N,\ B_D, Q_N,\ Q_D,\ H_N,\ H_D,\ S_N, \text{ and } S_D \text{ are binary.} \tag{45}$$

The following is (IP), whose direct objective has been stated before, as the maximization of the sum of two types of sub-n languages:

$$\max z = \sum_{m=1}^{2^n} B_N[m] + \sum_{m=1}^{2^n} B_N[m] B_D[m] \tag{46}$$

subject to

$$\text{Constraints (20) to (45).}$$

As mentioned earlier, the maximization nature of the objective (46) guarantees that for any sub-n language, the required nondeterministic and deterministic programs will be discovered so long as they exist.

For given r, n, t_N, and t_D, suppose we have solved (IP) to optimality. Then, we may see that those m's with $B_N[m] = 1$ constitute the set $\mathcal{P}_N^n(r, t_N)$, while those m's with $B_N[m] = 1$ and $B_D[m] = 0$ constitute the set $\mathcal{P}_N^n(r, t_N) \setminus \mathcal{P}_D^n(r, t_D)$, the set of "hardest" sub-n languages that are still nondeterministically recognizable in t_N time by a feasible program; also, the resulting $\sum_{m=1}^{2^n} B_N[m]$ is the $z_N^n(r, t_N)$ defined earlier and the resulting $\sum_{m=1}^{2^n} B_N[m]B_D[m]$ is the $z_{ND}^n(r, t_N, t_D)$ defined earlier. We may obtain $\Delta z_{ND}^n(r, t_N, t_D) = z_N^n(r, t_N) - z_{ND}^n(r, t_N, t_D) =\mid \mathcal{P}_N^n(r, t_N) \setminus \mathcal{P}_D^n(r, t_D) \mid$.

6 Conclusions

Theorems 1, 2, and 3 indicate that the P-NP conjecture is linked to the asymptotic behavior of (IP). If we can refute the conclusion of Theorem 2 for an infinite number of r's, then we will have disproved the conjecture; while if we can verify the hypothesis of Theorem 3 using the same language π^* for all r's that are large enough, then we will have proved the conjecture. At this moment, it is not clear whether the asymptotic behavior is more difficult to study than the P-NP conjecture itself. Nevertheless, we have offered a potential alternative for the final unraveling of the myth.

The successful solving of (IP) for sufficiently large r, n, t_N, and t_D values will also offer to us hints as to which side of the conjecture we should be more inclined to. For a large enough r, if we routinely find $\Delta z^n(r, n^{a_N}, n^{a_D})$ to be 0 for increasingly larger n and a_N values, then we should lean more toward the conclusion of P = NP; while if we can clearly identify the sub-n representations of the same language in our solutions for increasingly larger n and a_D values, then we should lean more toward the belief that P $\neq$ NP.

References

[1] Adleman, L. and K. Manders. 1977. Reducibility, Randomness, and Intractability. *Proceedings of the 9th Annual ACM Symposium on Theory of Computing*, pp. 151-163, Association for Computing Machinery, New York.

[2] Babai, L. and S. Moran. 1988. Arthur-Merlin Games: A Randomized Proof System, and a Hierarchy of Complexity Classes. *Journal of Computer and System Sciences*, **36**, pp. 254-276.

[3] Chvatal, V. 1972. On Hamilton's Ideals. *Journal of Combinatorial Theory, Series B*, **12**, pp. 163-168.

[4] Chvatal, V. 1977. Determining the Stability Number of a Graph. *SIAM Journal on Computing*, **6**, pp. 643-662.

[5] Cook, S.A. 1971. The Complexity of Theorem-proving Procedures. *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, pp. 151-158, Association for Computing Machinery, New York.

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

[7] Gill, J. 1977. Computational Complexity of Probabilistic Turing Machines. *SIAM Journal on Computing*, **6**, pp. 675-695.

[8] Goldwasser, S., S. Macali, and C. Rackoff. 1985. The Knowledge Complexity of Interactive Proof Systems. *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, pp. 291-304, Association for Computing Machinery, New York.

[9] Levin, L.V. 1984. Problem, Complete in "Average" Instances. In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing*, Association for Computing Machinery, New York.

[10] McDiarmid, C.J.H. 1979. Determining the Chromatic Number of a Graph. *SIAM Journal on Computing*, **8**, pp. 1-14.

[11] Meyer, A.R. and L.J. Stockmeyer. 1972. The Equivalence Problem for Regular Expressions with Squaring Requires Exponential Times. *Proceedings of the 13th Annual Symposium on Switching and Automata Theory*, pp. 125-129, IEEE Computer Society, Long Beach, California.

[12] Papadimitriou, C.H. and M. Yannakakis. 1984. The Complexity of Facets (and Some Facets of Complexity). *Journal of Computer and System Sciences*, **28**, pp. 244-259.

[13] Razborov, A.A. 1985. A Lower Bound on the Monotone Network Complexity of the Logical Permanent. *Mathematical Notes of the Academy of Sciences of the USSR*, **37**, pp. 485-493.

[14] Stockmeyer, L.J. 1976. The Polynomial-time Hierarchy. *Theoretical Computer Science*, **3**, pp. 1-22.

[15] Yao, A.C. 1985. Separating the Polynomial-time Hierarchy by Oracles. *Proceedings of the 26th Annual Symposium on Foundations of Computer Science*, pp. 1-10, IEEE Computer Society, Los Angeles.