A List of P-Complete Problems

Miyano, Satoru
Research Institute of Fundamental Information Science Kyushu University

Shiraishi, Shuji
Department of Applied Mathematics Fukuoka University

Shoudai, Takayoshi
Department of Control Engineering and Science Kyushu Institute of Technology

https://hdl.handle.net/2324/3123

出版情報：RIFIS Technical Report. 17, 1989-10-11. Research Institute of Fundamental Information Science, Kyushu University

RIFIS-TR-CS-17

RIFIS Technical Report

A List of P-Complete Problems

Satoru Miyano
Shuji Shiraishi
Takayoshi Shoudai

October 11, 1989

Revised: December 29, 1990

Research Institute of Fundamental Information Science
Kyushu University 33
Fukuoka 812, Japan

E-mail: miyano@rifis.sci.kyushu-u.ac.jp Phone: 092(641)1101 Ext.4471

A List of P-Complete Problems

Satoru Miyano
Research Institute of Fundamental Information Science
Kyushu University 33
Fukuoka 812, Japan
Shuji Shiraishi
Department of Applied Mathematics
Fukuoka University
Fukuoka 814-01, Japan
Takayoshi Shoudai
Department of Control Engineering and Science
Kyushu Institute of Technology
Iizuka 820, Japan

December 29, 1990

1 Introduction

One of the roles of parallel complexity theory is to investigate the problems which have no efficient parallel algorithms. It has been observed that some problems do not seem to allow any fast parallel algorithms although they are easily solvable in polynomial time by sequential algorithms. These problems have been shown P-complete.

The class of problems with efficient parallel algorithms is understood to be **NC** [Pippenger, 1979]. The class **NC** is a subclass of **P** but, no mathematical proof has been given that shows **NC**$\neq$**P**. Like **NP**$\neq$**P** question, we strongly believe that **NC** and **P** are different. With the assumption that **NC**$\neq$**P**, we can see that no P-complete problem is in **NC**. By this observation, we can use P-completeness to show the inherent difficulty of parallelization.

In recognizing the importance of P-completeness, we make a list of P-complete problems in a way similar to [Garey and Johnson, 1979] and [Greenlaw, Hoover and Ruzzo, 1989]. We hope this list would help to understand which problems have no efficient parallel algorithms.

2 NC-Reducibility and P-Completeness

In this section we give some definitions and notions related to P-completeness.

Definition 1 A *problem* (or *search problem*) S with a size parameter $h(n)$ is a family $\{S_n\}_{n\geq 0}$ of binary relations $S_n \subseteq \{0,1\}^n \times \{0,1\}^{h(n)}$ for $n \geq 0$. For $n \geq 0$, $x \in \{0,1\}^n$ is called an *instance* and an object $y \in \{0,1\}^{h(n)}$ satisfying $S_n(x,y)$, if any, is called a *solution* for x. For convenience, we assume that there exists y with $S_n(x,y)$ for each $x \in \{0,1\}^n$. If a solution y with $S_n(x,y)$ is unique for each $x \in \{0,1\}^n$, then the problem of finding a solution is exactly the same as computing the function defined by S. Moreover, if $h(n) = 1$ for all $n \geq 0$, then the problem S is regarded as a *decision problem.*

Example 1 A *maximal independent set* (MIS) of an undirected graph is a maximal set U of vertices such that no two vertices in U are adjacent. The problem of finding a MIS is formulated in the following way: A graph with n vertices is represented by an $n \times n$-adjacency matrix and a subset of vertices is represented by an n-bit vector. Then MIS=$\{\text{MIS}_n\}_{n\geq 0}$ is defined only for integers of the form n^2 as

$$\text{MIS}_{n^2} \subseteq \{0,1\}^{n^2} \times \{0,1\}^n,$$

where for $(x,y) \in\text{MIS}_{n^2}$, x is a symmetric matrix representing an undirected graph with n vertices and y a bit vector representing a MIS in the graph.

Definition 2 We say that a search problem S is *polynomial time solvable* if there is a polynomial time computable function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ such that $S_n(x, f(x))$ holds for all $x \in \{0,1\}^n$ and $n \geq 0$. We denote by **P** the class of polynomial time solvable search problems.

A very common parallel computation model is the parallel RAM (PRAM) model in which processors work together synchronously and communicate with a common random access memory. By read and write access abilities, PRAMs are classified to CREW PRAM, EREW PRAM and CRCW PRAM. In any case, we define as follows:

Definition 3 A parallel algorithm on a PRAM solving a problem is *efficient* if, given an input of size n, it runs

(1) in time $O((\log n)^k)$ for some constant $k \geq 0$,

(2) with a polynomial number of processors.

This definition is based on the observation that the time $O((\log n)^k)$ is very fast and a polynomial number of processors is feasible.

The class **NC** is defined by the uniform circuit model.

Definition 4 A *circuit* α with n inputs and m outputs is a finite labeled directed acyclic graph such that it has n input lines and m output lines and each node is labeled with a gate such as AND gate, OR gate, NOT gate, etc. We denote by $size(\alpha)$ (resp., $depth(\alpha)$) the size of a circuit α which is the number of nodes in α (resp., the depth of a circuit α which is the length of the longest path from some input to some output). The function $f_\alpha : \{0,1\}^n \rightarrow \{0,1\}^m$ computed by α is defined in an obvious way.

Definition 5 For a search problem $S = \{S_n\}_{n\geq 0}$, we say that $\{\alpha_n\}_{n\geq 0}$ *solves* S if $\{\alpha_n\}_{n\geq 0}$ computes a function $\{f_n\}_{n\geq 0}$ such that $S_n(x, f_n(x))$ for all $x \in \{0,1\}^n$ and $n \geq 0$, where f_n is the function computed by α_n.

Several kinds of uniformities of circuits have been proposed [Ruzzo, 1980], [Cook, 1985]. We use the following definition because of its simplicity.

Definition 6 A circuit family $\{\alpha_n\}_{n\geq 0}$ is *log-uniform* (or simply, *uniform*) if, given n in unary, the description of the nth circuit α_n is computed by a deterministic off-line Turing machine using $O(\log n)$ worktape space.

It is known that $\mathbf{NC}^k$ defined below does not change by the choice of uniformity for $k \geq 2$ [Ruzzo, 1980].

Definition 7

(1) $\mathbf{NC}^k$=$\{S \mid S$ is solvable by a uniform circuit family $\{\alpha_n\}_{n\geq 0}$ such that $size(\alpha_n)$ is bounded by some polynomial and simultaneously $depth(\alpha_n) = O((\log n)^k)\}$.

(2) $\mathbf{NC}$=$\bigcup_{k\geq 0}\mathbf{NC}^k$.

The following theorem relates the class **NC** and efficient parallel algorithms.

Theorem 1 (Stockmeyer and Vishkin, 1984) **NC** *is the class of problems solvable by PRAM algorithms with polynomial number of processors and running time* $O((\log n)^k)$, *where* k *is an arbitrary constant.*

Assuming that **P**$\neq$**NC**, we can prove that no P-complete problem allows any PRAM parallel algorithm running in time $O((\log n)^{O(1)})$ with $n^{O(1)}$ processors. Thus the P-completeness plays a very important role to convince of the hardness of efficient parallelization.

The knowledge that a problem is P-complete provides algorithm designers valuable information about the approaches they should take. It would relieve wasting efforts for devising drastically fast parallel algorithms and, instead, direct toward ways which lead to useful algorithms. Proofs of P-completeness may also tell us which parts of problems are hard to parallelize.

The P-completeness due to [Cook, 1985] adopted the NC^1-reducibility that uses NC^1-computable $O(\log n)$ depth uniform circuits with oracle gates since it deals with reductions between functions. Another commonly used reducibility is the many-one log space reducibility for decision problems [Hopcroft and Ullman, 1979]. Here we use the following definition of reducibility.

Definition 8 Let S and T be problems of size parameters $g(n)$ and $h(n)$, respectively. We say that S is *NC^k-reducible* to T, denoted $S \leq^{NC^k} T$, if there is a uniform circuit family $\{\alpha_n\}_{n\geq 0}$ satisfying the following conditions:

1. Each α_n can involve several oracle gates which can solve T, where an oracle gate for T is a gate such that, for each input bit sequence $(y_1, \ldots, y_r)$, it produces an output sequence $(z_1, \ldots, z_{h(r)})$ satisfying $T_r(y_1, \ldots, y_r, z_1, \ldots, z_{h(r)})$.
2. The depth of α_n is $O((\log n)^k)$, where the depth of an oracle gate with r input lines is measured as $\lceil \log r \rceil$. For each input $x \in \{0,1\}^n$, the circuit α_n outputs z with $S_n(x, z)$ so far as the input-output relation satisfies T at each oracle gate. Namely, the output z of α_n may differ according to the solutions selected at oracle gates, but for any solutions the relation $S_n(x, z)$ holds.

We say that S is *NC-reducible* to T, denoted $S \leq^{NC} T$, if $S \leq^{NC^k} T$ for some k.

Obviously, the NC-reducibility is an extension of the NC^1-reducibility. It is also known that if S is log space reducible to T then S is NC^2-reducible to T. Hence the NC-reducibility is stronger than these two kinds of reducibilities and can deal with reductions among relations. However, all problems in the list of Section 4 can be shown P-complete via NC^2-reductions.

Fact 1

(1) *The relation $\leq^{NC}$ is transitive.*

(2) *If $S \leq^{NC} T$ and $T \in$***NC***, then $S \in$***NC***.*

Definition 9 A problem T is said to be *P-complete* if the following conditions are satisfied:

(1) T is in **P**.

(2) For each problem S in **P**, $S \leq^{NC} T$.

The importance of P-completeness is based on the following fact:

Fact 2 *No P-complete problem is in* **NC** *if* **P$\neq$NC**.

Finally we should remark that proving the P-completeness of a problem is just the start of work on that problem. Even if a problem is shown P-complete, it tells us that no drastic speed up may be expected theoretically. However, it does not deny the possibility of reducing the degree of polynomial of the time complexity.

3 Proving P-Completeness

The following problem is the most widely used problem for proving P-completeness.

CIRCUIT VALUE PROBLEM (CVP)
INSTANCE: A circuit $B = (B_1, \ldots, B_n)$, where each B_i is either (i) $B_i = 1$ (*true*) or 0 (*false*), (ii) $B_i = \neg B_j$ $(j < i)$, (iii) $B_i = B_j \wedge B_k$ $(j, k < i)$, or (iv) $B_i = B_j \vee B_k$ $(j, k < i)$. B_n is called the output gate and the gates with *true* or *false* are called the input gates.
PROBLEM: Decide whether the value of B_n is *true*.

Theorem 2 (Ladner, 1975) *CIRCUIT VALUE PROBLEM is P-complete.*

We note that CVP is complete for the class of search problems which have polynomial time algorithms. The MONOTONE CIRCUIT VALUE PROBLEM (MCVP) is a restricted version of the CIRCUIT VALUE PROBLEM for which instances are circuits with AND and OR gates only [Goldschlager, 1977].

We give two examples of problems complete for P. The first one is originally a problem of computing a function but it can be regarded as a decision problem. However, the second problem does not seem to have an equivalent decision problem. Both of them are shown P-complete by NC-reductions from MCVP.

LEXICOGRAPHICALLY FIRST MAXIMAL INDEPENDENT SET (LFMIS)
INSTANCE: A graph $G = (V, E)$ with $V = \{1, \ldots, n\}$ and a vertex u.
PROBLEM: Decide whether u is contained in the lexicographically first subset U of V such that no two vertices in U are adjacent.

Theorem 3 (Cook, 1985) *LEXICOGRAPHICALLY FIRST MAXIMAL INDEPENDENT SET is P-complete.*

Proof. The following greedy algorithm solves LFMIS in polynomial time. An example of the lexicographically first maximal independent set is shown in Fig. 1:

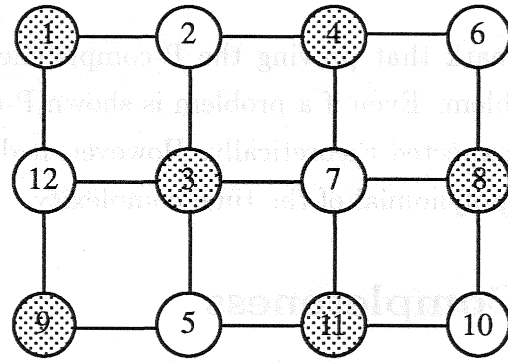


Fig. 1: The lexicographically first maximal independent set

begin /* $G = (V, E)$ with $V = \{1, ..., n\}$ */
    $U \leftarrow \emptyset$;
    for $i \leftarrow 1$ **to** n **do**
        if $U \cup \{i\}$ is an independent set **then** $U \leftarrow U \cup \{i\}$
end

We give a reduction from MCVP to LFMIS. Let $B = (B_1, \ldots, B_n)$ be a monotone circuit. We construct a graph G such that each gate B_i is associated with two vertices u_i and v_i. A linear order $\prec$ on vertices is defined so that $u_i, v_i \prec u_j, v_j$ for $i < j$. We define edges of G and the order between u_i and v_i as follows:

(1) If $B_i = true$ (resp. $false$), then $u_i \prec v_i$ (resp. $v_i \prec u_i$) and an edge $\{u_i, v_i\}$ is added.

(2) If $B_i = B_j \wedge B_k$ with $j, k < i$, then $u_i \prec v_i$ and edges $\{u_i, v_i\}$, $\{v_j, u_i\}$, and $\{v_k, u_i\}$ are added.

(3) If $B_i = B_j \vee B_k$ with $j, k < i$, then $v_i \prec u_i$ and edges $\{u_i, v_i\}$, $\{u_j, v_i\}$, and $\{u_k, v_i\}$ are added.

An example of construction from the monotone circuit in Fig. 2 is shown in Fig. 3. It is not hard to see that this reduction is computable in NC and the lexicographically first maximal independent set of G includes u_n iff the value of B_n is $true$.$\square$

The next problem is a search problem $S = \{S_n\}_{n \geq 0}$ such that a solution y with $S_n(x, y)$ is not necessarily unique for $x \in \{0,1\}^n$.

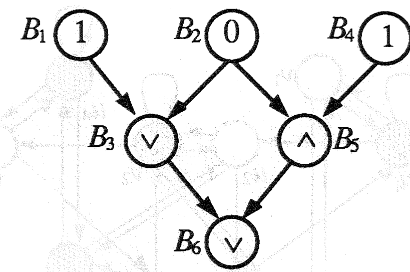


Fig. 2: Monotone circuit

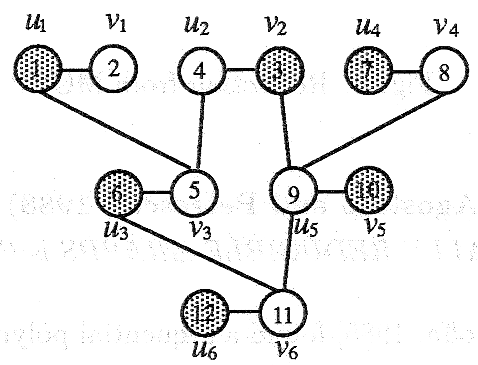


Fig. 3: Reduction from MCVP to LFMIS

MINIMUM FEEDBACK VERTEX SET FOR CYCLICALLY REDUCIBLE GRAPHS

INSTANCE: A cyclically reducible directed graph $D = (V, A)$.

PROBLEM: Find a minimum feedback vertex set of D.

Let $D = (V, A)$ be a directed graph. A set $X \subseteq V$ is a *feedback vertex set* if it contains at least one vertex from every cycle in D. The *associated graph of vertex x with respect to D*, denoted $A(D, x)$, consists of x and all vertices which are not connected, after eliminating x from D, by any path to vertices on the cycles. Let $y_1, \ldots, y_k$ be a sequence of vertices of D. Then we define graphs $D_0, \ldots, D_k$ inductively as follows: $D_0 = D, D_i = D_{i-1} - A(D_{i-1}, y_i)$ for $i = 1, \ldots, k$. We say that the sequence $y_1, \ldots, y_k$ is a *complete D-sequence* of D if each of $A(D_0, y_1), \ldots, A(D_{k-1}, y_k)$ contains a cycle but D_k is acyclic. Note that D_{i-1} contains a cycle for $i = 1, \ldots, k$ if $y_1, \ldots, y_k$ is a complete D-sequence. A directed graph D is *cyclically reducible* if there exists a complete D-sequence for D.

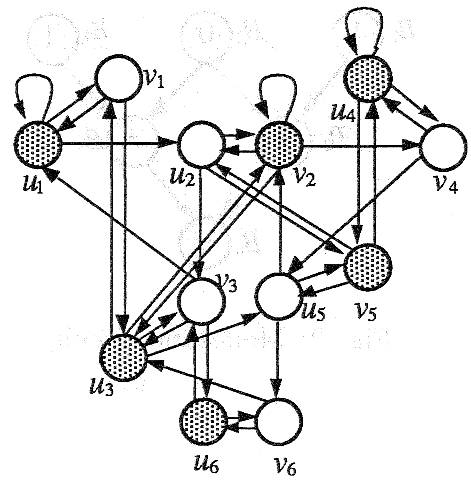


Fig. 4: Reduction from MCVP

Theorem 4 (Bovet, de Agostino and Petreschi, 1988) *MINIMUM FEEDBACK VERTEX SET FOR CYCLICALLY REDUCIBLE GRAPHS is P-complete.*

Proof. [Wang, Lloyd and Soffa, 1985] found a sequential polynomial time algorithm finding a complete D-sequence. Hence, the problem is in P. We give a reduction from MCVP. Let $B = (B_1, \ldots, B_n)$ be a monotone circuit. We construct a cyclically reducible graph D such that each gate B_i is associated with two vertices u_i and v_i. We define edges of D as follows:

(1) If $B_i = true$ (resp., $false$), then a loop edge (u_i, u_i) (resp., (v_i, v_i)) is added.

(2) If $B_i = B_j \wedge B_k$ with $j, k < i$, then edges (u_i, v_j), (v_j, v_k), (v_k, u_i), (v_i, u_k), (u_k, v_i), (u_j, v_i), and (v_i, u_j) are added.

(3) If $B_i = B_j \vee B_k$ with $j, k < i$, then edges (v_i, u_j), (u_j, u_k), (u_k, v_i), (u_i, v_k), (v_k, u_i), (v_j, u_i), and (u_i, v_j) are added.

(4) Finally, for each $i = 1, ..., n$, edges (u_i, v_i) and (v_i, u_i) are added.

An example of construction from the monotone circuit in Fig. 2 is shown in Fig. 4. Any minimum feedback vertex set of D has at least n vertices by (4). By induction on n, we can show the following two facts:

(i) D has a unique minimum feedback vertex set having n vertices.

(ii) The value of B_i is $true$ iff u_i is included in this minimum feedback vertex set.□

It should be remarked that in the reduction of the proof of Theorem 4 a cyclically reducible graph is constructed from a circuit so that the graph has a unique minimum feedback vertex set. Hence the problem itself allows several solutions but an instance in the reduction has only one solution. This is the reason why the reduction from MCVP has succeeded.

4 Acknowledgements

The first version of this P-complete list was organized in October, 1989. After that, however, some amount of P-complete problems have been reported. Recognizing this situation, the authors recompiled the list. The authors would like to thank especially the following people for providing us useful information: S. de Agostino, J. Balcázar, H. Ebara, R. Greenlaw, J. Hershberger, H. Imai, P. Kanellakis, G. Mauri, E.W. Mayr, P. Rajčáni, J. Peters, W.L. Ruzzo, M. Warmuth, O. Watanabe, S. Yamasaki, H. Yasuura. Finally, special thanks to S. Arikawa for encouragements and to A. Shinohara for helping us in organizing this list.

5 P-Complete Problems

We classify the problems into the following categories:

1 Circuits
2 Logic
3 Graph
4 Optimization
5 Formal Language
6 Algebraic Problems
7 Miscellaneous
8 Remarks

5.1 Circuits

5.1.1 CIRCUIT VALUE PROBLEM (CVP)

INSTANCE: A circuit $B = (B_1, \ldots, B_n)$, where each B_i is either (i) $B_i = true$ or $false$, (ii) $B_i = \neg B_j$ $(j < i)$, (iii) $B_i = B_j \wedge B_k$ $(j, k < i)$, or (iv) $B_i = B_j \vee B_k$ $(j, k < i)$.
PROBLEM: Decide whether the value of B_n is *true*.
Reference: [Ladner, 1975].
Comment: The problem is still P-complete even when the instances are planar circuits with NOT and AND gates (PLANAR CVP) [Goldschlager, 1977]. A circuit only with AND and OR gates and constants *true*, *false* is called a *monotone circuit*. The circuit value problem for monotone circuits (MONOTONE CVP) is P-complete [Goldschlager, 1977]. MONOTONE CVP is P-complete even if each gate has fanout at most two and B_n is an OR gate. Moreover, MONOTONE CVP is P-complete for circuits such that OR gates and AND gates appear alternatingly (ALTERNATING MONOTONE CVP). It is also obvious that CVP with only NAND gates (resp., NOR gates) and constants is P-complete (NAND CVP (resp., NOR CVP)). However, for the monotone planar circuits defined in [Goldschlager, 1980], the problem falls in NC^2 [Goldschlager, 1980].

5.1.2 ARITHMETIC CIRCUIT VALUE PROBLEM (ARITHMETIC CVP)

INSTANCE: A circuit $B = (B_1, \ldots, B_n)$, where each B_i is either (i) $B_i = 1$ or 0, (ii) $B_i = B_j * B_k$ $(j, k < i)$ (iii) $B_i = B_j - B_k$ $(j, k < i)$, or (iv) $B_i = B_j + B_k$ $(j, k < i)$, where $*, -, +$ are arithmetic operations on integers.
PROBLEM: Decide whether the value of B_n is 1.
Reference: This problem is stated in [Greenlaw, Hoover and Ruzzo, 1989] as a private communication of [Venkateswaran, 1983].

Comment: Reduction from CVP by replacing *true*, *false*, $\neg x$, $x \wedge y$, $x \vee y$ by 1, 0, $1 - x$, $x * y$, $x + y - x * y$, respectively.

5.1.3 MIN-PLUS CIRCUIT VALUE PROBLEM (MIN-PLUS CVP)

INSTANCE: A circuit $B = (B_1, \ldots, B_n)$, where each B_i is either (i) $B_i = 1$ or 0, (ii) $B_i = \min\{B_j, B_k\}$ $(j, k < i)$, or (iii) $B_i = B_j + B_k$ $(j, k < i)$.
PROBLEM: Decide whether the value of B_n is 1.
Reference: This problem is stated in [Greenlaw, Hoover and Ruzzo, 1989] as a private communication of [Venkateswaran, 1983].
Comment: Reduction from CVP by replacing *true*, *false*, $x \wedge y$, $x \vee y$ by 1, 0, $\min\{x, y\}$, $\min\{1, x + y\}$, respectively.

5.1.4 APPROXIMATION OF CIRCUIT DEPTH OF ONES

INSTANCE: A circuit B with AND and OR gates and constants 0 (*false*) and 1 (*true*), and an integer K.
PROBLEM: For a gate v of B, $depth(v)$ is the length of the longest path from some input to v. Let $depth\text{-}ones(B)$ be $\max\{depth(v) \mid \text{the value of } v \text{ is } 1\}$ called the *depth of ones*. Then decide whether $depth\text{-}ones(B) \geq K \geq \epsilon \cdot depth\text{-}ones(B)$, where ϵ is a fixed rational number in $(0, 1]$.
Reference: [Kirousis and Spirakis, 1988].
Comment: This problem is to approximate the depth of ones in a circuit with a performance ratio $1/\epsilon$. If $\epsilon = 1$ and K the depth of the output gate, the problem is the same as CIRCUIT VALUE PROBLEM.

5.2 Logic

5.2.1 SOLVABLE PATH SYSTEM (SPS)

INSTANCE: A quadruple $Q = (X, R, S, T)$ called a *path system*, where X is a finite set, $R \subseteq X \times X \times X$, $S \subseteq X$ and $T \subseteq X$.
PROBLEM: Let A be the least subset of X such that (a) $S \subseteq A$ and (b) if $x, y \in A$ and $(x, y, z) \in R$ then $z \in A$. Decide whether $A \cap T \neq \emptyset$.
Reference: [Cook, 1974].
Comment: This is the first problem that was shown P-complete by a generic reduction from deterministic polynomial time oblivious Turing machines. An *oblivious Turing machine* is a one-tape Turing machine such that the head position at time t is determined only by the

input length and is not dependent of the contents of the input.

5.2.2 UNIT RESOLUTION

INSTANCE: A set $S = \{C_1, ..., C_m\}$ of clauses in the propositional calculus.
PROBLEM: Decide whether the empty clause $\Box$ can be deduced from S by unit resolution.
Reference: [Jones and Laaser, 1977], [Dobkin, Lipton and Reiss, 1979].
Comment: A *literal* is an atom (*positive literal*) or the negation of an atom (*negative literal*). A *clause* is a disjunction of literals. When a clause contains no literal, it is called the *empty clause* denoted $\Box$. Let C and C' be the clauses of the form $C = \alpha \vee \beta_1 \vee \cdots \vee \beta_p$ and $C' = (\sim \alpha) \vee \gamma_1 \vee \cdots \vee \gamma_q$, respectively. The *resolvent* of C and C' is $\beta_1 \vee \cdots \vee \beta_p \vee \gamma_1 \vee \cdots \vee \gamma_q$. This is called the unit resolvent in case $p = 0$ or $q = 0$. A *deduction by unit resolution* from S is a sequence $D_1, ..., D_n$ in which each D_i is either a clause in S or follows from two earlier clauses by unit resolution. Namely, a unit resolution is a deduction in which a resolvent is obtained by using at least one unit clause, i.e., a clause consisting of a single literal. Remains P-complete even if it is restricted to Horn clauses.

5.2.3 PROPOSITIONAL HORN SATISFIABILITY

INSTANCE: A set S of Horn clauses in the propositional calculus.
PROBLEM: Decide whether S is satisfiable.
Reference: [Plaisted, 1984], [Kasif, 1986].
Comment: A *Horn clause* is a clause which has at most one positive literal. The positive literal in a Horn clause (if exists) is called the *head of the clause.* The negative literals (if exist) are called the *body of the clause.* S is *satisfiable* if there is a truth assignment satisfying all clauses in S. Remains P-complete even if every clause in S has at most three literals. The satisfiability problem for Horn clauses is also discussed in [Yamasaki and Doshita, 1983].

5.2.4 DEPTH-RESTRICTED HYPER-RESOLUTION FROM PREDICATE UNIQUE MATCH 2-HORN CLAUSES

INSTANCE: A set S of Horn clauses in the predicate calculus such that every clause has at most two literals and S has unique matches, and an integer d in unary.
PROBLEM: Decide whether there exists a hyper-resolution refutation (or natural deduction refutation) of depth d from S.
Reference: [Plaisted, 1984].

Comment: Here we deal with the predicate calculus. For necessary definitions, see [Plaisted, 1984] and [Chang and Lee, 1973]. A set S of Horn clauses has *unique matches* if for every clause C in S and every literal L of the body of C, and every ground instance L' of L, there exists at most one clause D of S such that the head of D is unifiable with L'. A *hyper-resolution proof* from a set S of Horn clauses is a sequence of positive literals in which each literal in the sequence is an instance of a clause in S or is derived from previous literals using the following rule: Suppose that $L_1, ..., L_k$ have already been derived. Suppose that $M_1 \wedge M_2 \wedge \cdots \wedge M_k \rightarrow M$ is a clause C of S expressed as an implication, where all M_i are positive literals. Let $M_1' \wedge M_2' \wedge \cdots \wedge M_k' \rightarrow M'$ be the most general instance of C such that M_i' is an instance of L_i for $1 \leq i \leq k$. Then we say that M' is derived from $L_1, ..., L_k$. The depth of a literal L in such a proof is zero if L is an instance of a clause of S. If M' is as above, then the depth of M' is one plus the maximum of the depths of the M_i''s. The depth of a proof is the maximum depth of any of its literals. A *refutation* is a proof of $\square$. If a formula is of the form $\exists y_1 \exists y_2 \cdots \exists y_n \forall x_1 \forall x_2 \cdots \forall x_m A$, where A is an expression not containing any quantifier symbols, then we say that it is in *Schönfinkel-Bernays form.* Remains P-complete even for clauses in Schönfinkel-Bernays form.

5.2.5 DEPTH-RESTRICTED HYPER-RESOLUTION FROM PROPOSITIONAL 3-HORN CLAUSES

INSTANCE: A set S of Horn clauses in the propositional calculus such that every clause has at most three literals, and an integer d in unary.
PROBLEM: Decide whether there exists a hyper-resolution refutation (or natural deduction refutation) of depth d from S.
Reference: [Plaisted, 1984].
Comment: This is a propositional calculus version of DEPTH-RESTRICTED HYPER-RESOLUTION FROM PREDICATE UNIQUE MATCH 2-HORN CLAUSES.

5.2.6 UNIFICATION

INSTANCE: Two terms s and t represented by a labeled directed acyclic graph D with two specified vertices u_s and u_t.
PROBLEM: Decide whether s and t are unifiable.
Reference: [Dwork, Kanellakis and Mitchell, 1984]., [Yasuura, 1984]
Comment: Let V be an infinite set of variable symbols and F an infinite set of function symbols. The arity of $f \in F$ is denoted by $a(f)$. A function symbol $g \in F$ with $a(g) = 0$ is called a *constant.* A *term* is defined inductively as follows:

(a) A variable symbol $x \in V$ or constant $g \in F$ is a term.

(b) If $f \in F$ and $t_1, \ldots, t_{a(f)}$ are terms, then $f(t_1, \ldots, t_{a(f)})$ is a term.

The set of terms is denoted by T. A *substitution* σ is a mapping from V to the set of terms such that $\sigma(x) = x$ for all but finitely many $x \in V$. For a term s, $\sigma(s)$ also denotes the term obtained by replacing every occurrence of a variable x by $\sigma(x)$. Two terms s and t are *unifiable* if there is a substitution σ, called a *unifier* for s and t, such that $\sigma(s) = \sigma(t)$. A substitution σ is said to be more general than a substitution τ if there is a substitution ρ with $\tau = \rho \circ \sigma$. It is known [Robinson, 1965] that whenever terms s and t are unifiable, there is the *most general unifier* for s and t.

For the representation of a term, we consider a labeled directed acyclic graph (dag) D labeled as follows: Each vertex is labeled with a variable symbol or a function symbol. A vertex labeled with a variable symbol or a constant has no outedge. For a vertex labeled with a function symbol of arity k, there are k outedges labeled with $1, ..., k$, respectively. Then by specifying a vertex u_t of D, we can associate, in a very natural way, a term t with the subgraph formed from the vertices and edges reachable from u_t. In this way, a term is represented by a labeled dag.

The unification is P-complete even if both terms are linear, i.e., each variable appears at most once in each term, are represented by trees, and have all function symbols with arity at most 2, but they may share variables. Moreover, this problem is P-complete even if both terms are represented by trees, no variable appears in both terms, each variable appears at most twice in some term and all function symbols have arity at most 2. In contrast, if there are no sharings of variables and one of the terms is linear, then the problem can be solved in NC [Dwork, Kanellakis and Stockmeyer, 1988]. A term s matches a term t if $t = \sigma(s)$ for some substitution σ. The *term matching* is the problem of finding such substitution. An $O((\log n)^2)$ time CREW PRAM algorithm for term matching using $O(M(n)^2)$ processors is known [Dwork, Kanellakis and Mitchell, 1984], where $M(n)$ is the number of arithmetic operations required for $n \times n$ matrix multiplication. There is also a Las Vegas parallel term matching algorithm which runs in $O((\log n)^2)$ time using $M(n)$ processors [Dwork, Kanellakis and Stockmeyer, 1988].

5.2.7 LOGICAL QUERY PROGRAM

INSTANCE: An atom $p(a, b)$ and a finite set of atoms $P_E = \{q_{j_1}(a_1, b_1), \ldots, q_{j_n}(a_n, b_n)\}$, where q_{j_k} is a predicate symbol from $\{q_0, q_1\}$ and a_k and b_k are constants for $k = 1, \ldots, n$.
PROBLEM: Decide whether $p(a, b)$ is a theorem of $P = P_I \cup P_E$ for the following logical query program P_I:

$p(X, Y) : -q_1(X, A), p(A, B), p(B, C), q_1(C, Y).$

$p(X,Y):-q_0(X,Y).$

Reference: [Ullman and Van Gelder, 1988].

Comment: A *logical query program* is a function free Horn logic program. This problem for the following program is also P-complete,

$p(X,Y):-p(X,A),p(A,B),q_1(B,C),p(C,D),q_2(D,Y).$

$p(X,Y):-q_0(X,Y).$

The logical query program has the *polynomial fringe property* if every atom in the minimum model of the resulting extended logic program has a derivation tree whose fringe defined as the set of its leaves, is of polynomial length in n, where n is the number of atoms in P_E. A logical query program which have a polynomial fringe property falls in NC. The following logical query programs have the polynomial fringe property, hence, it is in NC,

(i) $p(X,Y):-p(X,A),p(A,B),p(B,Y).$
$p(X,Y):-q_0(X,Y).$

(ii) $p(X,Y):-p(X,A),q_1(A,B),p(B,Y).$
$p(X,Y):-q_0(X,Y).$

(iii) $p(X,Y):-q_1(X,A),p(A,B),q_2(B,C),p(C,Y).$
$p(X,Y):-q_0(X,Y).$

5.2.8 UNIQUE RECOVERABILITY FOR INCOMPLETE TABLES OF INFINITE TYPE

INSTANCE: A collection F of functional dependencies on a finite attribute set $\{A_1, ..., A_m\}$ and a matrix $T = (T_{ij})_{0\le i\le n,\ 1\le j\le m}$ called an *incomplete table* such that $T_{0j} = A_j$ for $1 \le j \le m$ and the value of each T_{ij} is either a nonnegative integer or the null value $*$ for $1 \le i \le n$ and $1 \le j \le m$.

PROBLEM: Decide whether T is uniquely recoverable under F when the domain of each attribute A_j is the set of nonnegative integers.

Reference: [Miyano and Haraguchi, 1982].

Comment: If a table T' contains no null value and coincides with T except in the entries with the null value, T' is called an extension of T, where $T'_{1j}, ..., T'_{nj}$ contains values in the domain of A_j for $1 \le j \le n$. A *functional dependency* is of the form $A_j \leftarrow A_{i_1}, ..., A_{i_k}$. We say that a table T satisfies $A_j \leftarrow A_{i_1}, ..., A_{i_k}$ if the values in the columns $A_{i_1}, ..., A_{i_k}$ determine the value in the column A_j. An incomplete table T is *uniquely recoverable* under F if there is exactly one extension T' which satisfies all functional dependencies in F.

5.3 Graph

5.3.1 AND/OR GRAPH ACCESSIBILITY PROBLEM (AGAP)

INSTANCE: An and/or graph $D = (V, A)$ and vertices s and t of D.
PROBLEM: Decide whether t is reachable from s.
Reference: [Jones and Laaser, 1977].
Comment: This problem is essentially the same as TWO-PERSON GAME [Jones and Laaser, 1977]. An *and/or graph* or *alternating graph* is a directed graph such that each vertex is assigned a label from $\{\wedge, \vee\}$. A vertex with label $\wedge$ (resp. $\vee$) is called an *AND vertex* (resp. *OR vertex*). We say that t is *reachable* from s in an and/or graph $D = (V, A)$ if a pebble can be placed on the specified vertex t by using the following rules: For $v \in V$, let pred(v)=$\{u \mid (u, v) \in A\}$.

(a) For a vertex v with pred(v)=$\emptyset$, we can place a pebble on v. We can also place a pebble on s.

(b) For an AND vertex v, a pebble can be placed if *all* vertices in pred(v) are pebbled.

(c) For an OR vertex v, a pebble can be placed if *at least one* vertex in pred(v) is pebbled.

This problem is also discussed in [Immerman, 1981]. A modified version of this problem is considered in [Yasuura, 1983].
When the vertices of an and/or graph $D = (V, A)$ is linearly ordered, the problem of deciding whether the specified vertex t is reachable from s and simultaneously the vertices of D are breadth-first ordered is also P-complete [Lengauer and Wagner, 1987].

5.3.2 HIERARCHICAL GRAPH ACCESSIBILITY PROBLEM (HGAP)

INSTANCE: A hierarchical graph $\Gamma = (D_1, \ldots, D_k)$ and two vertices $s = (i_1, \ldots, i_m, u_1)$ and $t = (j_1, \ldots, j_n, u_2)$ of the expansion graph $E(\Gamma)$.
PROBLEM: Decide whether there is a path from s to t in the expansion graph $E(\Gamma)$.
Reference: [Lengauer and Wagner, 1987].
Comment: The graph accessibility problem for directed graphs is complete for NLOG [Savitch, 1970]. A *hierarchical graph* is a sequence $\Gamma = (D_1, \ldots, D_k)$ of directed graphs $D_i = (V_i, A_i)$, $1 \le i \le k$, called *subcells*. An example is given in Fig. 5 and Fig. 6. As seen in the example, each subcell D_i consists of three kinds of vertices, *pins* (square vertices), *inner vertices* (black round vertices) and *nonterminals* (labeled with j, $j < i$, called type). The pins are used to connect the corresponding subcell. The inner vertices just form a part of the graph. Each nonterminal v of type i stands for a copy of subcell D_i and there is a one-to-one correspondence between the neighbors of v and the pins of D_i. The *expansion*

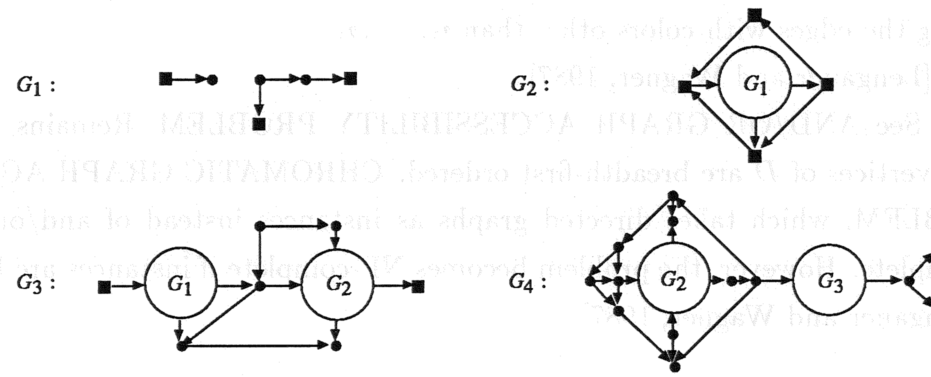


Fig. 5: $\Gamma = (D_1, D_2, D_3, D_4)$

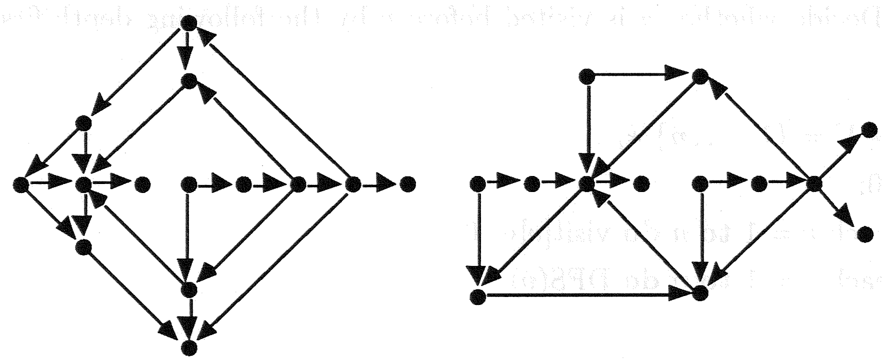


Fig. 6: $E(\Gamma)$

graph $E(\Gamma)$ is defined by expanding D_k recursively. Namely, D_i is expanded by replacing each nonterminal with type j, $j < i$, by a copy of the expansion of subcell D_j through the corresponding pins. By this definition of $E(\Gamma)$, a vertex s of the expansion graph $E(\Gamma)$ is represented by a tuple $(i_1, \ldots, i_m, u)$, where the sequence $i_1 > \cdots > i_m$ represents the hierarchical relation of expansion and u is the vertex of a copy of D_{i_m} corresponding to s in $E(\Gamma)$. Some P-completeness results about hierarchical graphs are also shown in [Lengauer and Wagner, 1987].

5.3.3 AND/OR CHROMATIC GRAPH ACCESSIBILITY PROBLEM

INSTANCE: An and/or graph $D = (V, A)$, positive integers k, m, an edge coloring $\gamma : A \to \{1, \ldots, m\}$, and vertices s, t, where V is linearly ordered as $V = \{1, \ldots, n\}$.
PROBLEM: Decide whether (1) $k \leq m \leq \log |V|$, (2) there are k different colors $i_1, \ldots, i_k$ in

$\{1,\ldots,m\}$ such that t is reachable from s in the graph restricted to the subgraph obtained by removing the edges with colors other than $i_1,\ldots,i_k$.
Reference: [Lengauer and Wagner, 1987].
Comment: See AND/OR GRAPH ACCESSIBILITY PROBLEM. Remains P-complete even if the vertices of D are breadth-first ordered. CHROMATIC GRAPH ACCESSIBILITY PROBLEM, which takes directed graphs as instances instead of and/or graphs, is NLOG-complete. However, the problem becomes NP-complete if instances are hierarchical graphs [Lengauer and Wagner, 1987].

5.3.4 LEXICOGRAPHICALLY FIRST DEPTH-FIRST SEARCH ORDER (LFDFS-ORDER)

INSTANCE: A directed graph $D = (V, A)$ with $V = \{1, ..., n\}$ and two vertices u and v.
PROBLEM: Decide whether u is visited before v by the following depth-first search algorithm:

```
begin /* V = {1,...,n} */
    i ← 0;
    for each v = 1 to n do visit[v] ← 0;
    for each v = 1 to n do DFS(v)
end
```

The procedure DFS numbers the vertices as follows:

```
procedure DFS(v)
    /* For each v ∈ V, we assume a fixed adjacency list ADJ(v) */
    /* of vertices linked by an edge to v */
    begin
      if visit[v]=0 then
          begin
              local u;
              i ← i + 1;
              visit[v] ← i;
              for each u ∈ADJ(v) in the increasing order
                do DFS(u)
          end
    end
```

Reference: [Reif, 1985].

Comment: Remains P-complete even for undirected graphs. But, constructing *any* depth-first search tree is in RNC for an undirected graph [Aggarwal and Anderson, 1988] and for a directed graph [Aggarwal, Anderson and Kao, 1990]. The algorithm runs in $O(TMM(n)(\log n)^3)$ time using $PMM(n)$ processors, where $TMM(n)$ and $PMM(n)$ are the time and the number of processors needed to find a minimum weight perfect matching on an n vertex graph with maximum edge weight n. NC algorithms for depth-first search are known for the following restricted classes of graphs. If G is an undirected planar connected graph with n vertices, there exists a CREW PRAM algorithm for constructing a depth-first spanning tree of G that runs in $O((\log n)^3)$ time using $O(n^4)$ processors [Smith, 1986]. The number of processors of his algorithm can be reduced to $O(n)$ [Ja'Ja' and Kosaraju, 1988], [He and Yesha, 1988]. A CREW PRAM algorithm is also known for a directed acyclic graph with n vertices. The algorithm runs in time $O((\log n)^2)$ using $O(M(n)/\log n)$ processors, where $M(n)$ is the time complexity of $n \times n$ matrix multiplication [Ghosh and Bhattacharjee, 1984]. But it contains an error and is corrected [Zhang, 1986].

5.3.5 LEXICOGRAPHICALLY FIRST MAXIMAL PATH

INSTANCE: A graph $G = (V, E)$ with $V = \{1, \ldots, n\}$ and a vertex u.

PROBLEM: Decide whether u is contained in the lexicographically first maximal path $(v_1, \ldots, v_t)$.

Reference: [Anderson and Mayr, 1987].

Comment: The lexicographically first maximal path can be computed by the following greedy algorithm:

```
begin
  i ← 1; visit i; U ← {i};
  while there is j ∈ V − U with {i, j} ∈ E do
    begin
      let j be the least such vertex;
      i ← j; visit i; U ← U ∪ {i}
    end
end
```

The problem is P-complete even if the instances are restricted to planar graphs with degree at most 3 [Anderson and Mayr, 1987]. The problem of finding *any* maximal (not necessarily lexicographically first) path starting at a given vertex can be solved in RNC [Anderson, 1987]. For graphs with maximum degree $D(n)$, where n is the number of vertices of an instance graph, the maximal path problem can be solved in $O(D(n)(\log n)^3)$ time using $O(n^2)$ processors on a CREW PRAM. Furthermore, the maximal path problem for planar graphs allows a parallel algorithm running in $O((\log n)^3)$ time using $O(n^2)$ processors on a CREW PRAM [Anderson and Mayr, 1987].

5.3.6 LEXICOGRAPHICALLY FIRST MAXIMAL INDEPENDENT SET (LFMIS)

INSTANCE: A graph $G = (V, E)$ with $V = \{1, \ldots, n\}$ and a vertex u.
PROBLEM: Decide whether u is contained in the lexicographically first maximal independent set U.
Reference: [Cook, 1985].
Comment: An *independent set* is a subset U of V such that no two vertices in U are adjacent. A *maximal independent set* is a independent set which is maximal with respect to the order defined by the set inclusion relation. The *lexicographically first maximal independent set* is a maximal independent set which is the first maximal independent set with respect to the lexicographic order on the family of all subsets of $V = \{1, ..., n\}$. The problem is P-complete even for the planar graphs with degree 3 or the bipartite graphs with degree 3 [Miyano, 1989]. But it allows an NC^2 algorithm for forests [Miyano, 1988b]. On the other hand, the problem of finding *any* maximal independent set of a graph (MIS) was shown to be in NC [Karp and Wigderson, 1985]. An NC^2 algorithm is devised using the technique of transforming a random parallel algorithm [Luby, 1986]. MIS is solvable in time $O((\log n)^4)$ using $O(n)$ processors on an EREW PRAM [Goldberg and Spencer, 1989]. A similar problem, LEXICOGRAPHICALLY FIRST MAXIMAL CLIQUE, is also P-complete [Cook, 1985].

5.3.7 LEXICOGRAPHICALLY FIRST MAXIMAL SUBGRAPH PROBLEM FOR π (LFMSP(π))

INSTANCE: A graph (resp., directed graph) $G = (V, E)$ with $V = \{1, \ldots, n\}$ and a vertex v.
PROBLEM: Decide whether u is contained in the lexicographically first maximal (abbreviated to lfm) subset U of V such that the vertex-induced subgraph $G[U]$ of U satisfies the property π, where π is assumed to be polynomial-time testable, hereditary on induced subgraphs and nontrivial.
Reference: [Miyano, 1989].
Comment: A graph property π is said to be *nontrivial* if infinitely many instance graphs satisfy π and at least one instance violates π. The property π is said to be *hereditary*

on (resp., vertex-induced) subgraphs if, whenever a graph G satisfies π, all (resp., vertex-induced) subgraphs of G also satisfy π. For a polynomial-time testable property π which is hereditary on induced subgraphs, LFMSP(π) is solved in polynomial time by the following algorithm:

begin
  $U \leftarrow \emptyset$;
  for $i \leftarrow 1$ **to** n
    if the induced subgraph of $U \cup \{i\}$ satisfies π **then** $U \leftarrow U \cup \{i\}$
end

If the property π is nontrivial, hereditary on induced subgraphs and polynomial-time testable, then LFMSP(π) is P-complete. The reduction is given from LEXICOGRAPHICALLY FIRST MAXIMAL INDEPENDENT SET. The following properties all satisfy the conditions the above: clique, independent set, planar, bipartite, outerplanar, edge graph, chordal, comparability graph, forest, without cycles of length k, maximum degree k, acyclic, etc. When a linear order is given on the edge set as $E = \{e_1 < e_2 < \cdots < e_m\}$, we can define the lfm edge-induced subgraph problems . There is no known general P-completeness result similar to the lfm vertex-induced subgraph problems. The lfm edge-induced forest problem and the lfm edge-induced bipartite subgraph problem are in NC^2. But the lfm edge-induced k-cycle free subgraph problem is P-complete for $k \geq 3$. There are some problems termed with LEXICOGRAPHICALLY FIRST that are shown in NC [Miyano, 1988a, 1988b, 1988c, 1989], [Shoudai, 1989]. For example, the lexicographically first topological order problem is complete for NLOG [Shoudai, 1989]. Therefore the problem is in NC^2. Similar problems are also discussed in [Greenlaw, 1990b].

5.3.8 LEXICOGRAPHICALLY FIRST MAXIMAL BOUNDED DEGREE SUBGRAPH PROBLEM

INSTANCE: A graph $G = (V, E)$ with $V = \{1, ..., n\}$ and a vertex u.
PROBLEM: Decide whether u is contained in the lexicographically first maximal subset U of V such that the vertex-induced subgraph $G[U]$ of U is of degree at most k, where $k \geq 1$ is a fixed integer.
Reference: [Miyano, 1989].
Comment: P-completeness follows from LEXICOGRAPHICALLY FIRST MAXIMAL SUBGRAPH PROBLEM FOR π. The problem of finding *any* maximal bounded degree subgraph is solvable in NC^2 for any degree bound $k \geq 1$ [Shoudai and Miyano, 1990] .

When the edge set E is linearly ordered as $E = \{e_1 < e_2 < \cdots < e_m\}$, we can consider the lexicographically first maximal matching problem (LEXICOGRAPHICALLY FIRST MAXIMAL MATCHING (LFMM)). In contrast with LFMIS, LFMM seems to be neither P-complete nor in NC. LFMM is shown CC-complete [stated as an indirect personal communication of S.A. Cook in [Mayr and Subramanian, 1989]]. In the same way, LEXICOGRAPHICALLY FIRST MAXIMAL BOUNDED DEGREE EDGE-INDUCED SUBGRAPH PROBLEM is CC-complete [Shoudai and Miyano, 1990] .

5.3.9 LEXICOGRAPHICALLY FIRST Δ+1 VERTEX COLORING

INSTANCE: A graph $G = (V, E)$ with $V = \{1, \ldots, n\}$, a vertex u and a color $c \in C = \{1, \ldots, \Delta + 1\}$, where Δ is the maximum vertex degree of G.
PROBLEM: Decide whether vertex u is colored with c by the following sequential algorithm:

begin
  for $v \leftarrow 1$ **to** n **do**
    color(v) $\leftarrow \min\{c \in C \mid c$ is not assigned to any vertex which is adjacent to $v\}$
  end

Reference: [Luby, 1986].
Comment: Any graph can be colored using no more than $\Delta + 1$ colors. There is a CREW PRAM algorithm which computes $\Delta + 1$ vertex coloring. with running time $O((\log n)^3 \log \log n)$ using $O(n)$ processors [Luby, 1988]. G can be colored using at most Δ colors iff G is not an odd cycle and G does not contain a complete subgraph on Δ+1 vertices [Brooks, 1941]. There is an EREW PRAM algorithm which finds a Δ vertex coloring using $O(n^4)$ processors in time $O((\log n)^5)$ [Karloff, 1989]. The problem of deciding whether a given graph can be colored with $\Delta - 1$ colors is NP-complete [Garey and Johnson, 1979].

5.3.10 GREEDY BREADTH-DEPTH SEARCH ORDER

INSTANCE: A graph $G = (V, E)$ with a numbering on the vertices in V, a start vertex s, and two vertices u and v.
PROBLEM: Decide whether u is visited before v by the greedy breadth-depth search algorithm induced by the numbers in the numbering of V.
Reference: [Greenlaw, 1990a].
Comment: Reduction from LEXICOGRAPHICALLY FIRST DEPTH-FIRST SEARCH ORDER. The greedy breadth-depth search algorithm is described as follows:

```
begin
    /* A graph G = (V, E) with V = {1,...,n} is given */
    /* For each v ∈ V, we assume a fixed adjacency list ADJ(v) */
    /* of vertices linked by an edge to v */
    count ← 1;
    for i ← 1 to n do visit[i] ← 0;
    push s on stack S;
    while S ≠ null do
        begin
            v←pop S;
            for each u ∈ ADJ(v) in the increasing order do
                if visit[u]= 0 then
                    begin
                        visit[u]←count;
                        push u on S;
                        count←count+1
                    end
        end
end
```

5.3.11 MINIMUM FEEDBACK VERTEX SET FOR CYCLICALLY REDUCIBLE GRAPHS

INSTANCE: A cyclically reducible directed graph $D = (V, A)$.

PROBLEM: Find a minimum feedback vertex set of D.

Reference: [Bovet, de Agostino and Petreschi, 1988].

Comment: A set $X \subseteq V$ is a *feedback vertex set* if it contains at least one vertex from every cycle in G. The *associated graph of vertex x with respect to D*, denoted $A(D, x)$, consists of x and all vertices not connected by any path to vertices included in cycles after eliminating x from D. Let $y_1, \ldots, y_k$ be a sequence of vertices of D. Then we define graphs $D_0, \ldots, D_k$ inductively as follows: $D_0 = D, D_i = D_{i-1} - A(D_{i-1}, y_i)$ for $i = 1, \ldots, k$. We say that the sequence $y_1, \ldots, y_k$ is a *complete D-sequence* of D if each of $A(D_0, y_1), \ldots, A(D_{k-1}, y_k)$ contains a cycle but D_k is acyclic. Note that D_{i-1} contains a cycle for $i = 1, \ldots, k$ if $y_1, \ldots, y_k$ is a complete D-sequence. A directed graph D is *cyclically reducible* if there exists a complete D-sequence for D. The problem of finding a minimum feedback vertex set for general directed graphs is NP-complete [Garey and Johnson, 1979]. A polynomial time algorithm for finding a minimum feedback vertex set for cyclically reducible graphs is presented by using the fact that $\{y_1, \ldots, y_k\}$ is a minimum feedback vertex set if $y_1, \ldots, y_k$ is a complete D-sequence [Wang, Lloyd and Soffa, 1985]. There is a CREW PRAM algorithm which runs in $O(k(\log n)^2)$ time using $O(n^4)$ processors, where k is the cardinality of the minimum feedback vertex set [Bovet, de Agostino and Petreschi, 1988].

5.3.12 HIGH DEGREE SUBGRAPH (HDS)

INSTANCE: A graph $G = (V, E)$.
PROBLEM: Decide whether there is a subset $U \subseteq V$ of vertices such that the degree of each vertex of the vertex-induced subgraph $G[U]$ of U is at least k, where $k \geq 3$ is a fixed integer.
Reference: [Anderson and Mayr, 1984], [Mayr, 1988].
Comment: HDS for $k = 2$ falls in NC [Mayr, 1988]. It is easy to see that the *maximum* size vertex-induced subgraph each of whose vertices is of degree at least k is unique. The problem of deciding whether a given vertex u is in the maximum size high degree subgraph is also P-complete.

5.3.13 MINIMUM DEGREE ELIMINATION SEQUENCE

INSTANCE: A graph $G = (V, E)$.
PROBLEM: Find a minimum degree elimination sequence.
Reference: [Vishwanathan and Sridhar, 1988].
Comment: Reduction from CIRCUIT VALUE PROBLEM with fanout at most 2. A *minimum* (resp., *maximum*) *degree elimination sequence* of a graph G is an ordering $v_1, v_2, \ldots, v_n$ of the vertices of G such that, for every i, the vertex v_i is of minimum (resp., maximum) degree in the subgraph G_i induced by the vertices $\{v_i, v_{i+1}, \ldots, v_n\}$. A similar problem, MAXIMUM DEGREE ELIMINATION SEQUENCE, can be shown P-complete. For a directed graph, the minimum *indegree* elimination order problem can be defined and it is also P-complete.

With a numbering on the vertices, the *lexicographically first minimum* (resp., *maximum*) *degree elimination sequence* $v_1, ..., v_n$ is defined by choosing the *lowest* numbered vertex v_i which is of minimum (resp., maximum) degree in the subgraph induced by $V - \{v_1, ..., v_{i-1}\}$ for $1 \leq i \leq n$, where $|V| = n$. Obviously, the lexicographic version of these problems are P-complete [Greenlaw, 1988]. Moreover, the following similar problems are considered and some P-completeness results are shown in [Greenlaw, 1988]. For an integer $k \geq 1$, the *lexicographically first less than* k (resp., *greater than* k) *degree elimination sequence* $v_1, ..., v_n$ is defined as follows: If the subgraph induced by $V - \{v_1, ..., v_{i-1}\}$ contains a vertex of degree less than k (resp., greater than k) then v_i is the lowest numbered such vertex. Otherwise v_i is the lowest numbered vertex in $V - \{v_1, ..., v_{i-1}\}$. Then it is shown, for example, that given a graph $G = (V, E)$ with a numbering on the vertices, a vertex u and an integer $t \geq 1$, the problem of deciding whether $u = v_t$ and v_i is of degree greater than k for $1 \leq i \leq t-1$ but v_i is of degree at most k in the lexicographically first greater than k degree elimination sequence is P-complete.

5.3.14 FOUR COLOR INDEX

INSTANCE: A graph G.
PROBLEM: Decide whether the color index of G is at most 4.
Reference: [Vishwanathan and Sridhar, 1988].
Comment: Reduction from MINIMUM DEGREE ELIMINATION ORDER. The *color index* of a graph G is the maximum, over all subgraphs H of G, of the minimum vertex degree of H.

5.3.15 CONGRUENCE CLOSURE

INSTANCE: A directed graph $D = (V, A)$, vertices u, v and an equivalence relation C on V.
PROBLEM: Decide whether u and v are equivalent under the congruence closure of C.
Reference: [Kanellakis and Revesz, 1989].
Comment: The *congruence closure* of C (denoted $\approx_c$) is the finest equivalence relation on V that contains C such that for all vertices v and w with corresponding successors v_1, w_1 and v_2, w_2 we have: $v_1 \approx_c w_1, v_2 \approx_c w_2 \Longrightarrow v \approx_c w$. An algorithm for solving CC is the following: Let D and C, u and v be as above. We define symmetric and reflexive relations E on pairs of vertices of D. This relation is represented by undirected edges added to D and labeled E. For each vertices of u and v that are in the same equivalence class of C we add undirected edge uEv to the graph. Also, add undirected edge uEv if it is not present and either

(a) u_1Ev_1 and u_2Ev_2 are present, where u_1, u_2, and v_1, v_2 are the ordered successors of u, v. In this case u and v are distinct vertices. This is called an *up-propagation step* uPv.

(b) uEw and wEv are present, where w is some vertex in D. In this case u and v are distinct vertices. This is called a *transitivity step* uTv.

The following characterization of the congruence closure relation is known:

$u \approx_c v$ iff undirected edge uEv is added after some finite sequence of up-propagation and transitivity steps.

Remains P-complete even if C is the reflexive, symmetric, and transitive closure of two pairs of distinct vertices (we say that C has two axioms). When C has no axioms, that is each distinct vertex is an equivalence class, the problem is in NC^2. When C has only one axiom, then the complexity is open.

If D is acyclic, the followings are known: (1) P-complete if C has three axioms. (2) NC^2 if C has one axiom. (3) When C has two axioms, then the complexity is open. (4) When D is a simple directed acyclic graph (simple dag) with k axioms ($k \geq 2$), the complexity is open. A simple dag is a dag D such that the only vertices of D with indegree greater than 1 are leaves.

5.3.16 UNIFICATION CLOSURE

INSTANCE: A directed graph $D = (V, A)$, vertices u, v of D and an equivalence relation C on V.
PROBLEM: Decide whether u and v are equivalent under the unification closure of C.
Reference: [Kanellakis and Revesz, 1989].
Comment: The *unification closure* of C (denoted $\sim_c$) is the finest equivalence relation on V that contains C such that for all vertices v and w with corresponding successors v_1, w_1 and v_2, w_2 we have: $v \sim_c w \Longrightarrow v_1 \sim_c w_1, v_2 \sim_c w_2$. If D has at most a fixed number of leaves, then the complexity is in NC^2.

5.3.17 ACYCLIC CONGRUENCE CLOSURE

INSTANCE: A directed graph $D = (V, A)$, and an equivalence relation C on V.
PROBLEM: Decide whether the graph contracted by the congruence closure of C is acyclic.
Reference: [Kanellakis and Revesz, 1989].
Comment: We say that D is acyclic under the equivalence relation C if the directed graph we get by the contracting the vertices in each equivalence class of C is still acyclic. The problem is in NC^2 if C has a fixed number of nontrivial equivalence classes.

5.3.18 ACYCLIC UNIFICATION CLOSURE

INSTANCE: A directed graph $D = (V, A)$, and an equivalence relation C on V.
PROBLEM: Decide whether the graph contracted by the unification closure of C is acyclic.
Reference: [Kanellakis and Revesz, 1989].
Comment: See ACYCLIC CONGRUENCE CLOSURE.

5.3.19 GENERAL GRAPH CLOSURE

INSTANCE: A graph $G = (V, E)$ and a set $E' \subseteq V \times V - E$.
PROBLEM: Decide whether the general closure $gc(G, E')$ of G contains an edge $\{u, v\}$ of E'.
Reference: [Khuller, 1989].
Comment: The *general closure* $gc(G, E')$ of G is the graph obtained from $G = (V, E)$ by repeatedly adding an edge in E' such that the sum of degrees of its two endpoints is at least n, until no such edge remains. Obviously, the general graph closure $gc(G, E')$ is uniquely determined.

5.3.20 HIGH VERTEX CONNECTIVITY SUBGRAPH (HVCS)

INSTANCE: A graph $G = (V, E)$.
PROBLEM: Decide whether G contains an induced subgraph of vertex connectivity at least k, where k is any fixed integer with $k \geq 3$.
Reference: [Kirousis, Serna and Spirakis, 1989].
Comment: Reduction from ALTERNATING MONOTONE CVP with fanout 2 (see CIRCUIT VALUE PROBLEM). The *vertex (edge) connectivity* $k(G)$ $(\lambda(G))$ of a graph G is the minimum number of vertices (edges) whose removal results in a disconnected or trivial graph. A graph $G = (V, E)$ is m-vertex (m-edge) connected if $k(G) \geq m$ $(\lambda(G) \geq m)$. The optimization version of HVCS (resp., HIGH EDGE CONNECTIVITY SUBGRAPH (HECS)) asks what is the largest k such that there is an induced subgraph of vertex (edge) connectivity k. Let HVCS(G) (resp., HECS(G)) denote this largest k. An approximate solution to this problem is to find d such that HVCS(G)$\geq d \geq c$ HVCS(G) (resp., HECS(G)$\geq d \geq c$ HECS(G)) for some fixed $c < 1$. This problem cannot be approximated for $c > 1/2$ $(c > 1/2)$ in NC. However HVCS (resp., HECS) can be approximated in NC for $c < 1/4$ (resp., $c < 1/2$).

5.3.21 HIGH EDGE CONNECTIVITY SUBGRAPH (HECS)

INSTANCE: A graph $G = (V, E)$.
PROBLEM: Decide whether G contains an induced subgraph of edge connectivity at least k, where k is any fixed integer with $k \geq 3$.
Reference: [Kirousis, Serna and Spirakis, 1989].
Comment: See HIGH VERTEX CONNECTIVITY SUBGRAPH.

5.3.22 APPROXIMATION OF RELIABLY LONG PATH

INSTANCE: A directed acyclic graph $D = (V, A)$, a vertex v, a subset $U \subseteq V$ of vertices and an integer K.
PROBLEM: Assume that for each $u \in U$ exactly one of the incoming edges to u fails with probability $1/indegree(u)$. The length L of the reliably longest path is defined to be the maximum number l such that D has a path starting from v of length at least l, with probability 1. Then decide whether $L \geq K \geq \epsilon L$, where ϵ is a fixed rational number in $(0,1]$.
Reference: [Kirousis and Spirakis, 1988].
Comment: Reduction from APPROXIMATION OF CIRCUIT DEPTH OF ONES [Kirousis and Spirakis, 1988].

5.3.23 UNARY NETWORK FLOW FOR VERTEX MULTIPLICITY GRAPHS

INSTANCE: A directed vertex multiplicity graph $M = (V, A, b)$ with two distinguished vertices, a source s and a sink t, a positive integer capacity $c(u, v)$ represented in unary on every edge $(u, v) \in A$, and a positive unary integer K.
PROBLEM: Decide whether there exists a flow of at least K units from s to t in a directed graph represented by M.
Reference: [Karpinski and Wagner, 1988].
Comment: See MAXIMUM FLOW. Reduction from MAXIMUM FLOW. A *directed vertex multiplicity graph* is a triple $M = (V, A, b)$, where V is a finite set, A is a subset of $V \times V$ and b is a positive integer function on V. The directed vertex multiplicity graph $M = (V, A, b)$ represents a directed graph $D = (V', A')$ such that $V' = \{(v, i) \mid v \in V \text{ and } 1 \leq i \leq b(v)\}$ and $A' = \{((v, i), (u, j)) \mid (u, v) \in A, 1 \leq i \leq b(v) \text{ and } 1 \leq j \leq b(u)\}$. An *undirected vertex multiplicity graph* is also defined similarly.

5.3.24 PERFECT MATCHING FOR VERTEX MULTIPLICITY GRAPHS

INSTANCE: An undirected vertex multiplicity graph $M = (V, E, b)$.
PROBLEM: Decide whether there is a perfect matching in a graph represented by M.
Reference: [Karpinski and Wagner, 1988].
Comment: See UNARY NETWORK FLOW FOR VERTEX MULTIPLICITY GRAPH. Remains P-complete even for undirected vertex multiplicity graphs which represent bipartite graphs. It is known that this problem for usual graphs is in RNC [Karp, Upfal and Wigderson, 1985], [Mulmuley, Vazirani and Vazirani, 1987], but not known to be in NC.

5.3.25 PERFECT B-MATCHING WITH CAPACITIES

INSTANCE: A graph $G = (V, E)$, functions $b : V \to \{1, 2, \ldots\}$ and $c : E \to \{1, 2, \ldots\}$.
PROBLEM: Decide whether there is a function $c' : E \to \{0, 1, \ldots\}$ such that $c'(\{u, v\}) \leq c(\{u, v\})$ for every $\{u, v\} \in E$ and $\sum_{v \in adj(u)} c'(\{u, v\}) = b(u)$ for all $u \in V$.
Reference: [Karpinski and Wagner, 1988].
Comment: This problem was shown to be in P by [Grötschel, Lovasz and Schrijver, 1988].

5.4 Optimization

5.4.1 LINEAR PROGRAMMING (LP)

INSTANCE: An integer $n \times d$ matrix A, an integer n-vector b, an integer d-vector c and an integer B.
PROBLEM: Decide whether there is a vector x such that $Ax \leq b$ and $cx \geq B$.
Reference: [Dobkin, Lipton and Reiss, 1979], [Khachian, 1979], [Dobkin and Reiss, 1980].
Comment: The number d represents the number of variables called the dimension and n is the number of inequality constraints. This problem was shown to be in P by [Khachian, 1979], [Karmarkar, 1984]. For any fixed dimension d, LP with n inequalities can be solved on a probabilistic CRCW PRAM with $nd/(\log d)^2$ processors almost surely in $O(d^2(\log d)^2)$ time [Alon and Megiddo, 1990]. The algorithm always finds the correct solution. The probability that the algorithm will not finish within $O(d^2(\log d)^2)$ time tends to zero exponentially with respect to n. A parallel algorithm solving LP (not an NC algorithm) is also developed [Vaidya, 1990].

5.4.2 LINEAR INEQUALITIES (LI)

INSTANCE: An integer $n \times d$ matrix A and an integer n-vector b.
PROBLEM: Decide whether there is a rational d-vector $x > 0$ such that $Ax \leq b$.
Reference: This problem is stated in [Greenlaw, Hoover and Ruzzo, 1989] as a private communication with [Cook, 1982].
Comment: The reduction from CIRCUIT VALUE PROBLEM to LI is as follows: An input $x = true$ (resp. $x = false$) is represented by the equation $x = 1$ (resp. $x = 0$). A NOT gate $y = \neg x$ with input x and output y is represented by the inequalities $y = 1 - x$ and $0 \leq y \leq 1$. An AND gate $z = x \wedge y$ with inputs x, y and output z is represented by the inequalities $0 \leq z \leq 1$, $z \leq x$, $z \leq y$, and $x + y - 1 \leq z$.

5.4.3 MAXIMUM FLOW

INSTANCE: A directed graph $D = (V, A)$ with two distinguished vertices, a source s and a sink t, a positive integer capacity $c(u, v)$ on every edge $(u, v) \in A$, and a positive integer i.
PROBLEM: Decide whether the ith bit of the value of the maximum flow from s to t in D is 1.
Reference: [Goldschlager, Shaw and Staples, 1982].
Comment: Reduction from MONOTONE CIRCUIT VALUE PROBLEM with fanout at most 2. A *flow* f is an assignment of a nonnegative integer to each edge of D such that
(a) for every $(u, v) \in A$, $f(u, v) \leq c(u, v)$, and
(b) for every $v \in V - \{s, t\}$, $\Sigma_{(u,v)\in A} f(u, v) = \Sigma_{(v,w)\in A} f(v, w)$.
The *maximum flow* is a flow f whose total flow $|f| = \Sigma_{(v,t)\in A} f(v, t)$ is maximum. The edge capacities defined in [Goldschlager, Shaw and Staples, 1982] are bounded by $2^{|V|}$. It is an open question whether the problem for polynomial capacities or 0-1 capacities is P-complete. The problem restricted to planar directed graphs can be solved in $O((\log n)^3)$ time using $O(n^4)$ processors or $O((\log n)^2)$ time using $O(n^6)$ processors on a CREW PRAM [Johnson, 1987]. A parallel algorithm for MAXIMUM FLOW that runs in $O(n^2 \log n)$ time using $O(n)$ processors on a CRCW PRAM is also known [Shiloach and Vishkin, 1982]. Constructing a maximum flow in a directed graph whose edge weights are given in unary lies in RNC using the technique of maximum perfect matching [Karp, Upfal and Wigderson, 1985]. It seems that the parallel complexity of the maximum flow problem depends critically on whether the capacities are given in unary or in binary. Nevertheless, there is a randomized parallel algorithm to construct a maximum flow in a directed graph whose edge weights are given in binary, such that the number of processors is bounded by a polynomial in the number of vertices and the time is $O((\log n)^k \log C)$ for some constant k, where C is the largest capacity of any edge [Karp, Upfal and Wigderson, 1985]. The maximum flow problem in the following form is also P-complete: Given a directed graph $D = (V, A)$ with two vertices s and t, a capacity $c(u, v)$ for $(u, v) \in A$, and an integer K, decide whether the maximum flow is as large as K [Lengauer and Wagner, 1987].

5.4.4 LEXICOGRAPHICALLY FIRST BLOCKING FLOW

INSTANCE: A directed acyclic graph $D = (V, A)$ with a numbering on the vertices, two distinguished vertices s and t, a positive integer capacity $c(u, v)$ on every edge $(u, v) \in A$, and a positive integer i.
PROBLEM: Decide whether the ith bit of the value of the lexicographically first blocking flow from s to t in D is 1.

Reference: [Cheriyan and Maheshawari, 1989].

Comment: See MAXIMUM FLOW. A flow f is a *blocking flow* if for every path from s to t contains a *saturated edge* (u,v), i.e., $f(u,v) = c(u,v)$. The *lexicographically first blocking flow* is the flow which is generated by the sequential depth-first search blocking flow algorithm [Dinic, 1970]: Find a path from s to t by using the lexicographic depth-first search method. Then find an edge e with the least capacity, say c, on the path from s to t. Push the capacity c on each edge of the path to saturate some edges including e and delete all newly saturated edges from the graph. Repeat this procedure until t is not reachable from s. A maximum flow can be found by iterating any blocking flow algorithm at most n times. There is a simple parallel algorithm for finding a blocking flow which runs in $O(n \log n)$ time using m processors on an EREW PRAM, where m is the number of edges [Goldberg and Tarjan, 1989]. A *layered network* is an acyclic graph in which all s-t paths have the same length. Remains P-complete even if D is a layered network of length 3 [Cheriyan and Maheshawari, 1989].

5.4.5 SUPERINCREASING KNAPSACK PROBLEM

INSTANCE: An integer w and a sequence of integers $w_1, w_2, ..., w_n$ such that for all $2 \leq i \leq n$ $w_i > \sum_{j=1}^{i-1} w_j$.

PROBLEM: Decide whether there is a sequence of 0-1 valued variables $x_1, x_2, ..., x_n$ such that $w = \sum_{j=1}^{n} x_j w_j$.

Reference: [Karloff and Ruzzo, 1989].

Comment: Reduction from NAND CIRCUIT VALUE PROBLEM. The general well-known 0-1 knapsack problem is NP-complete. Superincreasing knapsacks are known as the basis for the Merkle-Hellman cryptosystem [Merkle and Hellman, 1978]. However, the system is shown to be of questionable security [Adleman, 1983], [Shamir, 1982].

5.4.6 FIRST FIT DECREASING BIN PACKING

INSTANCE: A sequence $p_1, ..., p_n$ of pieces with values $v_1, ..., v_n$, respectively, which are rational numbers satisfying $1 \geq v_1 \geq \cdots \geq v_n \geq 0$ and an assignment $I : \{p_1, ..., p_n\} \rightarrow \{b_1, ..., b_k\}$, called a *packing*, of the pieces into bins $b_1, \ldots, b_k$, where the sum of the values of the pieces in each bin must be ≤ 1, i.e., $\sum_{p_j \in I^{-1}(b_i)} v_j \leq 1$ for $1 \leq i \leq k$.

PROBLEM: Decide whether I is the packing produced by the first fit decreasing (FFD) algorithm.

Reference: [Anderson, Mayr and Warmuth, 1989].

Comment: The FFD algorithm considers the pieces in the order of non-increasing size.

Starting with v_1 until v_n, the FFD algorithm places each piece into the first (lowest numbered) bin that has enough remaining capacity. It has been shown that the length of the packing generated by FFD is at most $\frac{11}{9}\mathrm{OPT}(I) + 3$, where $\mathrm{OPT}(I)$ is the optimal number of bins for I [Garey and Johnson, 1979].

5.4.7 GENERAL LIST SCHEDULING

INSTANCE: A list of jobs $(J_1, \ldots, J_n)$, a positive integer execution time $T(J_i)$ for each job, and a nonpreemptive schedule L with m processors.
PROBLEM: Decide whether L is the schedule produced by the list scheduling algorithm.
Reference: [Helmbold and Mayr, 1987].
Comment: In nonpreemptive scheduling, a job once begun must be executed to completion. Whenever a processor becomes free, the scheduler simply performs a fixed directed scan of the list and assigns to the processor the first job encountered which has not yet been executed and whose predecessors have all been completed.

5.4.8 HEIGHT-PRIORITY SCHEDULE

INSTANCE: A directed acyclic graph $D = (V, A)$ and a profile μ.
PROBLEM: Find a height-priority schedule for D and μ.
Reference: [Dolev, Upfal and Warmuth, 1986].
Comment: A *profile* μ is a function $\mu : \{0, 1, ...\} \to \{1, 2, ...\}$, where $\mu(i)$ represents the number of processors available at the ith time slot. A profile is called *straight* if it is a constant function.

A *schedule* s for a directed acyclic graph $D = (V, A)$ and a profile μ is an onto function $s : V \to \{0, 1, ..., l-1\}$ for some l satisfying (a) and (b).

(a) $|s^{-1}(r)| \leq \mu(r)$ for each r in $\{0, 1, ..., l-1\}$. This means that the number of processors used at the rth time slot is at most $\mu(r)$.

(b) If $(x, y) \in A$, then $s(x) < s(y)$.

Let q be a function from V to $\{0, 1, ...\}$. A schedule is called a *q-priority schedule* if it satisfies (c) (q is called a *priority function*).

(c) $s(x) > s(y)$ and $q(x) > q(y)$ implies that x is a successor of some vertex z with $s(z) = s(y)$.

A *height-priority schedule* is a schedule with $height(x)$ as a priority function, where $height(x)$ represents the length of the longest path starting at x.

A directed acyclic graph D is said to be an *outforest* (resp., *inforest*) if every vertex has at most one immediate predecessor (resp., immediate successor). The problem is still

P-complete even if D is an outforest and μ is nondecreasing.

When the profile is assumed to be straight, the problem of finding the following schedule is P-complete.

(1) A height-priority schedule for a graph consisting of an outforest and an inforest.

(2) A height-priority schedule for a graph in which the vertices of every component are partitioned into levels according to the height of the vertices and the vertices of a level precede all vertices of the same component of the levels below it.

(3) A weighted-priority schedule for a weighted outforest using only three different weights, where the priority function is given by the weights on the vertices.

On the other hand, an optimal height-priority schedule for a straight profile and an outforest can be found by an $O(\log n)$ time (resp., $O((\log n)^2)$ time) EREW PRAM algorithm using n^2 processors (resp., n processors).

5.4.9 NEAREST NEIGHBOR TRAVELING SALESMAN PROBLEM

INSTANCE: A distance matrix $(D_{ij})_{1\le i,j\le n}$ and a vertex v_1, where $0 \le D_{ij} \le \infty$ is an integer or the symbol ∞ specifying the distance between vertices i and j. If the distance $D_{ij} = \infty$, it means that there is no edge between i and j.

PROBLEM: Find a nearest neighbor heuristic tour starting at v_1.

Reference: [Kindervater, Lenstra and Shmoys, 1989].

Comment: A sequence of vertices $v_1, ..., v_n$ with $\{v_1, ..., v_n\} = \{1, ..., n\}$ is called a *nearest neighbor heuristic tour* starting at v_1 such that the distance $D_{v_{i-1}v_i}$ between v_{i-1} and v_i is equal to $\min\{D_{v_{i-1}v_j} \mid i \le j \le n\}$ for each $1 < i \le n$. It should be noted that there may be several nearest neighbor heuristic tours since the nearest neighbors are not necessarily unique. However, the reduction is given from CVP so that the instance of NEAREST NEIGHBOR TRAVELING SALESMAN PROBLEM transformed from a circuit has a unique solution.

A nearest neighbor heuristic tour is found by the following *nearest neighbor heuristic*:

1. Start at vertex v_1.

2. Among all vertices not yet visited, choose as a vertex which is nearest to the current vertex with respect to the distance matrix $(D_{ij})_{1\le i,j\le n}$. Repeat this step until all vertices have been visited.

The problem is still P-complete even if the distance matrix satisfies the triangle inequality. On the other hand, the double minimum spanning tree heuristic and the nearest addition heuristic described below can be implemented in NC.

(1) *Double minimum spanning tree heuristic:* Construct a minimum-weight spanning tree, double its edges and construct an Eulerian circuit of this double edged spanning tree.

Then start at a given vertex and traverse the Eulerian circuit, skipping vertices visited before.

(2) *Nearest addition heuristic:* Start with a tour consisting of a given vertex with a self-loop. Find vertices j not on the tour and k on the tour such that D_{jk} is minimum. Then insert j directly before k. Repeat this step until all vertices are inserted.

However, it is not known whether the Christofides heuristic described below is in NC. This is simply because the parallel complexity of the maximum matching problem is unresolved.

(3) *Christofides heuristic:*

1. Construct a minimum-weight spanning tree T.

2. Find the vertices of odd degree and find the minimum perfect matching M in the complete graph consisting of these vertices of odd degree.

3. Construct an Eulerian circuit of the multigraph with vertices $1, 2, ..., n$ and edges in $T \cup M$. Then start at a given vertex and traverse the Eulerian circuit, skipping vertices visited before.

5.4.10 NEAREST MERGER TRAVELING SALESMAN PROBLEM

INSTANCE: A distance matrix $(D_{ij})_{1 \le i,j \le n}$, where $0 \le D_{ij} \le \infty$ is an integer or the symbol ∞ specifying the distance between vertices i and j.

PROBLEM: Find a nearest merger heuristic tour.

Reference: [Kindervater, Lenstra and Shmoys, 1989].

Comment: The *nearest merger heuristic* is described as follows:

1. Start with n partial tours, each consisting of a single vertex with a self-loop.

2. Find tours C and C' such that the distance $dist(C, C')$ between C and C' is minimum, where $dist(C, C') = \min\{D_{ik} \mid i \in C,\ k \in C'\}$. Let $\{i, j\}$ be an edge of C and $\{k, l\}$ an edge of C' for which $D_{ik} + D_{jl} - D_{ij} - D_{kl}$ is minimum. Then merge C and C' by replacing edges $\{i, j\}$ and $\{k, l\}$ by $\{i, k\}$ and $\{j, l\}$, respectively. Repeat this step until a complete tour is obtained.

Since tours C, C' and edges $\{i, j\}$, $\{k, l\}$ are not necessarily uniquely determined, the resulting tour depends on the choices of C, C', $\{i, j\}$, and $\{k, l\}$. Therefore the above heuristic does not specify a unique tour. A *nearest merger heuristic tour* is a tour obtained by the above heuristic by appropriately specifying the choices in step 2. However, the reduction is given from CVP so that a circuit is transformed to an instance of NEAREST MERGER TRAVELING SALESMAN PROBLEM for which the nearest merger heuristic is unique.

The problem is still P-complete even if the distance matrix satisfies the triangle inequality.

5.4.11 NEAREST INSERTION TRAVELING SALESMAN PROBLEM

INSTANCE: A distance matrix $(D_{ij})_{1\leq i,j\leq n}$ and a vertex v, where $0 \leq D_{ij} \leq \infty$ is an integer or the symbol ∞ specifying the distance between vertices i and j.
PROBLEM: Find a nearest insertion heuristic tour starting at v.
Reference: [Kindervater, Lenstra and Shmoys, 1989].
Comment: A *nearest insertion heuristic tour* starting at v is a tour obtained by the *nearest insertion heuristic* described as follows:

1. Start with a tour consisting of v with a self-loop.
2. Find a vertex not on the tour which is nearest to a vertex already contained in the tour. Insert this vertex between two neighboring vertices on the tour in the cheapest possible way. Repeat this step until the tour is complete.

By a reason similar to NEAREST MERGER TRAVELING SALESMAN PROBLEM, the nearest insertion heuristic does not specify a unique tour. However, the reduction is given from CVP so that a circuit is transformed to an instance of NEAREST INSERTION TRAVELING SALESMAN PROBLEM which has a unique nearest insertion heuristic tour.

The problem is still P-complete even if the distance matrix satisfies the triangle inequality.

5.4.12 CHEAPEST INSERTION TRAVELING SALESMAN PROBLEM

INSTANCE: A distance matrix $(D_{ij})_{1\leq i,j\leq n}$ and a vertex v, where $0 \leq D_{ij} \leq \infty$ is an integer or the symbol ∞ specifying the distance between vertices i and j.
PROBLEM: Find a cheapest insertion heuristic tour starting at v.
Reference: [Kindervater, Lenstra and Shmoys, 1989].
Comment: A *cheapest insertion heuristic tour* starting at v is a tour obtained by the *cheapest insertion heuristic* described below. The cheapest insertion heuristic does not necessarily specify a unique tour. However, CVP is reduced so that a circuit is transformed to an instance of CHEAPEST INSERTION TRAVELING SALESMAN PROBLEM which has a unique solution.

1. Start with a tour consisting of v with a self-loop.
2. Find a vertex not on the tour which can be inserted between two neighboring vertices on the tour in the cheapest possible way. Insert this vertex between two neighboring vertices on the tour in the cheapest possible way. Repeat this step until the tour is complete.

The problem is still P-complete even if the distance matrix satisfies the triangle inequality.

5.4.13 FARTHEST INSERTION TRAVELING SALESMAN PROBLEM

INSTANCE: A distance matrix $(D_{ij})_{1\leq i,j\leq n}$ and a vertex v, where $0 \leq D_{ij} \leq \infty$ is an integer or the symbol ∞ specifying the distance between vertices i and j.
PROBLEM: Find a farthest insertion heuristic tour starting at v.
Reference: [Kindervater, Lenstra and Shmoys, 1989].
Comment: A *farthest insertion heuristic tour* starting at v is a tour obtained by the *farthest insertion heuristic* described below. By a reason similar to NEAREST MERGER TRAVELING SALESMAN PROBLEM, the farthest insertion heuristic does not necessarily specify a unique tour. The reduction is given from CVP so that the resulting instance of FARTHEST INSERTION TRAVELING SALESMAN PROBLEM transformed from a circuit has a unique solution.

1. Start with a tour consisting of v with a self-loop.
2. Find a vertex not on the tour for which the minimum distance to a vertex on the tour is maximum. Insert this vertex between two neighboring vertices on the tour in the cheapest possible way. Repeat this step until the tour is complete.

The problem is still P-complete even if the distance matrix satisfies the triangle inequality.

5.4.14 FINITE HORIZON MARKOV DECISION PROCESS

INSTANCE: A Markov decision process $M = (S, c, p)$ and two integers T and K.
PROBLEM: Decide whether there is a policy δ such that the expectation of the cost $\sum_{t=0}^{T} c(s_t, \delta(s_t, t), t)$ is at most K.
Reference: [Papadimitriou and Tsitsiklis, 1987].
Comment: A Markov decision process $M = (S, c, p)$ consists of the following: S is a finite set of states with a special state s_0 called the initial state. For each state $s \in S$, a finite set D_s of decisions is given. For each time $t = 0, 1, 2, \ldots$, each state s and each decision $i \in D_s$, an integer cost $c(s, i, t)$ is given. $p(s, s', i, t)$ denotes the probability that the process transits from state s to state s' with decision $i \in D_s$ at time t. For each time t, we denote by s_t the state at time t. The process is *stationary* if c and p are independent of t, i.e., $c(s, i, t) = c(s, i, t')$ and $p(s, s', i, t) = p(s, s', i, t')$ for all t, t'. A *policy* δ is a mapping which gives a decision $\delta(s, t) \in D_s$ for each time t and state s. It is not known whether the stationary finite horizon problem is in P. For deterministic cases, i.e., the probability

is either 0 or 1, the stationary finite horizon problem (resp. nonstationary finite horizon problem, discounted problem, average cost problem) is in NC.

5.4.15 DISCOUNTED MARKOV DECISION PROCESS

INSTANCE: A Markov decision process $M = (S, c, p)$, a rational number $0 < \beta \leq 1$, and an integer K.
PROBLEM: Decide whether there is a policy δ such that the expectation of the discounted cost $\sum_{t=0}^{\infty} c(s_t, \delta(s_t, t), t)\beta^t$ is at most K.
Reference: [Papadimitriou and Tsitsiklis, 1987].
Comment: See FINITE HORIZON MARKOV DECISION PROCESS.

5.4.16 AVERAGE COST MARKOV DECISION PROCESS

INSTANCE: A Markov decision process $M = (S, c, p)$ and an integer K.
PROBLEM: Decide whether there is a policy δ such that the expectation of the cost $\lim_{T\to\infty}(\sum_{t=0}^{T} c(s_t, \delta(s_t, t), t))/T$ is at most K.
Reference: [Papadimitriou and Tsitsiklis, 1987].
Comment: See FINITE HORIZON MARKOV DECISION PROCESS.

5.5 Formal Language

5.5.1 CONTEXT-FREE GRAMMAR EMPTINESS

INSTANCE: A context-free grammar $G = (N, \Sigma, P, S)$.
PROBLEM: Decide whether $L(G) = \emptyset$.
Reference: [Jones and Laaser, 1977].
Comment: Reduction from GENERATABILITY.

5.5.2 CONTEXT-FREE GRAMMAR INFINITY

INSTANCE: A context-free grammar $G = (N, \Sigma, P, S)$.
PROBLEM: Decide whether $L(G)$ is infinite.
Reference: [Jones and Laaser, 1977].
Comment: Reduction from CONTEXT-FREE GRAMMAR EMPTINESS.

5.5.3 CONTEXT-FREE GRAMMAR MEMBERSHIP

INSTANCE: A context-free grammar $G = (N, \Sigma, P, S)$ and $x \in \Sigma^*$.

PROBLEM: Decide whether $S \Rightarrow^* x$.
Reference: [Jones and Laaser, 1977].
Comment: Reduction from GENERATABILITY. For a fixed context-free grammar G, it can be determined in NC^2 that a given string x is in $L(G)$ or not with respect to $|x|$ [Ruzzo, 1980].

5.5.4 STRAIGHT-LINE PROGRAM MEMBERSHIP

INSTANCE: A string $x \in \Sigma^*$ and a straight-line program I with operations in $\Phi = \Sigma \cup \{\{\epsilon\}, \emptyset, \cup, \cdot\}$, where Σ is a finite alphabet.
PROBLEM: Decide whether x is a member of the set constructed by the program I.
Reference: [Goodrich, 1983].
Comment: A straight-line program is a sequence of assignments of the following forms: $X \leftarrow Y \cup Z$, $X \leftarrow Y \cdot Z$, $X \leftarrow \emptyset$, $X \leftarrow \{\epsilon\}$, $X \leftarrow \{a\}$, where $a \in \Sigma$ and X, Y and Z are variables.

5.5.5 STRAIGHT-LINE PROGRAM NONEMPTINESS

INSTANCE: A straight-line program I with operations in $\Phi = \Sigma \cup \{\{\epsilon\}, \emptyset, \cup, \cdot\}$, where Σ is a finite alphabet.
PROBLEM: Decide whether the set constructed by the program I is not empty.
Reference: [Goodrich, 1983].
Comment: See STRAIGHT-LINE PROGRAM MEMBERSHIP.

5.5.6 LABELED GRAPH ACCESSIBILITY PROBLEM (LGAP)

INSTANCE: A directed graph $D = (V, A)$ with edges labeled by strings over $\{a_1, \bar{a}_1, a_2, \bar{a}_2\}$ and two vertices s and t.
PROBLEM: Decide whether there is a path from s to t such that the concatenation of its labels on the edges is in the semi-Dyck set D_2.
Reference: [Greenlaw, Hoover and Ruzzo, 1985].
Comment: The *semi-Dyck set* D_2 is the language generated by the context-free grammar with productions $S \rightarrow S a_i S \bar{a}_i S | \epsilon$ $(i = 1, 2)$, where S is the start symbol. If D is acyclic, the problem is complete for LOGCFL, the class of sets log space reducible to context-free languages.

5.6 Algebraic Problems

5.6.1 INTEGER ITERATED MOD (IIM)

INSTANCE: Binary positive integers $a, b_1, b_2, ..., b_n$.
PROBLEM: Decide whether $((\cdots((a \bmod b_1) \bmod b_2)\cdots \bmod b_n) = 0$.
Reference: [Karloff and Ruzzo, 1989].
Comment: Reduction from NAND CIRCUIT VALUE PROBLEM. The *integer iterated mod problem* is related to the problem of computing the greatest common divisor (gcd) of two numbers. No NC integer gcd algorithm is known. The *polynomial iterated mod problem* is: given univariate polynomials $a(x), b_1(x), ..., b_n(x)$ over a field F, compute the residue $((\cdots((a(x) \bmod b_1(x)) \bmod b_2(x))\cdots \bmod b_n(x)) = 0$. The problem over any field F is in Arithmetic-NC^2, where Arithmetic-NC^2 is the class of problems solvable in time $O((\log n)^2)$ with $O(n^{O(1)})$ processors by using the computation model in which inputs and outputs are elements of the field F and operations are the primitive field operations. The problem is in NC^2 if F is a fixed finite field or the rationals Q.

5.6.2 GENERATABILITY (GEN)

INSTANCE: A set X, a binary operation $\cdot$ on X, a subset $S \subseteq X$, and an element $x \in X$.
PROBLEM: Decide whether x is contained in the smallest subset of X which contains S and is closed under the binary operation $\cdot$.
Reference: [Jones and Laaser, 1977].
Comment: Reduction from UNIT RESOLUTION. The binary operation $\cdot$ in the reduction is commutative but not associative. GENERATABILITY for associative $\cdot$ is complete for NLOG.

5.6.3 UNIFORM WORD PROBLEM FOR FINITELY PRESENTED ALGEBRAS

INSTANCE: A finitely presented algebra $A = (M, \Gamma)$ and terms x, y over M, where M is a finite ranked alphabet and Γ is a finite set of unordered pairs of terms called the axioms.
PROBLEM: Decide whether $x \equiv_\Gamma y$, x is equivalent to y under the congruence relation generated by Γ.
Reference: [Kozen, 1977].
Comment: Reduction from CIRCUIT VALUE PROBLEM. A finite ranked alphabet M is a pair (Σ, a), where Σ is a finite set of symbols and a assigns each symbol $\sigma \in \Sigma$ a nonnegative integer $a(\sigma)$ called the arity of σ. A term over M is defined inductively as

follows: (1) Every element $\sigma \in \Sigma$ with arity $a(\sigma) = 0$ is a term. (2) If $\theta \in \Sigma$ has arity $a(\theta) = m$ and $x_1, \ldots, x_m$ are terms, then $\theta(x_1, \ldots, x_m)$ is a term. We denote the set of terms over M by $W(M)$. An axiom $\{u, v\} \in \Gamma$ is denoted by $u \equiv_\Gamma v$. The relation $\equiv_\Gamma$ on $W(M)$ is the smallest congruence relation on $W(M)$ satisfying the axioms in Γ. Namely, $\equiv_\Gamma$ is the smallest equivalence relation satisfying (1) for each axiom $\{u, v\} \in \Gamma$, $u \equiv_\Gamma v$ and (2) if $\theta \in \Sigma$ has arity m and $s_1 \equiv_\Gamma t_1, \ldots, s_m \equiv_\Gamma t_m$, then $\theta(s_1, \ldots, s_m) \equiv_\Gamma \theta(t_1, \ldots, t_m)$.

5.6.4 TRIVIAL ALGEBRA

INSTANCE: A finitely presented algebra $A = (M, \Gamma)$.
PROBLEM: Decide whether A is a trivial algebra, i.e., consisting of one element.
Reference: [Kozen, 1977].
Comment: See UNIFORM WORD PROBLEM FOR FINITELY PRESENTED ALGEBRAS.

5.6.5 FINITE ALGEBRA

INSTANCE: A finitely presented algebra $A = (M, \Gamma)$.
PROBLEM: Decide whether A is finite.
Reference: [Kozen, 1977].
Comment: Reduction from CIRCUIT VALUE PROBLEM. See UNIFORM WORD PROBLEM FOR FINITELY PRESENTED ALGEBRAS.

5.6.6 SUBALGEBRA MEMBERSHIP

INSTANCE: A finitely presented algebra $A = (M, \Gamma)$ and terms $x_1, \ldots, x_n, y$.
PROBLEM: Decide whether $[y]$ is contained in the subalgebra generated by $[x_1], \ldots, [x_n]$, where $[z]$ denotes the equivalence class under $\equiv_\Gamma$.
Reference: [Kozen, 1977].
Comment: Reduction from GENERATABILITY. See UNIFORM WORD PROBLEM FOR FINITELY PRESENTED ALGEBRAS.

5.6.7 UNIFORM WORD PROBLEM FOR LATTICES

INSTANCE: A finite set E of equations and an equation $e_1 = e_2$.
PROBLEM: Decide whether $E \models e_1 = e_2$.
Reference: [Cosmadakis, 1988].

Comment: A *lattice* is a set L with two binary operations $+, \cdot$ satisfying the following axioms: For any $x, y, z \in L$,

(i) $(x \cdot y) \cdot z = x \cdot (y \cdot z)$, $(x + y) + z = x + (y + z)$ (associativity).

(ii) $x \cdot y = y \cdot x$, $x + y = y + x$ (commutativity).

(iii) $x \cdot x = x$, $x + x = x$ (idempotence).

(iv) $x + (x \cdot y) = x$, $x \cdot (x + y) = x$ (absorption).

Let U be a countably infinite set of symbols. A *term* over U is defined inductively as follows. The set of terms over U is denoted by $W(U)$.

(1) If $\alpha \in U$, then α is a term.

(2) If $p, q \in W(U)$, then $(p + q)$ and $(p \cdot q)$ are terms.

An *equation* is a formula of the form $e_1 = e_2$, where $e_1, e_2 \in W(U)$. Given a lattice L, a *valuation* is defined to be a function $\mu : U \to L$ and extended to $W(U)$ by defining $\mu(p+q) = \mu(p) + \mu(q)$, $\mu(p \cdot q) = \mu(p) \cdot \mu(q)$. A lattice L satisfies an equation $e_1 = e_2$ under a valuation μ, denoted $L \models_\mu e_1 = e_2$, if $\mu(e_1) = \mu(e_2)$. We say that E implies $e_1 = e_2$, denoted $E \models e_1 = e_2$, if for every lattice L and every valuation μ such that $L \models_\mu E$, we have $L \models_\mu e_1 = e_2$. Remains P-complete even if $E = \emptyset$ and terms are represented by directed acyclic graphs. But under the tree representation of terms, if $E = \emptyset$, the problem is solvable in DLOG.

5.6.8 GENERATOR PROBLEM FOR LATTICES

INSTANCE: A finite set E of equations and terms $e, g_1, \ldots, g_n$.

PROBLEM: Decide whether $E \models gen(e, g_1, \ldots, g_n)$.

Reference: [Cosmadakis, 1988].

Comment: See UNIFORM WORD PROBLEM FOR LATTICES. Let X be a subset of a lattice L. The *sublattice* generated by X is the smallest subset of L which contains X and is closed under the operations $+, \cdot$ of L. Given a valuation $\mu : U \to L$ and terms $e, g_1, \ldots, g_n$ over U, we say that e is generated by $g_1, \ldots, g_n$ in L under μ, denoted $L \models_\mu gen(e, g_1, \ldots, g_n)$, if $\mu(e)$ is in the sublattice of L generated by the set $\{\mu(g_i) | i = 1, \ldots, n\}$. $E \models gen(e, g_1, \ldots, g_n)$ if, for every lattice L and valuation μ such that $L \models_\mu E$, we have $L \models_\mu gen(e, g_1, \ldots, g_n)$. Remains P-complete even if $E = \emptyset$ and terms are represented by directed acyclic graphs. But under the tree representation of terms, if $E = \emptyset$, the problem is solvable in DLOG.

5.6.9 GENERALIZED WORD PROBLEM (GWP)

INSTANCE: A finite subset $U = \{u_1, \ldots, u_n\}$ of $(\Sigma \cup \bar{\Sigma})^*$ and a word x in $(\Sigma \cup \bar{\Sigma})^*$, where

Σ is a finite alphabet and $\bar{\Sigma} = \{\bar{a} \mid a \in \Sigma\}$.

PROBLEM: Decide whether x is in $\langle U \rangle$.

Reference: [Avenhaus and Madlener, 1984a].

Comment: Generic reduction. Let F be the free group with generators Σ in which only trivial relations $a\bar{a} = \bar{a}a = e$ hold for $a \in \Sigma$. $\langle U \rangle$ denotes the subgroup of F generated by U. The word problem WP is to decide for x whether $x = e$ in F. It is known that WP is solvable in DLOG [Lipton and Zalcstein, 1977]. The decision algorithm for GWP is based upon the Nielsen reduction algorithm.

5.6.10 SUBGROUP

INSTANCE: Finite sets U and V of $(\Sigma \cup \bar{\Sigma})^*$, where Σ is a finite alphabet.

PROBLEM: Decide whether $\langle U \rangle$ is a subgroup of $\langle V \rangle$.

Reference: [Avenhaus and Madlener, 1984a].

Comment: Reduction from GENERALIZED WORD PROBLEM. It is also P-complete to decide whether $\langle U \rangle$ is a normal subgroup of $\langle V \rangle$.

5.6.11 SUBGROUP EQUALITY

INSTANCE: Finite sets U and V of $(\Sigma \cup \bar{\Sigma})^*$, where Σ is a finite alphabet.

PROBLEM: Decide whether $\langle U \rangle = \langle V \rangle$.

Reference: [Avenhaus and Madlener, 1984a].

Comment: Reduction from SUBGROUP.

5.6.12 GROUP INDEPENDENT SET

INSTANCE: Finite set U of $(\Sigma \cup \bar{\Sigma})^*$, where Σ is a finite alphabet.

PROBLEM: Decide whether U is independent, i.e., each $x \in \langle U \rangle$ has a unique freely reduced representation.

Reference: [Avenhaus and Madlener, 1984a].

Comment: Generic reduction. A word is called *freely reduced* if it contains no segment $a\bar{a}$ or $\bar{a}a$, for any $a \in \Sigma$.

5.6.13 SUBGROUP ISOMORPHISM

INSTANCE: Finite sets U and V of $(\Sigma \cup \bar{\Sigma})^*$, where Σ is a finite alphabet.

PROBLEM: Decide whether $\langle U \rangle$ is isomorphic to $\langle V \rangle$.

Reference: [Avenhaus and Madlener, 1984a].

Comment: Reduction from GROUP INDEPENDENT SET.

5.6.14 FINITE INDEX SUBGROUP

INSTANCE: Finite sets U and V of $(\Sigma \cup \bar{\Sigma})^*$, where Σ is a finite alphabet.

PROBLEM: Decide whether $\langle U \rangle$ is a subgroup of $\langle V \rangle$ that has finite index in $\langle V \rangle$.

Reference: [Avenhaus and Madlener, 1984a].

Comment: Reduction from SUBGROUP.

5.6.15 GROUP INDUCED ISOMORPHISM

INSTANCE: Finite sets $U = \{u_1, \ldots, u_m\}$ and $V = \{v_1, \ldots, v_m\}$ of $(\Sigma \cup \bar{\Sigma})^*$, where Σ is a finite alphabet.

PROBLEM: Decide whether the mapping φ defined by $\varphi(u_i) = v_i$ $(i = 1, \ldots, m)$ induces an isomorphism from $\langle U \rangle$ onto $\langle V \rangle$.

Reference: [Avenhaus and Madlener, 1984a].

Comment: Reduction from GROUP INDEPENDENT SET.

5.6.16 GROUP RANK

INSTANCE: Finite set U of $(\Sigma \cup \bar{\Sigma})^*$ and a positive integer k, where Σ is a finite alphabet.

PROBLEM: Decide whether there is an independent set V of size k such that $\langle U \rangle = \langle V \rangle$.

Reference: [Avenhaus and Madlener, 1984a].

Comment: Reduction from GROUP INDEPENDENT SET.

5.6.17 GROUP COSET INTERSECTION

INSTANCE: Finite sets U and V of $(\Sigma \cup \bar{\Sigma})^*$ and words x and y in $(\Sigma \cup \bar{\Sigma})^*$, where Σ is a finite alphabet.

PROBLEM: Decide whether $\langle U \rangle x \cap y \langle V \rangle \neq \emptyset$.

Reference: [Avenhaus and Madlener, 1984b].

Comment: It is also P-complete to decide whether $\langle U \rangle x \cap \langle V \rangle y \neq \emptyset$ (resp., $x \langle U \rangle \cap y \langle V \rangle \neq \emptyset$). Remains P-complete even if $x = y = e$, i.e., $\langle U \rangle \cap \langle V \rangle \neq \langle e \rangle$.

5.6.18 CONJUGATE SUBGROUPS

INSTANCE: Finite sets U and V of $(\Sigma \cup \bar{\Sigma})^*$, where Σ is a finite alphabet.

PROBLEM: Decide whether there is $x \in (\Sigma \cup \bar{\Sigma})^*$ such that $x^{-1} \langle U \rangle x = \langle V \rangle$.

Reference: [Avenhaus and Madlener, 1984b].

Comment: It is also P-complete to decide whether there is $x \in (\Sigma \cup \bar{\Sigma})^*$ such that $x^{-1}\langle U \rangle x$ is a subgroup of $\langle V \rangle$.

5.6.19 GROUP COSET EQUALITY

INSTANCE: Finite sets U and V of $(\Sigma \cup \bar{\Sigma})^*$ and words x and y in $(\Sigma \cup \bar{\Sigma})^*$, where Σ is a finite alphabet.
PROBLEM: Decide whether $\langle U \rangle x = y \langle V \rangle$.
Reference: [Avenhaus and Madlener, 1984b].
Comment: It is also P-complete to decide whether $\langle U \rangle x = \langle V \rangle y$.

5.6.20 GROUP COSET EQUIVALENCE

INSTANCE: Finite sets U and V of $(\Sigma \cup \bar{\Sigma})^*$, where Σ is a finite alphabet.
PROBLEM: Decide whether there are $x, y \in (\Sigma \cup \bar{\Sigma})^*$ such that $\langle U \rangle x = y \langle V \rangle$.
Reference: [Avenhaus and Madlener, 1984b].
Comment: It is also P-complete to decide whether there are $x, y \in (\Sigma \cup \bar{\Sigma})^*$ such that $\langle U \rangle x = \langle V \rangle y$.

5.6.21 GROUP CONJUGACY EQUIVALENCE

INSTANCE: Finite sets U and V of $(\Sigma \cup \bar{\Sigma})^*$, where Σ is a finite alphabet.
PROBLEM: Decide whether there is $x \in (\Sigma \cup \bar{\Sigma})^*$ such that $x^{-1}\langle U \rangle x = \langle V \rangle$.
Reference: [Avenhaus and Madlener, 1984b].
Comment: It is also P-complete to decide whether there is $x \in (\Sigma \cup \bar{\Sigma})^*$ such that $x^{-1}\langle U \rangle x$ is a subgroup of $\langle V \rangle$.

5.7 Miscellaneous

5.7.1 ZIV-LEMPEL CODING

INSTANCE: A binary string S and a positive integer z.
PROBLEM: Decide whether there is a codeword, whose binary value is z, in the Ziv-Lempel coding of S.
Reference: [de Agostino, 1990].
Comment: Reduction from CIRCUIT VALUE PROBLEM. A data compression method called the *Ziv-Lempel coding* consists of a rule for parsing strings of symbols from a finite alphabet into substrings and a coding scheme that maps these substrings into uniquely decipherable words called the *codewords*. The parsing of the data string is executed by a

procedure that creates a new phrase as soon as a prefix of the still unparsed part of string differs from all preceding phrases. The first phrase is the first symbol of the string and the last one is the only one allowed to be equal to a preceding one. We denote by p_i the ith phrase of the parsing (p_0 is the empty string). For each phrase p_i, there is one nonnegative integer $j \leq i$ such that $p_j x = p_i$, where x is the last symbol of p_i. Then, the codeword for p_i is composed by $\lceil \log(i\alpha) \rceil$ bits that are the binary representation of the number $j\alpha + f(x)$, where α is the cardinality of the alphabet and f a mapping from the alphabet onto the set of the integers $\{0, ..., \alpha - 1\}$. For example, for the word $a\ b\ b\ a\ a\ b\ a\ b\ b\ c\ c$, the Ziv-Lemple parsing is the sequence a, b, ba, ab, abb, c, c.

5.7.2 MILLER-WELGMAN CODING

INSTANCE: A binary string S and a positive integer z.

PROBLEM: Decide whether there is a codeword, whose binary value is z, in the Miller-Welgman coding of S.

Reference: [de Agostino, 1990].

Comment: Reduction from CIRUCIT VALUE PROBLEM. The *Miller-Welgman coding* is a variation of the Ziv-Lempel coding in order to improve compression efficiency. The Miller-Welgman coding is the following: A table is initialized to contain all strings of length 1 Then, the parsing rule creates as a new phrase p_i the longest prefix of the still unparsed part of string, that is matched in the table and encodes it with a codeword of $\lceil \log(i + \alpha) \rceil$ bits that are the binary representation of its concatenation of p_i with the first character of the next phrase. Afterwards, the table is augmented by adding the string which is the concatenation of p_i with the first character of the next phrase. For example, for the word $a\ b\ b\ a\ a\ b\ a\ b\ b\ c\ c$, the Miller-Welgman parsing is the sequence a, b, b, a, ab, ab, b, c, c with the table a, b, c, ab, bb, ba, aa, aba, abb, bc, cc.

5.7.3 CONVEX HULL

INSTANCE: A finite set $S \subseteq \mathbf{Q}^n$ and $x \in \mathbf{Q}^n$, where $\mathbf{Q}$ is the set of rational numbers.

PROBLEM: Decide whether x is in the convex hull of S.

Reference: [Long and Warmuth, 1990].

Comment: The *convex hull* of S is the set of all convex combinations on elements in S. By using linear programming [Khachian, 1979] , one can test in polynomial time whether x is in the convex hull of S. The reduction is from MONOTONE CIRCUIT VALUE PROBLEM and the dimension n of $\mathbf{Q}^n$ depends on the number of gates in a circuit.

5.7.4 ENVELOPE LAYERS

INSTANCE: A set of line segments $L = \{\overline{p_1q_1}, \overline{p_2q_2}, \ldots, \overline{p_nq_n}\}$ in the plane, a distinguished point x on some line segment of L and a positive integer k.
PROBLEM: Decide whether the layer depth of a point x is k.
Reference: [Hershberger, 1990].
Comment: Reduction from MONOTONE CIRCUIT VALUE PROBLEM. A line segment $\overline{pq}$, where p and q are points in the plane, is a linear combination $\alpha p + (1 - \alpha)q$ ($\alpha \in R, 0 \leq \alpha \leq 1$). Remains P-complete even for a set which consists of disjoint segments. An *upper envelope* is a collection of the most upper line segments, which consists of pieces of segments in L. The set of line segments are scanned by repeatedly computing the upper envelope and discarding the pieces of segments that appear on it. A *layer depth* of a point on some segment in L is an iteration number at which it appears on the upper envelope.

5.7.5 GAUSSIAN ELIMINATION WITH PARTIAL PIVOTING (GEPP)

INSTANCE: A matrix A with entries over the reals rounded up to two decimal places, and integers i, j.
PROBLEM: When Gaussian elimination with partial pivoting is done on A, decide whether the pivot used to eliminate the jth column entries is taken from row i.
Reference: [Vavasis, 1989].
Comment: Reduction from NAND CIRCUIT VALUE PROBLEM. In Gaussian elimination, the elements along the diagonal are used to eliminate the entries in the columns below them. The number a_{ii} and position (i, i) on the diagonal of A used for elimination are called the *pivot value* and the *pivot position*, respectively. A problem arises if zero or a very small number appears in the pivot position. A solution to this problem with the elimination algorithm is to swap the rows of the matrix before each selection of a new pivot. In particular, when the algorithm starts to eliminate below the diagonal in column i, it first searches all the entries in the column from i downward to find the entry with the largest magnitude, say at position (j, i). Then row i is swapped with row j in order to bring the larger number into the pivot position. This row-swapping technique is called *partial pivoting.* A *complete pivoting* is to swap both rows and columns in order to bring the largest element in the remaining uneliminated submatrix into the pivot position.
Gaussian elimination is the oldest and best known method for solving systems of linear equations, and partial pivoting is the most common technique to make elimination numerically stable.

The problem is also P-complete if complete pivotings are used. If exact rational arithmetic is used instead of arithmetic rounded to two decimals, the problem is again P-complete. The problem of deciding whether the pivot value for column j is positive is P-complete both for partial and complete pivotings.

If the problem uses scaling, it is not known P-complete. There are other numerical algorithms that use interchanges; for example, column interchanges are used in QR-factorization if rank-deficiency is a possibility. This may also lead to a P-complete problem.

If all the pivot positions were somehow known in advance, then the Gaussian elimination could be carried out in NC. A parallel algorithm based on Newton iterations is shown [Pan and Reif, 1985]. Their algorithm runs in $O((\log n)^{O(1)})$ time and $O(n^3)$ processors.

5.7.6 TWO-PLAYER GAME

INSTANCE: A 5-tuple $J = (P_1, P_2, W_0, s, M)$ called a *two-player game*, where $P_1 \cap P_2 = \emptyset$, $W_0 \subseteq P_1 \cup P_2$, $s \in P_1$, and $M \subseteq P_1 \times P_2 \cup P_2 \times P_1$.
PROBLEM: Decide whether s is a winning position for player one.
Reference: [Jones and Laaser, 1977].
Comment: Reduction from GENERATABILITY. Informally, P_1 (resp., P_2) is the set of positions in which it is player one's (resp., player two's) turn to move. W_0 denotes the set of positions in which player one has an immediate win, and s is the starting position. M denotes the set of allowable moves: if $(p, q) \in M$ and $p \in P_1$ (resp., $p \in P_2$) then player one (resp., player two) may move from position p to position q in a single step. The set W of *winning positions* for player one is defined inductively as follows: (1) $W_0 \subseteq W$. (2) If $x \in P_1$ and $(x, y) \in M$ for some winning position y, then x is in W. (3) If $x \in P_2$ and y is a winning position for every move (x, y) in M, then x is in W.

5.7.7 STRATEGY ELIMINATION IN TWO PERSON GAME

INSTANCE: A two person game $J = (A, B)$, where $A = (a_{ij})$ and $B = (b_{ij})$ are $m \times n$ matrices with integer entries, and an integer i.
PROBLEM: Decide whether a strategy i is eliminated in a reduced game.
Reference: [Knuth, Papadimitriou and Tsitsiklis, 1988].
Comments: Reduction from MONOTONE CIRCUIT VALUE PROBLEM. Player x chooses a row i from A (this choice is called a strategy) and player y simultaneously chooses a column j from B. We say that strategy i (resp., j) of player x (resp., y) dominates strategy i' (resp., j') of the same player if $a_{ik} \geq a_{i'k}$ (resp., $b_{kj} \geq b_{kj'}$) for $k = 1, \ldots, n$ (resp., $k = 1, \ldots, m$). A game $J = (A, B)$ is reduced by eliminating the ith row (resp., jth column) from both matrices A and B if some row of A (resp., column of B) dominates row

i of A (resp., column j of B). A *reduced game* is a game in which no further elimination is possible. We note that the reduced game is unique up to row and column permutation. Remains P-complete even for *zero-sum games*, i.e., $A + B = O$. But it is easy to see that the problem of deciding whether a game is already reduced is in NC.

5.7.8 GENERAL DEADLOCK DETECTION

INSTANCE: Two $n \times m$ matrices $P = (P_{ij})$ and $Q = (Q_{ij})$ and integers $w_1, \ldots, w_m$ with P_{ij}, $Q_{ij} \leq w_j$ for $j = 1, ..., m$, where P_{ij} and Q_{ij} denote, respectively, the number of units of resource R_j held and requested by process p_i for $i = 1, ..., n$.
PROBLEM: Decide whether P and Q represent a deadlock state.
Reference: [Spirakis, 1987].
Comment: We say that matrices P and Q represent a *deadlock state* if there exists a subset $S \subseteq \{p_1, \ldots, p_n\}$ of processes such that every $p_i \in S$ waits for some $p_k \in S$ $(i \neq k)$ to release resources, i.e., for every $p_i \in S$, there is a resource R_j satisfying $Q_{ij} > w_j - \Sigma_{p_k \in S} P_{kj}$. There is an NC algorithm for the problem restricted to single unit resources, i.e., $w_j = 1$ for all j. This algorithm takes $O((\log n)^2)$ time using a CRCW PRAM of $O(n^3/\log n)$ processors for instances which represent an expedient state, i.e., all satisfiable requests have been granted and single unit requests are occurred.

5.7.9 BOOLEAN RECURRENCE EQUATION

INSTANCE: An $m \times n$ boolean matrix M, an $n \times n$ boolean matrix B, an $n \times 1$ boolean vector F, and an integer j $(0 \leq j \leq n)$.
PROBLEM: Decide whether the first entry of $M \cdot Y_j$ is 1, where Y_j is an $n \times 1$ boolean vector defined inductively as $Y_0 = F$, $Y_k = B \cdot \bar{Y}_{k-1}$ for $k \geq 1$ ($\bar{Y}_{k-1}$ is a boolean vector obtained by negating each entry of Y_{k-1}).
Reference: [Bertoni, Bollina, Mauri and Sabadini, 1985].
Comment: If the integer j is in range $0 \leq j \leq (\log n)^k$ for some constant k, this problem is complete for AC^k, which is the class of sets computed by alternating Turing machines with $O((\log n)^k)$ alternations.

5.7.10 MOSTOWSKI EPIMORPHISM

INSTANCE: A directed acyclic graph $D = (V, A)$ which satisfies the axiom of extensionality, i.e., $\forall u \forall v((\forall x[(x,u) \in A \Leftrightarrow (x,v) \in A]) \Rightarrow u = v)$, and two vertices x_1 and $x_2 \in V$.
PROBLEM: Decide whether $M_D(x_1) = M_D(x_2)$, where M_D is the Mostowski epimorphism for D.

Reference: [Dahlhaus, 1988].
Comment: Reduction from MONOTONE CIRCUIT VALUE PROBLEM. For a finite set V, we define $\tilde{V} = \cup_{i=0}^{\infty} V_i$, where $V_0 = V$ and $V_{i+1} = V_i \cup 2^{V_i}$ for $i \geq 0$. A *Mostowski epimorphism* for a directed acyclic graph $D = (V, A)$ satisfying the axiom of extensionality is a function M_D from V to $\tilde{V}$ which satisfies $M_D(x) = \{M_D(y) \mid (y, x) \in A\}$. A sequential polynomial time algorithm which computes a Mostowski epimorphism is known.

5.7.11 STRONG BISIMILARITY

INSTANCE: A finite labeled transition system $M = (Q, \Sigma, T)$ and two states $p, q \in Q$.
PROBLEM: Decide whether p and q are strongly bisimilar.
Reference: [Àlvarez, Balcázar, Gabarró and Santha, 1990].
Comment: A *finite labeled transition system* is a triple $M = (Q, \Sigma, T)$, where Q is a finite set of states, Σ is a finite alphabet of actions and $T \subseteq Q \times \Sigma \times Q$ is a set of transitions. A relation $S \subseteq Q \times Q$ is a *strong bisimulation* if $(p, q) \in S$ implies, for all $x \in \Sigma$, the following bisimilarity conditions hold:

(a) If $(p, x, p') \in T$, then for some $q' \in Q$, $(q, x, q') \in T$ and $(p', q') \in S$.

(b) If $(q, x, q') \in T$, then for some $p' \in Q$, $(p, x, p') \in T$ and $(p', q') \in S$.

The *strong bisimilarity relation* is defined as the union of all strong bisimulations.

The problem of deciding observation equivalence and the problem of deciding observation congruence of two states in a finite labeled transition system are also P-complete [Àlvarez, Balcázar, Gabarró and Santha, 1990].

5.8 Remarks

An NC^0 *permutation* is a one-to-one onto function $f : \{0,1\}^* \to \{0,1\}^*$ with $f(\{0,1\}^n) = \{0,1\}^n$ for each $n \geq 0$ such that some NC^0 family of circuits $\{C_n\}_{n \geq 0}$ computes $\{f_n\}_{n \geq 0}$, where $f_n = f \mid_{\{0,1\}^n}: \Sigma^n \to \Sigma^n$. There are NC^0 permutations whose inverses are as hard to compute as the parity function [Boppana and Lagarias, 1987]. These permutations can be said to be *one-way* since the parity function is known not to be computable even by unbounded fanin polynomial size circuits of constant depth [Furst, Saxe and Sipser, 1984]. There is an NC^0 permutation f such that its inverse f^{-1} is P-complete. The reduction is given from CVP [Håstad, 1988].

References

[1] Adleman, L.M. [1983], On breaking generalized knapsack public key cryptosystems, *Proc 15th ACM STOC*, 402-412.

[2] Aggarwal, A. and Anderson, R.J. [1988], A random NC algorithm for depth first search, *Combinatorica* **8**, 1-12.

[3] Aggarwal, A., Anderson, R.J. and Ming-Yang Kao [1990], Parallel depth-first search in general directed graphs, *SIAM J. Comput.* **19**, 397-409.

[4] Alon, N. and Megiddo, N. [1990], Parallel linear programming in fixed dimension almost surely in constant time, *Proc. 31th IEEE FOCS*, 574-582.

[5] Àlvarez, C., Balcázar, J.L., Gabarró, J. and Santha, M. [1990], Parallel complexity in the design and analysis of concurrent systems, manuscript.

[6] Anderson, R. [1987], A parallel algorithm for the maximal path problem, *Combinatorica* **7**, 315-326.

[7] Anderson, R. and Mayr, E.W. [1984], Parallelism and greedy algorithms, Report No. STAN-CS-84-1003, Dept. of Computer Science, Stanford University.

[8] Anderson, R. and Mayr, E.W. [1987], Parallelism and the maximal path problem, *Inf. Process. Lett.* **24**, 121-126.

[9] Anderson, R., Mayr, E.W. and Warmuth, M.K. [1989], Parallel approximation algorithms for bin packing, *Inform. and Comput.* **82**, 262-277.

[10] Avenhaus, J. and Madlener, K. [1984a], The Nielsen reduction and P-complete problems in free groups, *Theoret. Comput. Sci.* **32**, 61-76.

[11] Avenhaus, J. and Madlener, K. [1984b], On the complexity of intersection and conjugacy problems in free groups, *Theoret. Comput. Sci.* **32**, 279-295.

[12] Bertoni, A., Bollina, M.C., Mauri, G. and Sabadini, N. [1985], On characterizing classes of efficiently parallelizable problems, VLSI: Algorithms and Architectures. *Proc. Int. Workshop on Parallel Computing and VLSI*, 13-26, North-Holland, Amsterdam, Netherlands.

[13] Boppana, R. and Lagarias, J. [1987], One way functions and circuit complexity, *Inform. and Comput.* **74**, 226-240.

[14] Bovet, D.P., de Agostino, S. and Petreschi, R. [1988], Parallelism and the feedback vertex set problem, *Inf. Process. Lett.* **28**, 81-85.

[15] Brooks, R.L. [1941], On coloring the nodes of a network, *Proc. Cambridge Philos. Soc.* **37**, 194-197.

[16] Chang, C.L. and Lee, R.C.T. [1973], Symbolic Logic and Mechanical Theorem Proving, Academic Press, New York.

[17] Cheriyan, J. and Maheshawari, S.N. [1989], The parallel complexity of finding a blocking flow in a 3-layer network, *Inf. Process. Lett.* **31**, 157-161.

[18] Cook, S.A. [1974], An observation on time-storage trade off, *J. Comput. System Sci.* **9**, 308-316.

[19] Cook, S.A. [1985], A taxonomy of problems with fast parallel algorithms, *Inform. and Control* **64**, 2-22.

[20] Cosmadakis, S.S. [1988], The word and generator problems for lattices, *Inform. and Comput.* **77**, 192-217.

[21] Dahlhaus, E. [1988], Is SETL a suitable language for parallel programming - a theoretical approach, *Proc. CSL 87* (Lecture Notes in Computer Science **329**), 56-63.

[22] de Agostino, S. [1990], P-complete problems in data compression, Report No. URLS-DM/NS-90/001(INFO), Dept. of Mathematics, University of Rome "La Sapienza".

[23] Dinic, E.A. [1970], Algorithm for solution of a problem of maximum flow in a network with power estimation, *Soviet Math. Dokl.* **11**, 1277-1280.

[24] Dobkin, D., Lipton, R.J. and Reiss, S. [1979], Linear programming is log-space hard for P, *Inf. Process. Lett.* **9**, 96-97.

[25] Dobkin, D. and Reiss, S. [1980], The complexity of linear programming, *Theoret. Comput. Sci.* **11**, 1-18.

[26] Dolev, D., Upfal, E. and Warmuth, M.K. [1986], The parallel complexity of scheduling with precedence constraints, *J. Parallel and Distributed Computing* **3**, 553-576.

[27] Dwork, D., Kanellakis, P.C. and Michell, J.C. [1984], On the sequential nature of unification, *J. Logic Programming* **1**, 35-50.

[28] Dwork, D., Kanellakis, P.C. and Stockmeyer, L.J. [1988], Parallel algorithms for term matching, *SIAM J. Comput.* **17**, 711-731.

[29] Furst, M., Saxe, J. and Sipser, M. [1984], Parity, circuits, and the polynomial time hierarchy, *Math. Syst. Theory* **17**, 13-27.

[30] Garey, M.R. and Johnson, D.S. [1979], Computers and Intractability: a Guide to the Theory of NP-completeness, W.H. Freeman and Company, San Francisco.

[31] Ghosh, R.K. and Bhattacharjee, G.P. [1984], A parallel search algorithm for directed acyclic graphs, *BIT* **24**, 134-150.

[32] Goldberg, M. and Spencer, T. [1989], A new parallel algorithm for the maximal independent set problem, *SIAM J. Comput.* **18**, 419-427.

[33] Goldberg, M. and Tarjan, R.E. [1989], A parallel algorithm for finding a blocking flow in an acyclic network, *Inf. Process. Lett.* **31**, 265-271.

[34] Goldschlager, L.M. [1977], The monotone and planar circuit value problems are log space complete for P, *SIGACT News* **9**, 25-29.

[35] Goldschlager, L.M. [1980], A space efficient algorithm for the monotone planar circuit value problem, *Inf. Process. Lett.* **10**, 25-27.

[36] Goldschlager, L.M., Shaw, R.A. and Staples, J. [1982], The maximum flow problem is log-space complete for P, *Theoret. Comput. Sci.* **21**, 105-111.

[37] Goodrich, G.B. [1983], The complexity of finite languages, Ph.D Dissertation, University of Washinton.

[38] Greenlaw, R. [1988], Ordered vertex removal and subgraph problems, *J. Comput. System Sci.* **39**, 323-342.

[39] Greenlaw, R., Hoover, H. and Ruzzo, W.L. [1989], A compendium of problems complete for P, manuscript.

[40] Greenlaw, R. [1990a], Breadth-depth search is P-complete, Report No. 90-60, Department of Computer Science, University of New Hampshire.

[41] Greenlaw, R. [1990b], The parallel complexity of approximation algorithms for the acyclic subgraph problem, Report No. 90-61, Department of Computer Science, University of New Hampshire.

[42] Grötschel, M., Lovasz, L. and Schrijver, A. [1988], Geometric algorithms and combinatorial optimization, To appear in Springer-Verlag.

[43] Håstad, J. [1988], One-way permutation in NC^0, *Inf. Process. Lett.* **26**, 153-155.

[44] He, X. and Yesha, Y. [1988], A nearly optimal parallel algorithm or constructing depth first spanning trees in planar graphs, *SIAM J. Comput.* **17**, 486-491.

[45] Helmbold, D. and Mayr, E. [1987], Fast scheduling algorithms on parallel computers, F.P. Preparata, ed., Advances in Computing Research: Parallel and Distributed Computing, 39-68, JAI Press.

[46] Hershberger, J. [1990], Upper envelope onion peeling, manuscript.

[47] Hopcroft, J.E. and Ullman, J.D. [1979], Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, Mass.

[48] Immerman, N. [1981], Number of Quantifiers is better than number of tape cells, *J. Comput. System Sci.* **22**, 384-406.

[49] Ja'Ja, J. and Kosaraju, S. [1988], Parallel algorithms for planar graphs and related problems, *IEEE Trans. Circ. Syst.* March.

[50] Johnson, D.B. [1987], Parallel algorithms for minimum cuts and maximum flows in planar networks, *J. Assoc. Comput. Mach.* **34**, 950-967.

[51] Jones, N.D. and Laaser, T. [1977], Complete problems for deterministic polynomial time, *Theoret. Comput. Sci.* **3**, 105-117.

[52] Kanellakis, P.C. and Revesz, P.Z. [1989], On the relationship of congruence closure and unification, *J. Symbolic Computation* **7**, 427-444.

[53] Karloff, H.J. [1989], An NC algorithm for Brooks' theorem, *Theoret. Comput. Sci.* **68**, 89-103.

[54] Karloff, H.J. and Ruzzo, W.L. [1989], The iterated mod problem, *Inform. and Comput.* **80**, 193-204.

[55] Karmarkar, N. [1984], A new polynomial-time algorithm for linear programming, *Combinatorica* **4**, 373-395.

[56] Karp, R.M., Luby, M. and Madras, N. [1989], Monte-Carlo approximation algorithms for enumeration problems, *J. Algorithms* **10**, 429-448.

[57] Karp, R.M., Upfal, E. and Wigderson, A. [1985], Constructing a perfect matching in random NC, *Proc. 17th ACM STOC*, 22-32.

[58] Karp, R.M. and Wigderson, A. [1985], A fast parallel algorithm for the maximal independent set problem, *J. Assoc. Comput. Mach.* **32**, 762-773.

[59] Karpinski, M. and Wagner, K.W. [1988], The computational complexity of graph problems with succinct multigraph representation, *ZOR, Methods Models Oper. Res.* **32**, 201-211.

[60] Kasif, S. [1986], On the parallel complexity of some constraint satisfaction problems, *Proc. AAAI-86*, 349-353.

[61] Khachian, L.G. [1979], A polynomial time algorithm for linear programming, *Doklady Akad. Nauk SSSR* **244**, No. 4, 1093-1096. Translated in *Soviet Math. Doklady* **20**, 191-194.

[62] Khuller, S. [1989], On computing graph closures, *Inf. Process. Lett.* **31**, 249-255.

[63] Kindervater, G.A.P., Lenstra, J.K. and Shmoys, D.B. [1989], The parallel complexity of TSP heuristics, *J. Algorithms* **10**, 249-270.

[64] Kirousis, L.M., Serna, M. and Spirakis, P. [1989], The parallel complexity of the subgraph connectivity problem, *Proc. 30th IEEE FOCS*, 294-299.

[65] Kirousis, L.M. and Spirakis, P. [1988], Probabilistic log-space reductions and problems probabilistically hard for P, *Proc. SWAT 88* (Lecture Notes in Computer Science **318**), 163-175.

[66] Knuth, D.E., Papadimitriou, C.H. and Tsitsiklis, J.N. [1988], A note on strategy elimination in bimatrix games, *Oper. Res. Lett.* **7**, 103-107.

[67] Kozen, D. [1977], Complexity of finitely presented algebras, *Proc. 9th ACM STOC*, 164-177.

[68] Ladner, R.E. [1975], The circuit value problem is log space complete for P, *SIGACT News* **7**, 18-20.

[69] Landau, G.M. and Vishkin, U. [1989], Fast parallel and serial approximate string matching, *J. Algorithms* **10**, 157-169.

[70] Lengauer, T. and Wagner, K.W. [1987], The correlation between the complexities of the non-hierarchical and hierarchical versions of graph problems, *Proc. STACS 87* (Lecture Notes in Computer Science **247**), 100-113.

[71] Long, P.M. and Warmuth, M.K. [1990], Composite geometric concepts and polynomial predictability, Report No. UCSC-CRL-90-31, University of California, Santa Cruz.

[72] Lipton, R.J. and Zalcstein, Y. [1977], Word problems solvable in logspace, *J. Assoc. Comput. Mach.* **24**, 522-526.

[73] Luby, M. [1986], A simple parallel algorithm for the maximal independent set problem, *SIAM J. Comput.* **15**, 1036-1053.

[74] Luby, M. [1988], Removing randomness in parallel computation without a processor penalty, *Proc. 29th IEEE FOCS*, 162-173.

[75] Mayr, E.W. [1988], Parallel approximation algorithms, *Proc. Fifth Generation Computer Systems 1988*, 542-551.

[76] Mayr, E.W. and Subramanian, A. [1989], The complexity of circuit value and network stability, *Proc. Structure in Complexity Theory 1989*, 114-123.

[77] Merkle, R.C. and Hellman, M.E. [1978], Hiding information and signatures in trapdoor knapsacks, *IEEE Trans. Inform. Theory* **IT-24**, 525-530.

[78] Miyano, S. and Haraguchi, M. [1982], Recovery of incomplete tables under functional dependencies, *Bulletin of Informatics and Cybernetics* **20**, 25-41.

[79] Miyano, S. [1988a], Δ_2^p-complete lexicographical first maximal subgraph problems, *Proc. Mathematical Foundation of Computer Science* (Lecture Notes in Computer Science **324**), 454-462.

[80] Miyano, S. [1988b], Parallel complexity and P-complete problems, *Proc. Fifth Generation Computer Systems 1988*, 532-541.

[81] Miyano, S. [1988c], A parallelizable lexicographically first maximal edge-induced subgraph problem, *Inform. Process. Lett.* **27**, 75-78.

[82] Miyano, S. [1989], The lexicographically first maximal subgraph problems: P-completeness and NC algorithms, *Math. Syst. Theory* **22**, 47-73.

[83] Mulmuley, K., Vazirani, U.V. and Vazirani, V.V. [1987], Matching is as easy as matrix inversion, *Proc. 19th ACM STOC*, 345-354.

[84] Pan, V. and Reif, J.H. [1985], Efficient parallel solution of linear systems, *Proc. 17th ACM STOC*, 143-152.

[85] Papadimitriou, C.H. and Tsitsiklis, J.N. [1987], The complexity of markov decision processes, *Mathematics of Operations Research* **12**, 441-450.

[86] Pippenger, N. [1979], On simultaneous resource bounds, *Proc. 20th IEEE FOCS*, 307-311.

[87] Plaisted, D.A. [1984], Complete problems in the first-order predicate calculus, *J. Comput. System Sci.* **29**, 8-35.

[88] Reif, J.H. [1985], Depth-first search is inherently sequential, *Inf. Process. Lett.* **20**, 229-234.

[89] Robinson, J.A. [1965], A machine oriented logic based on the resolution principle, *J. Assoc. Comput. Mach.* **12**, 23-41.

[90] Ruzzo, W.L. [1980], Tree-size bounded alternation, *J. Comput. System Sci.* **21**, 218-235.

[91] Savitch, W.J. [1970], Relationships between nondeterministic and deterministic tape complexities, *J. Comput. System Sci.* **4**, 177-192.

[92] Shamir, A. [1982], A polynomial time algorithm for breaking the basic Merkle-Hellman cryptosystem, *Proc. 23rd IEEE FOCS*, 145-152.

[93] Shiloach, Y. and Vishkin, U. [1982], An $O(n^2 \log n)$ parallel MAX-FLOW algorithm, *J. Algorithms* **3**, 128-146.

[94] Shoudai, T. [1989], The lexicographically first topological order problem is NLOG-complete, *Inform. Process. Let.* **33**, 121-124.

[95] Shoudai, T. and Miyano, S. [1990], Bounded degree maximal subgraph problems are in NC, Technical Report RIFIS-TR-CS-27, Kyushu University, 1990.

[96] Smith, J.R. [1986], Parallel algorithms for depth-first searchs I. Planar graphs, *SIAM J. Comput.* **15**, 814-830.

[97] Spirakis, P. [1987], The parallel complexity of deadlock detection, *Theoret. Comput. Sci.* **52**, 155-163.

[98] Stockmeyer, L.J. and Vishkin, U. [1984], Simulation of parallel random access machines by circuits, *SIAM J. Comput.* **13**, 409-422.

[99] Ullman, J.D. and van Gelder, A. [1988], Parallel complexity of logical query programs, *Algorithmica* **3**, 5-42.

[100] Vaidya, P.M. [1990], Reducing the parallel complexity of certain linear programming problems, *Proc. 31th IEEE FOCS*, 583-589.

[101] Vavasis, S.A. [1989], Gaussian elimination with pivoting is P-complete, *SIAM J. Disc. Math* **2**, 413-423.

[102] Vishwanathan, S. and Sridhar, M.A. [1988], Some results on graph coloring in parallel, *Proc. 1988 Int. Conf. on Parallel Processing III*, 299-303.

[103] Wang, C.C., Lloyd, E.L. and Soffa, M.L. [1985], Feedback vertex sets and cyclically reducible graphs, *J. Assoc. Comput. Mach.* **32**, 296-313.

[104] Yamasaki, S. and Doshita, S. [1983], The satisfiability problem for a class consisting of Horn sentences and some non-horn sentences in proportional logic, *Inform. and Contr.* **59**, 1-12, (Erratum, *Inform. and Contr.* **61**, 174).

[105] Yasuura, H. [1983], The reachability problem on directed hypergraphs and computational complexity, Tech. Rep. of IECEJ, AL83-42, 87-100.

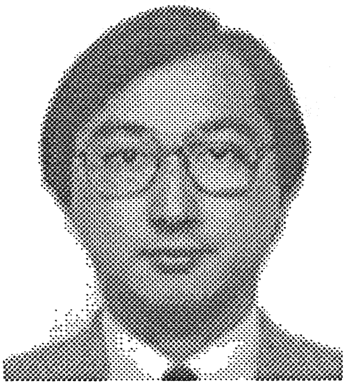
[106] Yasuura, H. [1984], On parallel computational complexity of unification, *Proc. Int. Conf. on Fifth Generation Computer Systems 1984*, 235-243.

[107] Zhang, Y. [1986], Ph.D thesis, Drexel University, Philadelphia, Pennsylvania.

Contents

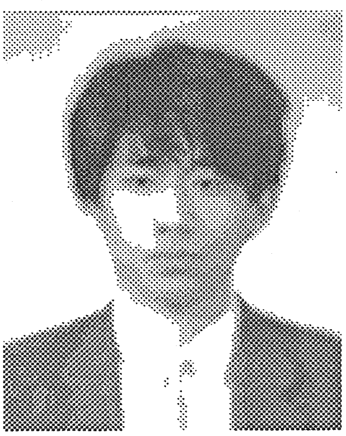
Index

About the Authors



Satoru Miyano (宮野　悟) was born in Oita on December 5, 1954. He received the B.S. in 1977, the M.S. degree in 1979 and the Dr. Sci. in 1984 all in Mathematics from Kyushu University. Presently, he is an Associate Professor of Research Institute of Fundamental Inforamtion Science, Kyushu University. His present interests include parallel algorithms, computational complexity and computational learning theory.

Shuji Shiraishi (白石修二) was born in Kagoshima on September 23, 1955. He received the B.S. degree from Kagoshima University in 1979, the M.S. degree in 1981 and the Dr. Sci. in 1984 from Kyushu University. Presently he is a Lecturer of Department of Applied Mathematics, Fukuoka University. His interests are in parallel graphs algorithms.



Takayoshi Shoudai (正代隆義) was born in Fukuoka on December 30, 1961. He received the B.S. degree in 1986 and the M.S. degree in 1988 in Mathematics from Kyushu University. Presently, he is a Research Associate of Department of Control Engineering and Science, Kyushu Institute of Technology at Iizuka. His research interests are in parallel algorithms, probabilistic algorithms and computational complexity.

Research Institute of Fundamental Information Science, Kyushu University, Fukuoka 812, Japan.