アルゴリズム 69－2
（1999．9．2）

P 完全問題の実用的な並列性について

藤原 暁宏† 井上 美智子‡ 増澤 利光‡

†九州工業大学 情報工学部 電子情報工学科
〒820-8502 福岡県飯塚市川津 680-4

‡奈良先端科学技術大学院大学 情報科学研究科
〒630-0101 奈良県生駒市高山町 8916-5

E-mail: fujiwara@cse.kyutech.ac.jp

概要： 本稿では，いくつかの P 完全問題に関して，実用的な並列性があることを示す．まず最初に，多重凸包問題について並列性を示すパラメータを提案する．また，そのパラメーにより問題が P 完全である場合の完全性の証明，および，コスト最適な並列アルゴリズムの提案を行う．次に，辞書式順5和問題を取り上げ，辞書式順独立点集合問題からの帰着により，この問題が P 完全であることを示す．また，この問題に対するコスト最適な並列アルゴリズムを提案するとともに，アルゴリズムを PVM を使用した PC クラスタ上に実装する．実験結果では，辞書式順5和問題に対して，ほぼ理想的なスピードアップが得られた．

Practical parallelizability of some P-complete problems

Akihiro Fujiwara†, Michiko Inoue‡ and Toshimitsu Masuzawa‡

†Department of Computer Science and Electronics
Kyushu Institute of Technology
680-4 Kawazu, Iizuka, Fukuoka 820-8502, Japan

‡Graduate School of Information Science
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara, 630-0101 Japan

Abstract: In this paper, we consider practical parallelizability of some P-complete problems. First we propose a parameter of parallelizability for a convex layers problem. We prove P-completeness of the problem and propose a cost optimal parallel algorithm according to the parameter. Second, we consider a lexicographically first maximal 5 sums problem, and prove P-completeness of the problem by reducing a lexicographically first maximal independent set problem. In addition, we propose a practical cost optimal parallel algorithm for the problem and implement the algorithm on a PC cluster using PVM. The results achieves almost ideal speedup for the problem.

1 Introduction

In parallel computation theory, one of the primary measures of parallel complexity is the class NC. Let n be the input size of a problem. The problem is called to be in the class NC if there exist an algorithm which solves the problem in $T(n)$ time using $P(n)$ processors where $T(n)$ and $P(n)$ are polylogarithmic and polynomial functions for n, respectively. Many problems in the class P, which is the class of problems solvable in a polynomial time sequentially, are also in the class NC.

On the other hand, some problems in P seem to have no parallel algorithm which runs in polylogarithmic time using a polynomial number of processors. Such problems are usually called *P-complete*. The P-complete problem is a problem to which we can reduce any problem in P using NC-reduction. Although there are some efficient parallel *probabilistic* algorithm for some P-complete problems, it is believed that the P-complete problem is inherently sequential and hard to be parallelized.

However polylogarithmic time complexity is not so important for a real parallel computation because the number of processors p is usually small in comparison with the size of a problem n, that is, $p << n$. Thus, *cost optimality* is the most important measure for parallel algorithms in practice. The cost of parallel algorithm is defined as the product of the running time and the number of processors of the algorithm. A parallel algorithm is called cost optimal if its cost is equal to the lower bound of a sequential time complexity for the same problem. In other words, the cost optimal parallel algorithm achieves optimal speedup, which is equal to the number of processors.

Therefore one way to parallelize P-complete problems is to find cost optimal parallel algorithms which runs in polynomial time. Let $\Omega(n^k)$ be lower bound of sequential time complexity for a P-complete problem A. It seems that the problem A has no parallel algorithm which runs in polylogarithmic time since A is P-complete. However, The problem A may have a parallel algorithm which runs in $O(n^{k-\epsilon})$ time using n^{ϵ} processors where $0 < \epsilon < k$. Since the parallel algorithm is cost optimal, the algorithm probably achieves optimal speedup in practice if the number of processors is not so large. Thus, in this paper, we aim to propose cost optimal parallel algorithms, which run in polynomial time, for P-complete problems.

Some convenient classifications were proposed to define such parallelizable P-complete problems. Vitter and Simons[11] proposed the class of problems PC^*. A problem in P is called to be in the class PC^* if and only if there exist a cost optimal parallel algorithm which solves the problem. Since the speedup is not bounded in the definition of PC^*, the definition is applicable to P-complete

problems. Kruskal et al.[9] proposed six classes of parallel algorithms. They described a class EP is the most practically interesting among the classes. The class EP means "efficient and polynomially fast", and a parallel algorithm is in EP if and only if there exist a cost optimal algorithm whose time complexity $T(n) = O(t(n)^\epsilon)$ with $\epsilon < 1$ and $T(n) \times P(n) = O(t(n))$, where $t(n)$ is the best time complexity of sequential algorithms which solves the same problem. These two classes are almost the same for P-complete problems because P-complete problems seem to have no polylogarithmic time parallel algorithm. In this paper, we call P-complete problems in the above classes *parallelizable in practice*, that is, P-complete problems which are parallelizable in practice have cost optimal parallel algorithms.

Among many P-complete problems, only some graph problems are known to be parallelizable in practice. In [11], Vitter and Simons showed that the unification, path system accessibility, monotone circuit value and ordered depth-first search problems have cost optimal parallel algorithms if their input graphs are dense graphs, that is, the number of edges is $m = \Omega(n^{1+\epsilon})$ for a constant ϵ where the number of vertices is n. For example, they showed that the monotone circuit value problem can be solved in $O(\frac{m}{p} + n)$ time using p processors on the common CRCW PRAM. The time complexity becomes $O(\frac{n^2}{p})$ if $m = \Theta(n^2)$, then the algorithm achieves cost optimality.

In this paper, we consider practical parallelizability of two P-complete problems. First we propose a parameter of parallelizability for *a convex layers problem*. The convex layers is a geometric problem and closely relates to other layering problems, such as visibility layers. Dessmark et al.[4] proved that the problem is P-complete, and Chazelle[1] proposed an optimal sequential algorithm which runs in $O(n \log n)$ time, where n is the input size of the problem. First we consider restricting positions of input points on d parallel lines. (The d is a parameter of the problem.) For the parameterized convex layers, we prove the problem is still P-complete if $d = n^\epsilon$ with $0 < \epsilon < 1$ by reducing the original convex layers. Next we propose a parallel algorithm which runs in $O(\frac{n \log n}{p} + \frac{d^2}{p} + d \log d)$ time using p processors $(1 \leq p \leq d)$ on the EREW PRAM. From the complexity, the problem is in NC if $d = (\log n)^k$ where k is a positive constant, and has a cost optimal parallel algorithm if $d = n^\epsilon$ with $0 < \epsilon \leq \frac{1}{2}$. We also consider complexity of the problem in case where all inputs are sorted and show that the complexity achieves similar cost optimality.

The second P-complete problem is *a lexicographically first maximal 5 sums*, which is a problem to compute lexicographically a maximal set of 5 elements whose sum is 0 among n integers. We prove the problem is also P-complete by reducing a lexicographically first maximal independent set, which is a graph problem known as P-complete[10]. We propose a cost optimal parallel algorithm, which runs in $O(\frac{n^4}{p} + n^2)$ time using p processors on the CREW PRAM, for the problem. In addition, we modify the algorithm for distributed memory model and implement the algorithm on a PC cluster using PVM. Although the PVM on a cluster is a primitive parallel environment, the algorithm achieves almost ideal speedup. This implies that some P-complete problems are parallelizable in practice within the reasonable number of processors.

2 Preliminaries

Let n be the input size of a problem. The problem is called to be in the class P if there is a sequential algorithm which solves the problem in $t(n)$ where $t(n)$ is a polynomial function for n. The P is well known class which denotes sequential efficiency. An analogous class of efficiency for parallel computation is the class NC. A problem is called to be in the class NC if there exist an algorithm which solves the problem in $T(n)$ time using $P(n)$ processors, and $T(n)$ and $P(n)$ are polylogarithmic and polynomial functions for n, respectively.

Using the above classes, the P-completeness is defined as follows. (For details of the P-completeness, see [8].)

Definition 1 (P-complete problem) *A problem Q is called to be P-complete if the following two conditions are satisfied.*

(1) *The problem Q is in P.*

(2) *For every problem S in P, S is NC-reducible to Q.* □

From the above definitions, we can prove P-completeness of a problem if we can reduce a known P-complete problem to the problem using NC-reduction.

Parallel computation model used in this paper is the PRAM. The PRAM employs processors which have capability to access any memory cell in a shared memory synchronously. (For details of the PRAM, see [8].) Since the PRAM is a theoretical model, we use the model to prove P-completeness of problems and propose cost optimal parallel algorithms in this paper.

3 Parameterized convex layers

3.1 Definitions

First we give some definitions for convex layers.

Definition 2 (Convex layers) *Let S be a set of n points in the Euclidean plane. The convex layers is a problem to compute convex polygons obtained by the following algorithm.*

(1) *Compute a convex hull of S, and remove points contained in the convex hull from S.*

(2) *Repeat (1) until $S = \phi$.* □

Dessmark et al.[4] proved P-completeness of the convex layers, and Chazelle[1] proposed an optimal sequential algorithm which runs in $O(n \log n)$ time. The sequential algorithm is time optimal because computation of a convex hull, which is one hull of convex layers, requires $\Omega(n \log n)$ time[12].

In this paper, we consider a parameter d for the problem, and restrict its input points on d horizontal lines.

Definition 3 (Convex layers for d lines) *The convex layers for d lines is a parameterized convex layers problem whose input points are on d horizontal lines.* □

The d is at most n if there is no restrictions for positions of input points. In the following, $CL(d)$ denotes the convex layers for d lines. We can solve the problem in $O(n \log n)$ time using the algorithm[1].

The above two convex layers problems are illustrated in Figure 1.

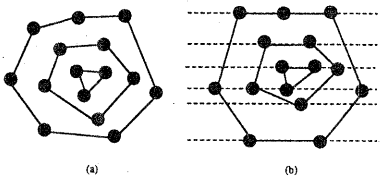

Figure 1: Convex layers problems: (a) convex layers, and (b) CL(6).

3.2 P-completeness of $CL(d)$

We discuss relationship between P-completeness and the number of lines d in this subsection. First we prove the problem $CL(d)$ is P-complete if $d = n^\epsilon$ with $0 < \epsilon \leq 1$. We prove the P-completeness by NC-reduction from the original convex layers.

Theorem 1 *The problem $CL(n^\epsilon)$ with $0 < \epsilon \leq 1$ is P-complete.*

(Proof) First of all, it is obvious that $CL(n^\epsilon)$ is in P because the problem has an $O(n \log n)$ time sequential algorithm. If $\epsilon = 1$, the $CL(n^\epsilon)$ is the original convex layers problem, which is proved to be P-complete. Thus, we consider the case that $0 < \epsilon < 1$ in the following. Let $U_0 = \{u_0, u_1, \ldots, u_{n-1}\}$ be input points of the convex layers in the Euclidean plain. We assume that $u_N = (x_N, y_N)$ and $u_S = (x_S, y_S)$ are points which have the largest and the smallest y-coordinates in U_0, and also assume that $u_E = (x_E, y_E)$ and $u_W = (x_W, y_W)$ are points which have the largest and the smallest x-coordinates in U_0, respectively.

First we add 4 points to the input. The points are $U_1 = \{u_{NW}, u_{NE}, u_{SE}, u_{SW}\} = \{(x_W - 1, y_N + 2), (x_E + 1, y_N + 1), (x_E + 1, y_S - 2), (x_W - 1, y_S - 1)\}$. (These 4 points forms a parallelogram which contains all points in U_0.) Next we add a set of $k = (n+4)^{\frac{1}{\epsilon}} - (n+4)$ points which are on a line $y = y_N + 2$, $U_2 = \{(x'_0, y_N + 2), (x'_1, y_N + 2), \ldots, (x'_{k-1}, y_N + 2)\}$ so that $x_W - 1 < x'_0 < x'_1 < \ldots < x'_{k-1} < x_W + 1$. (Since $0 < \epsilon < 1$, $k > 0$ holds.) These added points are illustrated in Figure 2.

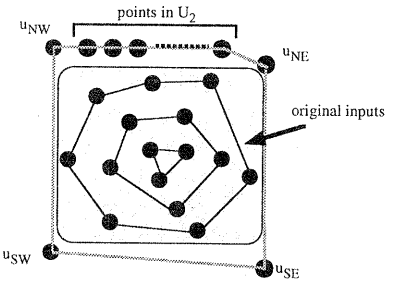

Figure 2: Points in U_1 and U_2.

We give a set of points $U_0 \cup U_1 \cup U_2$ as input points for $CL(d)$. The size of the input is $n + 4 + (n+4)^{\frac{1}{\epsilon}} - (n+4) = (n+4)^{\frac{1}{\epsilon}}$ and the number of horizontal lines is $d = n + 4$. Therefore, by letting $m = (n+4)^{\frac{1}{\epsilon}}$, the problem becomes $CL(m^\epsilon)$ with $0 < \epsilon < 1$.

The result of $CL(m^\epsilon)$ are convex hulls, and the outmost convex hull consists of all points in $U_1 \cup U_2$. Thus, after peering the outmost convex hull of the results, the remaining convex hulls are equal to the result of the original convex layers. Since m is a polynomial function of n, all of the above steps can be done using NC reduction. □

In the next subsection, we propose a cost optimal algorithm for $CL(d)$. Using the algorithm, we also prove that $CL((\log n)^k)$ is in NC where k is a positive constant.

3.3 Cost optimal parallel algorithm

The basic idea of the cost optimal parallel algorithm for $CL(d)$ is as follows. We assume that input points are on lines $\{l_0, l_1, \ldots, l_{d-1}\}$ and a line l_i is above l_{i+1} for each i $(0 \leq i \leq d-2)$. First we compute a set of points on each line and store the points in a double-ended queue in order of x coordinates for each line. (The double-ended queue is a queue which allows insertion and deletion at both ends.) Next we compute an outmost convex hull. The outmost convex hull consists of the following points.

(a) Points on lines l_0 and l_{d-1}.

(b) A subset of leftmost and rightmost points on lines $l_1, l_2, \ldots, l_{d-2}$.

We can compute points included in (b) in parallel for each line because points on each line are stored in each double-ended queue. Since obtained points are sorted in order of y coordinates, we can compute the outmost convex hull among the points using a cost optimal parallel algorithm which computes a convex hull for sorted points. Finally we repeat the above computation after peering the outmost convex hull until no point remains. The number of convex hulls are at most $\lceil \frac{d}{2} \rceil$ because top and bottom lines are removed by peering the outmost convex hull. Therefore the number of repetition is also at most $\lceil \frac{d}{2} \rceil$.

We summarize the overall algorithm in the following.

Algorithm for computing $CL(d)$

Input: A set of points $\{u_0, u_1, \ldots, u_{n-1}\}$ on lines $\{l_0, l_1, \ldots, l_{d-1}\}$.

Step 1: Set variables $TOP = 0$ and $BOT = d-1$. (l_{TOP} and l_{BOT} denote top and bottom lines respectively.) Compute a set of points on each line l_i ($TOP \leq i \leq BOT$). Points on l_i are stored a double-ended queue Q_i in order of x coordinates.

Step 2: For each line l_i ($TOP \leq i \leq BOT$), compute a leftmost point u^i_{left} and a rightmost point u^i_{right}.

Step 3: Let U_{left} and U_{right} denote sets of points $\{u^{TOP}_{left}, u^{TOP+1}_{left}, \ldots, u^{BOT}_{left}\}$ and $\{u^{TOP}_{right}, u^{TOP+1}_{right}, \ldots, u^{BOT}_{right}\}$ respectively. Compute a left hull of U_{left} and a right hull of U_{right}, and store points on each hull in CH_{left} and CH_{right}, respectively. (The left hull of U_{left} consists of points from u^{BOT}_{left} to u^{TOP}_{left} in clockwise order among points on a convex hull of U_{left}. The right hull of U_{right} is defined similarly.)

Step 4: Remove points in Q_{TOP}, Q_{BOT}, CH_{left} and CH_{right} as the outmost convex hull.

Step 5 Compute top and bottom lines on which there is at least one point. Set TOP and BOT to the number of the top and bottom lines respectively.

Step 6: Repeat Step 2, 3, 4 and 5 until no point remains.

We discuss complexities of the above parallel algorithm on the EREW PRAM. We use at most p processor ($1 \leq p \leq d$) in the algorithm except for Step 1.

Step 1 takes $O(\frac{n \log n}{p} + \log n)$ using Cole's merge sort[3] and primitive operations, and Step 2 takes $O(\frac{d}{p})$ time obviously. We can compute the left hull and right hull in Step 3 using a known parallel algorithm[2] for computing a convex hull for sorted points. The algorithm runs in $O(\frac{d}{p} + \log d)$ time for each hull. Step 4 takes $O(\frac{d}{p})$ time to remove the points. (Points in Q_{TOP}, Q_{BOT} are automatically removed by changing TOP and BOT in Step 5.) We can compute the top and bottom lines in Step 5 in $O(\frac{d}{p} + \log d)$ time using a basic parallel algorithm computing the maximum and minimum. As we described above, the number of the repetition of Step 6 is $\lceil \frac{d}{2} \rceil$. Therefore We can compute $CL(d)$ in $O(\frac{n \log n}{p} + \log n + (\frac{d}{p} + \log d) \times \lceil \frac{d}{2} \rceil) = O(\frac{n \log n}{p} + \frac{d^2}{p} + d \log d)$, and obtain the following theorem.

Theorem 2 *We can solve $CL(d)$in $O(\frac{n \log n}{p} + \frac{d^2}{p} + d \log d)$ time using p processors ($1 \leq p \leq d$) on the EREW PRAM.* □

We show that the class of the problem changes according to the number of lines d. First we consider the case of $d = (\log n)^k$ where k is a positive constant. The complexity is $O(\frac{n \log n}{p} + \log n \log \log n)$ in the case. If we use n processors in Step 1, the complexity becomes $O(\log n \log \log n)$. Consequently, we obtain the following corollary.

Corollary 1 *We can solve $CL((\log n)^k)$, where k is a positive constant, in $O(\log n \log \log n)$ time using n processors on the EREW PRAM, that is, $CL((\log n)^k)$ is in NC.* □

Next we consider the time complexity in case of $d = n^\epsilon$ where $0 < \epsilon < 1$. The complexity is $O(\frac{n \log n}{p} + \frac{n^{2\epsilon}}{p} + n^\epsilon \log n)$ in the case. In addition this, we assume that $\epsilon \leq \frac{1}{2}$ and $p \leq d = n^\epsilon$. Under the assumption, $\frac{n \log n}{p} > \frac{n^{2\epsilon}}{p}$ holds because $n \log n > n \geq n^{2\epsilon}$, and $\frac{n \log n}{p} > n^\epsilon \log n$ holds because $\frac{n \log n}{p} \geq n^{1-\epsilon} \log n \geq n^\epsilon \log n$. Therefore we can obtain a cost optimal parallel algorithm for $CL(n^\epsilon)$.

Corollary 2 *We can solve $CL(n^\epsilon)$ with $0 < \epsilon \leq \frac{1}{2}$ in $O(\frac{n \log n}{p})$ time using p processors ($1 \leq p \leq n^\epsilon$) on the EREW PRAM.* □

Finally we notice that the above algorithm runs in $O(\frac{n}{p} + \frac{d^2}{p} + d \log d)$ time except for Step 1. This implies that we can solve $CL(d)$ efficiently if given input points on each line are sorted in order of x coordinates. We call the problem $CLS(d)$ *(convex layers for sorted points on d horizontal lines)*. From the above complexity, we can solve the $CLS(d)$ sequentially and in parallel with the following complexities. (Both complexities are optimal.)

Corollary 3 *We can solve $CLS(n^\epsilon)$ with $0 < \epsilon \leq \frac{1}{2}$ in $O(n)$ time sequentially, and in $O(\frac{n}{p})$ time using p processors ($1 \leq p \leq n^\epsilon$) on the EREW PRAM.* □

4 Lexicographically first maximal 5 sums

4.1 Definitions

We first define the maximal 5 sums problem as follows.

Definition 4 (Maximal 5 sums) *Let I be a set of n integers. The maximal 5 sums is a problem to compute the maximal set of 5 integers $MS5I = \{(a_0, b_0, c_0, d_0, e_0), (a_1, b_1, c_1, d_1, e_1), \ldots, (a_m, b_m, c_m, d_m, e_m)\}$ obtained by the following algorithm.*

(1) *Set $MS5I = \phi$.*

(2) *Repeat the following substeps until no 5 sum is found in (2-1).*

(2-1) *Find a sequence of 5 integers (a, b, c, d, e) which satisfies the following two conditions.*

(a) *a, b, c, d, e are distinct elements in I.*

(b) *$a + b + c + d + e = 0$.*

(2-2) *Add (a, b, c, d, e) to $MS5I$ and remove a, b, c, d, e from I.* □

Note that result of the maximal 5 sums is not unique. For example, let $I = \{-6, 3, 7, 1, -9, 3, 1, 6, -2, -5, -1\}$ be an input for the maximal 5 sum. Then both of $\{(-6, -5, 1, 3, 7), (-9, -1, 1, 3, 6)\}$ and $\{(-9, -5, 1, 6, 7), (-6, -1, 1, 3, 3)\}$ are results for the problem. (There are a lot of other results.)

The lexicographically first maximal 5 sums problem is a modified maximal 5 sums so as to have the unique result. Let $A = (a_0, a_1, \ldots a_m)$ and $B = (b_0, b_1, \ldots, b_m)$ be two sequence of m numbers. We call that A is *lexicographically less than* B if $a_0 < b_0$, or there exists an integer i $(1 \leq i \leq m)$, such that $a_j = b_j$ for all j $(0 \leq j < i)$ and $a_i < b_i$. We call that a sequence A is the lexicographically first among a set of sequences if A is less than all of the other sequences in the set.

Definition 5 (Lexicographically first maximal 5 sums (LFM5S)) *The lexicographically first maximal 5 sums (LFM5S) is a problem to compute the unique result of the maximal 5 sums by adding the following condition to (2-1) of the algorithm in Definition 4.*

(c) *(a, b, c, d, e) is the lexicographically first sequence among all of the other sequences which satisfy (a) and (b).* □

In case of the above example, $\{(-9, -5, 1, 6, 7), (-6, -1, 1, 3, 3)\}$ is the unique result of $LFM5S$ for I.

We can propose a sequential algorithm which solves $LFM5S$ in $O(n^4)$ by modifying an algorithm computing the 3 sum problem[7]. (The 3 sum is a decision problem which decides whether there exists $a, b, c \in I$ satisfying $a + b + c = 0$.) The algorithm is the known fastest sequential algorithm for $LFM5S$. Note that the lower bound of $LFM5S$ is not known. However I guess that there exist no efficient sequential algorithm for $LFM5S$ from the following facts.

- The 3 sum has no $o(n^2)$ algorithm and has an $\Omega(n^2)$ lower bound on a weak model of computation[5].
- The subset sum, which is generalization of the 3 sum or $LFM5S$, is a well known NP-complete problem.

In the next subsection, we prove $LFM5S$ is P-complete.

4.2 P-completeness of $LFM5S$

We show a reduction from *the lexicographically first maximal independent set (LFMIS) problem* to $LFM5S$. The $LFMIS$ is a well known P-complete problem and defined as follows.

Definition 6 (Lexicographically first maximal independent set) *Let $G = (V, E)$ be an input graph with $V = \{v_0, v_1, \ldots, v_{n-1}\}$. The lexicographically first maximal independent set is a problem to compute the maximal independent set of G obtained by the following algorithm.*

(1) *Set $LFMIS = \phi$.*

(2) *Repeat the following substep from $i = 0$ to $i = n - 1$.*

(2-1) *If a vertex v_i is not connected to any vertex in $LFMIS$ then add v_i to $LFMIS$.* □

In [10], Miyano proved the following lemma for $LFMIS$.

Lemma 1 *The $LFMIS$ restricted to bipartite graphs with degree at most 3 is P-complete.* □

Using the above lemma, we prove the following theorem.

Theorem 3 *The problem $LFM5S$ is P-complete.*

(Proof) It is obvious that $LFM5S$ is in P.

Let $G = (V, E)$ with $V = \{v_0, v_1, \ldots, v_{n-1}\}$ be an input bipartite graph with degree at most 3. First we define *a vertex value* $VV(v_i)$ for each vertex v_i. The vertex value is a negative integer and defined as $VV(v_i) = i - n$. Thus vertices $v_0, v_1, \ldots, v_{n-1}$ have vertex values $-n, -(n-1), \ldots, -1$ respectively.

The main idea of our reduction is to create inputs of $LFM5S$ so that the sum of vertex values among a vertex and its adjacent vertices becomes 0. We consider a bipartite graph G illustrated in Figure 3 for example. We create three kinds of inputs of $LFM5S$ for each vertex

v_i. Let v_j, v_k and v_l be adjacent vertices of v_i. The inputs of $LFM5S$ for v_i is (1) $VV(v_i)$, (2) $|VV(v_i) + VV(v_j) + VV(v_k) + VV(v_l)|$ and (3) three "0". (If adjacent vertices are less than 3, the vertex value for non-existing vertex in the expression (2) is 0.) For example, we create input of $LFM5S$ from a graph G in Figure 3 as $\{-4, -3, -2, -1, 0, 0, \ldots, 0, 5(= |-4-1|), 6(= |-4-2|), 7, (= |-4-3|), 10(= |-4-3-2-1|)\}$. The result of $LFM5S$ is $\{(-4, -3, -2, -1, 10), (0, 0, 0, 0, 0), (0, 0, 0, 0, 0)\}$. (Since $LFM5S$ compute results lexicographically, a result (a, b, c, d, e) of $LFM5S$ means that $a \leq b \leq c \leq d \leq e$.) We ignore all $(0, 0, 0, 0, 0)$ in the result and find the smallest value in each sequence of the result (-4) as a vertex value of a vertex in the independent set of $LFMIS$ ($VV(v_0)$). (All $(0, 0, 0, 0, 0)$ in the result are ignored in the following.)

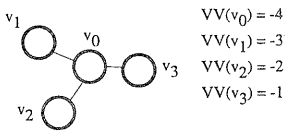

Figure 3: An example of a graph and vertex values

However the above simple reduction has two problems. The first problem is that there may be 5 integers a', b', c', d', e' in the input which satisfy $a' + b' + c' + d' + e' = 0$ and at most three of the integers are vertex values, or four vertices, whose vertex values are a', b', c', d', are not connected. The second problem is that a vertex value cannot be in the result of $LFM5S$ if a vertex value of its adjacent vertex is in the result. We explain the two problems using Figure 4. From the above reduction, the result of $LFM5S$ is $\{(-7, -6, -5, 0, 18), (-4, -1, 0, 0, 5)\}$. (Since $LFMIS$ of the graph is $\{v_0, v_2, v_3, v_4\}$, the first elements of sequences in the result must be $-7, -5, -4, -3$.) In the wrong result, $(-7, -6, -5, 0, 18)$ denotes the first problem, which is a wrong set of vertex values. In addition a vertex value -3 ($=VV(v_4)$) and is not the first element of the sequences because of the second problem, that is, a vertex value -1 ($=VV(v_6)$) is in the previous sequence $(-4, -1, 0, 0, 5)$.

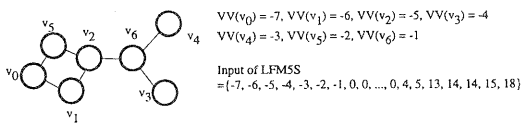

Figure 4: Another example of a graph and vertex values

To resolve the above problems, we propose the following reduction. Since the input is a bipartite graph, we assume that V can be partitioned into two sets V_1 and V_2 such that $(u, v) \in E$ implies either $u \in V_1$ and $v \in V_2$ or $u \in V_2$ and $v \in V_1$. First we create a 7-tuple $t_0^i = [VV(v_i), VV(v_i), VV(v_i)^2, VV(v_i)^3, 0, 0, 0]$ for each $v_i \in V_1$ or $t_0^i = [VV(v_i), 0, 0, 0, VV(v_i), VV(v_i)^2, VV(v_i)^3,]$ for each $v_i \in V_2$ as an input.

Next we create the following 7 kinds of 7-tuples t_m^i $(1 \leq m \leq 7)$ for each vertex v_i, where v_j, v_k and v_l $(j < k < l)$ be adjacent vertices of v_i. In the following, $t_0^\alpha + t_0^\beta = [\alpha_0, \alpha_1, \ldots, \alpha_6] + [\beta_0, \beta_1, \ldots, \beta_6] = [\alpha_0 + \beta_0, \alpha_1 + \beta_1, \ldots, \alpha_6 + \beta_6]$ and $\overline{t_0^\alpha} = \overline{[\alpha_0, \alpha_1, \ldots, \alpha_6]} = [-\alpha_0, -\alpha_1, \ldots, -\alpha_6]$.

(1) $t_1^i = \overline{t_0^i + t_0^j + t_0^k + t_0^l}$

(2) $t_2^i = \overline{t_0^i + t_0^k + t_0^l}$

(3) $t_3^i = \overline{t_0^i + t_0^j + t_0^l}$

(4) $t_4^i = \overline{t_0^i + t_0^j + t_0^k}$

(5) $t_5^i = \overline{t_0^i + t_0^j}$

(6) $t_6^i = \overline{t_0^i + t_0^k}$

(7) $t_7^i = \overline{t_0^i + t_0^l}$

We call that v_i is *a parent vertex of tuples* t_m^i $(0 \leq m \leq 7)$.

Finally, we add 2n 7-tuples $z = [0, 0, 0, 0, 0, 0, 0]$ to the input. (We call the tuples *zero tuples*.)

As an example, we consider the input of a bipartite graph in Figure 3. The inputs of tuples are in Table 1. (All zero-tuples are omitted.)

We compute $LFM5S$ for the input. In case of the above example, we obtain a result $\{(t_0^0, t_0^1, t_0^2, t_0^3, t_1^0)\}$ which satisfies $t_0^0 + t_0^1 + t_0^2 + t_0^3 + t_1^0 = 0$. (The results consists of zero tuples are omitted.) Let T_1 be the first 7-tuples of each result of $LFM5S$, and T_2 be a set of t_0^i $(0 \leq i \leq n)$ which is not contained in the result of $LFM5S$. Then all of the first elements of tuples in $T_1 \cup T_2$ denotes vertex values of the results obtained by $LFMIS$.

We describe the correctness of the above reduction. First, a tuple in T_2 denotes a vertex value such that the vertex has no neighbor. Next we consider tuples included in T_1. Let v_0 be the first vertex in V. Then v_0 is always in the result of $LFMIS$. Let v_j, v_k, v_l be adjacent vertices of v_0. (We can also prove the theorem similarly if the number of adjacent vertices is less than 3.)

In this case, t_0^0 is a corresponding tuple to v_0, and it is sufficient that we can prove that a sequence $(t_0^0, t_0^j, t_0^k, t_0^l, t_1^0)$ is in the result of $LFM5S$. Since $t_0^0 + t_0^j + t_0^k + t_0^l + t_1^0 = 0$ from the above reduction, the sequence is in the result of $LFM5S$ if there is no sequence which is lexicographically less than the sequence.

Table 1: Input tuples for Figure 3

m	t_m^0	t_m^1	t_m^2	t_m^3
0	$[-4,-4,16,-64,0,0,0]$	$[-3,0,0,0,-3,9,-27]$	$[-2,0,0,0,-2,4,-8]$	$[-1,0,0,0,-1,1,-1]$
1	$[10,4,-16,64,6,-14,36]$	$[7,4,-16,64,3,-9,27]$	$[6,4,-16,64,2,-4,8]$	$[5,4,-16,64,1,-1,1]$
2	$[7,4,-16,64,3,-5,9]$	-	-	-
3	$[8,4,-16,64,4,-10,28]$	-	-	-
4	$[9,4,-16,64,5,-13,35]$	-	-	-
5	$[7,4,-16,64,3,-9,27]$	-	-	-
6	$[6,4,-16,64,2,-4,8]$	-	-	-
7	$[5,4,-16,64,1,-1,1]$	-	-	-

We assume that $v_0 \in V_1$. (We can prove in case of $v_0 \in V_2$ similarly.) The tuple t_0^0 is necessarily in the sequence because the sequence must be lexicographically less than $(t_0^0, t_0^j, t_0^k, t_0^l, t_1^0)$. Then there exist at least one tuple t_m^x, which satisfies $1 \leq m \leq 7$, in the sequence because the first element of t_0^0 is a negative integer, and then there exist at least one tuple t_0^y whose parent vertex $v_y \in V_2$ because the fifth, sixth and seventh elements of t_m^x are not zero.

Next we use a simple fact for integers and prove that $(t_0^0, t_0^j, t_0^k, t_0^l, t_1^0)$ is the lexicographically first sequence which contain t_0^0.

Fact 1 *Let a, b, c $(a < b < c)$ be three integers, and S_1, S_2, S_3 be constants. If the following equations are satisfied, then a, b, c are uniquely determined.*

$$\begin{aligned} a+b+c &= S_1 \\ a^2+b^2+c^2 &= S_2 \\ a^3+b^3+c^3 &= S_3 \end{aligned}$$

(Proof is omitted.)

Since the tuple t_0^0 is in the sequence, we can prove that one of t_m^0 $(1 \leq m \leq 7)$ is also in the sequence as follows. We assume $t_0^0 = [\alpha, \alpha, \alpha^2, \alpha^3, 0, 0, 0]$ and the second, third and fourth elements of t_0^0 is equal to S_1, S_2 and S_3 in Fact 1, respectively. From the above discussion, there exist at least one tuple whose parent vertex is in V_2. Since the second, third and fourth elemetns of the tuple are zero, there are at most three tuples whose second, third and fourth elements are not zero except for t_0^0. Since $S_1 = \alpha$, $S_2 = \alpha^2$ and $S_3 = \alpha^3$, only one of t_m^0 $(1 \leq m \leq 7)$ satisfies Fact 1.

For each pair of (t_0^0, t_1^0), (t_0^0, t_2^0), ..., (t_0^0, t_7^0), the other three tuples are also uniquely determined from fifth, sixth and seventh elements of tuples and Fact 1. Among the determined sequences of tuples, the sequence of tuple $(t_0^0, t_0^j, t_0^k, t_0^l, t_1^0)$ is the lexicographically first sequence.

It is easy to see that the above reduction is in NC. Although we define that inputs of $LFM5S$ is integers, we can easily reduce each 7-tuple to an integer. □

4.3 Cost optimal PRAM algorithm

The parallel algorithm for computing $LFM5S$ on the CREW PRAM is simple as follows. (The algorithm is also a sequential algorithm which runs in $O(n^4)$ if we use one processor.)

Algorithm for computing $LFM5S$
Input: A set of n integers I.

Step 1: Sort all elements in I. Let $S = (s_0, s_1, \ldots, s_{n-1})$ be the sorted input.

Step 2: Repeat the following substeps from $i = 0$ to $i = n - 5$.

(2-1) For each pair of $b, c \in S$ with $s_i < b < c$, compute two elements d, e with $d < e$ which satisfy $d + e = -(s_i + b + c)$ in parallel as follows.

(2-1-1) Compute BOT which satisfies $s_{BOT-1} = c$, and set $TOP = n - 1$.

(2-1-2) Compare $s_{BOT} + s_{TOP}$ and $-(s_i+b+c)$, and execute one of the followings according to the comparison until $BOT \geq TOP$.

- Output $d = s_{BOT}, e = s_{TOP}$ and quit (2-1). (If $s_{BOT}+s_{TOP} = -(s_i+b+c)$.)
- Set BOT = BOT+1 and repeat (2-1-2). (If $s_{BOT} + s_{TOP} < -(s_i + b + c)$.)
- Set TOP = TOP-1 and repeat (2-1-2). (If $s_{BOT} + s_{TOP} > -(s_i + b + c)$.)

(2-2) Compute the lexicographically first 5 sum (s_i, b, c, d, e) among sequences obtained in (2-1).

(2-3) Output the lexicographically first sequence of 5 elements (a, b, c, d, e) if they exist, as a result of $LFM5S$, and delete all elements a, b, c, d, e from S.

We can sort n elements in $O(\frac{n \log n}{p} + \log n)$ time using Cole's merge sort[3] in Step 1. In step 2, there are at most n^2 pairs and substeps (2-1-1) and (2-1-2) can be computed in $O(n)$ time. Thus (2-1) $\sim$ (2-3) takes in $O(\frac{n^3}{p} + n)$ time. Since the number of repetitions in Step 2 is n, we obtain the following theorem.

Theorem 4 *We can solve LFM5S in* $O(\frac{n^4}{p})$ *time using p processors* $(1 \leq p \leq n^2)$ *on the CREW PRAM.* □

4.4 Experimental results

We modify our PRAM algorithm for computing *LFM5S* and implement on a cluster of PC using PVM (Parallel Virtual Machine)[6]. (Although PRAM is a shared memory model and a PC cluster is a distributed memory environment, we can implement the algorithm easily since the algorithm enough simple.) We used at most 16 PCs (Pentium II 233, 64MB, Solaris 2.6), which are connected by Ethernet (10BASE-T). The program is written in C and compiled using GNU C.

Inputs of the experiments are integers and generated randomly. (The size of the input is 1,000.) Since n is enough small in this problem, we prepare whole input data for each processor, that is, every processor stores the same n inputs. (This assumption is also made as a simulation of concurrnet read ability on the CREW PRAM.)

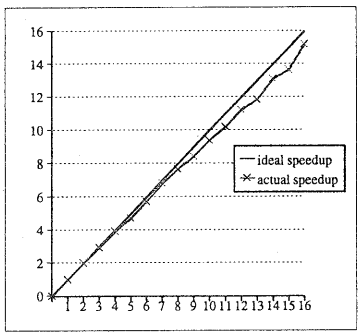

Figure 5: Speedup of our algorithm for *LFM5S*.

Figure 5 describes an experimental results of our algorithm. The PVM on a cluster is a primitive environment for parallel computing. In addition, our network, which is connected with 10BASE-T, is slow. Nevertheless the algorithm achieves almost ideal speedup. This implies that some P-complete problems are easy to parallelize within the reasonable number of processors.

5 Conclusions

In this paper, we proved two problems are P-complete, and proposed cost optimal algorithms for the problems. We implemented the second algorithm on a PC cluster using PVM, and obtained almost ideal speedup. The fact implies that some P-complete problems are parallelizable within the reasonable number of processors.

In the future research, we investigate other parallelizable P-complete problems. The result may imply new classification of problems in P. Another future topic is proposition of fast parallel algorithms which runs in $O(n^\epsilon)$ time where $0 < \epsilon < k$ for P-complete problems. Only a few P-complete problems are known to have such algorithms[11].

References

[1] B. Chazelle. On the convex layers of a planar set. *IEEE Transactions on Information Theory*, IT-31(4):509–517, 1985.

[2] W. Chen. *Parallel Algorithm and Data Structures for Geometric Problems.* PhD thesis, Osaka University, 1993.

[3] R. Cole. Parallel merge sort. *SIAM Journal of Computing*, 17(4):770–785, 1988.

[4] A. Dessmark, A. Lingas, and A. Maheshwari. Multi-list ranking: complexity and applications. In *10th Annual Symposium on Theoretical Aspects of Computer Science (LNCS665)*, pages 306–316, 1993.

[5] J. Erickson and R. Seidel. Better lower bounds on detecting affine and spherical degeneracies. In *34th Annual IEEE Symposium on Foundations of Computer Science (FOCS '93)*, pages 528–536, 1993.

[6] G. A. Geist et al. *PVM: Parallel Virtual Machine A Users' Guide and Tutorial for Networked Parallel Computing.* MIT Press, 1993.

[7] A. Gajentaan and M. H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Computational geometry*, 5:165–185, 1995.

[8] R. Greenlaw, H.J. Hoover, and W.L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory.* Oxford university press, 1995.

[9] C.P. Kruskal, L. Rudolph, and M. Snir. A complexity theory of efficient parallel algorithms. *Theoretical Computer Science*, 71:95–132, 1990.

[10] S. Miyano. The lexicographically first maximal subgraph problems: P-completeness and NC algorithms. *Mathematical Systems Theory*, 22:47–73, 1989.

[11] J.S. Vitter and R.A. Simons. New classes for parallel complexity: A study of unification and other complete problems for P. *IEEE Transactions of Computers*, C-35(5):403–418, 1986.

[12] A. C. Yao. A lower bound to finding convex hulls. *Journal of the ACM*, 28(4):780–787, 1981.