京都大学学術情報リポジトリ
KURENAI 紅
Kyoto University Research Information Repository

京都大学
KYOTO UNIVERSITY

Title	Cost optimal parallel algorithms for P-complete problems (Algorithm Engineering as a New Paradigm)
Author(s)	Fujiwara, Akihiro; Inoue, Michiko; Masuzawa, Toshimitsu
Citation	数理解析研究所講究録 (2001), 1185: 53-62
Issue Date	2001-01
URL	http://hdl.handle.net/2433/64640
Right	
Type	Departmental Bulletin Paper
Textversion	publisher

Kyoto University

数理解析研究所講究録
1185 巻 2001 年 53-62

Cost optimal parallel algorithms for P-complete problems

Akihiro FUJIWARA[1], Michiko INOUE[2] and Toshimitsu MASUZAWA[2]

[1]Department of Computer Science and Electronics,
Kyushu Institute of Technology
680-4 Kawazu, Iizuka, Fukuoka, 820-8501, Japan
fujiwara@cse.kyutech.ac.jp

[2] Graduate School of Information Science,
Nara Institute of Science and Technology
8916-5, Takayama, Ikoma, 630-0101, Japan
{kounoe, masuzawa}@is.aist-nara.ac.jp

Abstract: In this paper, we consider parallelizability of two P-complete problems, the convex layers problem and the lexicographically first maximal 3 sums problem. For the convex layers problem, we first introduce a parameter which indicates parallelizability of the problem. We prove P-completeness of the problem and propose a cost optimal parallel algorithm according to the parameter. For the lexicographically first maximal 3 sums problem, we prove P-completeness of the problem, and propose two cost optimal parallel algorithms for the problem and a related problem. The above results show that some P-complete problems have efficient cost optimal parallel algorithms.

Keywords: parallel algorithms, P-completeness, convex layers, lexicographically first maximal 3 sums.

1 Introduction

In parallel computation theory, one of primary complexity classes is the class NC. Let n be the input size of a problem. The problem is in the class NC if there exists 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. The problem in the class NC are regarded as efficiently solvable by parallel algorithms. Many problems in the class P, which is the class of problems solvable in polynomial time sequentially, are shown to be in the class NC.

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

However polylogarithmic time complexity is not so important for real parallel computation. The polylogarithmic time complexity $O((\log n)^k)$ needs at least $\frac{T_s(n)}{(\log n)^k}$ processors, where $T_s(n)$ is sequential time complexity for the same problem and usually greater than n. This fact implies that more than $n^{1-\epsilon}$ processors are needed for the polylogarithmic time complexity, where ϵ is a constant which satisfies $0 < \epsilon < 1$. On the other hand, the number of processor p for real parallel computation is independent of the size n and $p << n$. This fact implies that p is smaller than $n^{1-\epsilon}$ in most cases and is inconsistent with the polylogarithmic time complexity.

In consequence, *cost optimality* for $p << n$ is the most important measure for parallel algorithms in practice. The cost of a parallel algorithm is defined as the product of running time and the number of processors of the algorithm. A parallel algorithm is cost optimal if its cost is asymptotically equal to the lower bound of sequential time complexity for the same problem. In other words, the cost optimal parallel algorithm achieves optimal speedup, which is asymptotically equal to the number of processors.

Therefore one way to practically parallelize a P-complete problem is to find a cost optimal parallel algorithm which runs in polynomial time. Let $\Omega(n^k)$ be the 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 for $p << n$. In this paper, we call P-complete problems *parallelizable* if the problem has a cost optimal parallel algorithm whose number of processors is an increasing function of the input size, and aim to show some P-complete problems are parallelizable.

Among many P-complete problems, only some graph problems are known to be parallelizable. Vitter and Simons[11] 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 $\epsilon > 0$ 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 $m = \Theta(n^2)$ and $p < n$, then the algorithm achieves cost optimality.

In this paper, we consider parallelizability of two P-complete problems. First we consider *a convex layers problem.* The convex layers is a geometric problem and closely relates to the other layering problems, such as a visibility layers problem. Dessmark et al.[5] 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. We introduce a parameter d to the problem so that it restricts positions of input points: the input points are on d horizontal lines. We show that the parameter indicates parallelizability of the problem. For the parameterized convex layers problem, we prove that the problem is still P-complete if $d = n^{\epsilon}$ with $0 < \epsilon < 1$. 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 $0 < \epsilon \leq \frac{1}{2}$ for $1 \leq p \leq n^{\epsilon}$. We also consider complexity of the algorithm in case that all inputs are sorted, and show the complexity achieves similar cost optimality.

The second P-complete problem is *a lexicographically first maximal 3 sums problem*, which is a problem to compute a lexicographically first maximal set of sequences of 3 integers, $\{(a_0, b_0, c_0), (a_1, b_1, c_1), \ldots, (a_{m-1}, b_{m-1}, c_{m-1})\}$ which satisfy $a_i + b_i + c_i = 0$ for all i $(0 \leq i \leq m-1)$, from a given set of integers. We prove the problem is also P-complete by reducing a lexicographically first maximal independent set problem, which is a graph problem known as P-complete[10]. Next we propose a parallel algorithm, which runs in $O(\frac{n^2}{p} + n \log n)$ using p processors $(1 \leq p \leq n)$ on the CREW PRAM, for the problem. In addition, we propose a parallel algorithm for a related P-complete problem, which is *a lexicographically first maximal set of 3 arguments problem.* The algorithm runs in $O(\frac{n^3}{p} + n \log n)$ time for $1 \leq p \leq n^2$ on the CREW PRAM. The above two algorithms are cost optimal for $1 \leq p \leq \frac{n}{\log n}$ and $1 \leq p \leq \frac{n^2}{\log n}$, respectively. These results show that some P-complete problems have efficient cost optimal parallel algorithms.

2 Preliminaries

2.1 Definition of P-completeness

In this subsection, we describe a brief definition of P-completeness. (For details of precise definitions of P-completeness, see [8].)

Let n be the input size of a problem. The problem is in the class P if there exists a sequential algorithm which solves the problem in $t(n)$ time where $t(n)$ is a polynomial function for n. The class P is a well known class which denotes sequential efficiency. An analogous class of efficiency for parallel computation is the class NC. A problem is in the class NC if there exists 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.

Definition 1 (P-complete problem) *A prob-*

lem Q is P-complete if the following two conditions are satisfied.

(1) *Q is in P.*

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

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

3 Parameterized convex layers

3.1 Definitions

First we give some definitions for the 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 a set of convex hulls, $\{CH_0, CH_1, \ldots, CH_{m-1}\}$, which satisfies the following two conditions.*

(1) *$CH_0 \cup CH_1 \cup \ldots \cup CH_{m-1} = S$.*

(2) *Each CH_i $(0 \leq i \leq m-1)$ is a convex hull of a set of points $S-(CH_0 \cup CH_1 \cup \ldots \cup CH_{i-1})$.* □

Dessmark et al.[5] proved P-completeness of the convex layers problem, and Chazelle[1] proposed a sequential algorithm which runs in $O(n \log n)$ time. The time complexity is optimal because computation of a convex hull, which is the first hull of the convex layers, requires $\Omega(n \log n)$ time[12].

In this paper, we introduce 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 convex layers problem whose input points are on d horizontal lines.* □

The parameter 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 problem. We can solve the problem sequentially in $O(n \log n)$ time using the algorithm[1].

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

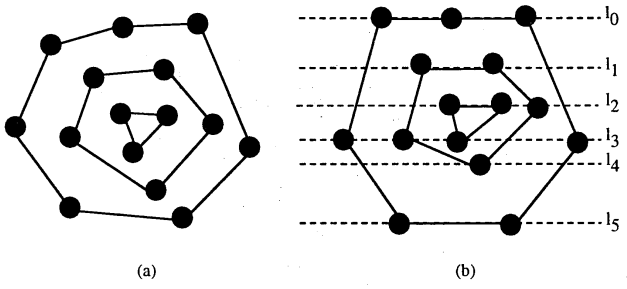

Figure 1: Convex layers problems: (a) convex layers, and (b) CL(6).

3.2 P-completeness of $CL(d)$

In this subsection, we discuss relationship between P-completeness and the parameter d. 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 problem.

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

(Proof) 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 problem 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 form 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 $y = y_N + 2$, that is, $U_2 = \{(x'_0, y_N + 2), (x'_1, y_N + 2), \ldots, (x'_{k-1}, y_N + 2)\}$ so that $x_W - 1 < x'_0 < x'_1 < \ldots < x'_{k-1} < x_E + 1$. (Since $0 < \epsilon < 1$, $k > 0$ holds.) These added points are illustrated in Figure 2.

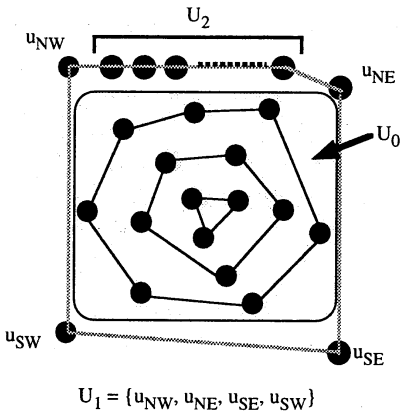

Figure 2: Points in U_0, U_1 and U_2.

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 prove that $CL((\log n)^k)$ is in NC where k is an arbitrary positive constant.

3.3 A cost optimal parallel algorithm for $CL(d)$

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 the outmost convex hull. The outmost convex hull consists of the following points.

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

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

We can compute points included in (b) in parallel for each line in $O(1)$ time because points on each line are stored in a double-ended queue. Since obtained points are sorted in order of y coordinates, we can compute the outmost convex hull of the points using a cost optimal parallel algorithm[2, 3] which computes a convex hull for sorted points. 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 the 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 horizontal 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 the top and bottom lines respectively.) Compute a set of points on each line l_i $(0 \leq i \leq d-1)$, and store them in a double-ended queue Q_i in order of x coordinates.

Step 2: For each line l_i $(TOP \leq i \leq BOT)$, compute the leftmost point u^i_{left} and the 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 the obtained points on each hull in CH_{left} and CH_{right}, respectively. (The left hull of U_{left} consists of points on a convex hull of U_{left}, which are from u^{BOT}_{left} to u^{TOP}_{left} in clockwise order. 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 the top and bottom lines on which there remains at least one point. Set TOP and BOT to obtained top and bottom lines respectively.

Step 6: Repeat Steps 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[4] and primitive operations, and Step 2 takes $O(\frac{d}{p})$ time obviously. We can compute the left and right hulls in Step 3 using a known parallel algorithm[2, 3] for computing a convex hull of 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 the 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 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 $O(\frac{n \log n}{p} + (\log n)^k \log \log n)$ in the case. If we use n processors in Step 1, the complexity becomes $O((\log n)^k \log \log n)$. Consequently, we obtain the following corollary.

Corollary 1 *We solve $CL((\log n)^k)$, where k is a positive constant, in $O((\log n)^k \log \log n)$ 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 to this, we assume that $\epsilon \leq \frac{1}{2}$ and $p \leq d = n^\epsilon$. Under the assumption, $\frac{n \log n}{p} \geq \frac{n^{2\epsilon}}{p}$ 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 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 Step1 runs in $O(\frac{n}{p})$ time if all points on each line are known and sorted in order of x coordinates. In this case, we can solve $CL(d)$ efficiently. We call the problem $CLS(d)$ *(convex layers for sorted points on d lines).* From the above discussion, 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 $0 < \epsilon < \frac{1}{2}$ 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 3 sums

4.1 Definitions

We first define the maximal 3 sums problem as follows.

Definition 4 (Maximal 3 sums) *Let I be a set of n distinct integers. The maximal 3 sums is a problem to compute the set of sequences of 3 integers $M3S = \{(a_0, b_0, c_0), (a_1, b_1, c_1), \ldots, (a_{m-1}, b_{m-1}, c_{m-1})\}$, which satisfies the following three conditions.*

1. *The set $S = \{a_0, b_0, c_0, a_1, b_1, c_1, \ldots, a_{m-1}, b_{m-1}, c_{m-1}\}$ is a subset of I.*
2. *For each (a_i, b_i, c_i) $(0 \leq i \leq m-1)$, $a_i + b_i + c_i = 0$.*
3. *There is no sequence of three integers (a', b', c') which satisfies $a', b', c' \in I - S$ and $a' + b' + c' = 0$.* □

Note that solution of the maximal 3 sums is not unique. For example, let $I = \{-10, -9, -2, 0, 9, 10, 11, 12\}$ be an input for the

maximal 3 sums. Then both of $\{(-10,-2,12),(-9,0,9)\}$, $\{(-10,0,10),(-9,-2,11))\}$ are solutions for the problem.

The lexicographically first maximal 3 sums problem is a modified maximal 3 sums problem so as to have the unique solution. Let $A = (a_0, a_1, \ldots a_{m-1})$ and $B = (b_0, b_1, \ldots, b_{m-1})$ be two sequences of m numbers. We call that A *is lexicographically less than* B if $a_0 < b_0$, or there exists an integer i $(1 \leq i \leq m-1)$ such that $a_j = b_j$ for all j $(0 \leq j \leq i-1)$ and $a_i < b_i$. We call that 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 3 sums (LFM3S)) *Let I be a set of n distinct integers. The lexicographically first maximal 3 sums is a problem to compute the set of sequences of 3 integers $LFM3S = \{(a_0, b_0, c_0), (a_1, b_1, c_1), \ldots, (a_{m-1}, b_{m-1}, c_{m-1})\}$, which satisfies the following two conditions.*

1. *The $LFM3S$ is a solution of the maximal 3 sums for I.*
2. *Let $s_i = \{a_i, b_i, c_i\}$ $(0 \leq i \leq m-1)$. Then, (a_i, b_i, c_i) is the lexicographically first sequence of 3 integers which satisfies $a_i + b_i + c_i = 0$ for $I - (s_0 \cup s_1 \cup \ldots \cup s_{i-1})$.* □

We can compute LFM3S sequentially by the following algorithm for a set of distinct integers I.

(1) Repeat the following substeps until no set of 3 integers is found in (1-1).

(1-1) Find a sequence of 3 integers (a, b, c) which satisfies the following two conditions.

- **(a)** a, b, c are elements in I which satisfy $a + b + c = 0$.
- **(b)** (a, b, c) is the lexicographically first sequence among all sequences which satisfy (a).

(1-2) Output the sequence (a, b, c) as a solution, and remove a, b, c from I, whenever they exist.

In case of the above example, $\{(-10,-2,12),(-9,0,9)\}$ is the unique solution of LFM3S for I.

4.2 P-completeness of LFM3S

We show reduction from *the lexicographically first maximal independent set (LFMIS) problem* to LFM3S. The LFMIS is a well known P-complete problem[10]. Let $G = (V, E)$ be an input graph for LFMIS. We assume that all vertices in $V = \{v_0, v_1, \ldots, v_{n-1}\}$ are totally ordered, that is, v_i is less than v_j if $i < j$. We can compute LFMIS sequentially as follows. (VS is an output of LFMIS for G.)

(1) Set $VS = \phi$.

(2) Repeat the following substeps until $V = \phi$.

(2-1) Compute the first vertex $v_i \in V$.

(2-2) Add v_i to VS, and remove v_i and its adjacent vertices from V, and also remove adjacent edges from E. □

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

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

Using the above lemma, we prove the following theorem.

Theorem 3 *The problem LFM3S is P-complete.*

(Proof)

It is obvious that LFM3S is in P.

First we introduce a key set of integers for LFM3S. The set is as follows.

$$\begin{aligned} Q &= \{q_0, q_1, \ldots, q_{12}\} \\ &= \{-64, -61, -32, -31, -29, -15, -14, \\ &\quad -13, -10, -8, 23, 46, 93\} \end{aligned}$$

We describe property of the above set Q. Let Q be the input for LFM3S. Then the solution of LFM3S is as follows.

$$\{(-64, -29, 93), (-32, -14, 46), (-15, -8, 23)\}$$

Next, letting $Q-\{-64\}$ be the input, the solution becomes as follows.

$$\{(-61,-32,93),(-31,-15,46),(-13,-10,23)\}$$

The most important difference between the above two solutions is that three numbers, $q_1 = -61$, $q_3 = -31$ and $q_7 = -13$, are not in the first solution but in the second solution.

We show precise reduction from LFMIS to LFM3S using the above key set Q in the following. Let $G = (V, E)$ with $V = \{v_0, v_1, \ldots, v_{n-1}\}$ be an input graph with degree at most 3. First we define *a vertex value* $VV(i)$ for each vertex v_i. The vertex value is a negative integer and defined as $VV(i) = i - n$. Thus vertices $v_0, v_1, \ldots, v_{n-1}$ have vertex values $-n, -(n-1), \ldots, -1$ respectively. Using the key set $Q = \{q_0, q_1, \ldots, q_{12}\}$ and the vertex values, we define the following 4-tuples for each vertex v_i $(0 \leq i \leq n-1)$ in V as inputs for LFM3S.

1. **Vertex tuple for** v_i:
   $VT(i) = [VV(i), q_0, VV(i), 0]$
2. **Auxiliary tuples for** v_i:
   (a) $AT_1(i) = [VV(i), q_1, 0, VV(i)]$
   (b) $AT_2(i) = [VV(i), q_2, VV(i), 0]$
   (c) $AT_3(i) = [VV(i), q_3, 0, VV(i)]$
   (d) $AT_4(i) = [VV(i), q_4, 0, VV(i)]$
   (e) $AT_5(i) = [VV(i), q_5, VV(i), 0]$
   (f) $AT_6(i) = [VV(i), q_6, 0, VV(i)]$
   (g) $AT_7(i) = [VV(i), q_7, 0, VV(i)]$
   (h) $AT_8(i) = [VV(i), q_8, VV(i), 0]$
   (i) $AT_9(i) = [VV(i), q_9, 0, VV(i)]$
   (j) $AT_{10}(i) = [2 * |VV(i)|, q_{10}, |VV(i)|, |VV(i)|]$
   (k) $AT_{11}(i) = [2 * |VV(i)|, q_{11}, |VV(i)|, |VV(i)|]$
   (l) $AT_{12}(i) = [2 * |VV(i)|, q_{12}, |VV(i)|, |VV(i)|]$
3. **Link tuples for** v_i: For each adjacent vertex v_j of v_i, which satisfies $i < j$, add one of the following tuples.
   (a) $LT_1(i,j) = [|VV(i)| + |VV(j)|, |q_0| + |q_1|, |VV(j)|, |VV(i)|]$
   (b) $LT_2(i,j) = [|VV(i)| + |VV(j)|, |q_0| + |q_3|, |VV(j)|, |VV(i)|]$
   (c) $LT_3(i,j) = [|VV(i)| + |VV(j)|, |q_0| + |q_7|, |VV(j)|, |VV(i)|]$

   (In case that v_i has only one adjacent vertex v_j which satisfies $i < j$, add $LT_1(i,j)$ for v_j. In case that v_i has the two adjacent vertices v_{j_1}, v_{j_2} which satisfies $i < j_1 < j_2$, add $LT_1(i,j_1)$ and $LT_2(i,j_2)$. In case that v_i has the three adjacent vertices, add all three tuples similarly.)

The above 4-tuples have the following special feature. Let $\{VT(i), AT_1(i), AT_2(i), \ldots, AT_{12}(i), LT_1(i,s), LT_2(i,t), LT_3(i,u), VT(s), VT(t), VT(u)\}$ be the input for LFM3S[1]. (We assume v_s, v_t and v_u are adjacent vertices which satisfy $i < s < t < u$.) Then the solution of LFM3S is as follows. (We call the solution *TYPE A sums.*)

$$\begin{gathered}\{(VT(i), AT_4(i), AT_{12}(i)),\\ (AT_2(i), AT_6(i), AT_{11}(i)),\\ (AT_5(i), AT_9(i), AT_{10}(i)),\\ (AT_1(i), VT(s), LT_1(i,s)),\\ (AT_3(i), VT(t), LT_2(i,t)),\\ (AT_7(i), VT(u), LT_3(i,u))\}\end{gathered}$$

Note that vertex tuples, $VT(s)$, $VT(t)$ and $VT(u)$, are in the sums. In other words, the above vertex tuples are not in the remaining inputs after the computation.

Next, we consider the solution without $VT(i)$ in the input. (We call the solution *TYPE B sums.*)

$$\begin{gathered}\{(AT_1(i), AT_2(i), AT_{12}(i)),\\ (AT_3(i), AT_5(i), AT_{11}(i)),\\ (AT_7(i), AT_8(i), AT_{10}(i))\}\end{gathered}$$

In this case, the vertex tuples, $VT(s)$, $VT(t)$ and $VT(u)$, remain in the inputs.

We give the above 4-tuples for all vertices in V of LFMIS, and compute LFM3S. Then the vertex $v_i \in V$ is in the solution of LFMIS if and only if

[1]The sum of tuples $A = [\alpha_0, \alpha_1, \alpha_2, \alpha_3]$ and $B = [\beta_0, \beta_1, \beta_2, \beta_3]$ is defined as $A + B = [\alpha_0 + \beta_0, \alpha_1 + \beta_1, \alpha_2 + \beta_2, \alpha_3 + \beta_3]$, and $A < B$ if A is lexicographically less than B. We assume that the sum is zero if the sum of tuples is $[0,0,0,0]$.

there exists a sum of three 4-tuples (T_1, T_2, T_3) which satisfies $T_1 = VT(i)$ in the solution of LFM3S.

In the following, we describe correctness of the reduction briefly. Let $[\alpha_0, \alpha_1, \alpha_2, \alpha_3]$ be one of input tuples. The α_0 is a vertex value, its absolute values or the sum of the values, and ensures that the sums of tuples are computed for each vertex lexicographically; that is, the sums are computed in order of the vertex values. The α_2 and α_3 are vertex values, its absolute values or 0, and ensures that no invalid sum is obtained. (The invalid sum means the sum between tuples of different vertices except for link tuples.) Since the α_1 is one element of the key set Q, only TYPE A or TYPE B sums for each vertex are obtained in computation of LFM3S.

It is easy to see that the above reduction is in NC. Although we define that inputs of $LFM3S$ are integers, inputs of the above reduction are 4-tuples. We can easily reduce each 4-tuple to an integer without loss of the features. Let $2^g \leq n < 2^{g+1}$ and $h = \max\{g, 6\}$. Then we can reduce each 4-tuple $[\alpha_0, \alpha_1, \alpha_2, \alpha_3]$ to $\alpha_0 * 2^{3(h+1)} + (\alpha_1 - 65) * 2^{2(h+1)} + \alpha_2 * 2^{h+1} + \alpha_3$. □

In addition to this, we consider P-completeness of the following problem as generalization of LFM3S.

Definition 6 Lexicographically first maximal set of 3 arguments (LFMS3A) *Let E be a totally ordered set of n distinct elements. The lexicographically first maximal set of 3 arguments is a problem to compute the set of sequences of 3 elements $LFMS3A = \{(a_0, b_0, c_0), (a_1, b_1, c_1), \ldots, (a_{m-1}, b_{m-1}, c_{m-1})\}$, which satisfies the following three conditions for a given function $f(x, y, z)$ whose value is $TRUE$ or $FALSE$.*

1. *The set $S = \{a_0, b_0, c_0, a_1, b_1, c_1, \ldots, a_{m-1}, b_{m-1}, c_{m-1}\}$ is a subset of E.*
2. *Let $e_i = \{a_i, b_i, c_i\}$ $(0 \leq i \leq m-1)$. Then, (a_i, b_i, c_i) is the lexicographically first set of 3 elements which satisfies $f(a_i, b_i, c_i) = TRUE$ for $I - (e_0 \cup e_1 \cup \ldots \cup e_{i-1})$.*
3. *There is no set of three elements (a', b', c') which satisfies $a', b', c' \in I - S$ and $f(a', b', c') = TRUE$.* □

Since LFMS3A is generalization of LFM3S, we can obtain the following corollary from Theorem 3.

Corollary 4 *The problem LFMS3A is P-complete.* □

4.3 Cost optimal parallel algorithms

In this subsection, we propose two cost optimal parallel algorithms for LFM3S and LFMS3A. The first and second algorithms run in $O(\frac{n^2}{p} + n \log n)$ and $O(\frac{n^3}{p} + n \log n)$ time using p processors, respectively.

4.3.1 A parallel algorithm for LFM3S

In this subsection, we consider a parallel algorithm for LFM3S on the CREW PRAM. We can propose a sequential algorithm which solves LFM3S in $O(n^2)$ by modifying an algorithm computing the 3 sum problem[7]. (The 3 sum is a decision problem which decides whether there exist $a, b, c \in I$ satisfying $a + b + c = 0$.) The algorithm is the known fastest sequential algorithm for LFM3S. Note that non-trivial lower bound of LFM3S is not known. However the 3 sum has no $o(n^2)$ algorithm and has an $\Omega(n^2)$ lower bound on a weak model of computation[6].

We propose a cost optimal parallel algorithm for LFM3S in the following.

Algorithm for computing $LFM3S$
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 sequence.)

Step 2: Repeat the following substeps from $i = 0$ to $i = n - 3$.

(2-1) Create the following two sorted sequences S' and S'_R from S.

$$S' = (s_{i+1}, s_{i+2}, \ldots, s_{n-1})$$
$$S'_R = (-s_{n-1} - s_i, -s_{n-2} - s_i, \ldots, -s_{i+1} - s_i)$$

(For $b \in S'$ and $c \in S'_R$ which satisfy $b = s_g$ and $c = -s_h - s_i$ respectively, $b = c$ if and only if $s_i + s_g + s_h = 0$.)

(2-2) Merge S' and S'_R into a sorted sequence $SS = (ss_0, ss_1, \ldots, ss_{2(n-i-1)-1})$.

(2-3) Compute the smallest element ss_j in SS which satisfies $ss_j = ss_{j+1}$.

(2-4) If the above ss_j is obtained, compute s_g and s_h in S such that $s_g = ss_j$ and $s_h = -s_g - s_i$, respectively. (It is obvious that $s_g \in S'$ and $-s_g - s_i \in S'_R$ since all elements in S are distinct.) Delete s_i, s_g, s_h from I, and output (s_i, s_g, s_h), whenever they exist.

We assume the number of processors p is restricted to $1 \leq p \leq n$. We can sort n elements in $O(\frac{n \log n}{p} + \log n)$ time using Cole's merge sort[4] in Step 1. In Step 2, we can compute a substep (2-1) in $O(\frac{n}{p})$ time easily. We can compute substeps (2-3) and (2-4) in $O(\frac{n}{p} + \log n)$ time using simple known algorithms and basic operations. In a substep (2-2), we can merge two sorted sequence in $O(\frac{n}{p} + \log \log n)$ time using a fast merging algorithm[9]. Since repetition of Step 2 is n, we obtain the following theorem.

Theorem 4 *We solve LFM3S in* $O(\frac{n^2}{p} + n \log n)$ *time using p processors* $(1 \leq p \leq n)$ *on the CREW PRAM.* □

In case of $1 \leq p \leq \frac{n}{\log n}$, the time complexity becomes $O(\frac{n^2}{p})$. Therefore the above algorithm is cost optimal for $1 \leq p \leq \frac{n}{\log n}$.

4.3.2 A parallel algorithm for LFMS3A with an unresolvable function

We consider the case that a given function $f(x, y, z)$ of LFMS3A is unresolvable, that is,

1. For given a, b, c, we can compute a value $f(a, b, c)$ in $O(1)$ time.
2. For a set E', we cannot resolve $a, b, c \in E'$ which satisfy $f(a, b, c) = TRUE$ in practical time.

In this case, the only way to solve LFMS3A is to compute $f(a, b, c)$ for all combinations (a, b, c) where $a, b, c \in E$. The computation obviously requires $\Omega(n^3)$ time sequentially.

A parallel algorithm for computing LFMS3A on the CREW PRAM is simple as follows.

Algorithm for computing LFMS3A with an unresolvable function
Input: A set of n elements E.

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

Step 2: Repeat the following substeps from $i = 0$ to $i = n - 3$.

(2-1) Compute all pairs (s_g, s_h) which satisfy $f(s_i, s_g, s_h) = TRUE$ $(i < g < h \leq n - 1)$.

(2-2) Compute the lexicographically first pair $(s_{g'}, s_{h'})$ among the pairs obtained in (2-1), delete $s_i, s_{g'}, s_{h'}$ from I, and output $(s_i, s_{g'}, s_{h'})$, whenever they exist.

We assume the number of processors is p $(1 \leq p \leq n^2)$. Since Step 1 is the same as the first algorithm, we consider complexity of Step 2. In a substep (2-1), there are at most $\frac{(n-1)(n-2)}{2}$ pairs, and we can compute the pairs in $O(\frac{n^2}{p})$ time. We can compute a substep (2-2) in $O(\frac{n^2}{p} + \log n)$ time easily. Since repetition of Step 2 is $O(n)$, we obtain the following theorem.

Theorem 5 *We solve LFMS3A with an unresolvable function in* $O(\frac{n^3}{p} + n \log n)$ *time using p processors* $(1 \leq p \leq n^2)$ *on the CREW PRAM.* □

The above algorithm is also cost optimal for $1 \leq p \leq \frac{n^2}{\log n}$.

5 Conclusions

In this paper, we proved that two problems, $CL(n^\epsilon)$ and $LFM3S$, are P-complete, and proposed cost optimal algorithms for the problems. The results imply 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 run in $O(n^\epsilon)$ time where $0 < \epsilon < 1$ 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] D. Z. Chen. Efficient geometric algorithms on the EREW PRAM. *IEEE transactions on parallel and distributed systems*, 6(1):41–47, 1995.

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

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

[5] 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.

[6] 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.

[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. Kruskal. Searching, merging and sorting in parallel computation. *IEEE Transactions on Computers*, C-32(10):942–946, 1983.

[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.