A Parallel Algorithm for the Maximum 2-Chain Edge Packing Problem

Shiraishi, Shuji
Department of Applied Mathematics Fukuoka University

出版情報：RIFIS Technical Report. 9, 1988-12-26. Research Institute of Fundamental Information Science, Kyushu University
バージョン：
権利関係：

RIFIS-TR-CS-9

RIFIS Technical Report

A Parallel Algorithm for the Maximum 2-Chain Edge Packing Problem

Shuji Shiraishi

December 26, 1988

Research Institute of Fundamental Information Science
Kyushu University 33
Fukuoka 812, Japan
E-mail: shuji@rifis1.sci.kyushu-u.junet Phone: 092(641)1101 Ext.4484

A PARALLEL ALGORITHM FOR THE MAXIMUM 2-CHAIN EDGE PACKING PROBLEM

Shuji SHIRAISHI

Department of Applied Mathematics
Fukuoka University, Fukuoka 814-01, Japan

We present a parallel algorithm for finding a maximum 2-chain edge packing of an undirected graph $G = (V, E)$. It runs in $O(\log n)$ time using $O(n+m)$ processors on a CRCW PRAM, where $n = |V|$ and $m = |E|$.

keywords: 2-chain edge packing, Euler tour technique, parallel algorithm

1. Introduction

A *k-chain* is a simple path of length k. A k-chain *edge packing* of an undirected graph $G = (V, E)$ is a set of k-chains which are pairwise edge disjoint in G. The task of selecting a maximum subset of such k-chains is called the *maximum k-chain edge packing problem.*

The edge packing problem has been studied as a generalization of the bin packing problem [6,8,9]. It is shown [8] that the maximum k-chain edge packing problem for $k \geq 3$ is NP-complete but the maximum 2-chain edge packing problem can be solved in linear time by using the depth first search technique. However it should be noted that the depth first search is not known to be in NC [1,10].

In the present paper, we give an efficient parallel algorithm for the maximum 2-chain edge packing problem which runs in time $O(\log n)$ using $O(n+m)$ processors

on a CRCW PRAM. The algorithm employs spanning trees instead of depth first search trees and then exploits the Euler tour technique [2,12].

2. Algorithm and its complexity

A *spanning tree* of G is a tree that contains all vertices of G. An *Euler graph* is a connected graph in which at most two vertices are of odd degree. An *Euler tour* in a graph is a circuit (path) that traverses every edge exactly once. It is known that Euler tours in graphs can be found in $O(\log n)$ parallel time using $O(n+m)$ processors on a CRCW PRAM [2,13]. We remark that if G is an Euler graph, the maximum k-chain edge packing problem is solvable in $O(\log n)$ parallel time with $O(n+m)$ processors on a CRCW PRAM. *A circular succession assignment* of a tree T is defined as follows [2]: Let $deg(x) = d$ in T and let $\{x, y_0\}, ..., \{x, y_{d-1}\}$ be the edges adjacent to x. Then we assign $NEXT((y_i, x)) := (x, y_{i+1 \ mod \ d})$ for $0 \leq i \leq d-1$. An Euler tour of T' (antiparallel directed graph of T) is computed into array $NEXT$.

The main result of this paper is the following:

Theorem 1. *A maximum 2-chain edge packing of an undirected graph can be computed in $O(\log n)$ time using $O(n+m)$ processors on a CRCW PRAM.*

The main idea of our algorithm is quite simple. First we consider a graph as a rooted tree. Second we compute the number of descendants of each vertex in the tree using Euler tours [2,12]. Finally we get 2-chains for each vertex following the vertex parity check.

The whole algorithm can be outlined as follows:

Algorithm

Input: A connected graph $G = (V, E)$ with $|V| = n$ and $|E| = m$.

Output: A maximum 2-chain edge packing of the graph.

1. Find a spanning tree S of G.

2. Replace each non-tree edge $\{x, y\}$ of $G - S$ by $\{x, y'\}$, where y' is a new vertex. Add every replaced edge $\{x, y'\}$ to the spanning tree S. We denote the resulting tree by T.

3. Get the directed graph T' from T by replacing each edge $\{x, y\}$ by a pair of antiparallel edges (x, y) and (y, x). T' is an Euler directed graph since each vertex in T' has even degree.

4. Find an Euler circuit of T' using the circular succession assignments.

5. Determine the root of the tree T' and compute the number of descendants, denoted $nd(x)$, of each vertex x in T.

6. Get 2-chains in the following way:

   Let $y_1, y_2, ..., y_k$ be children of a vertex x and let z be the parent of x.

   Case 1. $nd(x)$ is odd.

   (a) Get the 2-chain $\{z, x\}, \{x, y_{i_0}\}$, where $nd(y_{i_0})$ is even.

   (b) Get the 2-chains using all edges in $\{\{x, y_i\} | nd(y_i) \text{ is even}\} - \{x, y_{i_0}\}$.

   Case 2. $nd(x)$ is even.

   Get the 2-chains using all edges in $\{\{x, y_i\} | nd(y_i) \text{ is even}\}$.

In order to evaluate the complexity of the algorithm, we use the following two facts and lemma.

Fact 2 [11,12]. *Let G be an undirected graph with n vertices and m edges. Then a spanning tree of G is computed in $O(\log n)$ time using $O(n + m)$ processors on a CRCW PRAM.*

Fact 3. *Let T be an undirected rooted tree with n vertices. Then the number of descendants of each vertex is computed in $O(\log n)$ time using $O(n)$ processors on a EREW PRAM.*

The above fact uses the Euler tour technique in [12]. We construct a circular list corresponding to an Euler tour by Step 3 and Step 4. To determine the root of the tree, we cut the Euler tour at an arbitrary edge, say (x, y). Then (x, y) is the first edge of the list and vertex x becomes the root of the tree. The resulting list is called the traversal list. We number the edges of the traversal list. Of two edges (x, y) and (y, x), we take the lower-numbered one. Let y be the parent of x, then $nd(x)$ is just the number of the lower-numbered edges from the first edge of the list to (x, y) minus the number of lower-numbered edges from the first edge of the list to (y, x).

The following lemma guarantees that Step 6 of the algorithm is correct.

Lemma 4. Let $y_1, y_2, ..., y_k$ be children of a vertex x. Then $nd(x)$ *is odd if and only if* $|\{y_i | nd(y_i) \text{ is even}\}|$ *is odd.*

Step 1 uses Fact 2. Step 2 takes $O(1)$ time and $O(m)$ processors. Step 3 takes $O(\log m)$ time and $O(m)$ processors. Step 4 takes $O(1)$ time and $O(n)$ processors using the circular succession assignments. Step 5 takes $O(\log m)$ time and $O(m)$ processors. To compute the number of descendants $nd(x)$, we need the preorder numbering of the rooted tree. Step 6 takes $O(1)$ time and $O(n + m)$ processors. Hence the whole algorithm requires $O(\log n)$ time using $O(n + m)$ processors.

3. Concluding remarks and open questions

We have presented an efficient parallel algorithm for finding a maximum 2-chain edge packing of an undirected graph. It takes $O(\log n)$ time and $O(n + m)$ processors on a CRCW PRAM.

However, the maximum k-chain edge packing problem does not seem to allow feasible algorithms for $k \geq 3$ since the problem is NP-complete [9]. Instead of a maximum size k-chain edge packing, we can deal with a *maximal* k-chain ($k \geq 3$) edge packing in the sense that no k-chain can be added. The problem may be not solved by the maximal independent set (MIS) technique [5,7] and remains open.

Also a maximum k-chain *vertex* packing problem has been studied in [9]. If $k = 1$, the problem is well-known as the maximum matching problem that is in RNC [4]. If $k \geq 2$, the problem becomes NP-complete [9]. By the same reason as the edge packing, we can consider a maximal k-chain vertex packing problem. The maximal matching problem is solved in $O((\log n)^3)$ time and $O(n+m)$ processors on a CRCW PRAM [3]. But if $k \geq 2$, no parallel algorithm is known. The technique used in Theorem 1 does not work for the maximum k-chain edge packing problem for $k \geq 3$.

Acknowledgement

The author would like to thank Satoru Miyano for helpful discussions.

References

[1] A. Aggarwal and R.J. Anderson, A random NC algorithm for depth first search, *Proc. 19th ACM STOC* (1987) 325-334.

[2] M. Atallah and U. Vishkin, Finding Euler tour in parallel, *J. Comput. System Sci.* **29** (1984) 330-337.

[3] A. Israeli and Y. Shiloach, An improved parallel algorithm for maximal matching, *Inf. Process. Lett.* **22** (1986) 57-60.

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

[5] R.M. Karp and A. Wigderson, A fast parallel algorithm for the maximal independent set problem, *Proc. 16th ACM STOC* (1984) 266-272.

[6] D. G. Kirkpatrick and P. Hell, On the Complexity of general graph factor problems, *SIAM J. Comput.* **12** (1983) 601-609.

[7] M. Luby, A simple parallel algorithm for the maximal independent set problem, *Proc. 17th ACM STOC* (1985) 1-10.

[8] S. Masuyama and T. Ibaraki, Chain packing in graphs, *Technical Report 87014, Dept. of Applied Mathematics and Physics, Faculty of Engineering, Kyoto University, Kyoto 606, Japan* (1987); also submitted to Discrete Applied Mathematics.

[9] Z. S. Masuyama, T. Ibaraki and H. Mine, Graph packing over a rooted tree, *Memoirs of Faculty of Engineering*, Kyoto University, **49** (1987) 206-215.

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

[11] Y. Shiloach and U. Vishkin, An $O(\log n)$ parallel connectivity algorithm, *J. Algorithms* **3** (1982) 57-63.

[12] R.E. Tarjan and U. Vishkin, Finding biconnected components and computing tree functions in logarithmic parallel time, *Proc. 25th IEEE FOCS* (1984) 12-20.

[13] U. Vishkin, Implementation of simultaneous memory address access in models that forbid it, *J. Algorithms* 4 (1983) 45-50.