

AN OPTIMAL PARALLEL CIRCLE-COVER ALGORITHM

Dilip SARKAR

Department of Mathematics and Computer Science, University of Miami, P.O. Box 249085, Coral Gables, FL 33124, U.S.A.

Ivan STOJMENOVIĆ

Computer Science Department, University of Ottawa, Ottawa, Canada K1N 9B4

Communicated by W.M. Turski

Received 21 October 1988

Revised 6 March 1989

Given a set of n circular arcs, we provide an optimal parallel algorithm (on the CREW PRAM model of computation) for finding a minimum number of circular arcs whose union covers the circle. The algorithm runs in $O(\log n)$ time with $O(n)$ processors and uses $O(n)$ space. This is a significant improvement over the recent algorithm by Bertossi that runs in $O(\log n)$ time with $O(n^2)$ processors and uses $O(n^2)$ space.

Keywords: Analysis of algorithms, circle-cover, combinatorial problems, computational geometry, parallel processing

1. Introduction

Consider a set S of n circular arcs on a circle. Let $S = \{A_1, \dots, A_n\}$ and let each (circular) arc A_i be an ordered pair (x_i, y_i) where x_i and y_i denote respectively the starting and ending points of the arc with y_i following x_i in the clockwise direction. The set $S' \subseteq S$ of arcs whose union equals the circle is called a *cover*. A cover with the smallest number of arcs in it is called a *minimum cover*. The problem that we are concerned with is to see if a cover exists for any given set S of arcs and, if it does, to find a minimum cover; this is referred to as the *minimum cover* problem of n circular arcs. The minimum circle cover problem has applications in surveillance systems (cf. [4]).

In [4] an $O(n \log n)$ sequential algorithm for solving the minimum circle cover problem is presented and it is shown that the algorithm is optimal under the linear decision computation model. Bertossi [2] presented a parallel algorithm for solving the problem using the CREW PRAM model

of computation (synchronous shared memory single instruction multiple data model in which concurrent reads are allowed, but no two processors should attempt to write simultaneously in the same memory location). His algorithm runs in $O(\log n)$ time but requires $O(n^2/\log n + qn)$ processors (which is, in fact, $O(n^2)$ since in the worst case $q = O(n)$) and uses $O(n^2)$ space. We improve the efficiency of the parallel algorithm by presenting an $O(\log n)$ time parallel algorithm which uses only $O(n)$ processors and $O(n)$ space. The solution is optimal since the time-processor product $O(n \log n)$ matches the optimal sequential solution [4] and the space used in $O(n)$. Moreover, conceptually, the new algorithm is even simpler than that of Bertossi.

2. The algorithm

Without loss of generality we assume that no arc is covering the whole circle. We also assume

that all arcs are distinct, i.e., $(x_i \neq x_j)$ and $(y_i \neq y_j)$ for $i \neq j$ (this can be checked by a parallel sorting procedure [3]).

A_i is said to be contained in A_j if the endpoints appear in (clockwise) order as x_j, x_i, y_i and y_j . The first step in our algorithm is to remove those arcs that are contained in other arcs, since they are not essential. We perform this step by mapping the problem to a maximal elements problem, which is defined as follows: Given a set $\{p_1, \dots, p_n\}$ of n points in a plane, a point p_i dominates another point p_j iff $x(p_i) \geq x(p_j)$ and $y(p_i) \geq y(p_j)$, where $x(p)$ and $y(p)$ denote coordinates of a point p . The problem is to determine points which are dominated by no other point. These points are called maximal elements.

We assume an arbitrary origin O such that $0 \leq x_i < 2\pi$, and $x_i < y_i$ for each arc $A_i = (x_i, y_i)$. Thus $y_i \geq 2\pi$ if O is an interior point of A_i .

Let y be the maximum value of y_j , and x the minimum value of x_i among all arcs A_j containing O as an interior point. These values can be easily found in $O(\log n)$ time on a CREW PRAM with $O(n)$ processors. All arcs A_i which do not contain O as an interior point and satisfy $y_i < (y - 2\pi)$ or $x_i > x$ are contained inside another arc (this can be checked in constant time on a CREW PRAM). In this way all pairs of arcs such that one contains O and the other does not are tested for possible containment. The remaining arcs are divided into two sets: one set consists of arcs having origin O as an interior point and the other set consists of arcs not having origin O as an interior point. Each of these sets is considered separately. Let A_i and A_j be two arcs from the same set. Then A_i is contained in A_j if $x_j < x_i$ and $y_i < y_j$, or $-x_i < -x_j$ and $y_i < y_j$. Hence, A_i is contained in A_j if and only if $(-x_i, y_i)$ is dominated by $(-x_j, y_j)$. Thus, if each arc A_j is represented as a point $(-x_j, y_j)$ in a plane, then arcs which are not contained inside other arcs are exactly those corresponding to the points which are maximal elements. The maximal element problem can be solved in $O(\log n)$ time with $O(n)$ processors on a CREW PRAM [1,5]. Note that the maximal element algorithm can be also applied in the sequential solution in [4] (no technique for removing arcs contained in other arcs is given in [4]).

From now on we assume that n arcs are given such that no arc is contained inside another. Because of the noncontainment property, no two arcs can have the same starting or ending points. Thus, if starting points are ordered as $x_1 < x_2 < \dots < x_n$, then ending points of the arcs must also be ordered in the same fashion, i.e. $y_1 < y_2 < \dots < y_n$. The ordered set is obtained as output either from the maximal element algorithm or from a parallel sorting procedure.

We call A_j the successor of A_i , denoted $SUCC(A_i)$, if all of the following three conditions are satisfied:

- (1) the intersection of A_i and A_j is nonempty,
- (2) $x_j > x_i$, and
- (3) there exists no starting point in the interior of the arc defined by the intersection of A_i and A_j .

Note that the ending point of A_i may coincide with the starting point of A_j .

In order to determine $SUCC(A_i)$ for each arc A_i , we merge two sorted sets x_1, \dots, x_n and y_1, \dots, y_n into one sorted set z_1, \dots, z_{2n} . Let $y_i = z_{j'}$. Then the difference $j = i' - i$ is equal to the number of elements from the set x_1, \dots, x_n that are no greater than y_i , i.e., j is the index of the element x_j such that $x_j \leq y_i \leq x_{j+1}$. It does not give a correct answer only for arcs A_i for which $y_i \geq 2\pi$ (i.e. containing O as an interior point) and $(y_i - 2\pi) > x_1$ is satisfied. These arcs are processed separately, by merging x_1, \dots, x_n with $y_1 - 2\pi, \dots, y_n - 2\pi$, and finding j such that $x_j \leq y_i - 2\pi < x_{j+1}$. The merging of two sorted sets (each having $O(n)$ elements) into one sorted set can be done in $O(\log n)$ time with $O(n)$ processors on a CREW PRAM by the mergesort algorithm of [3], or even faster, in $O(\log \log n)$ time (cf. [1]).

It is clear that an arc has no successor, (which is recognized by $SUCC(A_i) = A_i$ for some i in the former algorithm) if and only if there is no solution to the circle cover problem (i.e., input arcs do not cover the circle). This can be easily checked in $O(\log n)$ time with $O(n)$ processors. Thus, from now on we assume that all arcs have successors (and hence, a circle cover exists).

In [4], a greedy algorithm is proposed for finding the minimal circle cover: starting with A_j proceed in the clockwise direction by forming an

array $SUCC_i(A_j)$, $0 \leq i \leq m_j$, where

$$SUCC_0(A_j) = A_j,$$

$$SUCC_i(A_j) = SUCC(SUCC_{i-1}(A_j)),$$

$$i = 1, 2, \dots, m_j$$

until the circle is covered (i.e., $SUCC_{m_j}(A_j)$ is the first element in the array which has positive index and contains the starting point x_j of A_j as an interior point). By $C_i(A_j)$ we denote the (contiguous) part of the circle covered by arcs $SUCC_l(A_j)$ for all l , $0 \leq l \leq i$. It is clear that

$$C_i(A_j) = C_{i-1}(A_j) \cup SUCC_i(A_j)$$

where \cup is used for set union of two arcs. From the definition it follows that the length of $C_{m_j}(A_j)$ is 2π , while for $i < m_j$ the length of $C_i(A_j)$ is less than 2π . Thus, m_j is the size of the minimum circle cover starting with arc A_j .

Our parallel algorithm is based on the following fact which is proved in [4]: If we recognize m_1 (the size of a minimum cover starting with A_1), then the size of an optimal solution is either m_1 or $m_1 - 1$. We will proceed in two steps. In the first step all n processors will "help" to find m_1 in $O(\log n)$ time and with $O(n)$ space. In the second step each processor i will check whether the size of the minimum cover starting with A_i "happened" to be $m_1 - 1$, since this is the only possible improvement in the size of optimal solution.

The first step is executed in two phases:

```

for each processor  $j = 1, \dots, n$  do in parallel
begin
 $l_j := 0$ ;  $SF(A_j) := A_j$ ;  $CF(A_j) := A_j$ ;
 $SP(A_j) := SUCC(A_j)$ ;  $CP(A_j) := A_j \cup$ 
                                $SUCC(A_j)$ ;
while  $CP(A_j)$  does not contain  $x_1$  do
begin
 $l_j := l_j + 1$ ;
 $SF(A_j) := SP(A_j)$ ;
 $CF(A_j) := CP(A_j)$ ;
 $SP(A_j) := SF(SF(A_j))$ ;
/*  $SP(A_j) := SUCC_{2^{l_j}}(A_j) */$ 
 $CP(A_j) := CF(A_j) \cup CF(SF(A_j))$ ;
/*  $CP(A_j) := C_{2^{l_j}}(A_j) */$ 
end
end.

```

The while loop terminates when x_1 becomes an interior point or endpoint of the arc $CP(A_j)$. However, processor 1 has the additional restriction $l_1 > 0$.

In the presented first phase each processor j finds the smallest nonnegative index l_j such that $C_{2^{l_j}}(A_j)$ contains x_1 (the starting point of A_1).

Now processor 1 finds m_1 in the second phase:

```

 $m_1 := 2$ ;
 $j := 1$ ;
while  $l_j \neq 0$  do
begin
 $m_1 := m_1 + 2^{l_j - 1}$ ;
 $j := k$  where  $A_k = SF(A_j)$ 
/*  $= SUCC_{2^{l_j - 1}}(A_j) */$ ;
end.

```

The second step starts by representing $m_1 - 2$ as a binary number $b_s b_{s-1} \dots b_1 b_0$ with $b_s = 1$, $b_i \in \{0, 1\}$ for $0 \leq i \leq s - 1$, i.e.,

$$m_1 - 2 = 2^s b_s + 2^{s-1} b_{s-1} + \dots + 2b_1 + b_0.$$

Each processor j checks whether there is a cover of size $m_1 - 1$ starting at A_j by "jumping" for 2^i whenever $b_i = 1$:

```

for each processor  $j = 1, \dots, n$  do in parallel
begin
 $SP(A_j) := SUCC(A_j)$ ;  $CP(A_j) := A_j$ ;
 $ST(A_j) := A_j$ ;  $CT(A_j) := A_j$ ;
for  $l = 0$  to  $s$  do
begin
if  $b_l = 1$ 
then  $CT(A_j) := CT(A_j) \cup CP(SP(A_j))$ 
 $CP(A_j) := CP(A_j) \cup CP(SP(A_j))$ ;
/*  $CP(A_j) := C_{2^{l_j}}(A_j) */$ 
 $SP(A_j) := SP(SP(A_j))$ ;
/*  $SP(A_j) := SUCC_{2^l}(A_j) */$ 
end
end.

```

Each processor will check whether the length of $CT(A_j)$ is equal to 2π (i.e., covers the whole circle). If there is one that covers the whole circle, it is a better and an optimal solution; otherwise the already found solution starting with A_1 is indeed an optimal solution.

References

- [1] M.J. Atallah, R. Cole and M.T. Goodrich, Cascading divide-and-conquer: A technique for designing parallel algorithms. In: *Proc. IEEE Symp. on Foundations of Computer Science* (1987) 151–160.
- [2] A.A. Bertossi, Parallel circle-cover algorithms, *Inform. Process. Lett.* **27** (1988) 133–139.
- [3] R. Cole, Parallel merge sort, In: *Proc. IEEE Symp. on Foundations of Computer Science* (1986) 511–516.
- [4] C.C. Lee and D.T. Lee, On a circle-cover minimization problem, *Inform. Process. Lett.* **18** (1984) 109–115.
- [5] I. Stojmenović and M. Miyakawa, An optimal parallel algorithm for solving maximal elements problem in the plane, *Parallel Comput.* **7** (1988) 249–251.