A Fast Randomized Parallel Algorithm for Finding Simple Cycles in Planar Graphs

Carsten F. Dorgerloh*
Institut für Informatik der Universität Bonn
Römerstr. 164, D-53117 Bonn
Germany

Abstract

We show that if a planar graph has a simple cycle of length k, where k is a fixed integer, such a cycle may be computed in $\mathcal{O}(\log n \log^* n)$ expected time by a randomized EREW-PRAM with $\mathcal{O}(n)$ processors.

1 Introduction

It is well known that finding the longest cycle in a graph is a hard problem, since finding a Hamiltonian cycle is NP-complete [GJ 79]. Hence finding a cycle of lenght k, for an arbitrary k, is NP-complete. This remains true if G is planar and under other restrictions. However, we are concerned only with the problem of finding simple cycles of a given length. Alon, Yuster and Zwick [AYZ 95] presented a randomized sequential algorithm for this task which uses $\mathcal{O}(n)$ expected time. Our randomized parallel algorithm for that problem, while using ideas from [JM 91], is largely a parallel implementation of the algorithm of [AYZ 95], with results of [CE 91] used crucially in the parallelization. We achieve an expected running time of $\mathcal{O}(\log n \log^* n)$ on a randomized EREW-PRAM with $\mathcal{O}(n)$ processors.

2 Notations and definitions

The terminology used in this paper follows that of Even [Ev 79]. Let $G = (V, E)$ be a graph. For each vertex v, $N(v)$ denotes the set of neighbours of v. A *simple cycle* of length k is a sequence $v_0, v_1, ..v_{k-1}$ of vertices with $v_i \neq v_j$ for $i \neq j$ and $(v_i, v_{(i+1) \bmod k}) \in E$ for $0 \leq i < k$. A *contraction* of an edge $(v, w) \in E$ is the following operation: Delete the edge (v, w), identify the vertices v and w as a new vertex vw adjacent to exactly those vertices that were adjacent to either v or w (or both) in the original graph. In practice, one of the vertices (v or w), called the *representative*, plays the role of the new vertex.
The computational model used in this paper is the randomized EREW-PRAM. This model is a synchronized parallel computation model for which simultaneous access to any memory location by different processors is forbidden. Furthermore, each processor

*email: carsten@cs.uni-bonn.de

has access to a random number generator which returns random numbers of $\log n$ bits in constant time. More details of the PRAM models can be found in the survey by Karp and Ramachandran [KR 88].

3 The Algorithm

Before we explain the technical details of the algorithm, we give our special view on the data structures which allows us to implement the algorithms efficiently.
We assume that the vertices of G are represented by positive numbers and that G is presented to the algorithm in the form of a set of *edge-lists* L: The graph is represented as an array of $|V|$ vertices, and each vertex u is equipped with a pointer to its list $L(u)$, a doubly-linked list that contains exactly one entry for each edge that connect u to another vertex in the graph. For implementation reasons it is convenient to assume that there is a fake edge at the end of each edge list. Additionally, each edge (u, v) appearing in the edge-list of u has a pointer to its *twin* edge (v, u) appearing in the edge-list of v. In the literature, pointers of this type are often called *cross links.*

3.1 Parallel Graph Contraction

Let $G = (V, E)$ be the input graph with $n = |V|$ vertices and $m = |E|$ edges. Since G is planar, from Euler's formula [Ev 79] follows that $m < 3n$. We will assume that there is one processor assigned to each vertex $v \in V$ and one processor to each edge $(u, v) \in E$. In the course of the algorithm we will need to contract all edges incident to a given vertex u wich will be the representative. Some of the algorithmic techniques used in the proof of the following lemma were introduced by Johnson and Metaxas [JM 91].

Lemma 1 *Let $G = (V, E)$ be a graph and $S \subset V$ be a set of vertices such that for each pair $u, v \in S$ with $u \neq v$ $N(u) \cap N(v) = \emptyset$ and $(u, v) \notin E$. Then the edges $e \in D := \{(u, v) \in E | u \in S$ and $v \in N(u)\}$ may be contracted by an EREW-PRAM in $\mathcal{O}(\log n)$ time using $\mathcal{O}(n + m)$ processors.*

PROOF: Let the *first* and *last* functions defined on $L(v)$ give the first and last edges, respectively, appearing in $L(v)$. The contraction is done by having each edge $(u, v) \in D$ first execute the following:

for each $(u, v) \in D$ **pardo**
    $next(last(L(v)))$:= $next(u, v)$
    $next(u, v)$:= $first(L(v))$
    for each (v, w) **pardo**
        rename the endpoint v to u
    end
end
for each (x_1, y_1) **pardo**
    (y_2, x_2) := $twin(x_1, y_1)$
    rename y_1 to y_2
end

The representative of each contraction (u, v) is always the vertex $u \in S$. In lines (02)-(03) the edge-list $L(v)$ of v is plugged into u's edge-list by redirecting a couple of pointers. The exact place into which $L(v)$ is plugged is after the edge (u, v) and before $next(u, v)$. The endpoints of the edges are renamed in lines (04)-(10). This is done by setting the first endpoint of each edge in $L(u)$, as well as the second endpoint of the corresponding twin edge, to u and the second to the first endpoint of the twin edge. All this can be done in constant time without memory access conflicts.
Now there may be multiple edges in $L(u)$ as well as loops. Loops are nullified and multiple edges are identified. One of them is kept while the rest are nullified as redundant. This is done as follows. Run standard list ranking on each edge-list to find the distance of each edge from the end of its list. Copy each edge-list in an array using the results of the list ranking as index. Sort the array using the parallel merge sort of Cole [Co 88] and use the results to form a sorted linked list. Then, blocks of redundant edges are nullified as follows.

```
for each (u, w) ∈ L(u) pardo
    if next(u, w) = (u, w) then
        nullify(u, w)
    end
end
```

Thus, the last edge in a block of consecutive edges having identical names is kept. Before we remove the nullified edges, we have to recompute the twin function. Each edge (v, w) wich is not nullified passes its address $prev(next(v, w))$ to a field $addr(v, w)$. From there, the edge (w, v) reads it. Now we can remove all nullified edges using pointer doubling. Each of these steps can be done in $\mathcal{O}(\log n)$ parallel time on an EREW-PRAM using $\mathcal{O}(n + m)$ processors. ∎

3.2 Planar Orientations

Given an undirected graph $G = (V, E)$, an orientation ω of G is a function which replaces each undirected edge $(u, v) \in E$ by one directed edge (u, v) or (v, u). By $deg_\omega(v)$ we will note the out-degree of v, under this orientation ω. We say that ω is *d-bounded* if for each vertex $v \in V$ we have $deg_\omega(v) \leq d$.
The following lemma can be found in [CE 91].

Lemma 2 *Let G be a planar graph. A 6-bounded acyclic orientation ω of G may be computed on an EREW-PRAM in time $\mathcal{O}(\log n \log^* n)$, using $\mathcal{O}(n/(\log n \log^* n))$ processors.*

Remark 3 *Using lemma 2 we can represent a planar graph by a compacted adjacency matrix A such that $A[v, 1, \ldots, 6]$ contains the set of neighbours of v under the orientation ω. This representation can be computed within the same resource bounds. Therefore, only minor changes are needed in the proof of lemma 2. We omit the details.*

3.3 Finding triangles

Lemma 4 *There is a EREW-PRAM algorithm for finding a triangle in a planar graph G, if one exists, which runs in time $\mathcal{O}(\log n \log^* n)$ with $\mathcal{O}(n)$ processors.*

Proof: The algorithm first constructs a compacted adjacency matrix A of G. By remark 3 to lemma 2 (Section 3.2) this takes parallel time $\mathcal{O}(\log n \log^* n)$.
For simplicity reasons, we describe the next step by a Priority CRCW-PRAM algorithm which runs in $\mathcal{O}(1)$ time using $\mathcal{O}(n)$ processors. Since any algorithm that works on this model can be simulated by an EREW-PRAM with the same number of processors and with the parallel time increased by only a factor of $\mathcal{O}(\log p)$, where p is the number of processors, one can design an $\mathcal{O}(\log n)$ time EREW-PRAM algorithm for this step (see e.g.[KR 88]). The Priority CRCW-PRAM algorithm which finds a triangle, if one exists, assigns one processor v to each $A[v, 1 \ldots 6]$ and proceeds as follows:

```
(01) for each v ∈ V pardo
(02)     for each i with 1 ≤ i ≤ 6 do
(03)         w := A[v, i]
(04)         for each j with 1 ≤ j ≤ 6 do
(05)             u := A[w, j]
(06)             for each l with 1 ≤ j ≤ 6 do
(07)                 if A[u, l] = v then
(08)                     triangle := (v, w, u)
(09)                 end
(10)             end
(11)         end
(12)     end
(13)     for each (i, j) with 1 ≤ i < j ≤ 6 do
(14)         for each l with 1 ≤ l ≤ 6 do
(15)             if A[A[v, i], l] = A[v, j] or
(16)                 A[A[v, j], l] = A[v, i] then
(17)                     triangle := (v, A[v, i], A[v, j])
(18)             end
(19)         end
(20)     end
(21) end
```

Let T be a triangle in G. Then either the edges of T form a directed cycle in A, or there is one vertex in T which has two descendants in A which are connected by an edge. In the first case (lines (02)-(12)) T is detected by 3 processors, in the latter (lines (13)-(20)) by 1 processor. The need of constant time is obvious. ∎

3.4 Finding simple cycles

Before we state our main theorem, we need the following definition. We say a cycle of length k in G is *well-coloured* if the vertices on it are consecutively coloured by $1, \ldots, k$.

Theorem 5 *Let $k > 3$ be a fixed integer. Given a planar graph G on n vertices, a simple cycle of size k in G, if one exists, may be computed in $\mathcal{O}(\log n \log^* n)$ expected time by a randomized EREW-PRAM using $\mathcal{O}(n)$ processors.*

Proof: We use the following algorithm:

```
(01) G_k = (V_k, E_k) := G
(02) for each u ∈ V_k pardo
(03)     choose a random colour c(u) ∈_R {1,...,k}
(04)     propagate c(u) to all edges (u,w)
(05) end
(06) delete all edges (u,v) where u,v are non-consecutively coloured
(07) i := k
(08) while i > 3 do
(09)     construct a compacted adjacency matrix A of G_i
(10)     the master processor guesses an direction δ ∈_R {0,1}
             and an index j ∈_R {0,...,6}
(11)     propagate δ and j to all A[w] and to all edges
(12)     for each A[w] pardo
(13)         if (δ = 0 and c(w) = i − 1 or δ = 1 and c(w) = i) and
                 A[w,j] is defined then
(14)             H[A[w,j],w] := 1;  H[w,A[w,j]] := 1
(15)         end
(16)     end
(17)     for each edge (u,v) with c(u),c(v) ∈ {i − 1, i} pardo
(18)         if H[u,v] ≠ 1 then delete (u,v)
(19)     end
(20)     for each edge (u,v) with c(u),c(v) ∈ {i − 1, i} pardo
(21)         if δ = 0 then
(22)             contract(u,v), vertex with colour k is the representative
(23)         else if δ = 1 then
(24)             contract(u,v), vertex with colour k − 1 is the representative
(25)         end
(26)     end
(27)     Let G_{i−1} = (V_{i−1}, E_{i−1}) be the graph obtained by recolouring
             each endpoint of an edge and each vertex with colour i by i − 1
(28)     i := i − 1
(29) end
(30) if G_3 contains at least one triangle then
(31)     Let C be one arbitrary triangle
(32)     for i = 3 to k do
(33)         reconstruct G_{i+1} from G_i and
(34)         extend thereby C by one vertex (and by one edge)
(35)     end
(36) end
```

Let G be the given planar graph that contains at least one cycle C of length k.
In lines (02)-(06) each processor u chooses a random colour from $\{1, \ldots, k\}$ and propagates it to each edge in its edge list. In addition, edges between vertices of non-consecutive colours are deleted, since they can not be a part of a well-coloured cycle. The colour of each vertex u is propagated to each edge in $L(u)$ as follows. Run list ranking to determine the length $l(L(u))$ of the edge-list of u. Now, $c(u)$ is copied $l(L(u))$ times in $\log l(L(u))$

time in an array. By using the unique index of the list ranking, each (u,v) in $L(u)$ reads the colour of u from the array. Finally, the colour of the second endpoint of each edge (u,v) is determined by reading the colour of the first endpoint of the twin edge (v,u). Edge deletions are made by first nullifying all edges to be deleted and then removing them by means of pointer jumping as in lemma 1.

There are k^k possible colourings of C. $2k$ of them are well-coloured. Thus the probability that C is well-coloured is $2/k^{k-1}$. Assume there exists a well-coloured cycle of length k in G_k.

The graph is reduced in $k-3$ phases (lines (08)-(29)). Consider the first phase where $i=k$. The well-coloured cycle of length k in G contains two vertices u and v with $c(u)=k-1$ and $c(v)=k$, such that either $v \in A[u,1,\ldots 6]$ or $u \in A[v,1,\ldots,6]$. The master processor guesses the orientation δ and the index j of this edge in the compacted adjacency matrix A of G_k. We assign one processor to each $A[w]$ and use them to propagate δ and j to each $A[w]$. The propagation of those two values to all edges is similar to the propagation of the colour described above.

Suppose from now on that the guess is $\delta = 0$ which means that the guess is $A[u,j]=v$ ($\delta = 1$ analogue). Then all edges between vertices coloured $k-1$ and vertices coloured k are deleted from G_k, except the edges (x,y), (y,x) with $c(y)=k-1$ and $A[y,j]=x$ (lines (12)-(19)). In order to avoid read conflicts, this is done as follows. Each processor assigned to $A[y]$ writes a 1 in the fields $H[x,y]$ and $H[y,x]$ if $A[y,j]=x$ and $c(y)=k-1$. Now all edges (x,y) in $L(x)$ with $c(x), c(y) \in \{k-1,k\}$ and $H(x,y)=0$ are nullified and then removed as described above.

The probability that G_k still contains a well-coloured cycle of length k is at least $1/12$. Note that each vertex having colour $k-1$ now has at most one neighbour coloured k. Thus we can apply lemma 1 where S is the set of vertices in G_k coloured k. This is done in lines (20)-(26). Before the next iteration of the while loop, each endpoint of an edge and each vertex with colour k will be assigned to the colour $k-1$.

There are $k-3$ iterations of the while loop. Thus the probability that G_3 contains a cycle of length 3 is at least $1/12^{k-3}$. Such a triangle can be found by using the algorithm of lemma 4.

Lines (04), (11) and (20)-(26) take $\mathcal{O}(\log n)$ time, whereas line (09) can be executed in time $\mathcal{O}(\log n \log^* n)$ by remark 3 (see Section 3.2). Hence lines (01)-(29) take $\mathcal{O}(\log n \log^* n)$ time. By running the process backwards, the reconstruction in lines (29)-(36) may be done within the same time bound. The algorithm finds C with a probability of at least $2/(12^{k-3}k^{k-1})$. By rerunning the algorithm, in case of failure, we obtain an $\mathcal{O}((12^{k-3}k^{k-1}) \log n \log^* n)$ expected time algorithm. ∎

4 Acknowledgements

I like to thank Marek Karpinski.

References

[AYZ 95] Alon, N., Yuster, R., Zwick, U., *Color-coding*, Proc. 42^{nd} Journal of the ACM (1995), pp. 844–856.

[CE 91] Chrobak, M., Eppstein, D., *Planar orientations with low out-degree and compaction of adjacency matrices*, Theoretical Computer Science **86** (1991), pp. 243–266.

[Co 88] Cole, R., *Parallel Merge Sort*, SIAM Journal on Computing **17**(4) (1988), pp. 770–785.

[Ev 79] Even, S., *Graph Algorithms*, Computer Science Press, 1979.

[GJ 79] Garey, M. R., Johnson, D. S., *Computers and Intractability*, W. H. Freeman and Company, 1979.

[JM 91] Johnson, D. B., Metaxas, P., *Connected Components in $O(\log^{3/2}|V|)$ Parallel Time for the CREW PRAM*, Proc. 32^{nd} IEEE FOCS (1991), pp. 688–697.

[KR 88] Karp, R. M., Ramachandran, V., *A Survey of Parallel Algorithms for Shared-Memory Machines*, Research Report UCB/CSD 88/408, University of California, Berkeley, 1988.