Once again: Finding simple cycles in graphs

Carsten Dorgerloh[§] Jürgen Wirtgen[¶]

January 21, 1997

Abstract

We present a randomized algorithm that computes a simple cycle of length k in general graphs, where k is a fixed integer, in $\mathcal{O}(\max\{m, n\log n\})$ expected time. This algorithm can be derandomized with only a small loss in efficiency, yielding a deterministic algorithm for this task which runs in $\mathcal{O}(\max\{m\log n, n\log n\})$ worst-case time. We show that the randomized algorithm may be parallelized. These algorithms improve upon previous results of many authors. Furthermore, we answer the question of [AYZ 94], whether deciding if a given graph contains a triangle is as difficult as boolean multiplication of two n by n matrices, in the negative.

[§]Institut für Informatik V, Universität Bonn, Römerstr. 164, D-53117 Bonn, Germany, email: carsten@cs.uni-bonn.de

[¶]Institut für Informatik V, Universität Bonn, Römerstr. 164, D-53117 Bonn, Germany, email: wirtgen@cs.uni-bonn.de

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 simple 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. The problem of finding given length simple cycles is one of the basic and natural algorithmic graph problems [Le 90] and was considered by many researchers, e.g. by [Mo 85], [RL 85], [Ri 86], [YZ 94], [Do 96], [DW 96]. Alon, Yuster and Zwick [AYZ 95] presented algorithms for this task which runs either in $\mathcal{O}(M(n))$ expected time or $\mathcal{O}(M(n)\log n)$ worst-case time, where $M(n) < n^{2.376}$ is the complexity of matrix multiplication. Our algorithms for that problem, while using ideas from [AYZ 95], takes advantage of techniques from [GK 87]. To be more precise, we combine the color-coding method of [AYZ 95] with the idea of Grigoriev and Karpinski [GK 87] to utilize the nice properties of prime numbers. [GK 87] make use of this properties to solve some matching problems efficiently in parallel (see also [KR 97]). We improve an $\mathcal{O}(nm)$ worst-case time bound of [Mo 85] (in many cases), and upon the results of [AYZ 95]. Furthermore, our results generalizes a result of [DW 96]. We achieve a randomized algorithm which runs in $\mathcal{O}(\max\{m, n\log n\})$ expected time and a deterministic algorithm which runs in $\mathcal{O}(\max\{m\log n, n\log n\})$ worst-case time. We show that our randomized method can be parallelized, yielding an efficient EREW-PRAM algorithm. The paper is organized as follows. Section 2 contains some definitions and notations used throughout the paper. In section 3 we present the sequential algorithms for finding simple cycles in graphs and in section 4 we show that there exists an efficient parallel algorithm for this problem.

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 $0 \leq i < k$.
The computational model we use is the unit-cost randomized RAM and its corresponding parallel version, the randomized EREW-PRAM. The latter model is a synchronized parallel computation model for which simultaneous access to any memory location by different processors is forbidden. Furthermore, each processor 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 Finding simple cycles

Before we state our theorems, we need the following definition. We say a cycle of length k in G is *well-colored* if the vertices on it are consecutively colored by $1, \ldots, k$.
Note that a well-colored cycle is always simple.

Theorem 1 *Let $k \geq 3$ be a fixed integer. Given a graph G on n vertices and m edges, a simple cycle of length k in G, if one exists, may be computed in $O(\max\{m, n \log n\})$ expected time.*

PROOF: We use the following algorithm:

```
(01) generate consecutively prime numbers $p_1, \ldots, p_n$
(02) for each vertex $u \in V$ do
(03)     assign one definite prime number $p(u)$ from $p_1, \ldots, p_n$ to $u$
(04)     $q(u) := p(u)$
(05)     choose a random color $c(u) \in_R \{1, \ldots, k\}$
(06) end
(07) delete all edges $(u, v)$ where $u, v$ are non-consecutively colored
(08) for each edge $(u, w) \in E$ do
(09)     $c_1(u, w) = c(u)$
(10)     $c_2(u, w) = c(w)$
(11) end
```

Let G be the given graph that contains at least one cycle C of length k.
In lines (01)-(06) we generate prime numbers $p_1, \ldots, p_n$, assign one of those numbers to each vertex u and initialize $q(u)$. In addition, each vertex u chooses a random color from $\{1, \ldots, k\}$. In lines (07)-(11) we delete edges between vertices of non-consecutive colors, since they can not be a part of a well-colored cycle. Furthermore, the color of each vertex u is propagated to each edge where u is one of the endpoints.
There are k^k possible colorings of C. $2k$ of them are well-colored. Thus the probability that C is well-colored is $2/k^{k-1}$. Assume there exists a well-colored cycle of length k in G.

```
(12) for $i := 2$ to $k$ do
(13)     for each edge $(u, w)$ do
(14)         if $c_1(u, w) = i$ and $c_2(u, w) = i - 1$ then
(15)             $q(u) := q(u) \cdot q(w)$
(16) for each vertex $v$ with $c(v) = 1$ do
```

```
(17)      $q(v) := 1$
(18) for each edge $(u,v)$ do
(19)      if $c_1(u,w) = 1$ and $c_2(u,w) = k$ then
(20)           $q(u) := q(u) \cdot q(w)$
```

The values of the array q are updated in $k-1$ phases (lines (12)-(15)). Consider the first phase where $i = 2$. The well-colored cycle of length k in G contains two vertices x and y with $c(x) = 1$ and $c(y) = 2$. For each edge (u,w) having the first endpoint colored 2 and the second colored 1 we multiply $q(u)$ by $q(w)$. We proceed in the next phases by performing the same computations with edges having the first endpoint colored 2 and the second colored 3 etc., until $i = k$. In lines (16)-(20) we set the values of the array q corresponding to vertices with color 1. Let v be a vertex with color 1. Then $q(v)$ is the product of the q-values of its neighbours having color k.

```
(21) if $\exists v \in V$ such that $c(v) = 1$ and $p(v)|q(v)$ then
(22)    extend $C$ by $v$
(23)    delete all vertices $u$ with $c(u) = 1$ and $p(u) \neq p(v)$
              and all edges which are incident to such vertices
(24)    for $i := k$ downto 2 do
(25)       $delflag := false$
(26)       for each edge $(u,w)$ do
(27)          if $c_1(u,w) = i$ and $c_2(u,w) = 1 + i$ mod $k$ then
(28)             if $p(v)|q(u)$ then
(29)                if not $delflag$ then
(30)                   extend $C$ by $u$
(31)                   $delflag := true$
(32)                end else if $delflag$ then
(33)                   delete $u$ and all edges which are incident to $u$
(34)    end
(35) end
```

Note that each vertex v with color 1 such that $p(v)$ divides $q(v)$ now must be a part of at least one well-colored cycle. We pick an arbitrary vertex with such properties, add it to the cycle C, delete all other vertices with color 1 together with all incident edges and determine the other vertices of C by running thru the colors in reverse order (lines (21)-(35)). Let v be the picked vertex and consider the phase where $i = k$. We check each edge (u,w) whether its first endpoint is colored by k and the second by 1 and if $p(v)|q(u)$. Let (x,y) be an edge with this properties. Then we know that $y = v$ and that x is as well as v a part of a well-colored cycle. Thus we add x to C and delete all other vertices with color k together with all incident edges. The correctness of the other phases follows by induction.
The first n prime numbers may be computed by the sieve of Pritchard [Pr 87] in $\mathcal{O}(n \log n)$ time. It is easy to see that lines (02)-(20) take $\mathcal{O}(m)$ time. By taking care in the implementation of the for-loop in lines (26)-(33) each edge (u,w) is processed at most

once. Hence lines (21)-(35) take $\mathcal{O}(m)$ time. The algorithm finds C with a probability of at least $2/k^{k-1}$. By rerunning the algorithm, in case of failure, we obtain an $\mathcal{O}(\max\{k^{k-1}m, n\log n\})$ expected time algorithm. ■

The following lemma is implicitly in [AYZ 95]. Using this lemma, the above algorithm can be derandomized with only a small loss of efficiency.

Lemma 2 *There is an explicit construction of a list of $k^{O(k)}\log n$ colorings that has the property that every sequence $v_1, \ldots v_k$ of k vertices from V is consecutively colored by $1, \ldots, k$ in at least one coloring of the list. Such a list may be computed in $\mathcal{O}(n\log n)$ time.*

Thus, the algorithm of Theorem 1 can be derandomized by exhausting a list with the properties of Lemma 2.

Theorem 3 *There exists a deterministic algorithm that decides whether a given graph contains a simple cycle of length k, and finds one, if one exist, in $\mathcal{O}(\max\{m\log n, n\log n\})$ worst-case time.*

4 Finding simple cycles in parallel

As mentioned in the introduction, the algorithm of Theorem 1 can be parallelized and we obtain the following result whose proof is omitted from this paper. This generalizes a previous result for planar graphs of [DW 96]. Some techniques which are useful to to obtain this result can be found in [Do 96].

Theorem 4 *Let $k \geq 3$ be a fixed integer. Given a graph G on n vertices, a simple cycle of length k in G, if one exists, may be computed in $\mathcal{O}(\log n)$ expected time by a randomized EREW-PRAM using $\mathcal{O}(n+m)$ processors.*

5 Open problems

As mentioned before, we answer by our results the question of [AYZ 94], whether solving the decision Problem "Does a given graph contain a triangle?" is as difficult as boolean multiplication of two n by n matrices, in the negative in the sense that this decision Problem may be solved substantially faster than the matrix multiplication by the currently best known algorithm with a running time of $\approx n^{2.376}$ [CW 87]. While answering this question, several other problems remain open. They are listed listed below.

- Can the $\log n$ factor appearing in the derandomization be omitted (Theorem 3)?
- The algorithm proposed in Theorem 4 finds a simple cycle of length k in a graph, if one exists, and is efficient in the sense that its work (processor-time product) is within a polylogarithmic factor of optimal speed-up. Can optimal speed-up be achieved? Can the algorithm be derandomized without processor penalty and with the parallel time increased by only a small factor?

References

[AYZ 94] Alon, N., Yuster, R., Zwick, U., *Finding and counting given length cyles*, Proc. 2nd European Symposium on Algorithms (1994), pp. 354–364.

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

[CW 87] Coppersmith, D., Winograd, S., *Matrix multiplication via arithmetic progressions*, Proc. 19th ACM STOC (1987), pp. 1–6.

[Do 96] Dorgerloh, C. F., *A Fast Randomized Parallel Algorithm for Finding Simple Cycles in Planar Graphs*, Research Report 85150-CS, Institut für Informatik der Universität Bonn, 1996.

[DW 96] Dorgerloh, C., Wirtgen, J., *A note on improving the running time of a class of parallel algorithms using randomization*, Research Report 85159-CS, Institut für Informatik der Universität Bonn, 1996.

[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.

[GK 87] Grigoriev, D. Y., Karpinski, M., *The Matching Problem for Bipartite Graphs with Polynomially Bounded Permanents is in NC*, Proc. 28th IEEE FOCS (1987), pp. 166–172.

[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.

[KR 97] Karpinski, M., Rytter, W., *Fast Parallel Algorithms for Graph Matching Problems*, Oxford University Press, 1997.

[Le 90] Leeuwen, J. v., *Graph Algorithms. Handbook of Theoretical Computer Science, Volume A, Algorithms and Comlexity*, chapter 10, pp. 525–631, Elsevier and The MIT Press, 1990.

[Mo 85] Monien, B., *How to find long paths efficiently*, Annals of Discrete Mathematics **25** (1985), pp. 239–254.

[Pr 87] Pritchard, P., *Linear prime-number sieves: A familiy tree*, Science of Computer Programming **9** (1987), pp. 17–35.

[Ri 86] Richards, D., *Finding short cycles in a planar graph using seperators*, Journal of Algorithms **7** (1986), pp. 382–394.

[RL 85] Richards, D., Liestman, A. L., *Finding cycles of a given length*, Annals of Discrete Mathematics **27** (1985), pp. 249–256.

[YZ 94] Yuster, R., Zwick, U., *Finding even cycles even faster*, Proc. 21st International Colloquium on Automata, Languages and Programming (1994), pp. 532–543.