SIAM J. DISCRETE MATH.
Vol. 10, No. 2, pp. 209–222, May 1997

005

FINDING EVEN CYCLES EVEN FASTER*

RAPHAEL YUSTER† AND URI ZWICK†

Abstract. We describe efficient algorithms for finding even cycles in undirected graphs. Our main results are the following: (i) For every $k \geq 2$, there is an $O(V^2)$ time algorithm that decides whether an undirected graph $G = (V, E)$ contains a simple cycle of length $2k$, and finds one if it does. (ii) There is an $O(V^2)$ time algorithm that finds a shortest even cycle in an undirected graph $G = (V, E)$.

Key words. graph algorithms, cycles

AMS subject classifications. 05C85, 05C38, 68R10

PII. S0895480194274133

1. Introduction.

Throughout this work, the term *cycle* refers to a simple closed walk and the term *path* refers to a simple nonclosed walk. An even (odd) cycle is a cycle whose length is even (odd). An even (odd) path is a path whose length is even (odd).

The problems of finding cycles of a given length and of finding a shortest even and a shortest odd cycle in undirected and directed graphs are among the most basic and natural algorithmic graph problems. These problems have been considered by many researchers; see [10] for a survey.

In this work we consider (almost exclusively) the undirected versions of these problems. The directed versions of some of them are believed to be much harder. The problem "does a given directed graph $G = (V, E)$ contain a directed cycle of an even length?", for example, is not known to be in P, nor is it known to be NP-complete (see [9]). Though we do not shed any new light on the directed versions of the problems, we obtain surprisingly fast algorithms for some of the undirected versions.

Monien [7] presented an $O(VE)$ algorithm for finding all pairs of vertices that are connected by paths of length $k-1$, where $k \geq 2$ is a fixed integer. (Note that if k is part of the input, the problem is NP-hard.) A simple consequence of his algorithm is an $O(VE)$ algorithm for finding a cycle of length k, if one exists. In [1], an $O(M(V) \log V)$ algorithm is obtained for the same problem, where $M(n) = O(n^{2.376})$ is the complexity of Boolean matrix multiplication. This algorithm is more efficient when G is dense. Both algorithms work on directed as well as undirected graphs. In this work we show that if k is even and if the graph is undirected, then both these bounds can be improved. We obtain an $O(V^2)$ algorithm for finding cycles of a given even length in undirected graphs. An $O(V^2)$ algorithm for finding quadrilaterals (cycles of length four) is part of the folklore (cf. [8]) but all other cases are new. To obtain this $O(V^2)$ algorithm we utilize a combinatorial theorem of Bondy and Simonovits [4] that states,

*Received by the editors August 10, 1994; accepted for publication (in revised form) April 9, 1996. A preliminary version of this paper appeared in the Proceedings of the 21st International Colloquium on Automata, Languages and Programming, Jerusalem, Israel, 1994, pp. 532–543. This research was supported in part by the Basic Research Foundation administrated by the Israel Academy of Sciences and Humanities.

http://www.siam.org/journals/sidma/10-2/27413.html

†Department of Computer Science, School of Mathematical Sciences, Tel Aviv University, Tel Aviv 69978, Israel (raphy@math.tau.ac.il, zwick@math.tau.ac.il).

roughly, that dense enough undirected graphs contain many even cycles. We also prove a constructive version of their theorem.

The $O(V^2)$ algorithm for finding cycles with a given even length leads to the following strange state of affairs: deciding whether a given undirected graph contains a cycle of length, say, 100, is *asymptotically* faster than deciding, using any known algorithm, whether this graph contains a triangle (a cycle of length 3)! The term "asymptotically" above should be stressed, because our $O(V^2)$ bound, as well as Monien's $O(VE)$ bound, hides huge multiplicative factors that depend exponentially on k. This exponential dependence on k is probably unavoidable since the problem is NP-hard if k is part of the input.

A shortest cycle in a directed or undirected graph $G = (V, E)$ can be easily found in $O(VE)$ time by conducting a BFS (breadth first search) from each vertex. Itai and Rodeh [5] show that a shortest cycle can also be found in $O(M(V))$ time in the undirected case, and in $O(M(V) \log V)$ time in the directed case. They also notice that by halting the BFS conducted from each vertex in the $O(VE)$ algorithm when the first nontree edge is found (this implies an $O(V)$ running time for each BFS), an *almost* shortest cycle (a cycle whose length exceeds the length of a shortest cycle by at most 1) in an undirected graph can be found in $O(V^2)$ time.

Monien [6] described a sophisticated $O(V^2\alpha(V))$ algorithm for finding shortest even length cycles (SELCs) in undirected graphs, where $\alpha(n) = \alpha(n, n)$ is the functional inverse of Ackermann's function. His algorithm uses the fast union-find data structure. We describe an $O(V^2)$ algorithm for finding SELCs. Our algorithm is somewhat simpler, and it does not use any sophisticated data structure. At the heart of our algorithm lies a combinatorial lemma which is of interest in its own right. The lemma states that if C is a shortest even cycle in a graph, then there exists a vertex v on C from which the paths, on the cycle, to all the other vertices on the cycle are almost the shortest possible. In fact, each of these paths is of length at most 1 greater than the distance between the endpoints of the path.

We also describe a simple $O(M(V) \log V)$ algorithm for finding a shortest odd length cycle (SOLC) in an undirected graph $G = (V, E)$ and a simple $O(VE)$ algorithm for finding a SOLC in a directed or undirected graph $G = (V, E)$. Monien [7] described an $O(VE)$ algorithm for the undirected case.

This paper is organized as follows. In section 2 we present the algorithm for finding fixed-length even cycles in undirected graphs. In section 3 we investigate the combinatorial structure of SELCs. In section 4 we describe the algorithm for finding a SELC and prove its correctness. In section 5 we describe the simple algorithms for finding SOLCs in directed and undirected graphs. We end, in section 6, with some concluding remarks.

2. Finding even cycles of a given length. Throughout this section we use C_l to denote a cycle of length l. The main result of this section is the following theorem.

THEOREM 2.1. *For every $k \geq 2$, there is an $O((2k)! \cdot V^2)$ time algorithm that decides whether an undirected graph $G = (V, E)$ contains a C_{2k} and finds one if it does.*

We also obtain the following result, which is an algorithmic version of a result by Bondy and Simonovits [4].

THEOREM 2.2. *Let $l \geq 2$ be an integer and let $G = (V, E)$ be an undirected graph with $|E| \geq 100l \cdot |V|^{1+1/l}$. Then G contains a C_{2k} for every $k \in [l, l \cdot |V|^{1/l}]$. Furthermore, such a C_{2k} can be found in $O(k \cdot V^2)$ time. In particular, a cycle of length* exactly *$\lfloor l \cdot |V|^{1/l} \rfloor$ can be found in $O(V^{2+1/l})$ time.*

It is interesting to comment on the relationship between these two theorems. In *any* undirected graph $G = (V, E)$ and any $k \geq 2$, we can find a C_{2k}, if one exists, in $O((2k)! \cdot V^2)$ time. This running time is $O(V^2)$ for every *fixed* $k \geq 2$. The running time is exponential, however, if k is part of the input. If the graph $G = (V, E)$ is *dense* enough, i.e., if it contains $\Omega(V^{1+1/k})$ edges, then it does contain a C_{2k}, and such a C_{2k} can be found in $O(k \cdot V^2)$. Note that this is now polynomial in both V and k. In dense enough graphs, we can therefore find extremely long cycles efficiently. In a graph containing $\Omega(V^{3/2})$ edges, for example, we can find, in $O(V^{2.5})$ time, a cycle of length $\Theta(V^{1/2})$. This should be compared with the fact that the problem of deciding whether an undirected graph $G = (V, E)$ contains a cycle (or an even cycle) of length $\Omega(V^{1/2})$ is NP-hard.

The first ingredient used in the proofs of Theorems 2.1 and 2.2 is a combinatorial lemma of Bondy and Simonovits [4] (see also [3]). Their proof of the lemma is nonconstructive. By slightly altering their arguments we obtain a constructive version of their lemma which is required in the proof of Theorem 2.2. Before stating the lemma we need the following definition.

DEFINITION 2.3. *A coloring of the vertices of an undirected graph $G = (V, E)$ is said to be* t-periodic *if the endpoints of every path of length t are colored by the same color.*

Note that the coloring in the definition above is not required to be proper; i.e., adjacent vertices may be colored by the same color. We can now state the lemma of Bondy and Simonovits [4] and present an algorithmic proof of it.

LEMMA 2.4. *Let t be a positive integer, and let $G = (V, E)$ be a connected undirected graph with $|E| \geq 2t \cdot |V|$. Then any coloring of the vertices of G that uses at least three distinct colors is not t-periodic. Furthermore, if G is nonbipartite, then any coloring of the vertices of G that uses at least two distinct colors is not t-periodic. In both the bipartite and nonbipartite cases, two vertices of distinct colors and a path of length t connecting them can be found in $O(E)$ time.*

Proof. We begin by showing that G contains two adjacent vertices joined by two vertex-disjoint paths, each of length at least t, and that such a subgraph, called a Θ*-graph*, can be found in $O(E)$ time. It is easy to see that G contains a subgraph G' whose minimal degree is at least $2t$. Such a subgraph can be easily found in $O(E)$ time by sequentially removing from G vertices whose degrees are less than $2t$. Let $v_1, v_2, \ldots, v_m$ be a maximal path in G', i.e., a path that cannot be further extended. Such a path can be greedily constructed in $O(E)$ time. The vertex v_1 is then adjacent to at least $2t$ vertices $v_{i_1}, v_{i_2}, \ldots, v_{i_{2t}}$ on this path, where $2 = i_1 < i_2 < \cdots < i_{2t}$. The path $v_1, v_2, \ldots, v_{i_{2t}}$, along with the edges (v_1, v_{i_t}) and $(v_1, v_{i_{2t}})$, forms the desired Θ-graph.

The Θ-graph found contains three distinct cycles L_1, L_2, L_3 of lengths l_1, l_2, l_3, respectively, such that $l_1, l_2, l_3 > t$ and $l_1 + l_2 - l_3 = 2$. Every vertex v of the Θ-graph has at most four distinct paths of length t in the Θ-graph that start at v. We can easily check in $O(V)$ time whether, for each v, the endpoints of these paths are colored by the same color of v. If this is not the case, then we are done, since we have found two vertices colored by distinct colors and a path of length t connecting them.

Assume therefore that the Θ-graph is t-periodic. It is easy to see that if one of the cycles L_1, L_2, or L_3 is t^*-periodic, then the other cycles, and therefore the Θ-graph, must also be t^*-periodic. Let t^* be the smallest integer for which the Θ-graph is t^*-periodic. It follows that t^* is also the smallest period of the cycles L_1, L_2, or L_3, and as a consequence $t^* | l_1, l_2, l_3$. As $l_1 + l_2 - l_3 = 2$, we get that $t^* | 2$. Thus $t^* = 1$ or

$t^* = 2$, and the number of colors used to color the Θ-graph is at most 2.

Every vertex of G is connected by a simple path whose length is a multiple of t to a vertex of, say, L_1. If a vertex $v \in V$ is colored by color not appearing on L_1, then a simple path t whose endpoints are colored by distinct colors can be easily found in $O(V)$ time.

Finally, note that a 2-periodic coloring of a graph $G = (V, E)$ that uses two colors is necessarily a proper coloring. Any graph $G = (V, E)$ that has a 2-periodic coloring that uses only two colors must therefore be bipartite. □

The second ingredient used in the proof of Theorem 2.1 is the following result of Monien [7].

LEMMA 2.5. *There is an $O(k! \cdot E)$ time algorithm that, given a (directed or undirected) graph $G = (V, E)$, an integer $k \geq 2$, and a vertex $s \in V$, finds all vertices $v \in V$ connected to s by paths of length k, and exhibits one such path for each such v.*

The following are immediate consequences of Lemma 2.5.

COROLLARY 2.6. *Let $G = (V, E)$ be a (directed or undirected) graph and let $k \geq 3$ be an integer. There is an $O((k-1)! \cdot E)$ time algorithm that, given a vertex $s \in V$, decides whether there is a C_k that passes through s, and finds such a C_k if one exists.*

Proof. Find all the vertices connected to s by paths of length $k-1$ and check whether one of them is also connected to s by an edge. □

COROLLARY 2.7. *Let $G = (V, E)$ be a (directed or undirected) graph and let $k \geq 1$ be an integer. There is an $O((k+1)! \cdot E)$ time algorithm that, given two disjoint subsets A and B of vertices, determines whether there is a path of length k connecting a vertex from A and a vertex from B, and finds such a path if one exists.*

Proof. Assume that the graph is directed. (If not, replace each undirected edge by two anti-parallel directed edges.) Add a new vertex s and connect it to all the vertices of A. Now find all the vertices to which there are directed paths of length $k+1$ from s. □

Alon, Yuster, and Zwick [1] have recently described a $2^{O(k)} \cdot E \log V$ time algorithm for performing the task of Lemma 2.5 and $2^{O(k)} \cdot E$ *expected* time algorithms for the tasks of Corollaries 2.6 and 2.7. The dependency on k in the above complexity bounds can be improved, therefore, from $k!$ to $2^{O(k)}$ if randomization or an extra $\log V$ factor is allowed.

We are now ready to prove Theorem 2.1. We prove, in fact, the following slightly stronger result.

THEOREM 2.8. *Let $k > 1$ be a fixed integer. There is an $O((2k)!V)$ time algorithm that, given an undirected graph $G = (V, E)$ and a vertex $s \in V$, either verifies that s is not contained in any C_{2k} or finds a C_{2k} in G (not necessarily passing through s).*

Proof. The algorithm starts a BFS from the vertex s. For $v \in V$, let $d(v)$ be the distance between s and v in G. Let $L_i = \{v \mid d(v) = i\}$ be the set of vertices at level i of the BFS tree. At stage i the algorithm scans the adjacency lists of the vertices of L_i. During this scan, the algorithm keeps a count of the number of edges found so far inside L_i (an edge is inside L_i if both its endpoints are in L_i). Similarly, it keeps a count of the number of edges found so far between L_i and L_{i+1}. We use L'_{i+1} to denote the set of vertices of L_{i+1} that were already discovered by the search. The search is halted when one of the following conditions holds:

1. Stage $k-1$ has completed or the BFS has ended.
2. At least $4k \cdot |L_i|$ edges were found inside L_i.
3. At least $4k \cdot (|L_i| + |L'_{i+1}|)$ edges were found between L_i and L'_{i+1}.

Since the L_i's are disjoint, the total number of edges scanned before the search is

halted is at most $12k\cdot|V|$. Hence, the search takes only $O(k\cdot V)$ time.

As in any BFS, when a vertex $v \in L_i$ is discovered, we let $\pi(v)$ be the vertex in L_{i-1} that discovered it. In such a way a shortest path tree rooted at s and consisting of all discovered vertices is maintained.

The algorithm continues in one of three possible ways, according to the condition that caused the BFS to halt.

Case 1. The BFS is halted because stage $k-1$ has completed.

In this case, the first $k+1$ levels $L_0, L_1, \ldots, L_k$ have all been discovered, and the subgraph G' induced by them (but not containing the edges inside L_k) contains at most $12k\cdot|V|$ edges. If s is on a C_{2k} then this C_{2k} is completely contained in G'. By Corollary 2.6, we can check whether such a cycle exists in $O((2k)!\cdot V)$ time.

Case 2. The BFS is halted because $4k\cdot|L_i|$ edges were found inside L_i for some $i < k$.

Stage i of the search is then left incomplete, but all the first $i+1$ levels $L_0, L_i, \ldots, L_i$ are already completely discovered. Consider the subgraph of G induced by L_i. This subgraph contains at least one connected component whose vertex set is $U \subseteq L_i$ and whose number of edges is at least $4k\cdot|U|$. Denote the subgraph composed of this connected component by H. Such a subgraph is easily found in $O(k\cdot L_i) = O(k\cdot V)$ time. Note that $|U| > 1$.

Assume at first that H is nonbipartite. (This is easily verified in $O(k\cdot V)$ time, since H contains $O(k\cdot V)$ edges.) Let c be the lowest common ancestor in the BFS tree of all the vertices in U. The vertex c is easily found in $O(k\cdot U) = O(k\cdot V)$ time in the following way: let $U_i = U$ and let $U_j = \{\pi(v) \mid v \in U_{j+1}\}$ for $j = i-1, i-2, \ldots,$ until a U_h with $|U_h| = 1$ is reached. Then $U_h = \{c\}$. As $|U| > 1$, c must have at least two children in U_{h+1}. Let d be one of them. Let $X_1 \subset U$ be the descendents of d in U, and let $X_2 = U - X_1$. Color the vertices of X_1 red and the vertices of X_2 blue. By Lemma 2.4, the subgraph H cannot be $2(k-i+h)$-periodic (since it is nonbipartite, connected and colored by two distinct colors). There must therefore be a path of length $2(k-i+h)$ in H between a red vertex and a blue vertex. As explained in the proof of Lemma 2.4, we can find such a path p in $O(k\cdot U) \le O(k\cdot V)$ time. (Such a path can also be found using Corollary 2.7, but the running time would be $O((2k)!\cdot V)$.) The path p can now be extended to a cycle of length $2k$ by adding the disjoint paths of the BFS tree from c to the two endpoints of p, each having length $i-h$. Note that this cycle contains s only if $c = s$.

Very similar actions are taken if H is bipartite. Let A and B be the vertex classes of H (i.e., A and B are disjoint, $A \cup B = U$, and all the edges in H are between A and B). Assume, without loss of generality, that $|A| > 1$. Let c be the lowest common ancestor in the BFS tree of all the vertices of A. The vertex c is found using the method described above. Assume again that c is in level h. As $|A| > 1$, c must have at least two children in level $h+1$. Let d be one of them. Let $X_1 \subset A$ be the descendents of d in A and let $X_2 = A - X_1$. Color the vertices of X_1 red, the vertices of X_2 blue, and the vertices of B green. By Lemma 2.4, the subgraph H cannot be $2(k-i+h)$-periodic, since it is connected and colored by three distinct colors. There must therefore be a path p of length $2(k-i+h)$ in H between two differently colored vertices. This path must be between a red vertex and a blue vertex, because any path of an even length that starts at a green vertex also ends at a green vertex. This path can again be found in $O(k\cdot V)$ time, and it can again be extended to a cycle of length $2k$.

Case 3. The BFS was halted because $4k\cdot(|L_i| + |L'_{i+1}|)$ edges were found be-

tween L_i and L'_{i+1}.

Find a connected subgraph H of the subgraph of G induced by L_i and L'_{i+1} with a vertex set U and with at least $4k{\cdot}|U|$ edges. Such a subgraph is easily found in $O(k{\cdot}V)$ time. Note that H is bipartite with vertex classes $A = U \cap L_i$ and $B = U \cap L'_{i+1}$. The algorithm can now proceed as in the previous case.

In any one of these three cases, the running time is $O((2k)! \cdot V)$. In fact, the running time of the algorithm in the second and third cases is only $O(k{\cdot}V)$. The only case in which a C_{2k} is not found by the algorithm is when no C_{2k} passes through s. This completes the proof of the theorem. □

Theorem 2.1 follows immediately from the above theorem. All we have to do is apply the algorithm described above from each vertex. We now turn to the proof of Theorem 2.2. The proof of Bondy and Simonovits actually shows that if $|E| \geq 100l \cdot |V|^{1+1/l}$ and $k \in [l, l \cdot |V|^{1/l}]$ then there *exists* a vertex $s \in V$ for which the algorithm of Theorem 2.8 stops *before* completing stage $k-1$. This immediately leads to the desired $O(k \cdot V^2)$ time algorithm. Theorem 2.8 has another interesting consequence.

THEOREM 2.9. *A C_{2k} in an undirected graph $G = (V, E)$ with $|E| \geq 101k \cdot |V|^{1+1/k}$ can be found in $O((2k)!{\cdot}V)$ expected time.*

Proof. Any graph on $|V|$ vertices and at least $100k{\cdot}|V|^{1+1/k}$ edges contains a C_{2k}. It follows immediately that the number of edges in a graph $G = (V, E)$ which are not contained in any C_{2k} is at most $100k{\cdot}|V|^{1+1/k}$. If $G = (V, E)$ contains at least $101k{\cdot}|V|^{1+1/k}$ edges, then a randomly chosen edge has a probability of at least $1/101$ of belonging to a C_{2k}. The randomized algorithm simply chooses a random edge and applies the algorithm of Theorem 2.8 to one of its endpoints. The expected number of applications before a desired C_{2k} is found is $O(1)$ and the expected running time is $O((2k)!{\cdot}V)$. □

3. The structure of shortest even length cycles. Let G be an undirected graph and let C be a SELC (shortest even length cycle) of it. Suppose the vertices on the cycle are consecutively labeled $v_0, v_1, \ldots, v_{2k-1}$. We denote by $d(x, y)$ the distance between two vertices x and y in G. Clearly, $d(v_0, v_i), d(v_0, v_{2k-i}) \leq i$ for every $1 \leq i \leq k$. If $d(v_0, v_i) = i$ and $d(v_0, v_{2k-i}) = i$, for every $1 \leq i \leq k$, then C, or some other SELC, can be easily found using a BFS from v_0. However, the paths on C between v_0 and v_i and between v_0 and v_{2k-i} are not necessarily shortest paths in G. As an example, consider K_4, the complete graph on four vertices. All the even cycles in K_4 are of length 4, but the distance between any two vertices is 1. It may be, therefore, that $d(v_0, v_i) < i$ or $d(v_0, d_{2k-i}) < i$ for some $1 \leq i \leq k$. It is not immediately clear how to find C, or any other SELC, in such a case.

The main result of this section is the following lemma that states that on every SELC C there is a vertex v_0 from which the paths, on C, to all the other vertices on C are *almost* shortest paths. An almost shortest path is a path whose length exceeds the length of a corresponding shortest path by at most one. Specifically, we have the following lemma.

LEMMA 3.1. *Let C be a SELC of G. Then the vertices on C can be consecutively labeled $v_0, v_1, \ldots, v_{2k-1}$, so that $i - 1 \leq d(v_0, v_i) \leq i$ and $i - 1 \leq d(v_0, v_{2k-i}) \leq i$ for every $1 \leq i \leq k$.*

This lemma is the cornerstone of the $O(V^2)$ algorithm for finding SELCs presented in the next section. We think, also, that this lemma is of interest in its own right. Before presenting a proof of Lemma 3.1, we present the following simple but useful lemma.

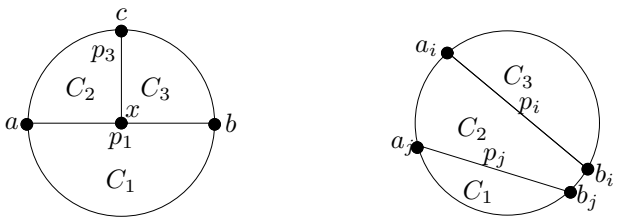

FIG. 3.1. *One of the cycles C_1, C_2, and C_3 is even.*

LEMMA 3.2. *If p_1 and p_2 are two distinct (but not necessarily disjoint) shortest paths in G between x and y, then C contains an even cycle whose length is at most $2d(x, y)$.*

Proof. Let $p_1 = (a_0, a_1, \ldots, a_{k-1}, a_k)$ and $p_2 = (b_0, b_1, \ldots, b_{k-1}, b_k)$ be two distinct shortest paths between $x = a_0 = b_0$ and $y = a_k = b_k$. Let $i \geq 0$ be the minimal index such that $a_i = b_i$ but $a_{i+1} \neq b_{i+1}$. Let j be the minimal index $j > i$ such that $a_j = b_j$. Then $(a_i, \ldots, a_j)$ and $(b_i, \ldots, b_j)$ are two shortest paths connecting a_i and a_j whose inner vertices are disjoint. We thus obtain a cycle of length $2(j - i) \leq 2k$. ∎

Proof of Lemma 3.1. Let H be a minimal subgraph of G (with respect to containment) containing C such that $d_H(x, y) = d(x, y)$ for every $x, y \in C$ ($d_H(x, y)$ denotes the distance between x and y in H). Let $e(H)$ be the edge set of H. If $H = C$, we are done. Otherwise, let $P = H \setminus e(C)$.

A path p whose two endpoints a and b are on C, but none of whose inner vertices are on C, that satisfies $|p| = d(a, b) < d_C(a, b)$, where $|p|$ is the length of p, is called an $a \sim b$ *shortcut*. Our first claim is that P is a collection of vertex disjoint shortcuts.

To see this, let P' be a connected component of P. The minimality of H implies that any edge of P' is contained in some shortcut. The component P' must therefore contain an $a \sim b$ shortcut p_1 for some $a, b \in C$. If P' is composed solely of this shortcut, we are done. Otherwise, let x be a vertex on p_1 incident to an edge e of P' which is not on p_1 (x may be a or b). The edge e is contained in some shortcut p_2. The shortcuts p_1 and p_2 meet only at x. If they had met in some other vertex y, a shorter even cycle would have existed, by Lemma 3.2, in the graph. Let p_3 be a portion of p_2 that connects x with some vertex c on C. Consider now the cycles C_1, C_2, and C_3 shown on the left of Fig. 3.1. Each of these cycles is of size less than $2k$. For C_1, this follows from the fact that $|p_1| < d_C(a, b)$. We show that $|C_2| < 2k$ as follows: let C_4 be the cycle comprised of p_1 with the part of C between a and b containing c. Since $|p_1| < d_C(a, b)$ we have that $|C_4| < 2k$. As p_3 is a shortest path between c and x, we get that $|C_2| \leq |C_4| < 2k$. The fact that $|C_3| < 2k$ follows from similar arguments. The sum of the lengths of these cycles is $2k + 2|p_1| + 2|p_3|$, which is even, and thus one of them must be even, contradicting the minimality of C. This contradiction shows that P' must simply be a shortcut.

We have shown that $P = \{p_1, \ldots, p_s\}$ is a set of disjoint shortcuts where $s \leq k$ (as each shortcut contains two vertices of C). We now claim that every two distinct shortcuts p_i and p_j must *cross* one another; i.e., each of the two paths on C between the endpoints of p_i contains an endpoint of p_j. See the left side of Fig. 3.2.

Assume, for contradiction, that the shortcuts p_i and p_j do not cross one another, as shown on the right side of Fig. 3.1. The length of each of the cycles C_1, C_2, and

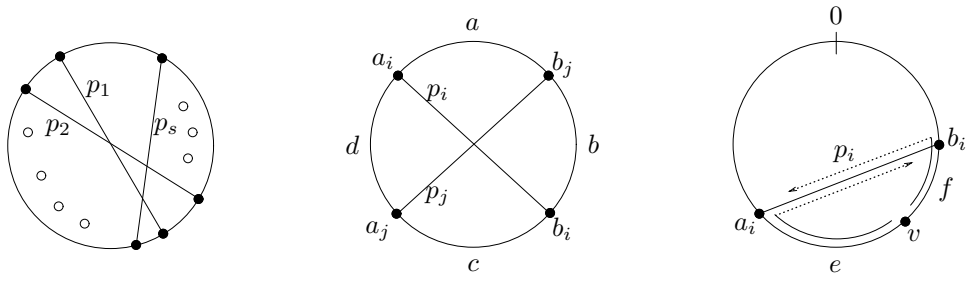

FIG. 3.2. *The shortcuts of P.*

C_3 there is less than $2k$. The sum of their lengths is $2k + 2|p_i| + 2|p_j|$, so one of them must be even, contradicting the minimality of C.

We have shown that the mutual position of p_i and p_j must be as shown in the middle of Fig. 3.2. Let a, b, c, d denote the four segments of C determined by the endpoints of these shortcuts. The minimality of C implies that $|p_i| + |a| + |b|$, $|p_i| + |c| + |d|$, $|p_j| + |b| + |c|$, and $|p_j| + |a| + |d|$ are all odd, since these are lengths of cycles smaller than $2k$. This, in turn, implies that $|p_i| + |p_j| + |a| + |c|$ and $|p_i| + |p_j| + |b| + |d|$ are even. These two expressions are the lengths of the "twisted" cycles a, p_i, c, p_j and b, p_i, d, p_j. As a consequence, these lengths are at least $2k$. In particular,

$$|p_i| + |p_j| + |a| + |c| \geq 2k = |a| + |b| + |c| + |d|. \tag{3.1}$$

Our third claim is that for any two vertices x, y on the cycle C there exists a shortest path between them that uses at most one shortcut. Consider a shortest path between x and y that contains at least two shortcuts. Let p_i and p_j be two consecutive shortcuts appearing on the path. Let c be the portion of the path that connects them, as shown again in the middle of Fig. 3.2. From (3.1), we get that $|p_i| + |c| + |p_j| \geq |b| + |c| + |d|$. We can therefore replace the portion p_i, c, p_j of the path by the path b, c, d without increasing the length. Continuing in this way, we can obtain a shortest path that uses at most one shortcut. In view of Lemma 3.2, a shortest path that uses more than one shortcut must connect two antipodal vertices, i.e., two vertices whose distance is k, on the cycle.

It is convenient at this point to fix a consecutive numbering $0, 1, \ldots, 2k-1$ of the vertices of the cycle C and to identify the vertices of C with their numbers. We let a_i and b_i, where $a_i < b_i$, be the two endpoints of the shortcut p_i. To every shortcut p_i we attach the following interval:

$$C_i = \left[\frac{a_i + b_i - |p_i| - 1}{2}, \frac{a_i + b_i + |p_i| + 1}{2} \right].$$

Both endpoints of this interval are integral since $b_i - a_i$ and $|p_i|$ have different parities; otherwise, C would not have been a SELC. As $|p_i| < b_i - a_i$, we get that $C_i \subseteq [a_i, b_i]$. The interval C_i corresponds to a subset of the vertices of C.

We claim that if $v \in C_i$, then for every vertex u on C, if a shortest path between v and u uses the shortcut p_i as its only shortcut, then the path between v and u along the cycle C is an almost shortest path between v and u. Recall that an almost shortest path between v and u is a path whose length is at most $d(v, u) + 1$. To see

this, suppose that $v \in C_i$ and that some shortest path from v to u uses p_i as its only shortcut. This shortest path must either go along portion e of the cycle C from v to a_i, then use p_i, and then go again along C, or go along portion f of the cycle C from v to b_i, then use p_i, and then go again along C. Both cases are shown on the right of Fig. 3.2. The definition of C_i implies, however, that

$$|e| = v - a_i \leq b_i - v + |p_i| + 1 = |f| + |p_i| + 1,$$

$$|f| = b_i - v \leq v - a_i + |p_i| + 1 = |e| + |p_i| + 1.$$

The path e, p_i can therefore be replaced by the path f, and the path f, p_i can be replaced by the path e while increasing the length by at most one, as required.

Our final task is to show that the intersection $\cap_{i=1}^{s} C_i$ of all these intervals is not empty. If $v_0 \in \cap_{i=1}^{s} C_i$, then the paths along C from v_0 to all other vertices on the cycle are almost shortest paths, as required. As all the C_i's are intervals, it is enough to show that any two of them intersect. Let C_i and C_j be two such intervals where $a_i < a_j$. The fact that p_i and p_j cross one another implies that $a_i < a_j < b_i < b_j$. To show that C_i and C_j intersect, we show that

$$\frac{a_j + b_j - |p_j| - 1}{2} \leq \frac{a_i + b_i + |p_i| + 1}{2}$$

and

$$\frac{a_i + b_i - |p_i| - 1}{2} \leq \frac{a_j + b_j + |p_j| + 1}{2}.$$

The first inequality is equivalent to $|p_i| + |p_j| + (2k - b_j + a_i) + (b_i - a_j) + 2 \geq 2k$. But $|p_i| + |p_j| + (2k - b_j + a_i) + (b_i - a_j)$ is the length of the twisted cycle a, p_i, c, p_j shown in the middle of Fig. 3.2. The length of this cycle is at least $2k$ by (3.1), proving the first inequality. The second inequality follows immediately from the fact that $a_i < a_j < b_i < b_j$. We have shown therefore that the intervals C_i and C_j, and therefore all the intervals, do intersect.

Any vertex $v_0 \in \cap_{i=1}^{s} C_i$ can play the role of v_0 in the statement of the lemma. This completes the proof of the lemma. □

If a SELC C is edge disjoint from all other SELCs, then a sharp inequality holds in (3.1). This can be used to show that all the intervals $C_i' = [\frac{a_i + b_i - |p_i| + 1}{2}, \frac{a_i + b_i + |p_i| - 1}{2}]$ intersect. Every vertex v_0 in this intersection has the property that the shortest paths along the cycle C from v_0 to all other vertices are in fact shortest paths. The intersection $\cap_{i=0}^{s} C_i'$ may, however, be empty if C is not edge disjoint from all other SELCs.

Let $v_0, \ldots, v_{2k-1}$ be an ordering of C that satisfies the conditions of Lemma 3.1. In view of Lemma 3.2, it is impossible that $d(v_0, v_{k-1}) = d(v_0, v_{k+1}) = k - 2$, because this yields two shortest paths of lengths $k - 1$ from v_0 to v_k. We may therefore assume, without loss of generality, that $d(v_0, v_{k-1}) = k - 1$. We call v_0 a *root* of C. If $d(v_0, v_k) = k$ we call C a cycle of *type one* with respect to (w.r.t.) v_0, and if $d(v_0, v_k) = k - 1$ we call C a cycle of *type two* w.r.t. v_0. Every cycle of type two w.r.t. v_0 has a unique $0 < j < k$ such that $d(v_0, v_{2k-j}) = j$, and $d(v_0, v_{2k-j-1}) = j$. We call j the *index* of C w.r.t. the root v_0.

Finally, we note that if $v_0, \ldots, v_{2k-1}$ is an ordering of C that satisfies the conditions of Lemma 3.1, then $v_k, \ldots, v_{2k-1}, v_0, \ldots, v_{k-1}$ is also such an ordering; i.e., v_k can play the role of v_0.

4. An $O(V^2)$ algorithm for finding a shortest even cycle. Relying on Lemma 3.1, we obtain an $O(V^2)$ algorithm for finding a SELC in an undirected graph $G = (V, E)$. The algorithm starts a BFS from every vertex, but stops it as soon as an even cycle is detected. This ensures that the time spent in each such BFS is at most $O(V)$. We show that the shortest even cycle found in this way by the algorithm is indeed a SELC of the graph.

The BFS performed is an augmented version of the standard BFS capable of detecting even cycles. Let a be a vertex from which such an augmented BFS is performed (a is called the root of the BFS). For every vertex v, we record a set of four variables. The first two variables are standard; the other two are used to detect even cycles. These four variables are:

$d(v)$**:** the distance of v from a, i.e., the level of v in the BFS tree; $d(v) = \infty$ if v has not yet been discovered.

$\pi(v)$**:** the parent of v in the BFS tree; $\pi(v) = 0$ if $v = a$ or if v has not yet been discovered. If $\pi(v) \neq 0$ then $d(v) = d(\pi(v)) + 1$.

$\theta(v)$**:** the *match* of v; if $\theta(v) \neq 0$ then $\theta(v)$ is a vertex in the same level of v such that $(v, \theta(v)) \in E$. A vertex v is said to be *matched* if $\theta(v) \neq 0$. If v is matched then $\theta(v)$ will also be matched and $\theta(\theta(v)) = v$. The set of edges $\{(v, \theta(v)) \mid \theta(v) \neq 0\}$ is therefore a matching.

$\rho(v)$**:** the highest proper ancestor of v in the BFS tree that is matched. If v has no matched proper ancestors, then $\rho(v) = 0$.

We now describe how we process a vertex v that has been popped out of the BFS queue. Before we start scanning v's neighbors, we assume that $\rho(v)$, $d(v)$, and $\pi(v)$ are correctly set (v may or may not be matched at this point depending on whether it is adjacent to a vertex in its level that has been processed before it). The action taken for an edge (v, u) depends on the value of $d(u)$, $\theta(v)$, and $\theta(u)$ in the following way:

1. If $d(u) = d(v) - 1$, do nothing (this edge has been processed before, in its opposite direction).
2. If $d(u) = \infty$, let $d(u) \leftarrow d(v) + 1, \pi(u) \leftarrow v$ and enqueue u to the BFS queue.
3. If $d(u) = d(v) + 1$, halt the BFS since an even cycle was found. Let c be the lowest common ancestor, in the BFS tree, of v and u. Then the $c \sim v$ and $c \sim u$ tree paths and the edge (v, u) form an even cycle of length $2(d(v) + 1 - d(c))$. This cycle is shown in Fig. 4.1.
4. If $d(u) = d(v)$ and $\theta(v) = u$ (which also means that $\theta(u) = v$), do nothing (this edge has been processed before, in its opposite direction).
5. If $d(u) = d(v)$ and $\theta(v) \neq u$, and $\theta(v)$ and $\theta(u)$ are not both zero, halt the BFS since an even cycle was found. Assume, for example, that $\theta(v) = x \neq 0$. Let c be the lowest common ancestor, in the BFS tree, of x and u. The $c \sim x$ and $c \sim u$ tree paths and the edges $(x, v), (v, u)$ form an even cycle of length $2(d(v) + 1 - d(c))$. This cycle is shown in Fig. 4.1.
6. If $d(u) = d(v)$ and $\theta(v) = \theta(u) = 0$, test whether $\rho(v) = \rho(u)$. If they are equal, let $\theta(v) \leftarrow u$, $\theta(u) \leftarrow v$. If they are not equal, halt the BFS since an even cycle is found as follows. Assume, for example, that $\rho(v) = x \neq 0$ and let $y = \theta(x)$. Let c be the lowest common ancestor, in the BFS tree, of y and u. Then the $c \sim y$ tree path followed by the edge (y, x) followed by the $x \sim v$ tree path followed by the edge (v, u) followed by the $u \sim c$ tree path closes an even cycle of length $2(d(v) + 1 - d(c))$. This cycle is shown in Fig. 4.1. Note that this is a cycle (i.e., it is simple) since x is not an ancestor

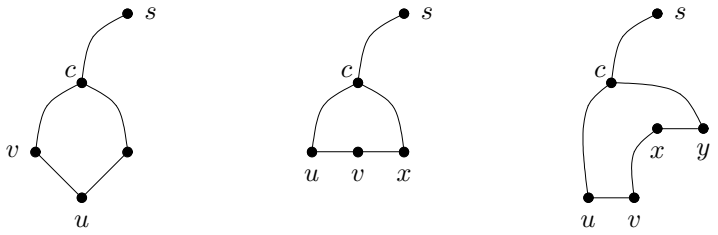

FIG. 4.1. *The even cycles detected by rules* 3, 5, *and* 6.

of u.

After we finish scanning all the neighbors of v, we rescan them to set $\rho(u)$ for every u that has become a child of v. We put $\rho(u) \leftarrow \rho(v)$ unless $\rho(v) = 0$ and $\theta(v) \neq 0$, in which case we put $\rho(u) \leftarrow v$. This completes the description of the algorithm.

THEOREM 4.1. *The augmented BFS scans no more than* $3|V|/2$ *edges and therefore runs in* $O(V)$ *time. Furthermore, if* C *is a SELC of length* $2k$ *and* v_0 *is a root of it, then an augmented BFS that starts from* v_0 *finds an even cycle of length* $2k$.

Proof. When the BFS halts (either because it has completed, or because an even cycle has been found), the only edges scanned are the BFS tree edges, the edges between matched vertices (these edges form a matching), and possibly an edge that closes an even cycle. There are at most $|V| - 1$ tree edges and at most $(|V| - 1)/2$ edges in the matching (the root of the BFS is never matched). The total number of edges scanned is therefore at most $3|V|/2$. The complexity claim is obvious because scanning an edge entails only a constant number of operations.

We now prove the second part of the theorem. Consider an augmented BFS that starts at a root v_0 of a SELC C. Note, according to the above six rules, that if the BFS halts while scanning the neighbors of a vertex v, the even cycle found has a length of at most $2(d(v) + 1)$.

Suppose that C is a SELC of type one w.r.t. v_0 (type-one and type-two SELCs were defined at the end of the previous section). Then v_{k-1} and v_{k+1} are both in level $k - 1$ of the BFS. Suppose that v_{k+1} is processed after v_{k-1}. If an even cycle is found before the edge (v_{k+1}, v_k) is scanned, its length must be $2k$ (it cannot be shorter, of course). Otherwise, an even cycle of length $2k$ is found, using rule 3, when the edge (v_{k+1}, v_k) is scanned.

Suppose that C is a SELC of type two, with index $j = k - 1$ w.r.t. v_0. Then v_{k-1}, v_k, v_{k+1} are all in level $k - 1$ of the BFS. If an even cycle of length $2k$ is not found before processing the vertex v_k, such a cycle is found, using rule 5, when v_k is processed since it is adjacent to two vertices in its level.

Finally, suppose that C is a SELC of type two, with $j < k - 1$. Then both v_{k-1} and v_k are in level $k - 1$ of the BFS (and v_{k+1} is in level $k - 2$). We claim that $\rho(v_{k-1}) \neq \rho(v_k)$, and therefore an even cycle is found (using rule 6) when the edge (v_{k-1}, v_k) is scanned, if such a cycle were not found before. First, note that $\theta(v_{2k-j-1}) = v_{2k-j}$. (Both are in level j; there is an edge between them; and we did not halt at level j.) Second, note that $(v_0, v_1, \ldots, v_{k-1})$, $(v_0, v_{2k-1}, \ldots, v_{2k-j})$, and $(v_k, v_{k+1}, \ldots, v_{2k-j-1})$ are shortest paths in G (refer to Fig. 4.2). As these shortest paths connect vertices whose distance is less than k, they must be the unique shortest

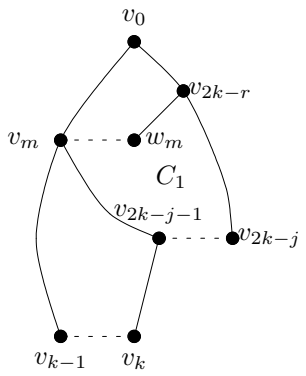

FIG. 4.2. *If* $\rho(v_k) = \rho(v_{k-1}) = v_m$ *then* $|C_1| = 2(j - r + 1) < 2k$.

paths between these vertices (cf. Lemma 3.2). These paths must therefore be tree paths; i.e., they must be contained in the BFS tree. It follows that v_{2k-j-1} is the ancestor of v_k at level j. Therefore, $\rho(v_k) \neq 0$. If $\rho(v_{k-1}) = 0$ we are done. Otherwise $\rho(v_{k-1}) = v_m$ where $1 \leq m < k-1$ since $v_0, v_1, \ldots, v_{k-2}$ are the proper ancestors of v_{k-1} (and v_0 is unmatched). Assume, for contradiction, that $\rho(v_k) = v_m$. Since v_{2k-j-1} is a matched ancestor of v_k, we have $m < j$, and v_m is an ancestor of v_{2k-j-1}. Let $w_m = \theta(v_m)$ be the match of v_m, and let v_{2k-r} be the lowest common ancestor, in the BFS tree, of w_m and v_{2k-j} (v_{2k-r} may be v_0). We obtain the following even cycle (cycle C_1 in Fig. 4.2) in G: $v_m \sim v_{2k-j-1} - v_{2k-j} \sim v_{2k-r} \sim w_m - v_m$, where $v_m \sim v_{2k-j-1}$, $v_{2k-j} \sim v_{2k-r}$, and $v_{2k-r} \sim w_m$ denote the tree paths between these vertices, and $v_{2k-j-1} - v_{2k-j}$, $w_m - v_m$ are the edges matching these vertices. The tree paths $v_m \sim v_{2k-j-1}$ and $v_m \sim v_{k-1}$ may coincide initially; this causes no problems. The cycle C_1 is indeed a cycle; i.e., it is simple, because it is composed of tree paths that cannot intersect one another. The length of C_1 is $2(j-r+1) \leq 2k-2$, contradicting the minimality of C. □

As a corollary of Theorem 4.1, we get that any graph containing more than $3(V-1)/2$ edges contains an even cycle. A simple example shows that this bound is the best possible. Just take any connected graph whose biconnected components are triangles. Furthermore, checking whether a graph contains an even cycle and exhibiting one if it does can be done in $O(V)$ time. Just perform *one* augmented BFS from an arbitrary vertex.

Finally, we point out that the result of this section is *not* implied by the results of section 2. We cannot afford checking, for $k = 1, 2, \ldots$, whether the graph contains a C_{2k} since the length of the smallest even cycle may be large.

5. Finding a shortest odd cycle in undirected and directed graphs. Shortest odd length cycles (SOLCs) can be found in polynomial time in both directed and undirected graphs. Our objective in this section is to describe very simple, yet efficient, algorithms for both of these problems. Monien [6] obtained a simple $O(VE)$ time algorithm for finding SOLCs in undirected graphs. Using fast Boolean matrix multiplication algorithms we obtain an $O(M(V) \log V)$ algorithm for the same task. This algorithm is more efficient than Monien's algorithm for dense graphs.

THEOREM 5.1. *There is an $O(M(V)\log V)$ time algorithm that finds a shortest odd cycle in an undirected graph $G=(V,E)$.*

Proof. Let A be the adjacency matrix of G. We assume that G is nonbipartite since otherwise it contains no odd cycles. Recall that $A^k(i,i)=1$ iff there is a closed walk of length k from i to itself (the multiplications used to obtain A^k are assumed to be Boolean). Since any closed walk of an odd length contains an odd cycle, the length of the SOLCs of G is the minimal odd k for which there exists an i such that $A^k(i,i)=1$. Since G is undirected, $A^t(i,i)=1$ implies $A^{t+2}(i,i)=1$. We can therefore look for this minimal k using the following approach. Start computing $A, A^3, A^7, \ldots, A^{2^i-1}, \ldots$ until $A^{2^s-1}(i,i)=1$ for some i. A binary search along the odd numbers between $2^{s-1}-1$ and 2^s-1 can then be used to find k. The number of Boolean matrix multiplications required is clearly $O(\log V)$. A specific SOLC of length k can be found without increasing the complexity of the algorithm. ∎

We turn our attention now to finding shortest odd cycles in directed graphs. Unlike in the undirected case, subpaths of SOLCs are not necessarily shortest paths, and therefore a simple BFS from every vertex does not suffice. Let $ed(u,v)$ be the length of a shortest even length directed walk from u to v. Similarly, let $od(u,v)$ be the length of a shortest odd length directed walk from u to v. If no odd (even) length walk exists we set $od(u,v)=\infty$ ($ed(u,v)=\infty$). Note that the existence of a walk of length $ed(u,v)$ ($od(u,v)$) does not imply the existence of a simple walk of length $ed(u,v)$ ($od(u,v)$).

LEMMA 5.2. *If $C=(v_0,v_1,\ldots,v_{k-1})$ is a SOLC of a directed graph G, then $ed(v_0,v_{2i})=2i$ and $od(v_0,v_{2i-1})=2i-1$ for every $1\le i\le \frac{k-1}{2}$.*

Proof. Any closed walk of an odd length contains an odd cycle. There is an odd length closed walk containing v_0 whose length is $ed(v_0,v_{2i})+k-2i$. The minimality of C implies that $ed(v_0,v_{2i})\ge 2i$. There is, however, a path of length $2i$ between v_0 and v_{2i}, and therefore $ed(v_0,v_{2i})=2i$. The second equality in the statement of the lemma follows using similar arguments. ∎

Given a vertex s, we can easily compute $ed(s,v)$ and $od(s,v)$ for every $v\in V$ as follows. Construct a graph $G'=(V',E')$ where

$$V'=\{v_e,v_o \mid v\in V\},$$

$$E'=\{(x_e,y_o),(x_o,y_e) \mid (x,y)\in E\}.$$

The graph G' is a directed bipartite graph that contains an *even representative* v_e and an *odd representative* v_o for every vertex $v\in V$. It is easily seen that $ed(u,v)=d'(u_e,v_e)$ and that $od(u,v)=d'(u_e,v_o)$, for every $u,v\in V$, where $d'(u',v')$ denotes the distance between u' and v' in G'. By performing a BFS on G' from s_e, we can therefore find $ed(s,v)$ and $od(s,v)$, for every $v\in V$, in $O(E)$ time (we assume the graph contains no isolated vertices).

For every $s\in V$, we can find, in $O(E)$ time, a shortest odd length closed walk that contains s. We simply compute $oc(s)=\min\{ed(s,v)+1 \mid (v,s)\in E\}$. If $oc(s)\ne\infty$, then a closed walk of length $oc(s)$, which is the minimal possible odd length, is easily found by tracing a shortest odd path from s to any vertex for which the minimum was achieved. The shortest odd length closed walk found in this way must be a SOLC. We thus obtain the following.

THEOREM 5.3. *A shortest odd length cycle in a directed graph $G=(V,E)$, if one exists, can be found in $O(VE)$ time.*

6. Concluding remarks. We have shown that interesting combinatorial properties of even cycles in undirected graphs lead to very efficient algorithms for finding even cycles of a given length and for finding shortest even cycles. Note that if the input graph is given using an adjacency matrix, then these $O(V^2)$ algorithms are optimal. It seems plausible to conjecture that $O(V^2)$ is the best possible bound, in terms of V, for these problems even if the adjacency lists of the graphs are given as input. Note that $O(V^2)$ time is currently the best known time even for finding quadrilaterals (C_4's).

Based on the results of this paper, Alon, Yuster, and Zwick [2] have recently obtained an $O(E^{4/3})$ algorithm for finding a C_4 in undirected graphs. More generally, a C_{4k} can be found in $O(E^{2-(\frac{1}{k}-\frac{1}{2k+1})})$ time. These algorithms are better than the $O(V^2)$ time algorithms presented here for relatively sparse graphs. It is interesting to note that the hardest cases for the C_4 problem, for example, are graphs with $\Theta(V^{3/2})$ edges.

Acknowledgment. The authors would like to thank Noga Alon for many helpful discussions.

REFERENCES

[1] N. Alon, R. Yuster, and U. Zwick, *Color-coding*, J. Assoc. Comput. Mach., 42 (1995), pp. 844–856. A preliminary version appeared in *Proc. 26th Annual ACM Symp. on Theory of Computing*, Montréal, Canada, 1994, pp. 326–335.

[2] N. Alon, R. Yuster, and U. Zwick, *Finding and counting given length cycles*, in Proc. 2nd European Symp. on Algorithms, Utrecht, the Netherlands, 1994, pp. 354–364; Algorithmica, to appear.

[3] B. Bollobás, *Extremal Graph Theory*, Academic Press, New York, 1978.

[4] J.A. Bondy and M. Simonovits, *Cycles of even length in graphs*, J. Combin. Theory, Ser. B, 16 (1974), pp. 97–105.

[5] A. Itai and M. Rodeh, *Finding a minimum circuit in a graph*, SIAM J. Comput., 7 (1978), pp. 413–423.

[6] B. Monien, *The complexity of determining a shortest cycle of even length*, Computing, 31 (1983), pp. 355–369.

[7] B. Monien, *How to find long paths efficiently*, Ann. Discrete Math., 25 (1985), pp. 239–254.

[8] D. Richards and A. L. Liestman, *Finding cycles of a given length*, Ann. Discrete Math., 27 (1985), pp. 249–256.

[9] C. Thomassen, *Even cycles in directed graphs*, European J. Combinatorics, 6 (1985), pp. 85–89.

[10] J. van Leeuwen, *Graph algorithms*, in Handbook of Theoretical Computer Science, Volume A, Algorithms and Complexity, J. van Leeuwen, ed., Elsevier and The MIT Press, 1990, Chap. 10, pp. 525–631.