Short Cycles in Planar Graphs

Łukasz Kowalik*
`kowalik@mimuw.edu.pl`

Institute of Informatics, Warsaw University
Banacha 2, 02-097 Warsaw, Poland

Abstract. We present new algorithms for finding short cycles (of length at most 6) in planar graphs. Although there is an $O(n)$ algorithm for finding *any* fixed subgraph H in a given n-vertex planar graph [5], the multiplication constant hidden in "O" notation (which depends on the size of H) is so high, that it rather cannot be used in practice even when $|V(H)| = 4$. Our approach gives faster "practical algorithms" which are additionally much easier to implement.
As a side-effect of our approach we show that the maximum number of k-cycles in n-vertex planar graph is $\Theta(n^{\lfloor \frac{k}{2} \rfloor})$.

1 Introduction

The subgraph isomorphism problem consists in deciding whether a given graph G, called "text", contains another graph H, called "pattern", as a subgraph. This is one of the most important and natural problems in the algorithmic graph theory, often arising in applications. In this paper we focus on the situation when G is planar and we consider one of the most natural class of patterns – cycles. It is easy to show that such restricted version of the problem is NP-complete, because of the reduction from the Hamiltonian cycle problem. However, treating the pattern as a fixed graph one can consider polynomial algorithms. Eppstein [5] presents a linear algorithm finding any fixed pattern in a planar graph. Unfortunately, for a given w-vertex pattern graph H his algorithm has to generate as much as $O(w^{3w+9})$ combinatorial objects. Therefore, in spite of great theoretical importance, the algorithm of Eppstein cannot be used in practice even for 4-vertex patterns. Thus constructing effective algorithms for particular patterns is still a challenging research area.

Related work. The problem of finding cycles of specified lengths in a planar graphs attracted many researchers. Papadimitriou et al. [9] gave first linear, but complicated algorithm for finding C_3's. Two simple linear algorithms for finding triangles were developed by Chiba et al. [3] and Chrobak et al. [4]. The first of that papers describes also simple linear algorithm for finding C_4's. Both algorithms from that paper can be also applied to d-degenerate graphs containing m edges with $O(d \cdot m)$ time complexity. Richards [10] gave $O(n \log n)$ algorithms

---

[*] Research supported by KBN grant 4T11C04425

for finding C_5's and C_6's. Alon et al. presented $O(d^2 \cdot m)$ time algorithm for finding C_5 (only one occurrence) in d-degenerate graph of m edges.

Apart from Eppstein's result [5] there were also a few other algorithms for finding cycles of *arbitrary* fixed length in planar graphs. Monien [8] presented an algorithm working in $O(m \cdot n)$ time for an arbitrary graph containing n vertices and m edges. Alon et al. designed an algorithm for planar graphs working in $O(n)$ expected time or $O(n \log n)$ worst case time. In our recent paper with M. Kurowski [7] we presented a data structure answering short paths queries in planar graphs. The structure can be also adapted to construct a linear time algorithm for finding cycles of any fixed length. However, similarly as in the Eppstein's algorithm, the constant hidden in the asymptotic notation describing the time complexity of the algorithm is very high.

New Results. We present a new approach to finding short cycles in planar graphs. We are able to apply our methods to cycles of lengths from 3 to 6. We obtain algorithms listing all $\#_c$ occurrences of cycles of a given length in n-vertex graph in time $O(n + \#_c)$. Each of the algorithms detects the first cycle in time $O(n)$. Moreover, the algorithms for cycles of length 3, 4, 5 work also for d-degenerate graphs. The running time in this case is $O(d^2 \cdot m + \#_c)$ (resp. $O(d^2 \cdot m)$ time when we search for only one cycle), where m denotes the number of edges and $\#_c$ is the number of cycles.

In section 6 we show how to extend the approach of Chiba and Nishizeki [3] to the case of 5-cycles obtaining a very simple algorithm. It can be also easily modified to count 5-cycles in d-degenerate graph of m edges in time $O(d^2 \cdot m)$. Both our algorithms for finding 5-cycles in d-degenerate graphs match the time complexity of the algorithm of Alon et al. being much simpler at the same time.

As a side-effect of our approach in section 4 we show that the maximum number of k-cycles in an n-vertex planar graph is $\Theta(n^{\lfloor \frac{k}{2} \rfloor})$. This fact apart from being interesting combinatorial result is used for time complexity analysis of our algorithms.

None of our algorithms requires a plane embedding of an input graph.

Applications in Graph Coloring. The problem of finding cycles in planar graphs is related to classical coloring and list coloring of planar graphs. Deciding whether a given planar graph is 3-colorable or 3-choosable is NP-complete. Nevertheless, we know that planar graphs without triangles are 3-colorable and planar graphs without C_3's and C_4's are 3-choosable [11]. Moreover, for any k, $3 \leq k \leq 6$ an arbitrary planar graph without k-cycles is 4-choosable [6]. Thus, algorithms for finding short cycles in planar graphs allow to recognize wide classes of 3-colorable, 3-choosable and 4-choosable planar graphs.

Our Approach. We start from transforming an n-vertex input graph G into new graphs G_2 and G_3. Each edge in G_2 (resp. G_3) corresponds to a certain path of length 2 (resp. 3) in G. Obviously, since the number of all paths of length 2 in planar graph can be $\Omega(n^2)$, we have to choose only some of the paths to guarantee a linear size of $E(G_2)$. On the other hand, we cannot lose information on any cycle that we search for. In the second phase our algorithms search for shorter cycles in graphs G_2 and G_3 or for pairs of internally disjoint paths corresponding to the same edges in G_2 or G_3.

2 Preliminaries

We assume the reader is familiar with standard terminology concerning graph theory and planar graphs in particular. In this section we recall some notions that are not so widely used.

Let G be an undirected graph. We say that G is *d-degenerate* when an arbitrary subgraph of G contains a vertex of degree at most d. A directed graph is said to be *k-oriented* if its every vertex has the out-degree at most k. If one can orient edges of graph G obtaining k-oriented graph G' we say that G is *k-orientable*. The *arboricity* $a(G)$ of graph G is the minimal number of forests needed to cover all the edges of G. The three defined notions are closely related. It is easy to show that $d = \Theta(k)$ and $d = \Theta(a(G))$. It is also well known that any planar graph is 5-degenerate, 3-orientable [4] and has arboricity at most 3.

3 Graphs of Paths of Length 2 and 3

Algorithm 1 described below transforms an input graph G into two graphs G_2 and G_3. The algorithm successively removes vertices of low degrees from graph G, storing in graphs G_2 and G_3 information about paths crossing the deleted vertices. $N(v)$ denotes the set of all neighbors of vertex v in current graph G and $N_0(v)$ denotes the set of neighbors of v in the input graph G. For every edge added to G_2 or G_3 we store the corresponding path of length 2 or 3 respectively. An edge x–$y \in G_2$ associated with a path xvy in G is denoted by $x \overset{v}{—} y$. Similarly, $x \overset{vw}{—} y$ denotes an edge in G_3 associated with path $xvwy$ in G. Observe that several edges joining the same vertices can be added to G_2 (resp. G_3). The graphs G_2 and G_3 are transformed to simple ones in step 12.

Algorithm 1 Transforming G to G_2 and G_3

```
1: $G_2 \leftarrow \emptyset$
2: $G_3 \leftarrow \emptyset$
3: while $G \neq \emptyset$ do
4:   $v \leftarrow$ a vertex with the lowest degree in $G$
5:   for all $x \in N(v)$ do
6:     for all $y \in N_0(v) - N(v)$ do
7:       $G_2 \leftarrow G_2 \cup \{x \overset{v}{—} y\}$
8:     for all $v \overset{w}{—} y \in G_2$ do
9:       $G_3 \leftarrow G_3 \cup \{x \overset{vw}{—} y\}$
10:    delete the edge $v - x$ from $G$
11:  delete the vertex $v$ from $G$
12: Replace multiple edges in $G_2$ and $G_3$ by single ones. For every single edge store all
    the paths corresponding to it.
```

Theorem 1. *Every cycle in the input graph consists of edge disjoint paths corresponding to certain edges in G_2 or G_3, treated as multigraphs.*

Proof. Let G_0 denote the input graph. We will show by the induction on the number of deleted vertices that whenever the algorithm starts or ends the execution of the while loop every cycle in G_0 consists of edge disjoint paths corresponding to certain edges in G_2 or G_3 and of edges in the current graph G.

Before any vertex was deleted the invariant is trivially satisfied, since then $G_0 = G \cup G_2 \cup G_3$. Now we assume that the invariant holds at the beginning of the while loop (statement 4) and we will show that it is still satisfied at the end (after statement 11). Let $C_0 \subseteq G_0$ be an arbitrary cycle. From the induction hypothesis there exists a cycle $C \subseteq G \cup G_2 \cup G_3$ corresponding to C_0 (as a union of edge disjoint paths). We assume that the vertex v chosen in statement 4 is incident with an edge v–$a \in G \cap C$ (otherwise C is not affected and there is nothing to prove). Let $b \in V(C)$ be the neighbor of v in C different from a. There are three cases to consider.

When v–$b \in G$ the edge $a \overset{v}{—} b$ is added to G_2 in statement 7. It is clear that $C' = C - \{v-a, v-b\} \cup \{a \overset{v}{—} b\}$ corresponds to C_0. Similarly when $v-b \in G_2$ the edge a–b is added to G_3 in statement 9. Again we see that $C'' = C - \{v-a, v-b\} \cup \{a - b\}$ corresponds to C_0. Finally, when $v \overset{cd}{—} b \in G_3$ the edge $a \overset{v}{—} c$ is added to G_2 in statement 7. It is easy to observe that both $v \overset{c}{—} d$ and $c \overset{d}{—} b$ are in G_2. Then we see that at the end of the while loop $C''' = C - \{v-a, v \overset{cd}{—} b\} \cup \{a \overset{v}{—} c, c \overset{d}{—} b\} \subseteq G \cup G_2 \cup G_3$ corresponds to C_0. That ends the proof. □

Proposition 1. *Let G be an arbitrary d-degenerate graph and let G_2, G_3 be graphs generated by Algorithm 1. Then at most $2d \cdot |E(G)|$ paths are stored in G_2, and at most $2d \cdot |E(G_2)|$ paths are stored in G_3.*

Proof. Consider a path xvy in G corresponding to an edge e in G_2. Assume that in the moment of adding the path to G_2, $x \in N(v)$ and $y \in N_0(v) - N(v)$. We say that the edge v–y *generates* e. Let us consider an arbitrary edge u–v in graph G. It generates at most $2d$ edges in G_2. Subsequently, at most $2d \cdot |E(G)|$ paths are stored in G_2. Similarly, at most $2d \cdot |E(G_2)|$ paths are stored in G_3. □

Corollary 1. *Let G be an arbitrary d-degenerate graph and let G_2, G_3 be graphs generated by Algorithm 1. Then $|E(G_2)| \leq 2d \cdot |E(G)|$ and $|E(G_3)| \leq 2d \cdot |E(G_2)| \leq 4d^2 \cdot |E(G)|$.*

Corollary 2. *For any d-degenerate graph G, Algorithm 1 can be implemented to work in time $O(d^2 \cdot |E(G)|)$.*

Proof. It suffices to show that step 12. of the algorithm can be implemented in linear time. It can be achieved by radix-sorting all the edges of G_2 and G_3. Assume that the vertices of G form a linear order. Every edge $x - y \in G_2$ with $x < y$, corresponding to a path xvy in G, is transformed into the sequence (x, y, v). All the sequences are sorted in $O(|V(G)| + d \cdot |E(G)|)$ time using radix-sort. Then we create the simple graph G_2 and for every edge we store the set of all paths corresponding to it. Similarly we sort all the 3-paths corresponding to edges in multigraph G_3 and we build a simple graph G_3. □

Corollary 3. *If G is planar then $|E(G_2)| \leq 10|E(G)|$ and $|E(G_3)| \leq 10|E(G_2)|$.*

Lemma 1. *For any planar graph G, graph G_2 generated by Algorithm 1 is a union of at most 20 planar graphs.*

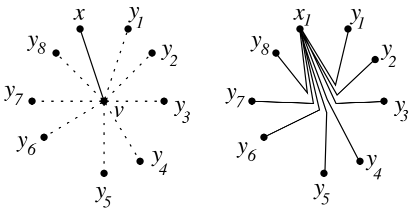
Proof. In the proof we consider multigraph G_2 generated by the algorithm before the execution of step 12.

Let $G = (V, E)$. It is well known that every planar graph is 4-colorable. Let $\mathcal{K} : V \longrightarrow \{1, 2, 3, 4\}$ be a 4-coloring of G. Let us consider a vertex v of degree ≤ 5 chosen by the algorithm in line 4. We can assign different numbers from set $\{1, \dots, 5\}$ to its neighbors. For a neighbor x of v let $no_v(x) \in \{1, \dots, 5\}$ denote its number. We define a partition of G_2,

$$G_2 = \bigcup_{\substack{i\in\{1,\dots,4\}\\ j\in\{1,\dots,5\}}} G_2^{i,j}$$

in which graph $G_2^{i,j}$ contains an edge $x \overset{v}{-} y$ considered in line 7 of the Algorithm 1 when $\mathcal{K}(v) = i$ and $no_v(x) = j$.

Now it suffices to show that each graph $G_2^{i,j}$ is planar. Let us take any plane embedding of G. Consider one of graphs $G_2^{i,j}$. We will show that it has a plane embedding. Every vertex in $G_2^{i,j}$ is embedded in exactly the same point as the corresponding vertex in G. An edge $x - y \in G_2^{i,j}$ corresponding to path $p = xvy$ in G is drawn *close* to the embedding of p in such a way that for given $x, v \in V$ none of the edges corresponding to paths xvy crosses any other edge (see Fig. 1).



graph G graph $G_2^{K(v),1}$

Fig. 1. Drawing edges incident with x.

Consider an arbitrary edge $e = x \overset{v}{-} y \in G_2^{i,j}$. Assume that e crosses another edge $e' = x' \overset{v'}{-} y' \in G_2^{i,j}$. If $v = v'$ either the edges were drawn in such a way that they do not cross or the edges cannot belong to the same graph $G_2^{i,j}$. Subsequently $v \neq v'$ and it remains to consider the case when $v = x'$ (the other ones are symmetric). Then v is adjacent to v' in G. We see that such a case cannot happen since v and v' are assigned different colors implying that both edges e and e' cannot belong to $G_2^{i,j}$.

□

Corollary 4. *Let G_2 be a graph generated by Algorithm 1. Then G_2 has arboricity 60, G_2 is 60-orientable and 120-degenerate.*

4 The Maximum Number of Given Length Cycles

In this short section we show how to use properties of graphs G_2 and G_3 to prove the following theorem.

Theorem 2. *Let $k \geq 3$ be a fixed integer constant. The maximum number of k-cycles in an n-vertex planar graph is $\Theta(n^{\lfloor \frac{k}{2} \rfloor})$.*

Proof. We start from the lower bound. We construct an n-vertex planar graph with $\Omega(n^{\lfloor \frac{k}{2} \rfloor})$ cycles of length k as follows. Assume that k is even. We start from a cycle C of length $\frac{k}{2}$. Then each edge u–v of cycle C is replaced by l paths of length 2 joining u and v. The construction is shown on Fig. 2. Note that $n = (l+1) \cdot \frac{k}{2}$ and $l = \Omega(n)$. It is clear that the resulting graph has $\Omega(l^{\frac{k}{2}}) = \Omega(n^{\lfloor \frac{k}{2} \rfloor})$ cycles of length k as required. When k is odd the construction is similar. We start from a cycle C of length $\lfloor \frac{k}{2} \rfloor + 1$ and we choose an edge $e \in C$. Every edge except e is replaced by l paths of length 2, as before. The resulting graph has $\Omega(l^{\lfloor \frac{k}{2} \rfloor}) = \Omega(n^{\lfloor \frac{k}{2} \rfloor})$ cycles of length k.

The upper bound is an immediate conclusion from Theorem 1 and Proposition 1. Let G be an arbitrary planar graph. It suffices to observe that a cycle of length k in G can consist of at most $\lfloor \frac{k}{2} \rfloor$ paths of length 2 and 3 stored in G_2 and G_3. Since there is $O(n)$ such paths (Prop. 1) and every cycle in G consists of these paths, there can be no more than $O(n^{\lfloor \frac{k}{2} \rfloor})$ cycles of length k in G. □

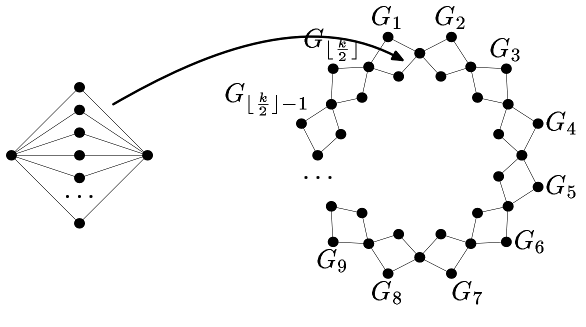


Fig. 2. Construction of an n-vertex graph containing $O(n^{\lfloor \frac{k}{2} \rfloor})$ k-cycles

5 Finding Cycles of Lengths From 3 To 6

In this section we show how to use graphs G_2 and G_3 to find short cycles. Algorithms for cycles of length 3, 4 and 5 apply also to d-degenerate graphs and have linear complexity, provided that $d = O(1)$. In the algorithm for cycles of length 6 we need Lemma 1. Therefore this algorithm apply only to planar graphs. In the rest of the section we assume that we search cycles in d-degenerate graph G of n vertices and m edges.

5.1 Length 3

As an immediate consequence of Theorem 1 we see that it suffices to find self-loops in G_3. Every such self-loop corresponds to certain cycle in G. The algorithm works in time $O(d^2 m)$.

5.2 Length 4

Using Theorem 1 we get that every cycle $xvyw$ of length 4 in G is formed by two paths xvy and xwy corresponding to certain edge $x - y$ in G_2. To find a cycle of length 4 it suffices to find an edge in G_2 that corresponds to more than one path in G. The algorithm works in time $O(d^2 \cdot m)$.

Similarly, if we want to list all 4-cycles in G it suffices to consider, for every edge $x - y \in G_2$, all pairs of paths corresponding to this edge. The algorithm works in time $O(d^2 \cdot m + \#_c)$, where $\#_c$ denotes the number of C_4's in the graph ($\#_c = O(n^2)$).

5.3 Length 5

Theorem 1 implies that every cycle of length 5 in G is formed by two paths of length 2 and 3 corresponding to certain edges x-$y \in G_2$ and x-$y \in G_3$ respectively. We can easily compute $E(G_2) \cap E(G_3)$ in linear time using radix-sort. Observe that two paths of length 2 and 3, corresponding to edges $x - y \in E(G_2)$ and $x - y \in E(G_3)$ respectively, do not necessarily form a cycle. For a path xvy of length 2 there can be even $\Theta(n)$ paths of length 3 joining x and y and crossing vertex v. Luckily the following proposition holds:

Proposition 2. *Any x-y path of length 3 can internally intersect with at most 2 x-y paths of length 2.*

To find one cycle of length 5, for every edge u-v in G_3 and for every path of length 3 in G corresponding to that edge we check at most 3 paths corresponding to edge u-v in G_2. The time will be $O(d^2 \cdot m)$ due to Proposition 1.

In the case when we want to find all 5-cycles, for every two edges, x-$y \in G_2$ and x-$y \in G_3$, we check all the pairs of corresponding paths. At most $O(d^2 m)$ pairs of paths do not form a cycle. Therefore the algorithm works in time $O(d^2 m + \#_c)$, where $\#_c$ denotes the number of C_5's in the graph ($\#_c = O(n^2)$).

It is possible that the algorithm returns one cycle more than once. In order get rid of unnecessary copies without increasing the time complexity it suffices to sort the cycles using radix-sort. The cycle C is transformed to a sequence $(v_1, v_2, v_3, v_4, v_5)$ such that $v_1 v_2 \dots v_5$ are successive vertices of C, $v_1 = \min V(C)$ and $v_2 = \min N_C(v_1)$ ($N_C(v_1)$ denotes the set of neighbors of v_1 in C).

5.4 Length 6

In this section we assume that G is an n-vertex planar graph. From Theorem 1 we know that each cycle of length 6 in G is formed either from two paths corresponding to the same edge in G_3 or it is formed from 3 paths of length 2 such that corresponding edges in G_2 form a triangle in G_2. The algorithm finding one occurrence of C_6 is linear. If we want to list all C_6's our algorithm will work in time $O(n + \#_c)$, where $\#_c$ denotes the number of C_6's in the graph ($\#_c = O(n^3)$). In the latter version of algorithm it is possible that the same cycle is found twice: either as a triple of 2-paths and as two paths corresponding to the same edge in G_3. To have every cycle printed exactly once it suffices to radix-sort all the cycles found in the last phase of the algorithm, similarly as it was shown is Section 5.3.

Finding Triangles in G_2 Since G_2 is $O(1)$-degenerate (see Corollary 4), we can use the algorithm from section 5.1 to find all triangles in linear time. Alternatively we can use any other algorithm finding C_3's in $O(1)$-degenerate graphs in linear time, e. g. [3,4]. Subsequently there are $O(n)$ triangles in G_2. For every such triangle we need to find 2-paths corresponding to its edges that form a 6-cycle in G. Consider a triangle $T = v_1v_2v_3$ in G_2. Observe that we can ignore 2-paths $v_1v_2v_3$, $v_1v_3v_2$, $v_2v_1v_3$. None of them can be a part of 6-cycle corresponding to T. All the other 2-paths corresponding to the edges of T will be called *needed.* For an edge e in triangle T let $Need(e)$ denote the set of needed paths corresponding to e. We check the triples of 2-paths using algorithm 2.

Algorithm 2 Searching for triples of 2-paths that form 6-cycles

```
Sort edges of triangle $T$ so $|Need(e_1)| \leq |Need(e_2)| \leq |Need(e_3)|$.
for all $p \in Need(e_1)$ do
  for all $q \in Need(e_2)$ do
    if $p$ and $q$ are internally disjoint then
      for all $r \in Need(e_3)$ do
        if $p$, $q$ and $r$ form a 6-cycle in $G$ then
          Print the cycle.
```

Proposition 3. *Every $O(1)$ steps Algorithm 2 either returns a 6-cycle or terminates.*

Proof. Let $T = v_1v_2v_3$ be a triangle in G_2. Consider needed paths corresponding to the edges v_1-v_2 and v_2-v_3. Let 2-paths v_1xv_2 and v_2yv_3 correspond to the edges v_1-v_2 and v_2-v_3 respectively. Assume that the paths are not internally disjoint. Since they are needed, $x = y$. Observe that there can be at most 2 pairs of such paths that are not internally disjoint because every 3 such pairs of paths form $K_{3,3}$ in G.

Thus, there can be at most 2 pairs of paths $p \in Need(e_1)$ and $q \in Need(e_2)$ that are not internally disjoint. Consider a pair of internally disjoint paths p and q checked by the algorithm. We can assume that $|Need(e_3)| \geq 3$, because otherwise there would be $O(1)$ triples to check and nothing remains to prove. Subsequently the algorithm can check at most 2 paths $r \in Need(e_3)$ that do not form a 6-cycle with p and q. Thus, when $|Need(e_3)| \geq 3$, among every 3 successively checked triples of paths at least one forms a cycle. □

Corollary 5. *One can find all 6-cycles in G corresponding to triangles in G_2 in time $O(n+\#_c)$ where $\#_c$ denotes the number of all such 6-cycles. The algorithm finds the first cycle in time $O(n)$.*

Checking paths corresponding to the same edge in G_3 Among all k paths corresponding to one edge in G_3 we need to find internally disjoint ones. If $k < 8$ there is $O(1)$ possible pairs and we can verify each of them. Otherwise we need the following proposition:

Proposition 4. *Let e be an edge in G_3 corresponding to $k \geq 8$ different 3-paths in G. There is an algorithm working in $O(k)$ time which either decides that there is no pair of paths corresponding to 6-cycle or returns $\Theta(k)$ cycles of length 6 and at least two paths such that there is no more cycles containing any of that paths.*

Proof. Assume that there is a cycle C compound of two internally disjoint 3-paths, p and q, each corresponding to e. Obviously there can be no more that 8 paths corresponding to e that are neither internally disjoint with p nor with q. For at most 9 paths $p_1, p_2, \dots p_9$ the algorithm verifies whether they form a cycle with all the other paths corresponding to e. Thus, after checking at most $9k$ pairs of paths, the algorithm finds a cycle compound of one of the paths p_i and a path p. Then for every path r corresponding to e, the algorithm checks whether r forms a cycle with p. Observe that the total number of cycles found by the algorithm is at least $k-9$ (at least $k-10$ paths form a cycle either with p_i or with p). Among these cycles are all the cycles containing paths $p_1, p_2, \dots p_i$ and p.

In the case when there is no pair of paths corresponding to e that form a 6-cycle, the algorithm checks $9k$ pairs of cycles and reports that there is no cycle formed by paths corresponding to e. □

Corollary 6. *There is an algorithm which lists all the cycles formed by paths corresponding to the same edges in G_3. Its time complexity is $O(n + \#_c)$. The algorithm finds the first cycle in $O(n)$ time.*

5.5 Longer cycles

From Theorem 1 we see that all cycles of length 7 are formed by two 2-paths and one 3-path corresponding to edges $e_1, e_2 \in G_2$ and to $e_3 \in G_3$, respectively. To find those edges we should search for triangles in graph $G_2 \cup G_3$. Unfortunately we were unable to show that G_3 is $O(1)$-degenerate.

6 Another Algorithm For Finding C_5 In Planar Graphs

In this section we show another linear algorithm for finding C_5's in planar graphs. This algorithm is more simple than the algorithm from section 5. It also seems to be faster in practice (see Section 7). Another important feature is that the new algorithm returns each cycle exactly once (additional sorting is not needed). The algorithm can be easily modified to count the number of 5-cycles in linear time. We extend the approach of Chiba and Nishizeki [3] to the case of cycles of length 5. The algorithm is presented below.

Algorithm 3 Listing all cycles of length 5 in d-degenerate graph

```
 1: Direct the edges of $G$ producing $d$-oriented directed graph $G'$.
 2: Sort the vertices of $G$ in such a way that $d(v_1) \geq d(v_2) \geq \dots d(v_n)$.
 3: for all $v \in V$ do $U(v) \leftarrow \emptyset$
 4: for $i \leftarrow 1$ to $n$ do
 5:   {finding all 5-cycles containing $v_i$}
 6:   for all $u \in N(v_i)$ do
 7:     for all $w' \in N(u) - \{v_i\}$ do
 8:       $U(w') \leftarrow U(w') \cup \{u\}$
 9:   for all $u \in N(v_i)$ do
10:     for all $w \in N(u) - \{v_i\}$ do
11:       for all $w \rightarrow x \in E(G')$ do
12:         if $x \neq u$ then
13:           for all $y \in U(x)$ do
14:             if $y \notin \{u, w\}$ then
15:               Print out the cycle $v_i uwxy$.
16:   for all $w$ such that $U(w) \neq \emptyset$ do
17:     $U(w) \leftarrow \emptyset$
18:   Delete $v_i$ from $G$.
```

Lemma 2 (Chiba, Nishizeki [3]). *Let G be a graph of m edges with arboricity a. Then*

$$\sum_{u-v \in E} \min\{d(u), d(v)\} \leq 2am$$

Proposition 5. *Algorithm 3 correctly lists all 5-cycles in d-degenerate n-vertex graph G in $O(d^2 \cdot m + \#_c)$ time, where $\#_c$ denotes the number of 5-cycles in G. Moreover, the algorithm finds first cycle (or reports that there is no one) in $O(d^2 \cdot m)$ time.*

Proof. It is straightforward that for every vertex v_i the algorithm lists all the 5-cycles containing v_i. This is achieved by finding edges joining vertices distant by 2 from v_i (statement 11). We will focus on the time complexity.

Statement 1 of the algorithm can be easily implemented to work in linear time. It suffices to successively choose a vertex v of degree at most d, make the edges incident with v outgoing from v in G' and mark them as deleted in G.

Consider a vertex v_i and its neighbor u at the beginning of the "for" loop. Let G_0 be the input graph and let G be the current version of G_0. Then $d_G(u) \leq d_{G_0}(u) = \min\{d_{G_0}(u), d_{G_0}(v)\}$. Subsequently, from Lemma 2, the variable w is assigned at most $4dm$ times in statement 10 (G has arboricity at most $2d$). Since G' is d-oriented the variable x is assigned at most $4d^2m$ times (statement 11). Moreover, among all checked sequences of vertices (v, u, w, x, y) at most $8d^2m$ sequences does not form a cycle. That ends a proof. □

Algorithm 3 can be easily adapted to count the number of 5-cycles in linear time. In this case $U(w)$ will denote the number of 2-paths from v_i to w. In algorithm 3 whenever a 3-path $p = v_i–u–w \to x$ is found we have to find those 2-paths $v_i–x$ stored in $U(x)$ which do not intersect with p (statement 14). We can easily get rid of this time-consuming fragment. Before checking all the 3-paths of the form $v_i–u\ldots$ we decrease the number $U(x)$ for all neighbors of u. As a result, when we find a 3-path $p = v_i–u–w \to x$ there is exactly $U(x)$ 2-paths from v_i to x that do not intersect with u. Then if w is adjacent to v_i exactly one of these paths intersects p (in vertex w). If w is not adjacent to v_i none of these paths intersects p. Thus in the first case we find $U(x) - 1$ paths that form 5-cycles with p and in the latter case $U(x)$ ones. See Algorithm 4 for details.

Proposition 6. *Algorithm 4 correctly counts cycles of a given n-vertex planar graph in $O(n)$ time.*

Algorithm 4 Counting cycles of length 5 in d-degenerate graph

```
$\#_c \leftarrow 0$
Direct the edges of $G$ producing $d$-oriented directed graph $G'$.
Sort the vertices of $G$ in such a way that $d(v_1) \geq d(v_2) \geq \ldots d(v_n)$.
for all $v \in V$ do $U(v) \leftarrow 0$
for $i \leftarrow 1$ to $n$ do
    for all $u \in N(v_i)$ do
        for all $w' \in N(u) - \{v_i\}$ do $U(w') \leftarrow U(w') + 1$
    for all $u \in N(v_i)$ do
        for all $w \in N(u) - \{v_i\}$ do $U(w) \leftarrow U(w) - 1$
        for all $w \in N(u) - \{v_i\}$ do
            for all $w \to x \in E(G')$ do
                if $x \neq u$ then
                    if $w \to v_i \in E(G')$ or $v_i \to w \in E(G')$ then
                        $\#_c \leftarrow \#_c + U(x) - 1$
                    else
                        $\#_c \leftarrow \#_c + U(x)$
        for all $w \in N(u) - \{v_i\}$ do $U(w) \leftarrow U(w) + 1$
    for all $w$ such that $U(w) \neq 0$ do $U(w) \leftarrow 0$
    Delete $v_i$ from $G$.
```

7 Experimental Results

Since the motivation of this paper was to give algorithms that can be used in practice in this section we show some of the experimental results concerning presented methods. The algorithms were tested on random 10^5-vertex triangulation generated by Donald Knuth's *Stanford GraphBase*. The computations were done on a Pentium II 500 MHz computer with 2 GB of memory. Below you can see time results for almost all the algorithms presented in the paper as well as the number of cycles found ($\#_c$). The algorithm for 6-cycles searches for triangles in graph G_2 using the algorithm by Chiba and Nishizeki [3].

algorithm	C_3	C_4	C_5	C_6	C_5 (Alg. 3)	counting C_5 (Alg. 4)
time [s]	28.03	20.70	51.55	93.43	26.94	5.89
$\#_c$	201,136	315,333	689,705	2,065,391	689,705	689,705

8 Acknowledgments

We would like to thank Krzysztof Diks for valuable comments on the preliminary version of this paper. Thanks go also to other members of Algorithms and Complexity Group for fruitful discussions on these results during seminar.

References

1. N. Alon, R. Yuster, U. Zwick, *Color-coding: a new method for finding simple paths, cycles and other small subgraphs within large graphs*, Proc. 26th ACM Symp. on Theory of Computing, 326-335, 1994.
2. N. Alon, R. Yuster, U. Zwick, *Finding and counting given length cycles*, Proc. European Symp. Algorithms, 354-364, 1994.
3. N. Chiba, T. Nishizeki, *Arboricity and subgraph listing algorithms*, SIAM J. Computing, **14**, 210-223, 1985.
4. M. Chrobak, D. Eppstein, *Planar orientations with low out-degree and compaction of adjacency matrices*, Theoretical Computer Science, **86**, 243-266, 1991.
5. D. Eppstein, *Subgraph isomorphism in planar graphs and related problems*, J. Graph Algorithms and Applications **3(3)**, 1-27, 1999.
6. G. Fijavz, M. Juvan, B.Mohar, R. Skrekovski, *Planar graphs without cycles of specified lengths*, European J. Combin., **23**, 377-388, 2002.
7. Ł. Kowalik, M. Kurowski, *Short path queries in planar graphs in constant time*, Proc. 35th ACM Symp. on Theory of Computing, 143-148, 2003.
8. B. Monien, *How to find long paths efficiently*, Ann. Disc. Math., **25**, 239-254, 1985.
9. C. H. Papadimitriou, M. Yannakakis, *The clique problem for planar graphs*, Inform. Proc. Lett. **13**, 131-133, 1981.
10. D. Richards, *Finding short cycles in planar graphs using separators* J. Algorithms, **7**, 382-394, 1986.
11. C. Thomassen, *3-list coloring planar graphs of girth 5*, Journal of Combinatorial Theory, Series B, **64**, 101-107, 1995.