Short Path Queries in Planar Graphs in Constant Time

[Extended Abstract]

Łukasz Kowalik
Institute of Informatics
Warsaw University
Banacha 2, 02-097 Warsaw, Poland
kowalik@mimuw.edu.pl

Maciej Kurowski
Institute of Informatics
Warsaw University
Banacha 2, 02-097 Warsaw, Poland
kuros@mimuw.edu.pl

ABSTRACT

We present a new algorithm for answering short path queries in planar graphs. For any fixed constant k and a given unweighted planar graph $G = (V, E)$ one can build in $O(|V|)$ time a data structure, which allows to check in $O(1)$ time whether two given vertices are distant by at most k in G and if so a shortest path between them is returned. This significantly improves the previous result of D. Eppstein [5] where after a linear preprocessing the queries are answered in $O(\log |V|)$ time. Our approach can be applied to compute the girth of a planar graph and a corresponding shortest cycle in $O(|V|)$ time provided that the constant bound on the girth is known.

Our results can be easily generalized to other wide classes of graphs – for instance we can take graphs embeddable in a surface of bounded genus or graphs of bounded tree-width.

Categories and Subject Descriptors

G.2.2 [**Discrete Mathematics**]: Graph Theory—*graph algorithms, path and circuit problems*

General Terms

Algorithms

Keywords

planar graph, shortest path, queries

1. INTRODUCTION

Computing shortest paths in planar graphs is a fundamental problem with numerous practical applications – for an extensive survey see [1, 7]. It is also often used as a black box in other graph algorithms. In this case the complexity of computing the shortest paths has often a crucial influence on the overall complexity.

STOC'03, June 9–11, 2003, San Diego, California, USA.

Recently the following problem attracts a lot of attention: for a given n-vertex planar graph, construct a data structure which allows to process quickly the queries concerning shortest paths between vertices. Results in this area are often given in a form of a trade-off between the preprocessing time and the time needed to answer the shortest path query. For instance the following scheme is shown in [2]: the preprocessing is performed in $O(n + p^2/\sqrt{r} + pr^{\frac{3}{4}})$ time where p depends on a graph structure ($p = O(n)$) and r is an arbitrarily chosen parameter in the range: $1 \leq r \leq p$; the query time is $O(\sqrt{r}log(r) + \alpha(n))$ where $\alpha(n)$ is the inverse of the Ackermann's function.

1.1 New Results

We study the following version of the shortest path problem. We fix a constant integer k. For an unweighted planar graph G (without given plane embedding) we describe a preprocessing which takes time $O(n)$. Then we provide a query algorithm which checks in $O(1)$ time whether two given vertices are distant by at most k and if so it computes a corresponding shortest path. The query algorithm searches for paths in a structure computed in the preprocessing phase. The structure can be updated in $O(1)$ time after deleting any edge of G thus it can work in a dynamic environment.

For any fixed constant k our approach can be used to construct a linear time algorithm which verifies whether a given graph has girth at most k and if so it computes the corresponding shortest cycle.

1.2 Related Work

The shortest path problem described above was considered previously by David Eppstein [5]. His approach uses also a linear preprocessing but the queries are answered in $O(\log n)$ time. The algorithm of Eppstein combine methods of tree decomposition with clustering techniques of Frederickson [6] while our algorithms are much simpler and direct. In the same paper, Eppstein presents another algorithm, based on tree decomposition technique, which uses $O(n \log n)$ time preprocessing and the queries are processed in $O(1)$ time. Both algorithms of Epptein need a plane embedding of the input graph.

The results of D. Eppstein [5] can be used to compute the girth of a bounded value in $O(n)$ time. We obtain exactly the same result but using simpler methods. If the constant bound on the girth is not known the problem seems to be harder – the best known algorithm is due to H. Djidjev [4] and runs in $O(n^{5/4} \log n)$ time.

1.3 Generalizations

In this paper we focus on the class of planar graphs because it is particularly important in applications and offers well-known terminology. Nevertheless our results can be further generalized. It is very easy to observe that for arbitrary constant number g instead of planar graphs we can take any class of graphs embeddable in a surface of genus g. All the results and proofs remain almost unchanged. Moreover all introduced notions and proofs can be adapted to any class $\mathcal{C}$ of graphs satisfying the following three conditions:

(i) $\mathcal{C}$ is sparse, i.e. there exists a constant c such that for every $G \in \mathcal{C}$, $|E(G)| \leq c|V(G)|$.

(ii) $\mathcal{C}$ is closed under taking subgraphs.

(iii) If a graph G from $\mathcal{C}$ is a subdivision of a graph H, then $H \in \mathcal{C}$.

Observe that every class of sparse graphs closed under taking minors satisfies the above conditions. For example the graphs of bounded tree-width form such a class.

2. KEY IDEAS

We start with some basic definitions. A plane graph is a planar graph $G = (V, E)$ together with its fixed planar drawing. In fact we assume that the set of graph vertices $V(G)$ is a set of points in the plane. When we refer to an edge of a plane graph we can consider it as a pair of vertices (v, w) or as a curve drawn in the plane. This ambiguity should not lead to a confusion because it is always clear from the context which meaning is used.

A graph in which edges are assigned directions is called a digraph. For a vertex v in a digraph G we define the *outdegree* as the number of edges outgoing from v and the *indegree* as the number of edges ingoing into v. We denote these values by $in\_deg_G(v)$ and $out\_deg_G(v)$ respectively. A digraph is *d-oriented* when the outdegree of each vertex is bounded by d.

It is well known that the edges of any planar graph G can be oriented in linear time, obtaining a directed graph G_1, in such a way that the maximum outdegree in G_1 is bounded by 5 (this constant can be decreased to 3, see [3]). Then u–$v \in E(G)$ iff $u \to v \in E(G_1)$ or $v \to u \in E(G_1)$. Thus after a linear preprocessing we can process the queries of the form:"Are vertices v and w adjacent?" in $O(1)$ time.

We generalize this result to much wider class of queries, i.e. we can check in $O(1)$ time whether given vertices are distant by at most k and if so we can compute a shortest path between them.

In this section we focus on the case $k = 2$ which reveals some key ideas of our approach. The general case is considered in the latter sections. Assume that G_1 is a planar 3-oriented digraph and v, w are vertices distant by 2. There are 3 cases to be considered excluding the symmetrical ones (see Fig. 1).

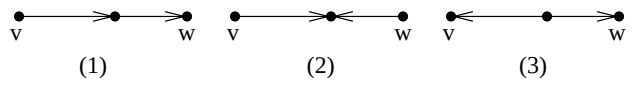

Figure 1: Possible cases of directing a 2-path.

As one can see the cases 1 and 2 are easy to detect in $O(1)$ time. The main obstacle is to detect the last case because the indegrees of v and w can be arbitrarily large. In order to resolve this problem we extend G by adding edges of weight 2 between each pair of vertices v and w which match the third case. The graph compound of edges with weight 2 is denoted by G_2. Formally v–$w \in G_2$ iff there exists a vertex $x \in V(G_1)$ such that $x \to v \in E(G_1)$ and $x \to w \in E(G_1)$. The vertex x is said to *support* the edge v–w. The above construction can be described more conveniently in terms of a special operation $\odot$ defined in the latter section.

Our goal is to show that the graph G_2 is a union of 15 plane graphs, hence it can be 45-oriented and subsequently the third case can be detected in $O(1)$ time.

We start with 5-coloring of the vertices of G and then we divide graph G_2 into 5 subgraphs. Each of them contains edges supported by vertices from the same color class. Each of the subgraphs can be further partitioned into 3 plane graphs (since G is 3-oriented, see Fig. 2). Hence G_2 is a union of 15 plane graphs. Moreover each of these graphs is a *subdrawing* of G: It can be drawn in such a way that its drawing is completely contained in the drawing of G. In another words its edges correspond with paths in G. The notion of subdrawing plays a central role in our proof.

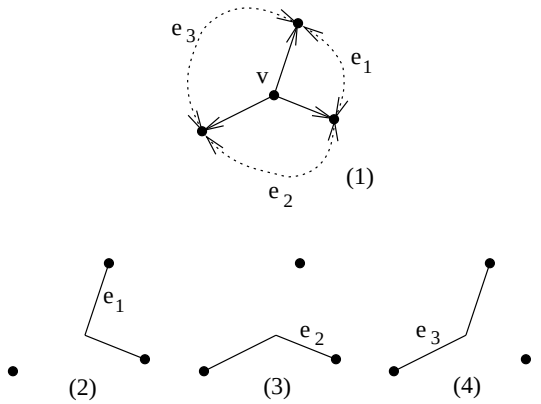

Figure 2: Edges e_1, e_2, e_3 supported in G_2 by v (1) are partitioned into 3 plane graphs – (2), (3), (4).

The reasoning described above can be extended to any fixed natural number k. We build successively graphs: $G_2, G_3, \ldots, G_k$. An edge u–v belongs to G_t iff there exists $x \in V(G)$ such that $x \to u \in E(G_i)$ and $x \to v \in E(G_j)$ for some i, j satisfying $i + j = t$. The edges in G_t have weights equal to t. In order to assure that the scheme works properly the following conditions should be satisfied:

(i) Each graph G_i is a union of a constant number of plane graphs and therefore it can be $O(1)$-oriented. Notice that after such an orientation only a constant number of vertices is reachable from any vertex v by paths of weight at most k in the graph $G_1 \cup G_2 \cup \ldots \cup G_k$ (this set is referred as $Reach_k(v)$).

(ii) For vertices v and w which are distant by at most k a shortest path between v and w is completely contained in the subgraph of G induced by the set of vertices $Reach_k(v) \cup Reach_k(w)$.

The first constraint is satisfied as a consequence of Corollary 2 while the other one follows from Theorem 3. They are presented in the latter sections.

3. PRELIMINARIES

In this section we give necessary definitions and we show some useful facts concerning the introduced notions.

Consider plane graphs G and H. We call H a *subdrawing* of G when the following two conditions are satisfied: *(i)* $V(H) \subseteq V(G)$; *(ii)* each edge in graph H is a curve compound of a sequence of edges of G (see Fig. 3). We call H a *t-subdrawing* of G, if H is a subdrawing of G and each edge is a concatenation of at most t edges of G.

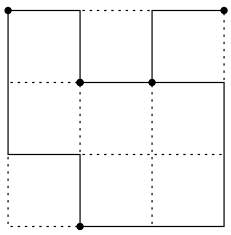

Figure 3: An exemplary subdrawing of the 4×4 mesh.

Let G and H be undirected graphs. If H can be decomposed into s t-subdrawings of G then such a decomposition is called *(G,t)-representation of H of thickness s*. In the case when H is directed (G,t)-representation of H of thickness s is a decomposition of H into s 3-oriented t-subdrawings of G.

Let G and H be a pair of plane digraphs. We define a relation on the set of edges of G. We say that an edge $e_1 = (v,w) \in E(G)$ *points with H* to a different edge $e_2 \in E(G)$ when there exists an edge $h = (v,x) \in H$ that intersects the interior of e_2 (see Fig. 4).

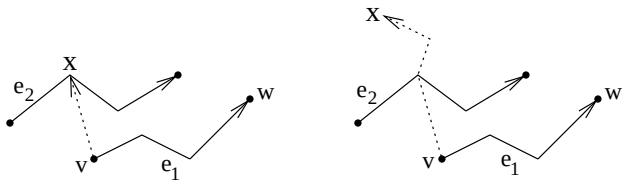

Figure 4: Edge e_1 points to edge e_2.

The following lemma binds the notions of the subdrawing and the pointing relation.

LEMMA 1. *Let G be a fixed plane graph. Let A and B be 1-oriented, plane digraphs such that A is subdrawing of G and B is b-subdrawing of G. Let e be an edge in graph A. Then e points with B to at most b edges in A.*

PROOF. Suppose that $e = (v,w)$ points to $b' > b$ edges. The outdegree of v in B is equal 1. Consider the only edge h in B outgoing from v. Edge h intersects the interiors of b' edges of A. Because both A and B are subdrawings of G the only candidates for the intersection points are the vertices of G. Due to assumption that B is a b-subdrawing of G there are at most $b+1$ vertices of G on the edge h. Therefore we have only b candidates for intersection points (the vertex v is excluded). It is a contradiction because some edges of A would have to share a common internal point and A would not be planar. □

LEMMA 2. *Every c-oriented graph can be vertex $(2c+1)$-colored in linear time.*

PROOF. Let G be an arbitrary c-oriented graph. We use induction on the number of vertices. Since $\sum_v in\_deg(v) = \sum_v out\_deg(v)$ there exists in G a vertex v such that $in\_deg(v) \leq out\_deg(v)$. Thus $deg(v) \leq 2c$. We can color inductively $G - \{v\}$ and assign to v a free color. □

4. OPERATION $\odot$

Consider an operation $\odot$ which takes as its arguments two digraphs G, H. The result is an undirected simple graph. Its vertex set is $V(G) \cup V(H)$ and two distinct vertices u and v are connected in $G \odot H$ iff there exists a vertex x such that $x \to u \in E(G)$ and $x \to v \in E(H)$. We say that an edge u–v is *supported by x*. Observe that one edge can be supported by many vertices.

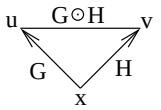

Figure 5: An edge supported by x in operation $G \odot H$.

We will show a few useful facts concerning the introduced operation.

THEOREM 1. *Let G be a plane graph. Let A and B be plane, 1-oriented digraphs. Moreover let A and B be a-subdrawing and b-subdrawing of G respectively. The numbers a and b are bounded by a fixed constant k. Then graph $A \odot B$ has a $(G, a+b)$-representation of thickness $5 \cdot (2a+1)(2b+1)$ which can be computed in linear time.*

PROOF. From Lemma 1 each edge of A points with B to at most b other edges. In another words the set of edges of A and the pointing relation define a b-oriented digraph. By Lemma 2 we can $(2b+1)$-color the edges of A in such a way that if one edge points to another then they are given different colors. Hence we can partition A in linear time into $2b+1$ graphs:

$$A = A_1 \cup A_2 \cup \ldots \cup A_{2b} \cup A_{2b+1},$$

each formed by the edges from the same color class. Similarly we can divide graph B in linear time into $2a+1$ graphs:

$$B = B_1 \cup B_2 \cup \ldots \cup B_{2a} \cup B_{2a+1}.$$

The above decomposition guarantees that in each graph A_i (respectively B_j) there is no edge which points to another edge with B (respectively A). In order to compute $(G, a+b)$-representation of $A \odot B$ we use the following formula:

$$A \odot B = \bigcup_{\substack{1 \leq i \leq 2b+1 \\ 1 \leq j \leq 2a+1}} A_i \odot B_j.$$

Now it suffices to show that each component $A_i \odot B_j$ is a union of at most five $(a+b)$ - subdrawings of G. To this end

we need one more decomposition. Consider an arbitrary graph $A_i \cup B_j$. Observe that it is 2-oriented. Applying Lemma 2 we get that $A_i \cup B_j$ is 5-colorable. The vertices from the same color class are neither connected in A_i nor in B_j.

Let c be one of the color classes. Denote by $A_{i,c}$ graph A_i restricted to the edges outgoing from the vertices colored c. Similarly we denote by $B_{j,c}$ graph B_j restricted to the edges outgoing from the vertices colored c. This partition can be done in linear time. Of course:

$$A_i \odot B_j = \bigcup_{1 \le c \le 5} A_{i,c} \odot B_{j,c}.$$

Notice that the latest decomposition guarantees that for each vertex $v \in A_{i,c} \cup B_{j,c}$, $out\_deg(v) > 0$ implies $in\_deg(v) = 0$.

Now it suffices to show that every graph $R_{i,j,c} = A_{i,c} \odot B_{j,c}$ is a $(a+b)$-subdrawing of G. According to the definition of the operation $\odot$ two distinct vertices v and w are adjacent in $R_{i,j,c}$ iff there exists a vertex x such that $x \to v$ is an edge in $A_{i,c}$ and $x \to w$ is an edge in $B_{j,c}$. Denote by x' the last point on the edge $x \to v$ shared with $x \to w$ (possibly $x' = x$). Clearly we can draw an edge $v - w$ as a concatenation of drawings $x' \to v$ (we call this part of the edge $v - w$ an A-part) and $x' \to w$ (we call this part of the edge $v - w$ a B-part). In the resulting graph $R_{i,j,c}$ each edge is compound of at most $a+b$ edges of G. It remains to prove that $R_{i,j,c}$ is a plane graph. Assume on the contrary that the interior of edge $e_1 \in E(R_{i,j,c})$ is intersected by another edge $e_2 \in E(R_{i,j,c})$. There are several cases to be considered:

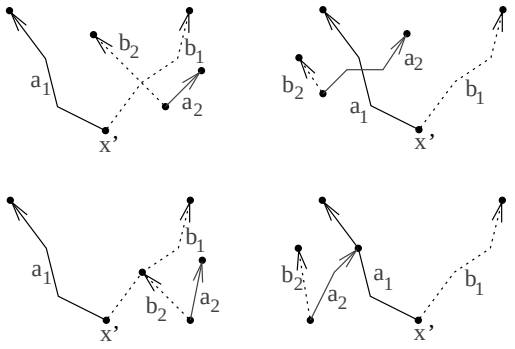

Figure 6: Case 1.

(i) The interior of the A-part of e_1 is intersected by the A-part of e_2. This is impossible because $A_{i,c}$ is planar. Similarly it is impossible that the interior of the B-part of e_1 is intersected by the B-part of e_2 (see Fig. 6).

(ii) The interior of the A-part of e_1 is intersected by the B-part of e_2. Denote the edges in $A_{i,c}$ corresponding to the A-parts of e_1 and e_2 as a_1 and a_2 respectively. Denote the edges in $B_{j,c}$ corresponding to the B-parts of e_1 and e_2 as b_1 and b_2 respectively. We have assumed that the interior of a_1 is intersected by b_2. It means that a_2 points with B to a_1. A contradiction. We can also exclude the symmetrical case: the interior of the B-part of e_1 is intersected by the A-part of e_2 (see Fig. 7).

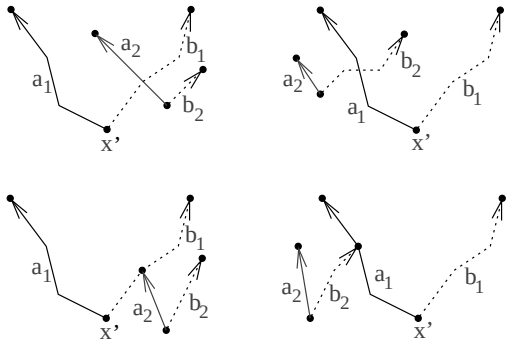

Figure 7: Case 2.

(iii) The common endpoint x' of the A-part and B-part of e_1 is intersected by e_2. This is impossible because indegree of x' is 0 in $R_{i,j,c}$.

We have shown that no internal point of e_1 (which was an arbitrarily chosen edge) can be intersected by another edge. This completes the proof of planarity of $R_{i,j,c}$.

The set of all graphs in decomposition of $A \odot B$ is a $(G, a+b)$-representation of $A \odot B$:

$$\{R_{i,j,c} : 1 \le i \le 2b+1;\ 1 \le j \le 2a+1;\ 1 \le c \le 5\}.$$

□

COROLLARY 1. *Let G be a plane graph. Let A and B be plane, 3-oriented digraphs. Moreover let A and B be a-subdrawing and b-subdrawing of G respectively. The numbers a and b are bounded by a fixed constant k. Then $A \odot B$ has a $(G, a+b)$-representation of thickness $45 \cdot (2a+1)(2b+1)$ which can be computed in linear time.*

PROOF. We decompose A into a union of 1-oriented subdrawings of G: A_1, A_2, A_3. Similarly we decompose B into a union of 1-oriented subdrawings of G: B_1, B_2, B_3. Notice that:

$$A \odot B = \bigcup_{1 \le i,j \le 3} A_i \odot B_j.$$

Now we apply Theorem 1 to each of 9 components of the union. □

COROLLARY 2. *Let G be a n-vertex plane graph. Let A be a graph with given (G, a)-representation of thickness s_a and let B be a graph with given (G, b)-representation of thickness s_b. The numbers a, b are bounded by a fixed constant k. Then the graph $A \odot B$ has a $(G, a+b)$-representation of thickness $45 \cdot s_a \cdot s_b \cdot (2a+1) \cdot (2b+1)$. Moreover such a representation can be computed in $O(n \cdot s_a \cdot s_b)$ time.*

PROOF. Let $A = A_1 \cup A_2 \cup \ldots \cup A_{s_a}$ and $B = B_1 \cup B_2 \cup \ldots \cup B_{s_b}$ be (G, a)-representation and (G, b)-representation of graphs A and B berespectively.

Notice that the following holds:

$$A \odot B = \bigcup_{i=1}^{s_a} \bigcup_{j=1}^{s_b} A_i \odot B_j.$$

It suffices to apply Corollary 1 to each component of the union. □

Since in the following sections we operate on planar graphs without given plane embeddings, we need to define (G,t)-representation of a graph H in the case when G is planar. Let G and H be planar graphs. H is *t-subdrawing* of G when there exist plane graphs G' and H' such that they are plane embeddings of G and H respectively and H' is t-subrawing of G'. Now the notion of (G,t)-representation is well defined even if G is a planar graph. Observe that all the results from this section hold also if we consider planar graphs instead of plane ones.

5. THE SHORTCUT GRAPH STRUCTURE

In this section we assume that k is a fixed constant and G is a planar graph with n vertices. We describe a data structure $S_k(G)$ called *a shortcut graph of* G with the following properties:

(i) construction of $S_k(G)$ takes $O(n)$ time and space,

(ii) for arbitrary vertices u and v one can check in $O(1)$ time whether u and v are distant by at most k' in graph G ($k' \leq k$).

5.1 Construction

$S_k(G)$ is a directed multigraph with the vertex set $V(G)$. Edges have integer weights from set $\{1 \dots k\}$. We denote by G_i the subgraph of $S_k(G)$ containing all edges with weight i. The undirected version of G_i is denoted as G_i^u.

We build $S_k(G)$ by constructing successively the graphs $G_1, G_2, \dots G_k$. For each graph G_i we compute and maintain its (G,i)-representation. This is achieved by repeating alternately the following two steps:

5.1.1 Step 1 – Computation of G_i^u

The initial graph G_1^u is simply equal to G. For $i > 1$ we compute the graph G_i^u applying operation $\odot$ to the previously constructed graphs:

$$G_i^u = \bigcup_{\substack{1 \leq p \leq q < i \\ p+q=i}} G_p \odot G_q$$

Each of the operations $G_p \odot G_q$ is performed separately by applying Corollary 2. As the result we obtain a (G,i)-representation of the graph G_i^u. Notice that the thickness of the representation is a function of i but independent of n, therefore it is $O(1)$.

5.1.2 Step 2 – Directing the Edges of G_i^u

We are given the graph G_i^u and its (G,i)-representation of thickness $O(1)$. Edges of each member of the representation are directed by applying the following result:

THEOREM 2 (CHROBAK, EPPSTEIN [3]). *Every planar graph can be 3-oriented in linear time.*

After directing all edges of G_i^u we obtain the resulting graph G_i. Notice that we have computed also a relevant (G,i)-representation for G_i. The thickness of this representation is $O(1)$ therefore the outdegrees in G_i are also bounded by $O(1)$.

COROLLARY 3. *The structure $S_k(G)$ satisfies the following conditions:*

(i) Edge $u \to v$ in $S_k(G)$ with weight t indicates that there is a walk in G from u to v of length t.

(ii) The graph $S_k(G)$ is d-oriented, $d = O(1)$.

(iii) The construction of $S_k(G)$ takes time $O(n)$.

5.2 Processing the Queries

Weight of a path p in $S_k(G)$ is the sum of weights of all edges that form p. A *t-path* is a path of weight less or equal t. We denote by $Reach_t(v)$ the set of vertices which are reachable in $S_k(G)$ from v via t-paths. For any directed graph D the statement $u - v \in E(D)$ means that $u \to v \in E(D)$ or $v \to u \in E(D)$. We start from the following theorem:

THEOREM 3. *Let G be a planar graph and let $S_k(G)$ be a shortcut graph of G. For any $l \leq k$, vertices u and v are distant by at most l in G if and only if the set $Reach_t(u) \cap Reach_{l-t}(v)$ is not empty for some $t \in \{0, \dots l\}$.*

PROOF. The implication ($\leftarrow$) is obvious. We show the other one. The proof is by the induction on l. As $u - v \in E(G_1)$ the theorem holds for $l = 1$.

If the distance between u and v is less than l we can use the induction hypothesis so we assume that $dist(u,v) = l$. Let p be a u-v path in G of length l. Let u_1 be the neighbor of u on the path p. By the induction hypothesis there is an integer t_1, $0 \leq t_1 < l$, and a vertex x such that $Reach_{t_1}(u_1) \cap Reach_{l-1-t_1}(v) \ni x$. Hence there are directed paths in $S_k(G)$ from u_1 to x and from v to x.

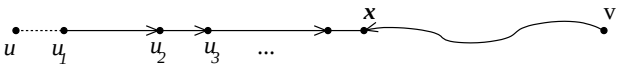

Figure 8: Proof of Theorem 3.

Let $u_1, u_2, \dots u_n = x$ be successive vertices of $u_1 \to x$ path in $S_k(G)$. We define a set Z as follows:

$$Z = \{i : 1 \leq i \leq n \text{ and } u_i - u \in E(S_k(G))\}$$

As $u - u_1 \in E(G_1)$, $u_1 \in Z$ and $Z \neq \emptyset$. Let $i^* = \max Z$. If $i^* = n$, i.e. $u_{i^*} = x$ we are done, because either $x \in Reach_{t_1+1}(u)$ or $u \in Reach_l(v)$. Assume $i^* < n$. Let $u_{i^*} \to u_{i^*+1}$ has weight a. If $u_{i^*} \to u \in E(S_k(G))$ and it has weight b, $u_{i^*+1} - u \in E(G_{a+b}) \subseteq E(S_k(G))$ and $i^* + 1 \in Z$, a contradiction. Finally $u \to u_{i^*} \in E(S_k(G))$ and $x \in Reach_{t_1+1}(u)$. □

5.2.1 The Algorithm

Theorem 3 yields a simple $O(1)$ time algorithm which verifies whether $dist(u,v) \leq k$ and if so it computes a relevant shortest path between u and v. As it was proved the outdegree in the shortcut graph $S_k(G)$ is bounded by $O(1)$. Let us denote the maximum outdegree in $S_k(G)$ by max_out_deg:

$$max\_out\_deg = max\{out\_deg(v) : v \in S_k(G)\}.$$

The computation of $Reach_t(u)$, for $t = 1, 2 \dots k$, can be done easily using dynamic programming: for every vertex $x \in Reach_i(u)$, $i < t$, and every edge $x \to y$ with weight $(t - i)$, we add vertex y to $Reach_t(u)$. Simultaneously we can build $T(u)$ – the tree of the shortest paths from u to the vertices of $Reach_t(u)$ in the graph $S_k(G)$. As one can see the tree $T(u)$ contains at most $max\_out\_deg^k = O(1)$ vertices. Similarly we build the shortest-paths tree $T(v)$ for the vertex v.

Theorem 3 shows that if $dist(u,v) = l \leq k$ then there exists a vertex $x \in S_k(G)$ such that $dist_{S_k(G)}(u,x) + dist_{S_k(G)}(v,x) = l$. Therefore it is enough to check all the vertices from $T(v) \cap T(u)$ and find a vertex x for which the value $dist_{S_k(G)}(u,x)+dist_{S_k(G)}(v,x)$ is minimal. The shortest path between u and v in G can be easily reconstructed in $O(1)$ time from the following shortest paths in $S_k(G)$ – the path from u to x and the path from v to x (we obtain the paths from the trees $T(u)$ and $T(v)$ resp.).

6. DYNAMIC EDGES DELETIONS AND THE GIRTH

An important asset of our data structure is that it can be adapted to work in a dynamic environment. We show in this section that after deleting an arbitrary edge from G the shortcut graph $S_k(G)$ can be updated in $O(1)$ time.

The idea behind the edge deletion algorithm is very simple. Each time we delete an edge e in G we also have to remove all the edges in G_2 which were caused by e. To make the shortcut graph up to date we have also to delete recursively unnecessary edges from graphs $G_3, \ldots, G_k$. To this end we have to answer two questions: *(i)* how numerous can be the set of deleted edges; *(ii)* how to find those edges quickly. It can be seen that the number of deleted edges is $O(1)$ and they can be found in $O(1)$ time.

To be more precise we introduce the following definitions: Let $e_1 = x \rightarrow v$, $e_2 = x \rightarrow w$ be edges in graphs G_i, G_j respectively. Let e be an edge of G_{i+j} equal $v \rightarrow w$ or $w \rightarrow v$. We say that the pair of edges (e_1, e_2) is a *parent* of e and e is a *child* of (e_1, e_2). Notice that each edge can have more than one parent.

6.1 The Extension of the Shortcut Graph

We assume that during the construction of $S_k(G)$ for each edge $e \in S_k(G)$ the following information are stored: *(i)* the number of parents of e; *(ii)* the list $L(e)$ of pairs (p_i, c_i) where c_i is a child of (p_i, e).

THEOREM 4. *Let G be a planar graph and $S_k(G)$ a shortcut graph for G. Each edge $e = v \rightarrow w \in S_k(G)$ has $O(1)$ children.*

PROOF. Let us denote the maximum outdegree in $S_k(G)$ by max_out_deg (recall that it is bounded by a constant number). The number of the edges outgoing from v different from e is at most $max\_out\_deg - 1$. It implies that the number of children of e is bounded by $max\_out\_deg - 1 = O(1)$. □

Assume than e is an edge of G_t and it is going to be deleted. For each pair $(p_i, c_i) \in L(e)$ we have to perform the following operations: remove the pair (e, c_i) from the list $L(p_i)$, decrease the counter of parents of c_i. If the counter reaches 0 we delete the edge c_i recursively. As each edge has $O(1)$ children and the depth of recursion is bounded by a constant k the whole operation takes $O(1)$ time. In order to delete an edge from the initial graph G it is enough to delete the directed edge from G_1.

6.2 The Girth of a Planar Graph

Let k be a fixed constant and G a planar graph. In order to check whether the girth of G is $\leq k$ we perform the following three operations for each edge $e = (v, w) \in G$:

(i) delete e from G;

(ii) check whether $dist(v, w) \leq k-1$ and if so compute the cycle compound of e and the shortest path between v and w;

(iii) restore the original state of $S_k(G)$ (undo the deletion of e).

Each step can be performed in $O(1)$ time. Therefore in a case when the girth of G is at most k the shortest cycle in G can be found in linear time.

7. ACKNOWLEDGEMENTS

We thank Krzysztof Diks for comments on early versions of this paper and many fruitful discussions.

8. REFERENCES

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice Hall, Englewood Cliffs, 1993.

[2] D. A. Chen and J. Xu. Shortest path queries in planar graphs. In *Proc. of the 32nd Annual ACM Symposium on Theory of Computing*, pages 469–478, 2000.

[3] M. Chrobak and D. Eppstein. Planar orientations with low out-degree and compaction of adjacency matrices. *Theoretical Computer Science*, 86(2):243–266, 1991.

[4] H. Djidjev. Computing the girth of a planar graph. In *Proc. 27th International Colloquium on Automata, Languages and Programming*, pages 821–831, 2000.

[5] D. Eppstein. Subgraph isomorphism in planar graphs and related problems. *J. Graph Algorithms & Applications*, 3(3):1–27, 1999.

[6] G. N. Frederickson. Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees. *SIAM Journal on Computing*, 26(2):484–538, April 1997.

[7] G. Ramalingam. *Bounded Incremental Computation*, volume 1089 of *LNCS*. Springer, 1996.