Exact Distance Oracles for Planar Graphs

Shay Mozes
Brown University

Christian Sommer
MIT

May 11, 2011

Abstract

We present new and improved data structures that answer exact node-to-node distance queries in planar graphs. Such data structures are also known as *distance oracles*. For any directed planar graph on n nodes with non-negative lengths we obtain the following: [1]

- Given a desired space allocation $S \in [n \lg\lg n, n^2]$, we show how to construct in $\tilde{O}(S)$ time a data structure of size $O(S)$ that answers distance queries in $\tilde{O}(n/\sqrt{S})$ time per query.
  As a consequence, we obtain an improvement over the fastest algorithm for k–many distances in planar graphs whenever $k \in [\sqrt{n}, n)$.
- We provide a linear-space exact distance oracle for planar graphs with query time $O(n^{1/2+\epsilon})$ for any constant $\epsilon > 0$. This is the first such data structure with provable sublinear query time.
- For edge lengths ≥ 1, we provide an exact distance oracle of space $\tilde{O}(n)$ such that for any pair of nodes at distance ℓ the query time is $\tilde{O}(\min\{\ell, \sqrt{n}\})$. Comparable query performance had been observed experimentally but could not be proven.

Our data structures are based on the following new tool: given a non-self-crossing cycle C with $c = O(\sqrt{n})$ nodes, we can preprocess G in $\tilde{O}(n)$ time to produce a data structure of size $O(n \lg\lg c)$ that can answer the following queries in $\tilde{O}(c)$ time: for a query node u, output the distance from u to all the nodes of C. This data structure builds on and extends a related data structure of Klein (SODA'05), which reports distances to the boundary of a face, rather than a cycle.

The best distance oracles for planar graphs until the current work are due to Cabello (SODA'06), Djidjev (WG'96), and Fakcharoenphol and Rao (FOCS'01). For $\sigma \in (1, 4/3)$ and space $S = n^\sigma$, we essentially improve the query time from n^2/S to $\sqrt{n^2/S}$.

[1] Asymptotic notation as in $\tilde{O}(\cdot)$ suppresses polylogarithmic factors in the number of nodes n.

1 Introduction

A fast shortest-path query data structure may be of use whenever an application needs to compute shortest path distances between some but not all pairs of nodes. Indeed, shortest-path query processing is an integral part of many applications, in particular in Geographic Information Systems (GIS) and intelligent transportation systems [JHR96]. These systems may help individuals in finding fast routes or they may also assist companies in improving fleet management, plant and facility layout, and supply chain management. A challenge for traffic information systems or public transportation systems is to process a vast number of queries on-line while keeping the space requirements as small as possible [Zar08]. Low space consumption is obviously very important when a query algorithm is run on a system with heavily restricted memory such as a handheld device [GW05] but it is also important for systems with memory hierarchies [HMZ03, AT05], where caching effects can have a significant impact on the query time.

While many road and public transportation networks are actually not exactly planar [EG08, AFGW10], they still share many properties with planar graphs; in particular, many road networks appear to have small separators as well. For this reason, planar graphs are often used to model various transportation networks.

In the following, we provide shortest-path query data structures (distance oracles) for planar graphs for essentially *any* specified space requirement. Throughout the paper we assume the edge lengths to be non-negative.[2] Our results extend a result of Cabello [Cab06] and improve upon a result of Djidjev [Dji96].

Theorem 1. *Let G be a directed planar graph on n vertices. For any value S in the range $S \in [n \lg\lg n, n^2]$, there is a data structure with preprocessing time $O(S \lg^3 n / \lg\lg n)$ and space $O(S)$ that answers distance queries in $O(nS^{-1/2} \lg^2 n \lg^{3/2} \lg n)$ time per query.*

As a corollary, we obtain the following result on k–many distances, improving upon Cabello [Cab06], who proves that the problem can be solved in time $\tilde{O}((kn)^{2/3} + n^{4/3})$. Our result is an improvement for $k = \tilde{o}(n)$. For the range roughly $k \in [\sqrt{n}, n)$, our algorithm is faster by a factor of $\tilde{O}((n/k)^{2/3})$.

Theorem 2. *Let G be a directed planar graph on n vertices. The distances between $k = \Omega(n^{1/2} \lg n / \lg\lg n)$ pairs of nodes $(s_1, t_1), (s_2, t_2), \ldots (s_k, t_k)$ can be computed in time $O((kn)^{2/3} (\lg n)^{7/3} (\lg\lg n)^{2/3})$.*

We also give a data-structure that does keep the space requirements as small as possible, i.e. linear in the size of the input. This is the first linear-space data structure with provable sublinear query time for exact point-to-point shortest-path queries. It has come to our attention that Nussbaum [Nus10] has simultaneously obtained a similar result.

Theorem 3. *For any directed planar graph G with non-negative arc lengths and for any constant $\epsilon > 0$, there is a data structure that supports exact distance queries in G with the following properties: the data structure can be created in time $O(n \lg n)$, the space required is $O(n)$, and the query time is $O(n^{1/2+\epsilon})$.*

The main techniques we use are Frederickson's r–division [Fre87], Fakcharoenphol and Rao's implementation of Dijkstra's algorithm [FR06], and Klein's Multiple-Source Shortest Paths (MSSP) data structure [Kle05], for which we propose a more general and more space-efficient alternative.

Theorem 4. *Given a directed planar graph G on n nodes and a simple cycle C with $c = O(\sqrt{n})$ nodes, there is an algorithm that preprocesses G in $O(n \lg^3 n)$ time to produce a data structure of size $O(n \lg c)$ that can answer the following queries in $O(c \lg^2 c \lg\lg c)$ time: for a query node u, output the distance from u to all the nodes of C.*

Since Klein's MSSP data structure has found quite a few applications, we believe that our data structure could be a useful tool in other algorithms as well.

Experimental results suggest that query times proportional to the shortest-path length are possible in practice using algorithms based on the so-called *arc-flag* technique [Lau04, KMS05, HKMS09]. [3] Hilger, Köhler, Möhring, and Schilling make a statement about the (experimental) worst-case behavior of their method:

[2] Our results apply to graphs with negative-length edges by using reduced lengths induced by a feasible price function [Joh77]. The current best bound for computing a feasible price-function in a planar graph is $O(n \lg^2 n / \lg\lg n)$ [MWN10]

[3] The preprocessing algorithm of this technique first partitions the graph into regions V_i and thereafter labels each edge e for all regions V_i with a boolean *flag* $sp_i(e)$ indicating whether e lies on any shortest path to V_i. At query time, only edges leading towards the target region need to be considered.

> In all cases, the search space of our arc-flag method is never larger than ten times the actual number of nodes on the shortest paths [HKMS09, Section 6].

We can now actually *prove* a similar statement:

> In (provably) all cases, the search space of our method is never larger than a polylogarithmic factor times the length ℓ of the shortest paths.

More precisely:

Theorem 5. *For any planar graph G with edge lengths ≥ 1 there is an exact distance oracle of space $O(n \lg n \lg\lg n)$ with query time $O(\min\{\ell \lg^2 \ell \lg\lg \ell, \sqrt{n}\lg^2 n\})$ for any pair of nodes at distance ℓ. The preprocessing time is at most $O(n^{1+\epsilon})$ for any constant $\epsilon > 0$.*

Note that our data structure has query time proportional to the path *length*. In fact, our algorithm maintains a *Bellman-Ford-type* invariant: after iteration i, the distance represents the minimum path length among all paths on $\tilde{O}(i)$ edges — the correct distance is computed after time roughly proportional to the minimum number of edges on a shortest path but we can only guarantee correctness after time $\tilde{O}(\ell)$. If we may further assume that, for some constant $\epsilon > 0$, all $s-t$ paths of length at most $(1+\epsilon)d_G(s,t)$ have $\Omega(h_G(s,t))$ edges (where $h_G(s,t)$ denotes the number of edges (*hops*) on a minimum-hop shortest-path), then our data structure can be constructed to have query time proportional to the minimum number of edges on a shortest path $\tilde{O}(h_G(s,t))$. This assumption essentially means that any $s-t$ path with significantly fewer edges than the shortest path is much longer. Graphs and weight functions considered in practice, in particular those in [HKMS09], appear to satisfy this assumption.

Our main contributions can be summarized as follows: *i)* We significantly improve the worst-case behavior of previously known distance oracles with low space requirements (in particular, we also provide the first one with linear space and sublinear query time), and *ii)* we also make an important step towards proving the behavior observed in practice. As our main tool, *iii)* we provide a more general multiple-source shortest-path data structure with many potential applications.

1.1 Related Work

Shortest-path query processing for planar graphs have been studied extensively. In this section, we give a brief review of previous results.

For exact shortest-path queries, the currently best result in terms of the tradeoff between space and query time is by Fakcharoenphol and Rao [FR06]. Their data structure of space $\tilde{O}(n)$ can be constructed in time $\tilde{O}(n)$ and processes queries in time $\tilde{O}(\sqrt{n})$. The preprocessing time and space can be improved by logarithmic factors [KMW10, Kle05, MWN10]. Note that, in these three articles [FR06, KMW10, MWN10], the main objective is actually a fast single-source shortest path algorithm for planar graphs with real (potentially negative) edge weights. Only [FR06] specifically discuss distance oracles, but the results of [KMW10, MWN10] immediately imply improvments to the distance oracle in [FR06]. In this work, we focus on the data structure and its space–query time tradeoff. Note that, when choosing $\epsilon = 1/\lg n$, our result (Theorem 3) slightly improves upon the product of space times query time in [FR06, KMW10, MWN10].

Some distance oracles have better query times. Djidjev [Dji96] proves that for any $S \in [n, n^2]$ there is an exact distance oracle with preprocessing time $O(S)$ (which increases to $O(n\sqrt{S})$ for $S \in [n, n^{3/2}]$), space $O(S)$, and query time $O(n^2/S)$. For a smaller range, he also proves that for any $S \in [n^{4/3}, n^{3/2}]$ there is an exact distance oracle with preprocessing time $O(nS^{1/2})$, space $O(S)$, and query time $\tilde{O}(nS^{-1/2})$. Chen and Xu [CX00], extending the range, prove that for any $S \in [n^{4/3}, n^2]$ there is an exact distance oracle using space $O(S)$ with preprocessing time $O(n\sqrt{S})$ and query time $\tilde{O}(n/\sqrt{S})$. Cabello [Cab06], mainly improving the preprocessing times, proves that for any $S \in [n^{4/3}, n^2]$ there is an exact distance oracle with preprocessing time and space $O(S)$ with query time $\tilde{O}(n/\sqrt{S})$. Compared to Djidjev's construction, the query time is slower by a logarithmic factor but the range for S is larger. In our construction, we sacrifice another root-logarithmic factor in the query time but we prove the bounds for essentially the whole range of S. See Figure 1 for a summary of known results in comparison with ours.

If constant query time is desired, storing a complete distance matrix is almost optimal. For unweighted graphs, Wulff-Nilsen [WN10a] recently improved the space requirements to $o(n^2)$. If the space is restricted

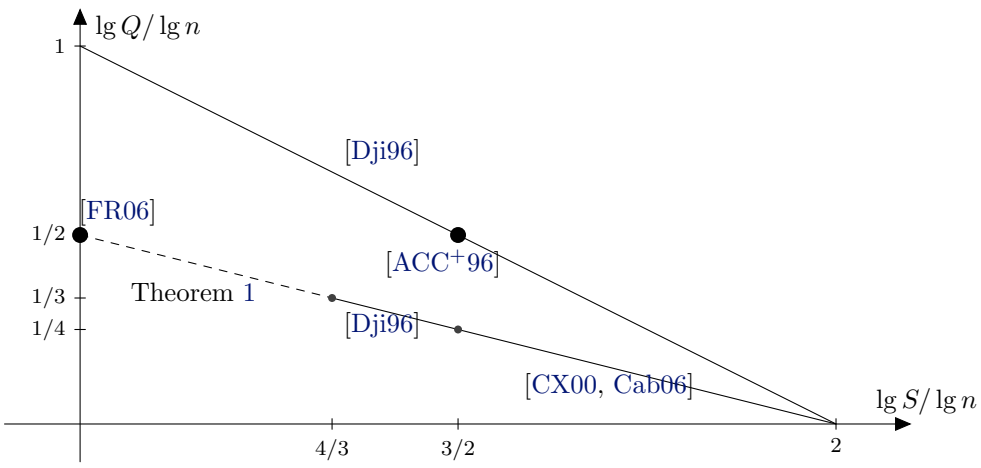

Figure 1: Tradeoff of the Space [S] vs. the Query time [Q] for different shortest-path query data structures on a doubly logarithmic scale, ignoring constant and logarithmic factors. The upper line represents the $Q = n^2/S$ tradeoff (completely covered by Djidjev [Dji96]; Arikati et al. [ACC+96] cover the point $S = n^{3/2}$; SSSP ($S = n$) and APSP ($S = n^2$) also lie on this line). The lower line represents the $Q = n/\sqrt{S}$ tradeoff; the result of Djidjev [Dji96] covers the range $S \in [n^{4/3}, n^{3/2}]$; Chen and Xu [CX00] and Cabello [Cab06] extend this to $S \in [n^{4/3}, n^2]$. Fakcharoenphol and Rao [FR06] cover the point $S = n$. We extend their results to the full range $S \in [n, n^2]$.

to linear, using the linear-time single-source shortest path algorithm of Henzinger et al. [HKRS97] is the fastest known for exact shortest paths queries until the current work. It has come to our attention that Nussbaum [Nus10] has simultaneously obtained a result similar to Theorem 3.

Efficient data structures for shortest-path queries have also been devised for restricted classes of planar graphs [DPZ00, CX00] and for restricted types of queries [Epp99, KK06, Sch98, Kle05]. If approximate distances and shortest paths are sufficient, (poly-)logarithmic query time has been achieved [Tho04, Kle02, Kle05, KKS11, Som11].

Based on separators, geometric properties, and other characteristics such as highway structures, many efficient practical methods have been devised [GSSD08, SS05, BFSS07], their time and space complexities are however difficult to analyze. Competitive worst-case bounds have been achieved under the assumption that actual road networks have small *highway dimension* [AFGW10]. While our preprocessing algorithm (Theorem 5) runs in almost linear time, some of the problems that appear in the preprocessing stage of practical route planning methods have recently been proven to be NP-hard [BDDW09, BCK+10].

2 Preliminaries

2.1 Recursive r–division of Planar Graphs

Let $G = (V, E)$ be a planar graph with $|V| = n$. Let E_P be a subset of the edges of G, and let $P = (V_P, E_P)$ be the subgraph of G induced by E_P. P is called a *piece* of G. The nodes of V_P that are incident in G to nodes of $V \setminus V_P$ are called the *boundary nodes* of P and denoted by ∂P.

An *r–division* [Fre87] of G is a decomposition into $O(n/r)$ edge-disjoint pieces, each with $O(r)$ nodes and $O(\sqrt{r})$ boundary nodes. We use an r–division with the additional property that, in each piece, there exists a constant number of faces, called *holes*, such that every boundary node is incident to some hole. Such a decomposition can be found in $O(n \lg r + nr^{-1/2} \lg n)$ [WN10b] by applying Miller's cycle separator [Mil86] iteratively.

We use this r–division recursively. Denote the base of the recursion as level 0, and the top of the recursion as level k. G is defined to be the only piece at level k. The pieces of level i of the recursion are obtained by computing an r_i–division for each level-$(i + 1)$ piece. The notation r_i suggests that we may use a different

parameter r in the r–division at every level of the recursion. Indeed, using a non-uniform recursion is important in obtaining Theorem 3. For a level-i piece P, the level-$(i-1)$ pieces obtained by applying the r–division to P are called the *subpieces* of P.

We stress that the classification of nodes of a piece at *any* level as boundary nodes is with respect to G (and not P). This implies that if v is a boundary node of a level-i piece, then v is also a boundary node of any lower level piece that contains v. This generalizes the decomposition used by Fakcharoenphol and Rao [FR06]. In that work, Miller's separator is used at each level rather than an r–decomposition.

2.2 Klein's Multiple-Source Shortest Paths Algorithm

Klein [Kle05] gave a multiple-source shortest path (MSSP) algorithm with the following properties. The input consists of a directed planar embedded graph G with non-negative arc-lengths, and a face f. For each node u in turn on the boundary of f, the algorithm computes (an implicit representation of) the shortest path tree rooted at u. This takes a total of $O(n \lg n)$ time and space. Subsequently, the distance between any pair (u, v) of nodes of G where u is on the boundary of f, can be queried in $O(\lg n)$ time. If the set of queries is known in advance, then the space requirement is $O(n)$. In particular, given a set S of $O(\sqrt{n})$ nodes on the boundary of a single face, the algorithm can compute all S-to-S distances in $O(n \lg n)$ time and $O(n)$ space.

We propose a more general and slightly more space-efficient alternative in Section 4, wherein we also provide a detailed comparison.

2.3 Dense Distance Graphs and Efficient Implementation of Dijkstra's Algorithm

The *dense distance graph* for a piece P, denoted DDG_P, is the complete graph on ∂P, the boundary nodes of P, such that the length of an arc corresponds to the distance (in P) between its endpoints. The dense distance graph for all pieces P in the r–division can be computed in $O(|G| \lg |G|)$ time and space using Klein's MSSP; For each piece P, compute MSSP data-structures in $O(|P| \lg |P|)$ time and space a constant number of times, specifying a different hole of P as the distinguished face at each run. Then query the MSSP data-structures for the distances between the boundary nodes in $O(|\partial P|^2 \lg |P|)$ time. Since in an r–division $|\partial P| = \sqrt{|P|}$, this takes $O(|P| \lg |P|)$ time and space per piece, for a total of $O(|G| \lg |G|)$ for all pieces.

Let $\mathcal{P}$ be a set of pieces (not necessarily at the same level), and let H be the union of the dense distance graphs of the pieces in $\mathcal{P}$. Fakcharoenphol and Rao [FR06] devised an ingenious implementation of Dijkstra's algorithm [Dij59] that computes a shortest path tree in H in time $O(|H| \lg^2 n)$, where $|H|$ is the number of nodes in H (i.e., the total number of boundary nodes in all the pieces in $\mathcal{P}$). We will refer to this implementation as *FR-Dijkstra*.

In fact, the proof of Fakcharoenphol and Rao's algorithm only relies on the property that the distances in each of the dense distance graphs given as input correspond to distances in a planar graph between a set of nodes that lie on a constant number of faces. It does not rely on any other properties of the r–decomposition.

FR-Dijkstra can be extended to the following setting (cf. [BSWN10]). Let J be a planar graph. Let n' denote the number of nodes of $J \cup H$. We can compute shortest paths in $H \cup J$ in $O(|H| \lg^2 |H| + n' \lg n')$ time. The edges of H are relaxed using the efficient data structure of Fakcharoenphol and Rao, while the edges of J are relaxed as in a traditional implementation of Dijkstra's algorithm using a heap.

2.3.1 External Dense Distance Graphs

Let $G - P$ be the graph obtained from G by deleting all nodes that belong only to P (i.e., all nodes of P that are not boundary nodes). The *external* dense distance graph for a piece P, denoted DDG_{G-P}, is defined to be the complete graph on ∂P such that the length of an arc corresponds to the distance in $G - P$ between its endpoints. External dense distance graphs were used recently in [BSWN10]. Computing the external dense distance graphs for all pieces in an r–division cannot be done efficiently using Klein's MSSP. The reason is that $|G - P|$ can be $\Theta(|G|)$ for all pieces. Instead, the computation can be done in a top-down approach as follows (cf. [BSWN10]). Recall that an r–division is obtained as the set of pieces of the deepest level

in a recursive division of the graph using Miller's simple cycle separators. Consider the set $\mathcal{Q}$ of all pieces at all levels of the recursion (rather than just the set of pieces at the deepest level). Note that there are $O(\lg |G|)$ recursive levels since the size of the pieces decreases by a constant factor every constant number of applications of Miller's cycle separator theorem. First, we compute DDG_Q for every piece $Q \in \mathcal{Q}$. As explained above, this can be done in $O(|G| \lg |G|)$ for all the pieces in a specific level of the recursion, for a total of $O(|G| \lg^2 |G|)$ for all pieces in $\mathcal{Q}$. Next, we consider a piece Q and denote the two subpieces of Q by Q_1 and Q_2. DDG_{G-Q_2} is obtained by computing distances in $DDG_{Q_1} \cup DDG_{G-Q}$ (see Figure 2), using multiple applications of FR-Dijkstra, once for each node in ∂Q_2. This takes $O(|Q| \lg^2 |\partial Q|)$ per piece, for an overall $O(|G| \lg^2 |G|)$ time for all pieces in a specific level. Since the number of levels is bounded by $\lg |G|$, the entire computation takes $O(|G| \lg^3 |G|)$ time.

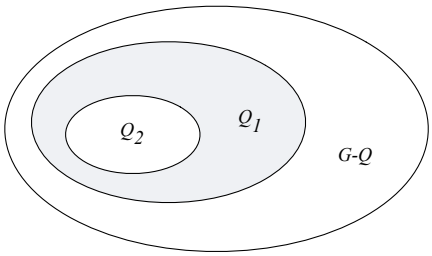

Figure 2: Pieces Q_1 and Q_2 in some level of the recursive application of Miller's cycle separator theorem. The piece Q is the union of Q_1 and Q_2. Distances in $G - Q_2$, the exterior of Q_2 are obtained by considering shortest paths in the interior of Q_1 and in $G - Q$, the exterior of Q.

3 A Linear-Space Distance Oracle

We first provide our linear-space data structure. The techniques used to construct and query our distance oracle are reused in the other more general constructions, in particular in the cycle MSSP data structure (Section 4).

In the following, we prove Theorem 3. We restate a more precise version.

Theorem 3. *For any directed planar graph G with non-negative arc lengths and for any constant $\epsilon > 0$, there is a data structure that supports exact distance queries in G with the following properties: the data structure can be created in time $O(n \lg n)$, the space required is $O(n)$, and the query time is $O(n^{1/2+\epsilon})$.*

For non-constant ϵ, the preprocessing time is $O(n \lg(n) \lg(1/\epsilon))$, the space required is $O(n \lg(1/\epsilon))$, and the query time is $O(n^{1/2+\epsilon} + n^{1/2} \lg^2(n) \lg(1/\epsilon))$.

Our distance oracle is an extension of the oracle in Fakcharoenphol and Rao [FR06]. The main ingredients of our improved space vs. query time tradeoff are *i)* instead of recursively using cycle separators, we use recursive r–divisions, and *ii)* we use an *adaptive* recursion, where the ratio between the boundary of a piece at level $i-1$ and the size of a piece at level i equals $\sqrt{n}$ (which is the query time we aim for).

We split the proof into descriptions and analysis of the preprocessing and query algorithms. Let $k = \Theta(\lg(1/\epsilon))$.

Preprocessing In the preprocessing step we compute the recursive r–division of the graph with k recursive levels and values of $\{r_i\}_{i=0}^{k}$ to be specified below. This takes $O(kn \lg n)$ time.

We then compute the dense distance graph for each piece. This is done for a piece P, with r nodes and $O(\sqrt{r})$ boundary nodes on a constant number of holes, by applying Klein's MSSP algorithm [Kle05] as described in Section 2.3. Thus, all of the boundary-to-boundary distances in P are computed in $O(r \lg r)$ time. Summing over all $O(n/r_i)$ pieces at level i, the preprocessing time per level is $O(n \lg r_i)$. The overall time to compute the dense distance graphs for all pieces over all recursive levels is therefore bounded by $O(kn \lg n)$.

The space required to store DDG_P is $O((\sqrt{r})^2) = O(r)$; summing over all pieces at level i we obtain space $O(\frac{n}{r_i} r_i) = O(n)$ per level; the total space requirement is $O(kn)$.

Query Given a query for the distance between nodes u and v, we proceed as follows.

For simplicity of the presentation, we initially assume that neither u nor v are boundary nodes.

Let P_0 be the level-0 piece that contains u. We compute distances from u in P_0. This is done in $O(r_0)$ time using the algorithm of Henzinger et al. [HKRS97]. Denote these distances by $\text{dist}_{P_0}(u, w)$. Let H_0 denote the star graph with center u and leaves ∂P_0. The arcs of H_0 are directed from u to the leaves, and their lengths are the corresponding distances in P_0.

Let S_u be the set of pieces that contain u. Note that S_u contains exactly one piece of each level. Let R_u be the union of subpieces of every piece in S_u. That is, $R_u = \bigcup_{P \in S_u} \{P' : P' \text{ is a subpiece of } P\}$. Let H_u be the union of the dense distance graphs of the pieces in R_u. We use FR-Dijkstra (see Section 2.3) to compute distances from u in $H_u \cup H_0$. Observe that any shortest path from u to a node of H_u can be decomposed into a shortest path in P_0 from u to ∂P_0 and shortest paths each of which is between boundary nodes of some piece in R_u. Since all u-to-∂P_0 shortest paths in P_0 are represented in H_0, and since all shortest paths between boundary nodes of pieces in R_u are represented in H_u, this observation implies that distances from u to nodes of H_u in $H_u \cup H_0$ are equal to distances from u to nodes of H_u in G. We denote these distances by $\text{dist}_G(u, w)$.

We repeat a similar procedure for v (reversing the direction of arcs) to compute $\text{dist}_G(w, v)$, the distances in G from every node of H_v to v.

Let P_{uv} be the lowest level piece that contains both u and v. Assume first that P_{uv} is not a level-0 piece. Let P_u (P_v) be the subpiece of P_{uv} that contains u (v). Since P_{uv} is both in S_u and in S_v, both P_u and P_v are in R_u as well as in R_v. This implies that we have already computed $\text{dist}_G(u, w)$ and $\text{dist}_G(w, v)$ for all $w \in \partial P_u$. Since we have assumed that P_{uv} is not a level-0 piece, the shortest u-to-v path must contain some node of ∂P_u. Therefore, the u-to-v distance can be found by computing

$$\min_{w \in \partial P_u} \text{dist}_G(u, w) + \text{dist}_G(w, v).$$

If P_{uv} is a level-0 piece, then $P_{uv} = P_0$, and the u-to-v distance can be found by computing

$$\min \left\{ \text{dist}_{P_0}(u, v), \min_{w \in \partial P_{uv}} \{\text{dist}_G(u, w) + \text{dist}_G(w, v)\} \right\}.$$

The case when u or v are boundary nodes is a degenerate case that can be solved by the above algorithm. Let Q_u be the highest level piece of which u is a boundary node. We have the preprocessed distances in Q_u from u to all other nodes of ∂Q_u. Therefore, it suffices to replace S_u above with the set of pieces that contain Q_u as a subgraph in order to assure that H_u is small enough and that the distances computed by the fast implementation of Dijkstra's algorithm are the distances from u to nodes of H_u in G.

Query Time We next analyze the query time. Computing the distances dist_{P_0} takes $O(r_0)$ time. Let $|H_u|$ denote the number of nodes of H_u. Fakcharoenphol and Rao's Dijkstra implementation runs in $O(|H_u| \lg^2 n)$ time. It therefore remains to bound $|H_u|$. Let P_i be the level-i piece in S_u. P_i has $O(\frac{r_i}{r_{i-1}})$ subpieces, each with $O(\sqrt{r_{i-1}})$ boundary nodes. Therefore, the contribution of P_i to $|H_u|$ is $O(\frac{r_i}{\sqrt{r_{i-1}}})$. The total running time is therefore

$$r_0 + \lg^2 n \sum_{i=1}^{k} \frac{r_i}{\sqrt{r_{i-1}}}.$$

Recall that $r_k = |G| = n$, and set $r_0 = \sqrt{n}$. For $i = 1 \ldots k-1$ we recursively define r_i so as to satisfy

$$\frac{r_i}{\sqrt{r_{i-1}}} = \sqrt{n}.$$

This implies

$$\begin{aligned} r_1 &= n^{\frac{1}{2}+\frac{1}{4}} = n^{1-\frac{1}{4}} \\ r_2 &= n^{\frac{1}{2}+\frac{3}{8}} = n^{1-\frac{1}{8}} \\ r_2 &= n^{\frac{1}{2}+\frac{7}{16}} = n^{1-\frac{1}{16}} \\ &\ldots \\ r_{k-1} &= n^{1-\frac{1}{2^k}}. \end{aligned}$$

The total running time is thus bounded by

$$\sqrt{n} + \lg^2 n \left((k-1)\sqrt{n} + \frac{n}{\sqrt{n^{1-\frac{1}{2^k}}}} \right) \leq \left(k\sqrt{n} + n^{\frac{1}{2}+\frac{1}{2^{k+1}}} \right) \lg^2 n. \tag{1}$$

By $k = \Theta(\lg(1/\epsilon))$ we obtain the claimed running times.

4 A Cycle MSSP Data Structure for Planar Graphs

In this section we provide our main technical tool. We prove Theorem 4, which we restate here.

Theorem 4. *Given a directed planar graph G on n nodes and a simple cycle C with $c = O(\sqrt{n})$ nodes, there is an algorithm that preprocesses G in $O(n \lg^3 n)$ time to produce a data structure of size $O(n \lg \lg c)$ that can answer the following queries in $O(c \lg^2 c \lg \lg c)$ time: for a query node u, output the distance from u to all the nodes of C.*

Comparison with Klein's MSSP data structure Our data structure can be seen as an alternative to Klein's MSSP data structure (see Section 2.2) with two main advantages (which we exploit in Section 5):

- our data structure can handle queries to an arbitrary not-too-long cycle as opposed to a single face, and
- the space requirements are only $O(n \lg \lg n)$ (even $O(n)$ is possible at the cost of increasing the query time) as opposed to $O(n \lg n)$,

and three main disadvantages:

- our data structure cannot efficiently answer queries from u to a *single* node on the cycle C; such a query requires the same time as computing the distances from u to all the nodes on C,
- our data structure requires amortized time $O(\lg^2 c \lg \lg c)$ per node on the cycle, as opposed to $O(\lg n)$ per node, which is slower for long cycles, and
- the preprocessing time of our data structure is $O(n \lg^3 n)$ as opposed to $O(n \lg n)$.

Preprocessing Let G_0 be the exterior of C. That is, the graph obtained from G by deleting all nodes strictly enclosed by C. Consider C as the infinite face of G_0. Similarly, Let G_1 be the interior of G. Namely, the graph obtained from G by deleting all nodes not enclosed by C. Consider C as the infinite face of G_1. The preprocessing step consists of the following:

1. Computing DDG_C and DDG_{G-C}. This can be done in $O((n + c^2) \lg n)$ time using Klein's MSSP algorithm [Kle05]. Storing DDG_C and DDG_{G-C} requires $O(c^2) = O(n)$ space.
2. For $i \in \{0, 1\}$, computing an r–division of G_i with $r = c^2$. Each piece has $O(c^2)$ nodes and $O(c)$ boundary nodes incident to a constant number of holes. Consider the nodes of C as boundary nodes of every piece in the division (each piece still has $O(c)$ boundary nodes). This step takes $O(n \lg n)$ time.
3. Computing, for each piece P, a recursive r–division of P identical to the one that would be computed in the preprocessing step of the oracle in Section 3 with $\epsilon = 1/\lg c$. That is, the number of levels in this recursive decomposition is $k = \Theta(\lg \lg c)$. The top level (level k) piece in this recursive decomposition is the entire piece P. In the description in Section 3, the top level piece is the entire graph and therefore it has no boundary nodes. Here, in contrast, we consider the boundary nodes of P as boundary nodes of the top-level piece in the decomposition (and thus, as boundary nodes of any lower-lever piece in which they appear). This does not asymptotically change the total number of boundary nodes at any level of the recursive decomposition since P has c boundary nodes, and every level of the recursive decomposition consists of a total of $\Omega(c)$ boundary nodes. The time to compute the recursive r–division for all pieces is bounded by $O(n \lg^2 n)$.

4. Computing, for each piece P, the dense distance graph for each of the pieces in the recursive decomposition of P. Let H_P denote the union of the dense distance graphs for all the pieces in the recursive decomposition of P. As discussed in Section 3, the space required to store H_P is $O(n \lg \epsilon) = O(n \lg \lg c)$. Using the methods presented in Section 3, computing H_P takes $O(|P| \lg |P| \lg \lg c) = O(c^2 \lg c \lg \lg c)$. This leads to a total running time of $O(n \lg c \lg \lg c)$ for computing H_P for all pieces P.

5. Computing, for each piece P, the dense distance graph $DDG_{G_i - P}$. Recall that we consider the nodes of C as boundary nodes of *every* piece in the division. These dense distance graphs can be computed as described in Section 2.3.1. As shown there, the entire computation (for all pieces combined) takes $O(n \lg^3 n)$ time.

The time required for the preprocessing step is therefore $O(n \lg^3 n)$ and the space required is $O(n \lg \lg c)$.

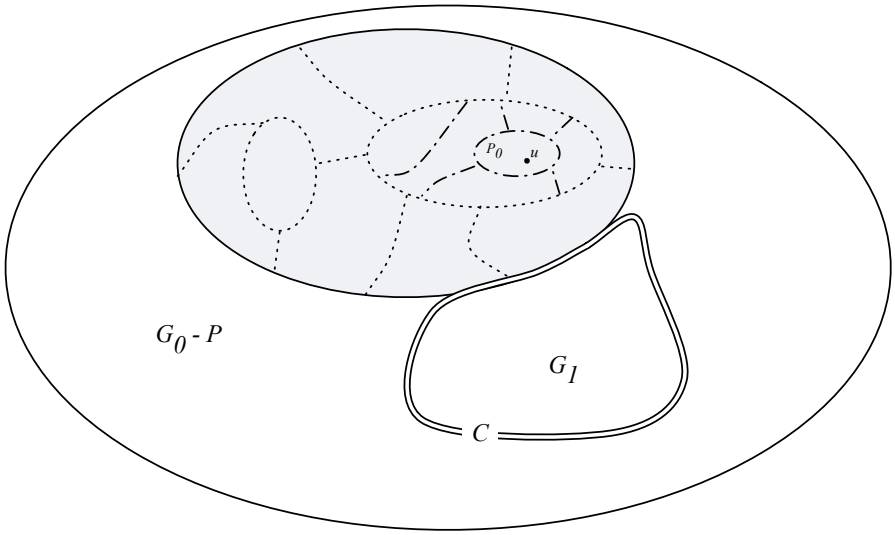

Figure 3: A schematic diagram showing the various subgraphs whose dense distance graphs are used in a query to the cycle distance oracle. The cycle C is double-lined. The interior of C is the subgraph G_1. The query node u is indicated by a small solid circle. The piece P in the r–decomposition of the exterior of C (G_0) is shown as a grey region with solid boundary. The boundaries of the pieces whose dense distance graphs are in H_u are shown as dotted lines (one level) and dashed-dotted lines (another level). P_0 is the smallest piece of P that contains u. Any shortest path from u to C can be decomposed into a shortest path from u to ∂P_0 followed by shortest paths between nodes on the boundaries shown in the figure.

Query When queried with a node u, the data structure outputs the distances from u to all the nodes of C. We describe the case where u is not enclosed by C. In this case we use the dense distance graphs computed in the preprocessing step for G_0. The symmetric case is handled similarly, by using the dense distance graphs computed for G_1.

Let P be a piece in the r–division of G_0 to which u belongs. Recall that P consists of $O(c^2)$ nodes. Consider the recursive r–division of P computed in item 3 of the preprocessing stage. Let P_0 be the level-0 piece of P that contains u. P_0 consists of $O(\sqrt{c^2}) = O(c)$ nodes.[4]

We first compute, in $O(c)$ time, the distances from u to all nodes of P_0, and store them in a table dist_{P_0}. We then compute, using FR-Dijkstra, the distances from u in the union of the following dense distance graphs (see Figure 3):

1. H_0, the star graph with center u and leaves ∂P_0. The arcs of H_0 are directed from u to the leaves and their lengths are the corresponding lengths in dist_{P_0}.

[4]Here, as in Section 3, we assume that u is not a boundary node in the recursive r–decomposition of P. The case where u is such a boundary node is degenerate, see Section 3.

2. H_u, the subset of dense distance graphs in H_P that correspond to pieces in the recursive decomposition of P that contain u and their subpieces. These dense distance graphs are available in H_u.

3. DDG_{G_0-P}

4. DDG_C

Note that the first two graphs are the analogs of H_0 and H_u from Section 3.

Distances from u in the union of the above graphs are equal to the distances from u in G. This is true since any u-to-C shortest path can be decomposed into (1) a shortest path in P_0 from u to ∂P_0, (2) shortest paths each of which is a shortest path in Q between boundary nodes of Q for some piece in the recursive r–decompostion of P that is represented in H_u, (3) shortest paths in $G_0 - P$ between nodes of $\partial P \cup C$, and (4) shortest paths in the interior of C between nodes of C

To bound the running time of FR-Dijkstra we need to bound the number of nodes in all dense distance graphs used in the FR-Dijkstra computation. H_0 has $O(\sqrt{c})$ nodes. The analysis in Section 3 shows that the graphs in the set H_u consist of $O(\sqrt{|P|}\lg\lg|P|)$ nodes (substitute $k = 1/\lg|P|$ in eq. (1)). DDG_{G_0-P} has $O(c+\sqrt{P}) = O(c)$ nodes, and DDG_C has c nodes. Combined, we get that the running time of the invocation of FR-Dijkstra is bounded by $O(c\lg^2 c\lg\lg c)$. This dominates the $O(c)$ time required for the computation of dist_{P_0}, so the overall query time is $O(c\lg^2 c\lg\lg c)$, as claimed.

5 A Distance Oracles with Space $S \in [n\lg\lg n, n^2]$

In this section, we prove Theorem 1. Using our new cycle MSSP data structure, the proof is rather straightforward.

Theorem 1. *Let G be a directed planar graph on n vertices. For any value S in the range $S \in [n\lg\lg n, n^2]$, there is a data structure with preprocessing time $O(S\lg^3 n/\lg\lg n)$ and space $O(S)$ that answers distance queries in $O(nS^{-1/2}\lg^2 n\lg^{3/2}\lg n)$ time per query.*

Let $r := (n^2\lg\lg n)/S$. Note that $r \in [\lg\lg n, n]$ for any $S \in [n\lg\lg n, n^2]$.

Preprocessing We start by computing an r–division. Each piece has $O(r)$ nodes and $O(\sqrt{r})$ boundary nodes incident to a constant number of holes. For each piece P we compute the following:

1. We compute a distance oracle as in Theorem 3 using $\epsilon = 1/\lg r$. This takes $O(r\lg r\lg\lg r)$ time and uses $O(r\lg\lg r)$ space.

2. For each hole of P (bounded by a cycle) we compute our new cycle MSSP data structures.[5] Since the number of holes per piece is constant, this requires $O(n\lg^3 n)$ time per piece, and $O(S\lg^3 n/\lg\lg n)$ overall, which dominates the preprocessing time.

For each region we store a distance oracle, $O(1)$ Cycle MSSP data structures, and the internal and external dense distance graphs. The total space requirement is thus $O((n/r)\cdot n\lg\lg n) = O(S)$.

Query Given a pair of nodes s, t, we compute a shortest s-to-t path as follows. Assume first that s and t are in different regions. Let P denote the piece that contains s and let ∂P denote its boundary. We compute the distances in G from ∂P to t using the cycle MSSP data structures. These distances can be obtained in time $O(|\partial P|\lg^2 n\lg\lg n) = O(\sqrt{r}\lg^2 n\lg\lg n)$. Analogously, we compute the distances in G from s to ∂P. It remains to find the node $p \in \partial P$ that minimizes $d_G(s,p) + d_G(p,t)$, which can be done in $O(|\partial P|) = O(\sqrt{r})$ time using a simple sequential search.

If s and t lie in the same piece, we also have to account for the possibility that the shortest s-to-t path does not visit ∂P. The length of such a path is found by querying the precomputed distance oracle for P, which takes $O(\sqrt{r}\lg^2 r\lg\lg r)$ time.

[5] Note that for $S = o(n\lg n)$ we cannot even afford to store Klein's MSSP data structure [Kle05].

Comparison The query time of our data structure is at most $O(\sqrt{r}\lg^2 r \lg\lg r)$, which, in terms of S, is $O(nS^{-1/2}\lg^2 n \lg^{3/2}\lg n)$. Let us contrast this with Cabello's data structure [Cab06] that, for any $S \in [n^{4/3}\lg^{1/3} n, n^2]$ has preprocessing time and space $O(S)$ and query time $O(nS^{-1/2}\lg^{3/2} n)$. In our construction, we sacrifice a factor of $O(\sqrt{\lg n(\lg\lg n)^3})$ in the query time but we gain a much larger regime for S. For the range $S \in [\omega(n\lg n/\lg\lg n), o(n^{4/3}\lg^{1/3} n)]$, only data structures of size $O(S)$ with query time $O(n^2/S)$ had been known [Dji96] (see also Figure 1).

k–many distances As a consequence, we also obtain an improved algorithm for k–many distances, for $k = \Omega(\sqrt{n}/\lg\lg n)$.

Proof of Theorem 2. For some value of r to be specified below, we preprocess G in time $O((n^2/r)\lg^3 n)$, and then we answer each of the k queries in time $O(\sqrt{r}\lg^2 r\lg\lg r)$. The total time is $O((n^2/r)\lg^3 n + k\sqrt{r}\lg^2 r\lg\lg r)$. This is minimized by setting $r = n^{4/3}k^{-2/3}(\lg n/\lg\lg n)^{2/3}$. Note that $r = O(n)$ since $k = \Omega(\sqrt{n}\lg n/\lg\lg n)$. The total running time is thus $O((kn)^{2/3}(\lg n)^{7/3}(\lg\lg n)^{2/3})$. □

6 Distance Oracles with Query Time Quasi-Proportional to the Shortest-Path Length

We use our new cycle MSSP data structure to prove Theorem 5, which states that there is a distance oracle with query time proportional to the shortest-path length. We actually prove two versions, the stronger one being a distance oracle with query time proportional to the minimum number of edges (hops) on a shortest path. For the stronger version, we need the following assumption, which essentially means that approximate shortest paths do not use significantly fewer edges.

Assumption 1. *Let $h_G(s,t)$ denote the number of edges (*hops*) on a minimum-hop shortest-path. For some constant $\epsilon > 0$, all $s-t$ paths of length at most $(1+\epsilon)d_G(s,t)$ have $\Omega(h_G(s,t))$ edges.*

We restate Theorem 5 and its stronger variant.

Theorem 5. *For any planar graph G with edge lengths ≥ 1 there is an exact distance oracle of space $O(n\lg n\lg\lg n)$ with query time $O(\min\{\ell\lg^2\ell\lg\lg\ell, \sqrt{n}\lg^2 n\})$ for any pair of nodes at distance ℓ. The preprocessing time is at most $O(n^{1+\epsilon})$ for any constant $\epsilon > 0$.*

Furthermore, if Assumption 1 holds for G, the query time is at most $O(\min\{h\lg^2 h\lg\lg h, \sqrt{n}\lg^2 n\})$ for any pair of nodes (u,v) at hop-distance *$h = h_G(u,v)$.*

Our main ingredient is a distance oracle for planar graphs with tree-width w.

Theorem 6. *Let G be a planar graph on n vertices with tree-width w. For any value S in the range $S \in [n\lg\lg n, n^2]$, there is a data structure with preprocessing time $O(S\lg^2 n)$ and space $O(S)$ that answers distance queries in $O(\min\{nS^{-1/2}\lg^{2.5} n, w\lg^2 w\lg\lg w\})$ time per query.*

Note that for S superlinear in n but less than roughly nw, the oracle cannot make any use of the additional space available and the query algorithm runs in time proportional to w (up to logarithmic factors). Any application should use either space close to linear or more than nw.

Using Theorem 6, the proof of Theorem 5 boils down to a combination of *slicing* (see for example [Bak94, Kle08]), *local tree-width* [Epp00, DH04], and *scaling*.

Proof of Theorem 5. We repeatedly use Theorem 6 for different subgraphs as follows.

Slicing and local tree-width We use standard techniques [Bak94, Kle08] and research on the related *linear local tree-width* property [Epp00, DH04] to *slice* a planar graph into subgraphs of a certain tree-width:

- We compute a breadth-first search tree in the planar dual rooted at an arbitrary face.
- The subgraph induced by the nodes at depth between d and $d' > d$ has tree-width $O(d'-d)$ [Epp00, DH04].

If we were only interested in paths using at most w edges, we could *i)* cut the graph into slices of depth w, and *ii)* check the union of any two consecutive levels containing both endpoints. It is straightforward to see that each path on w edges lies completely within one of these unions.

Scaling We apply the slicing step described in the previous paragraph for different scales.

- For every integer $i > 0$ with $2^i \leq \sqrt{n}$, we slice the graph into subgraphs G^i_j at depth $r = 2^i$; here, G^i_j denotes the graph induced by the nodes adjacent to all the faces at depth in $[jr, (j+1)r)$. Every subgraph G^i_j has tree-width at most $O(r)$.
- For any two consecutive G^i_j, G^i_{j+1} we compute a distance oracle of size $O(|G^i_j \cup G^i_{j+1}| \lg\lg |G^i_j \cup G^i_{j+1}|)$ with query time $O(r \lg^2 r \lg\lg r)$ as in Theorem 6. Since each node is in at most two graphs G^i_j and since each G^i_j participates in at most two distance oracles, the total size of all these distance oracles per level i is $O(n \lg\lg n)$. The total size of our data structure is thus $O(n \lg(n) \lg\lg(n))$.

Which Scale? Let the smallest number of edges (or *hops*) on any shortest path from u to v be $h = h_G(u, v)$. At query time we shall use an approximate distance oracle to determine the right scale. In the preprocessing algorithm, we also precompute the approximate distance oracle of Thorup [Tho04] for $\epsilon = 1/2$. This oracle can be computed in $O(n\epsilon^{-1} \lg^3 n)$ time, it uses space $O(n\epsilon^{-1} \lg n)$, and it answers $(1+\epsilon)$–approximate distance queries in time $O(1/\epsilon)$. If Assumption 1 holds, we instead use that value of ϵ in the construction of Thorup's distance oracle. The space consumption is not increased.

Query Algorithm At query time, given a pair of nodes (u, v) at distance ℓ, we need to find a level that contains a shortest path. We query the approximate distance oracle in time $O(1/\epsilon)$ to obtain an estimate for ℓ. Let $\tilde{\ell}$ denote this estimate. We then execute one of the following search algorithms.

In the case that Assumption 1 does not hold, we directly query level i for the smallest i with $2^i \geq \tilde{\ell}$. Since all the edge weights are at least 1, any path of length $\tilde{\ell}$ has at most $\tilde{\ell}$ edges and is thus contained in some graph G^i_j at level i. Since graphs at level i have tree-width $O(2^i)$, and since $2^i = O(\tilde{\ell}) = O(\ell)$, the running time of the query algorithm is $O(\ell \lg^2 \ell \lg\lg \ell)$ as claimed.

If Assumption 1 does hold, we search the data structure level by level with increasing i until the first time a distance at most $\tilde{\ell}$ is found. By Assumption 1, we know that any $u - v$ path of length $\leq (1+\epsilon)\ell$ uses at least $c \cdot h_G(u, v)$ edges for some constant $c \leq 1$. We therefore search the next $1 - \lg_2 c$ levels to make sure that we find a shortest path. The running time can be calculated using a geometric sum, which is dominated by the time to search the last level with tree-width $O(h_G(u, v))$. □

6.1 Distance Oracles for Planar Graphs with Tree-width $o(\sqrt{n})$

In this section we prove Theorem 6. There exist oracles for (not necessarily planar) graphs with tree-width w. Chaudhuri and Zaroliagis [CZ00] provide a distance oracle that uses space $O(w^3 n)$ and answers distance queries in time $O(w^3 \alpha(n))$, where $\alpha(n)$ denotes the inverse Ackermann function.

In our application, the tree-width w may be non-constant up to $O(\sqrt{n})$. For this reason we cannot use their distance oracle. In the following we improve upon their result since we further assume planarity: we can obtain space $O(n \lg\lg n)$ and query time $O(w \lg^2 w \lg\lg w)$ using our distance oracle, which is a better space vs. query time tradeoff for $w = \Omega(\sqrt[3]{\lg\lg n})$.

In some sense our proof can be seen as an improvement over Djidjev's data structure with space $O(S)$ and query time $O(n^2/S)$. His data structure works on any graph with recursive balanced separators of size $O(\sqrt{n})$ and, furthermore, similar results can be obtained for graphs with separators of general size $f(n) = o(n)$ [Dji96, Sections 3 and 4]. The data structure with the better trade-off (space $O(S)$ and query time $O(n/\sqrt{S})$) however only works for planar graphs [Dji96, Section 5], since it exploits the Jordan Curve Theorem. We can now exploit the Jordan Curve Theorem for planar graphs with smaller separators by observing that, for planar graphs with smaller tree-width ($o(\sqrt{n})$), the size of the Jordan Curve separating the inside from the outside decreases proportionally [ST94, DPBF10]. These separators are referred to as *sphere-cut separators* [ST94, DPBF10] and they can be found efficiently.

Lemma 1 (Gu and Tamaki [GT09, proof of Theorem 2]). *For any constant $\epsilon > 0$ and for any biconnected vertex-weighted planar graph on n nodes with tree-width w, there is an $O(n^{1+\epsilon})$–time algorithm that finds a non-self-crossing cycle C of length $O(\epsilon^{-1}w)$ such that any connected component of $G \setminus C$ has weight at most $2/3$ of the total weight.*

Proof. The algorithm computes a *branch decomposition* as in Gu and Tamaki [GT09, proof of Theorem 2].

A *branch decomposition* [RS91] of a graph G is a ternary tree T whose leafs correspond to the edges of G. Removing any tree edge $e \in T$ creates two connected components $T_1, T_2 \subseteq T$, each of which corresponds to a subgraph G_i of G, induced by the edges corresponding to the set of leafs of T in T_i. In the branch decomposition tree, we may thus choose the edge e^* that achieves the best balance among all the edges $e \in T$.

In the algorithm of Gu and Tamaki, each edge of the branch decomposition tree corresponds to a *self-non-crossing closed curve* passing through $O(\epsilon^{-1}w)$ nodes of G (as in sphere-cut decompositions) that encloses G_1 and does not enclose G_2 (or vice versa).

Since the degree of each node is at most three, it is possible to choose an edge e^* such that the total weight strictly enclosed by the non-self-crossing cycle C_{e^*} and the total weight strictly not enclosed by C_{e^*} are each at most a $2/3$–fraction of the total weight. The cycle C_{e^*} is our balanced separator of length $O(\epsilon^{-1}w)$. □

For an optimal sphere-cut decomposition, the fastest algorithm runs in cubic time [ST94, GT08]. For our purposes, the constant approximation in Lemma 1 suffices.

We define a variant of the r–division as in Section 2.1, wherein we use either Miller's cycle separator or sphere-cut separators, whichever is smaller. An $[r, w]$–*division* of G with tree-width w is a decomposition into $\Theta(n/r)$ edge-disjoint pieces, each with $O(r)$ nodes and $O(\min\{\sqrt{r}, w\})$ boundary nodes.

Lemma 2. *An* $[r, w]$–division *can be found in time* $O(n^{1+\epsilon} \lg n + n \lg r + nr^{-1/2} \lg n)$.

The proof is similar to that of [Fre87, WN10b] and appears in the appendix.

Proof of Theorem 6. The data structure for planar graphs with smaller tree-width is essentially the same as the data structure for general planar graphs as described in Section 5. The main difference is that, when computing a cycle separator, instead of Miller's algorithm to find a cycle separator of length $O(\sqrt{n})$, we use sphere-cut separators and the corresponding $[r, w]$–division as in Lemma 2. □

7 Conclusion

We introduce a new data structure to answer distance queries between any node v and all the nodes on a not-too-long cycle of a planar graph. Using this tool, we significantly improve the worst-case query times for distance oracles with low space requirements both for linear space (down from $O(n)$ to $O(n^{1/2+\epsilon})$) and for superlinear space S (down from $O(n^2/S)$ to $O(n/\sqrt{S})$). We also give the first distance oracle that actually exploits the tree-width of a planar graph, particularly if it is $o(\sqrt{n})$ and, as an application, we give a distance oracle whose query time is roughly proportional to the shortest-path length — notably without any sacrifices in the worst-case behavior. Similar behavior of practical methods had been observed experimentally before but could not be proven until the current work. We hope that future work will provide a distance oracle with provable query time proportional to the number of edges on the shortest path. In our opinion, an interesting open question is whether there is another tradeoff curve below the space $O(S)$ and query time $O(n/\sqrt{S})$ curve.

Acknowledgments

We thank Hisao Tamaki for helpful discussions on [GT09].

References

[ACC+96] Srinivasa Rao Arikati, Danny Z. Chen, L. Paul Chew, Gautam Das, Michiel H. M. Smid, and Christos D. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. In *Algorithms - ESA '96, Fourth Annual European Symposium, Barcelona, Spain, September 25-27, 1996, Proceedings*, pages 514–528, 1996.

[AFGW10] Ittai Abraham, Amos Fiat, Andrew V. Goldberg, and Renato Fonseca F. Werneck. Highway dimension, shortest paths, and provably efficient algorithms. In *ACM-SIAM Symposium on Discrete Algorithms (SODA'10) January 17-19, 2010, Austin, Texas*, 2010.

[AT05] Lars Arge and Laura Toma. External data structures for shortest path queries on planar digraphs. In *Algorithms and Computation, 16th International Symposium, ISAAC 2005, Sanya, Hainan, China, December 19-21, 2005, Proceedings*, pages 328–338, 2005.

[Bak94] Brenda S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994. Announced at FOCS 1983.

[BCK+10] Reinhard Bauer, Tobias Columbus, Bastian Katz, Marcus Krug, and Dorothea Wagner. Preprocessing speed-up techniques is hard. In *Algorithms and Complexity, 7th International Conference, CIAC 2010, Rome, Italy, May 26-28, 2010. Proceedings*, pages 359–370, 2010.

[BDDW09] Reinhard Bauer, Gianlorenzo D'Angelo, Daniel Delling, and Dorothea Wagner. The shortcut problem - complexity and approximation. In *SOFSEM 2009: Theory and Practice of Computer Science, 35th Conference on Current Trends in Theory and Practice of Computer Science, Spindleruv Mlýn, Czech Republic, January 24-30, 2009. Proceedings*, pages 105–116, 2009.

[BFSS07] Holger Bast, Stefan Funke, Peter Sanders, and Dominik Schultes. Fast routing in road networks with transit nodes. *Science*, 316(5824):566, 2007.

[BSWN10] Glencora Borradaile, Piotr Sankowski, and Christian Wulff-Nilsen. Min *st*-cut oracle for planar graphs with near-linear preprocessing time. In *51st Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 601–610, 2010.

[Cab06] Sergio Cabello. Many distances in planar graphs. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 1213–1220, 2006. A preprint of the journal version is available in the University of Ljubljana preprint series, Vol. 47 (2009), 1089.

[CX00] Danny Z. Chen and Jinhui Xu. Shortest path queries in planar graphs. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 469–478, 2000.

[CZ00] Shiva Chaudhuri and Christos D. Zaroliagis. Shortest paths in digraphs of small treewidth. part I: Sequential algorithms. *Algorithmica*, 27(3):212–226, 2000. Announced at ICALP 1995.

[DH04] Erik D. Demaine and Mohammad Taghi Hajiaghayi. Diameter and treewidth in minor-closed graph families, revisited. *Algorithmica*, 40(3):211–215, 2004.

[Dij59] Edsger Wybe Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[Dji96] Hristo Djidjev. Efficient algorithms for shortest path problems on planar digraphs. In *Graph-Theoretic Concepts in Computer Science, 22nd International Workshop, WG '96, Cadenabbia (Como), Italy, June 12-14, 1996, Proceedings*, pages 151–165, 1996.

[DPBF10] Frederic Dorn, Eelko Penninkx, Hans L. Bodlaender, and Fedor V. Fomin. Efficient exact algorithms on planar graphs: Exploiting sphere cut decompositions. *Algorithmica*, 58(3):790–810, 2010. Announced at ESA 2005.

[DPZ00] Hristo Djidjev, Grammati E. Pantziou, and Christos D. Zaroliagis. Improved algorithms for dynamic shortest paths. *Algorithmica*, 28(4):367–389, 2000.

[EG08] David Eppstein and Michael T. Goodrich. Studying (non-planar) road networks through an algorithmic lens. In *16th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACM-GIS 2008, November 5-7, 2008, Irvine, California, USA, Proceedings*, page 16, 2008.

[Epp99] David Eppstein. Subgraph isomorphism in planar graphs and related problems. *Journal of Graph Algorithms and Applications*, 3(3), 1999. Announced at SODA 1995.

[Epp00] David Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3-4):275–291, 2000.

[FR06] Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *Journal of Computer and System Sciences*, 72(5):868–889, 2006. Announced at FOCS 2001.

[Fre87] Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16(6):1004–1022, 1987.

[GSSD08] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *Experimental Algorithms, 7th International Workshop (WEA'08), Provincetown, MA, USA, May 30-June 1, 2008, Proceedings*, pages 319–333, 2008.

[GT08] Qian-Ping Gu and Hisao Tamaki. Optimal branch-decomposition of planar graphs in $O(n^3)$ time. *ACM Transactions on Algorithms*, 4(3), 2008. Announced at ICALP 2005.

[GT09] Qian-Ping Gu and Hisao Tamaki. Constant-factor approximations of branch-decomposition and largest grid minor of planar graphs in $O(n^{1+\epsilon})$ time. In *Algorithms and Computation, 20th International Symposium, ISAAC 2009, Honolulu, Hawaii, USA, December 16-18, 2009. Proceedings*, pages 984–993, 2009.

[GW05] Andrew V. Goldberg and Renato Fonseca F. Werneck. Computing point-to-point shortest paths from external memory. In *Proceedings of the Seventh Workshop on Algorithm Engineering and Experiments and the Second Workshop on Analytic Algorithmics and Combinatorics, ALENEX /ANALCO 2005, Vancouver, BC, Canada, 22 January 2005*, pages 26–40, 2005.

[HKMS09] Moritz Hilger, Ekkehard Köhler, Rolf H. Möhring, and Heiko Schilling. Fast point-to-point shortest path computations with arc-flags. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 74:41–72, 2009. Papers of the 9th DIMACS Implementation Challenge: Shortest Paths, 2006.

[HKRS97] Monika Rauch Henzinger, Philip Nathan Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*, 55(1):3–23, 1997. Announced at STOC 1994.

[HMZ03] David A. Hutchinson, Anil Maheshwari, and Norbert Zeh. An external memory data structure for shortest path queries. *Discrete Applied Mathematics*, 126(1):55–82, 2003. Announced at COCOON 1999.

[JHR96] Ning Jing, Yun-Wu Huang, and Elke A. Rundensteiner. Hierarchical optimization of optimal path finding for transportation applications. In *CIKM '96, Proceedings of the Fifth International Conference on Information and Knowledge Management, November 12 - 16, 1996, Rockville, Maryland, USA*, pages 261–268, 1996.

[Joh77] Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM*, 24(1):1–13, 1977.

[KK06] Lukasz Kowalik and Maciej Kurowski. Oracles for bounded-length shortest paths in planar graphs. *ACM Transactions on Algorithms*, 2(3):335–363, 2006. Announced at STOC 2003.

[KKS11] Kenichi Kawarabayashi, Philip Nathan Klein, and Christian Sommer. Linear-space approximate distance oracles for planar, bounded-genus, and minor-free graphs. In *Automata, Languages and Programming, 38th International Colloquium, ICALP 2011*, 2011. to appear.

[Kle02] Philip Nathan Klein. Preprocessing an undirected planar network to enable fast approximate distance queries. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA*, pages 820–827, 2002.

[Kle05] Philip Nathan Klein. Multiple-source shortest paths in planar graphs. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*, pages 146–155, 2005.

[Kle08] Philip Nathan Klein. A linear-time approximation scheme for TSP in undirected planar graphs with edge-weights. *SIAM Journal on Computing*, 37(6):1926–1952, 2008. Announced at FOCS 2005.

[KMS05] Ekkehard Köhler, Rolf H. Möhring, and Heiko Schilling. Acceleration of shortest path and constrained shortest path computation. In *Experimental and Efficient Algorithms, 4th International Workshop, WEA 2005, Santorini Island, Greece, May 10-13, 2005, Proceedings*, pages 126–138, 2005.

[KMW10] Philip Nathan Klein, Shay Mozes, and Oren Weimann. Shortest paths in directed planar graphs with negative lengths: A linear-space $O(n \log^2 n)$-time algorithm. *ACM Transactions on Algorithms*, 6(2), 2010. Announced at SODA 2009.

[Lau04] Ulrich Lauther. An extremely fast, exact algorithm for finding shortest paths in static networks with geographical background. In *Geoinformation und Mobilität – von der Forschung zur praktischen Anwendung*, volume 22, pages 219–230, 2004.

[Mil86] Gary L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *Journal of Computer and System Sciences*, 32(3):265–279, 1986. Announced at STOC 1984.

[MWN10] Shay Mozes and Christian Wulff-Nilsen. Shortest paths in planar graphs with real lengths in $O(n \log^2 n / \log \log n)$ time. In *Algorithms - ESA 2010, 18th Annual European Symposium, Liverpool, United Kingdom, September 6-8, 2010. Proceedings*, 2010.

[Nus10] Yahav Nussbaum. Improved distance queries in planar graphs. *CoRR*, abs/1012.2825, 2010.

[RS91] Neil Robertson and Paul D. Seymour. Graph minors. X. obstructions to tree-decomposition. *J. Comb. Theory, Ser. B*, 52(2):153–190, 1991.

[Sch98] Jeanette P. Schmidt. All highest scoring paths in weighted grid graphs and their application to finding all approximate repeats in strings. *SIAM Journal on Computing*, 27(4):972–992, 1998. Announced at ISTCS 1995.

[Som11] Christian Sommer. More compact oracles for approximate distances in unweighted planar graphs, 2011. submitted.

[SS05] Peter Sanders and Dominik Schultes. Highway hierarchies hasten exact shortest path queries. In *Algorithms - ESA 2005, 13th Annual European Symposium, Palma de Mallorca, Spain, October 3-6, 2005, Proceedings*, pages 568–579, 2005.

[ST94] Paul D. Seymour and Robin Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.

[Tho04] Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *Journal of the ACM*, 51(6):993–1024, 2004. Announced at FOCS 2001.

[WN10a] Christian Wulff-Nilsen. Constant time distance queries in planar unweighted graphs with subquadratic preprocessing time, 2010. Computational Geometry, special issue on the 25th European Workshop on Computational Geometry (to appear).

[WN10b] Christian Wulff-Nilsen. Min st-cut of a planar graph in $O(n \log\log n)$ time. *CoRR*, abs/1007.3609, 2010.

[Zar08] Christos Zaroliagis. Engineering algorithms for large network applications. In *Encyclopedia of Algorithms*. 2008.

A $[r, w]$–division; Proof of Lemma 2

An $[r, w]$–*division* of G with tree-width w is a decomposition into $\Theta(n/r)$ edge-disjoint pieces, each with $O(r)$ nodes and $O(\min\{\sqrt{r}, w\})$ boundary nodes.

Lemma 2. *An* $[r, w]$–division *can be found in time* $O(n^{1+\epsilon} \lg n + n \lg r + nr^{-1/2} \lg n)$.

Proof. The procedure for obtaining the $[r, w]$–division is the one described in [WN10b], but using either Miller's cycle separator or sphere-cut separators, whichever is smaller. If $\sqrt{r} = O(w)$ then since the separators we use are smaller than the separators assumed in the proof of [WN10b, Lemmata 2 and 3], the lemmas apply and the resulting decomposition has $\Theta(n/r)$ pieces, each with $O(r)$ nodes and $O(\sqrt{r})$ boundary nodes.

If, $\sqrt{r} = \omega(w)$, then first apply the separator theorem as in the procedure for obtaining a weak r-division [WN10b, Lemma 2] until every piece has size $O(r)$. Note that since $\sqrt{r} = \omega(w)\sqrt{r} = \omega(w)$, we will always use sphere-cut separators, whose size is cw, regardless of the size of the piece we separate. Therefore, by [WN10b, Lemma 2], the number of pieces is $\Theta(n/r)$, and the total number of boundary nodes is $O(n/\sqrt{r})$. However, here we need a tighter bound on the total number of boundary nodes. In the following we show that the total number of boundary nodes is $O(nw/r)$.

Consider the binary tree whose root corresponds to G, and whose leaves correspond to the pieces of the weak r-division. Each internal node in the tree corresponds to a piece in the recursive decomposition that either consists of too many nodes or contains too many holes and is therefore separated into two subpieces using a sphere-cut separator. As in the proof of [WN10b, Lemma 2], for any boundary vertex v in the weak r-division, let $b(v)$ denote one less than the number of pieces containing v as a boundary vertex. Let $B(n)$ be the sum of $b(v)$ over all such v. Every time a separator is used, cw boundary nodes are introduced, and the region to which these nodes belong is split into two regions, so the number of regions to which these nodes belong increases by one. Therefore, $B(n)$ is bounded by the number of internal nodes times cw. Since the tree is binary, the number of internal nodes is bounded by the number of leafs, so $B(n) = O(nw/r)$.

Next, consider the procedure in [WN10b, Lemma 3] which further divides the pieces of the weak r-division to make sure that the number of boundary nodes in each piece is small enough. In our case we want to limit the number of boundary nodes per piece to at most $c'w$. Let t_i denote the number of pieces in the weak r-division with exactly i boundary nodes. Note that $\sum_i t_i = \sum_{v \in V_B}(b(v) + 1)$, where V_B is the set of boundary vertices over all pieces in the weak r-division. Hence, $\sum_i t_i \leq 2B(v)$, so by the bound on $B(v)$, $\sum_i t_i = O(nw/r)$.

In the weak r-division, consider a piece P with $i > c'\sqrt{r}$ boundary vertices. When the above procedure splits P, each of the subpieces contains at most a constant fraction of the boundary vertices of P. Hence, after $di/(c'w)$ splits of P for some constant d, all subpieces will contain at most $c'w$ boundary vertices. This will result in at most $1 + di/(c'w)$ subpieces and at most cw new boundary vertices. We may choose c' to be sufficiently larger than c. The total number of new boundary vertices introduced by the above procedure is thus

$$\sum_i cw(di/(c'w))t_i \leq d\sum_i it_i = O(nw/r)$$

and the number of new pieces is at most

$$\sum_i (di/(c'w))t_i = O(n/r).$$

Hence, the procedure generates an r-division. Since a sphere-cut separator can be found in $O(n^{1+\epsilon})$ time, a weak r-division can be found in $O(n^{1+\epsilon} \lg n) = O(n^{1+\epsilon'})$ time, and refining it into an r-division can be done within this time bound.

□

B Decomposition of Shortest Paths into Subpaths

The following elementary trivial observations on shortest paths and their interaction with boundaries of pieces are the basis for the correctness of our query algorithms.

Lemma 3. *Any subpath of a shortest path is a shortest path.*

Proof. Assume some subpath is not a shortest path. It may be replaced with a shorter path yielding a shorter overall path, a contradiction. □

Lemma 4. *Let Q be a subgraph of G with boundary ∂Q. Any shortest path from $x \in Q$ to a node of $y \in G$ has a (possibly empty) x-to-∂Q prefix.*

Proof. Follows immediately from the definition of the boundary of a subgraph. □