On Counting Oracles for Path Problems

Ivona Bezáková
Department of Computer Science, Rochester Institute of Technology, Rochester, NY, USA
ib@cs.rit.edu

Andrew Searns
Rochester Institute of Technology, Rochester, NY, USA
abs2157@rit.edu

Abstract

We initiate the study of counting oracles for various path problems in graphs. Distance oracles have gained a lot of attention in recent years, with studies of the underlying space and time tradeoffs. For a given graph G, a distance oracle is a data structure which can be used to answer distance queries for pairs of vertices $s, t \in V(G)$. In this work, we extend the set up to answering counting queries: for a pair of vertices s, t, the oracle needs to provide the number of (shortest or all) paths from s to t. We present $O(n^{1.5})$ preprocessing time, $O(n^{1.5})$ space, and $O(\sqrt{n})$ query time algorithms for oracles counting shortest paths in planar graphs and for counting all paths in planar directed acyclic graphs. We extend our results to other graphs which admit small balanced separators and present applications where our oracle improves the currently best known running times.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Counting oracle, Path problems, Shortest paths, Separators

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2018.56

Funding Research supported by NSF grant CCF-1319987 and by an REU (Research Experience for Undergraduates) supplement.

1 Introduction

Shortest path problems have been heavily studied for decades and the developed algorithms are among the most important algorithmic building blocks. In the most traditional set up, one is given a graph G and two vertices s, t and the goal is to find a shortest path from s to t in G. Due to many applications querying for multiple s, t pairs, the design of so-called distance oracles has gained a lot of attention in recent years [13, 9, 18, 6, 11, 8, 10, 3]. In an oracle approach, for a given graph G, the goal is to pre-compute a not too large data structure (an oracle) which can then be used to answer distance queries for pairs of vertices s, t in as fast time as possible. Many previous works, which we discuss in more detail later, have studied the tradeoffs between the required space and the query time for such distance oracles for various graph classes. Among the prime applications of these oracle results is map querying, where a user often prefers knowing not just one of the optimal routes, but they would like to be shown a variety of options. Hence, we propose to amend distance oracles with counting: in addition to the distance from s to t, a counting path oracle returns also the number of all shortest paths from s to t. Such an oracle can then be used to generate the paths or provide a random sample when the total number of paths is prohibitively large.

We design counting oracles for the following two problems: #SHORTPATH-ORACLE, where one is given a positively weighted graph and the goal is to construct an oracle which answers queries of "how many shortest paths from s to t are there?", and #PATH-DAG-ORACLE, where one is given a directed acyclic graph (DAG) and the oracle answers queries of "how many paths from s to t are there?". We note that the second problem is #P-hard for general graphs [20], but both problems can be solved in polynomial time within the specified graph class. The second problem, which has applications of its own described below, helps us build an oracle for the first problem. For both problems, when the input is a planar graph with n vertices, we design oracles which take $O(n^{1.5})$ time to construct, take $O(n^{1.5})$ space, and each query can be answered within $O(\sqrt{n})$ time[1].

A straightforward approach to both problems yields an oracle which takes $O(n^2)$ space and $O(1)$ query time, by simply pre-computing all of the possible queries. In a DAG one can compute the number of paths from one vertex to all other vertices in linear time, leading to an $O(n(n+m))$ preprocessing time to compute the oracle for a graph with n vertices and m edges. For planar graphs, this preprocessing time is $O(n^2)$ since $m = O(n)$. For several applications, the number of queries can be linear, leading to an $O(n^2)$ preprocessing time and an overall $O(n)$ time across all queries in the planar setting. With our results, we speed up the running time for such applications to $O(n^{1.5} + n\sqrt{n}) = O(n^{1.5})$. For arbitrary positively weighted graphs, one can compute the number of shortest paths from one vertex to all other vertices in polynomial time, typically within the same running time as finding the distances to all other vertices. For example, one can extend the Dijkstra's algorithm to compute, in addition to the distances, also the respective path counts, and keep updating them throughout the computation. Our results speed up these traditional approaches.

Our techniques employ balanced separators, which are a staple of planar graph algorithms but to the best of our knowledge have not been used for any counting problems. The distance oracle results are ingenious and faster than our results but as far as we see they do not extend to counting. In an optimization problem, one can focus on a certain canonical type of the wanted object, such as a left-most shortest path, or one can even assume that there is a unique shortest path between any pair of vertices. (This can be obtained by small random perturbations of the edge weights.) For a counting problem there appears to be the need for more stored information or longer query time. In particular, we store, for each vertex in the separator, certain path counts to all other vertices in the graph, proceeding in a divide-and-conquer manner on the two parts of the graph. The main technical aspect of our contribution lies in a case analysis that proves that each path has been accounted for exactly once. We generalize our planar results to general graphs which admit small balanced separators.

1.1 Related work and applications

Distance oracles have been studied for several decades, with several very recent exciting results. The current state of the art exact distance oracle of Gawrychowski, Mozes, Weimann, and Wulff-Nilsen [13] requires $O(n^{1.5})$ space and can answer queries in $O(\log n)$ time. This work improved on a recent result of Cohen-Addad, Dahlgaard, and Wulff-Nilsen [9] who

[1] We note that the returned counts may be exponentially large, for example when the graph is a path where every edge has been duplicated – if s and t are the end-points, the number of shortest s-t paths is 2^{n-1}. Therefore, manipulating the counts can incur an additional $O(n \text{ polylog } n)$ factor in the running time. To simplify our presentation throughout this paper, we will (slightly optimistically) assume that each arithmetic operation (addition, multiplication of the counts) takes $O(1)$ time.

designed an oracle with $O(n^{5/3})$ space and $O(\log n)$ query time. Furthermore, both works obtained space/time tradeoffs: for a given S, they design an oracle which takes S space and the query time is a function of S. In particular, [13] obtain a query time $\tilde{O}(\max\{1, n^{1.5}/S\})$ for $S \in [n, n^2]$, while [9] answer queries within time $\tilde{O}(n^{5/2}S^{3/2})$ for $S \geq n^{3/2}$ (where the $\tilde{O}$ notation hides logarithmic factors). Other previous works, on which the two mentioned results build, also studied distance oracles and their space/time tradeoffs [18, 6, 11, 8, 10, 3]. As far as we see, these results do not extend to counting without significant increase in the running time (or space). Of note is also extensive study of approximate distance oracles, with either relative or absolute error, which can achieve a near-linear space and near-constant query time, see [1] and the references within. As for the counting variant in an approximate distance setting, Mihalák, Šrámek, and Widmayer [17] showed that counting all s-t paths up to a given length in a DAG is #P-complete. They also give a fully polynomial-time approximation scheme (FPTAS) for the problem, yielding an approximate counting approximate distance oracle in DAGs. However, the techniques heavily rely on the graph being acyclic. We also note two other hardness results for counting: Yamamoto [21] proved that there is no fully polynomial approximation scheme (FPRAS) for approximately counting all paths in a graph, unless RP = NP. On the fixed-parameter tractable side, Flum and Grohe [12] showed that the problem of counting paths of length k is #W[1]-complete.

Among applications of counting oracles for all paths in a DAG is the problem of counting minimum (s, t)-cuts in planar and bounded genus graphs. The problem of counting minimum (s, t)-cuts has been studied since the 1980's due to its connection to the (s, t) network reliability problem. Provan and Ball [19] proved that it is #P-complete for general graphs and in [4] they gave a general outline that reduces the problem in planar graphs with both s and t on the outerface to the problem of counting all paths in a planar DAG. Their technique was subsequently generalized to any location of s and t by Bezáková and Friedlander [5] and Chambers, Fox, and Nayyeri [7] further extended the approach to bounded genus graphs. In all these scenarios, one needs to count all paths between d pairs of vertices in a planar or bounded genus DAG. For planar graphs, this results in a running time of $O(n \log n + dn) = O(n^2)$ since $d = O(n)$, which our result improves to $O(n \log n + n^{1.5}) = O(n^{1.5})$ since instead of counting the paths for each pair in $O(n)$ time, we can make queries in $O(\sqrt{n})$ time per pair. The running time encompasses also the oracle preprocessing time. It is worth noting that our attempts to use more advanced decomposition techniques such as the r-divisions led to these same running times, making us wonder if a faster than $O(n^{1.5})$ algorithm exists.

The paper is organized as follows. In Section 2 we discuss preliminaries, including how to extend existing single source shortest path (SSSP) algorithms to counting, incurring an additional linear term in the running time. Section 3 presents counting oracles for all paths in planar DAGs, then we discuss counting oracles for shortest paths in positively weighted directed or undirected planar graphs in Section 4, and generalize the oracles beyond planar graphs in Section 5.

2 Preliminaries

An undirected graph $G = (V, E)$ is a set of vertices V and edges $E \subseteq (V \times V)$ of unordered pairs. In a directed graph $G = (V, E)$, the edges are ordered pairs and we refer to them as arcs, using the standard convention that (u, v) indicates an arc from vertex u to vertex v. Unless specifically noted, our results apply to both directed and undirected graphs. (We phrase all our results for graphs but they can be naturally extended to multigraphs which can

have multiple edges between the same pair of vertices.) A positively weighted graph, denoted by $G = (V, E, w)$, assigns a positive weight $w(e)$ to each edge $e \in E$ (i. e., $w : E \to \mathbf{R}^+$). For an $S \subseteq V$, we use $G[S]$ to denote the sub-graph of G induced by S. Throughout this text we use $n = |V|$ to denote the number of vertices and $m = |E|$ the number of edges.

A path p in a graph G is a sequence of vertices $v_1, v_2, \ldots, v_k$, $k \geq 1$, where $(v_i, v_{i+1}) \in E$ for each $i \in \{1, \ldots, k-1\}$. A path with no repeated vertices is called a simple path. For convenience, we refer to a path starting at vertex v_1 and ending at vertex v_k as a v_1-v_k path. We define the relation "is before on $p = v_1, \ldots, v_k$" by $u_1 \prec_p u_2$ where $u_1 = v_i$, $u_2 = v_j$, and $i \leq j$. We say that $v_i, v_{i+1}, \ldots, v_j$, where $i < j$, is a sub-path of a path $p = v_1, \ldots, v_k$. The length of a path $p = v_1, \ldots, v_k$ is $k-1$ in unweighted graphs, and $\sum_{i=1}^{k-1} w(v_i, v_{i+1})$ in weighted graphs. A u_1-u_2 path is shortest if its length is the smallest possible across all u_1-u_2 paths. The length of a shortest u_1-u_2 path is called the distance from u_1 to u_2. A cycle $v_1, \ldots, v_k$ is a path where $v_1 = v_k$ and $k \geq 2$. A graph is acyclic if it does not contain any cycles.

▶ **Observation 1.** *Any path in an acyclic graph is simple. Any shortest path in a positively weighted graph is simple. If p is a shortest path in a graph G, then any sub-path of p must also be shortest.*

We say that a class of graphs admits an $(\alpha, f(n))$-balanced separator, where $f(n)$ is a function and α is a constant, if for every graph G with n vertices its vertices can be partitioned into three sets A, B, C such that the size of A and B are each upper-bounded by αn, the size of C is $O(f(n))$, and there are no edges connecting a vertex in A with a vertex in B. We will refer to such a separator as an (A, B, C) separator.

A graph is planar if it has a planar embedding, that is, if it can be drawn in a plane without any of its edges crossing one another (except for their end-points). A graph is said to be of genus g if it has a crossing-free embedding into a surface of genus g. Planar and bounded genus graphs are sparse, in particular $m = O(n)$ and $m = O(n + g)$, respectively, and they admit small balanced separators:

▶ **Theorem 2** (Planar Separator Theorem, Lipton and Tarjan [16])**.** *Every planar graph has a $(2/3, \sqrt{n})$-balanced separator, which can be found in time $O(n)$.*

▶ **Theorem 3** (Bounded Genus Separator Theorem, Gilbert, Hutchinson, and Tarjan [14])**.** *Every graph of genus g has a $(2/3, \sqrt{gn})$-balanced separator, which can be found in time $O(n)$.*

In a DAG G, for a vertex u we can compute the number of all paths from u to v for every vertex v in time $O(m + n)$ via a simple application of topological sort: sum the number of paths to v's in-neighbors. We will refer to this algorithm as CountPaths(G, u). Next we show how to extend known single source shortest path (SSSP) algorithms with counting:

Except for the SSSP call, this algorithm runs in linear time and computes the number of shortest paths from s to every vertex in G. It does this by building a DAG of tight edges. By Observation 1, since every edge has weight greater than 0, shortest paths are simple. Thus, no cycles can be added into the DAG G' while looping over the edges. For every shortest path p between u and an arbitrary vertex $v \in V(G)$, every edge of p will be added in the direction of the path by Observation 1 (an edge is a two-vertex sub-path of the shortest path p). This leads to the following lemma:

▶ **Lemma 4.** *For any SSSP algorithm with running time $T(n)$ there exist an SSSP counting algorithm with running time $T(n) + O(m)$.*

Algorithm 1 Compute #shortest paths from u to every vertex in G.

```
procedure CountShortestPaths(G, u)
    SSSP(G, u) [Use an existing algorithm, assume d[v] stores the distance to vertex v.]
    initialize unweighted DAG G' = (V(G), ∅)
    for edge e = (v, w) in G do
        if d[w] = d[v] + w(e) then
            insert arc (v, w) into G'
        else if d[v] = d[w] + w(e) then
            insert arc (w, v) into G'
    CountPaths(G', u)
```

Of special importance is the application of this approach to planar graphs where Henzinger et al. [15] designed an $O(n)$ SSSP algorithm. Hence, in planar graphs we can count single source shortest paths in $O(n)$ time. The approach of [15] extends to bounded genus graphs, where it gives an $O(h(g)n)$ running time for graphs of genus g (where $h()$ is a function dependent only on g).

3 Counting Oracle for All Paths in Planar DAGs

In this section, we prove the following theorem:

▶ **Theorem 5.** *For any planar DAG G, there exists an oracle for #PATH-DAG-ORACLE which takes $O(n^{1.5})$ space, takes $O(n^{1.5})$ time to construct, and for any pair of vertices $s, t \in V(G)$ the oracle can answer queries about the number of paths from s to t in $O(\sqrt{n})$ time.*

3.1 Building the Oracle

A naive algorithm for counting the number of paths between two vertices in an unweighted DAG takes $O(n^2)$ time by running CountPaths from every possible source vertex. Instead, we induce Theorem 2 to construct the oracle in a divide-and-conquer manner: We first find a separator (A, B, C) for the given graph. Then we count the number of paths that intersect the separating set C. Finally, we count the number of paths that lie entirely within sub-graphs induced by A and B, respectively. For planar graphs this will lead to an $O(n^{1.5})$ construction time and $O(n^{1.5})$ space. The tricky aspect comes from the fact that many of the paths may cross the separator multiple times.

We start by defining a notion of paths intersecting sets and introduce two sets that are closely related to the oracle algorithm. Then we state a structural relation between these sets, the proof of which we defer to the full version of the paper.

▶ **Definition 6.** Let $G = (V, E)$ be a graph. We say a *path p intersects a set $S \subseteq V$* if p contains a vertex from S. A vertex v is a *first S-intersecting vertex of p* if and only if $v \in p \cap S$ and there is no other $u \in p \cap C$ such that $u \prec_p v$.

▶ **Definition 7.** Let (A, B, C) be a separator of G. For a pair of vertices u and v, define:

- $P_G(u, v)$ as the set of simple u-v paths in G, and
- $P'_{G,C}(u, v, c)$ as the set of simple u-v paths in G with first C-intersecting vertex $c \in C$.

▶ **Lemma 8.** *For a DAG G with separator (A, B, C) and $c \in C$,*

$$|P'_{G,C}(s, t, c)| = |P'_{G,C}(s, c, c) \times P_G(c, t)|.$$

Since there are $O(n^2)$ pairs of vertices, we cannot compute $|P'_{G,C}(s,c,c) \times P_G(c,t)|$ for every s,t pair in total $O(n^{1.5})$ time. However, if we use the fact that $|S \times T| = |S||T|$ for any two sets S and T, we can compute $|P'_{G,C}(s,c,c)|$ and $|P_G(c,t)|$ upfront and leave summing over all $c \in C$ to the query time. Using the CountPaths algorithm from Section 2 we can determine $|P_G(c,t)|$ for all $c \in C$ in $O(|C|n)$ time for an unweighted DAG. Since $|C| = O(\sqrt{n})$, we can compute all $|P_G(c,t)|$ in $O(n^{1.5})$ time. It now remains to show that we can compute $|P'_{G,C}(s,c,c)|$ in $O(n^{1.5})$ time.

Directly computing $|P'_{G,C}(s,c,c)|$ will take $O(|A \cup B|n)$ time. This already is $O(n^2)$ and takes too long. To reduce the running time, we instead count paths from the separator. This takes $O(|C|n) = O(n^{1.5})$ time. By reversing the direction of all arcs in the DAG G, the number of paths between a pair of vertices is preserved. By modifying G, we can guarantee that only s-c paths with first C-intersecting vertex c remain. We define a new notation for a specific modification of G that is necessary in computing $|P'_{G,C}(s,c,c)|$ efficiently.

▶ **Definition 9.** Let G be a graph and let (A,B,C) be its separator. For vertex $c \in C$, define G'_c as the graph constructed from G as follows:
- $V(G'_c) = (V(G) \setminus C) \cup \{c\}$, and
- $E(G'_c) = \{(u,v) | (v,u) \in E(G) \wedge u,v \in V(G'_c)\}$.

Intuitively, we remove all vertices from the separator which are not the vertex we are interested in for computing $|P'_{G,C}(s,c,c)|$. Since s-c paths with first C-intersecting vertex c can only have one vertex in the separator, we remove all paths which could intersect the separator at any other vertex. We also reverse all remaining arcs in the graph. The relationship between G and G'_c, which we prove in the full version of the paper, is as follows:

▶ **Lemma 10.** *For a DAG G with separator (A,B,C) and $c \in C$, $|P'_{G,C}(s,c,c)| = |P_{G'_c}(c,s)|$.*

Lemmas 8 and 10 suggest the following oracle construction algorithm:

Algorithm 2 Build a Path Counting Oracle for DAG G.

```
procedure ConstructAllPathsOracleDAG(G)
    if |V(G)| = 0 then return
    find a separator (A, B, C) in G (and store it)
    for c ∈ C do
        build G'_c by removing C \ {c} from G and reversing arcs
        call CountPaths(G, c) and store the results as P_G[c, v] for every v ∈ V(G)
        call CountPaths(G'_c, c) and store the results as P_{G'_c}[c, v] for every v ∈ V(G)
    ConstructAllPathsOracleDAG(G[A]), where G[A] is the A-induced subgraph
    ConstructAllPathsOracleDAG(G[B])
```

To bound the running time of the construction of the oracle, let α be the constant from the separator definition, used to bound the sizes of the sets A and B. By Theorem 2, $\alpha \leq 2/3$ for planar graphs. Then, we get the following recurrence for the running time:

$$T(n) = \begin{cases} T(|A|) + T(|B|) + O(n^{1.5}) \leq T(\alpha n) + T((1-\alpha)n) + O(n^{1.5}) & \text{if } n \geq 1 \\ O(1) & \text{if } n = 0 \end{cases}$$

where the $O(n^{1.5})$ term comes from doing $O(\sqrt{n})$ of the CountPaths computations, and the inequality is a worst case bound which follows from T's convexity. This recurrence can be evaluated using the Akra-Bazzi Method [2]. Trivially, the p value for which $(\sum_i a_i b_i^p = 1)$

is 1, since $a_0 = a_1 = 1$, $b_0 = \alpha$, and $b_1 = 1 - \alpha$. Since $p = 1$, the recurrence is evaluated as follows:

$$T(n) = O\left(n^1\left(1 + \int_1^n \frac{u^{1.5}}{u^2} du\right)\right) = O(n^{1.5}).$$

We note that in many cases both the running time and the space requirements can be significantly smaller, for example for graphs with $O(1)$ size separators such as outerplanar graphs.

At each recursive call of the oracle construction we store, for each $c \in C$, both $P_G[c, v]$ and $P_{G'_c}[c, v]$. This results in $O(|C|n) = O(n^{1.5})$ space per recursive call, yielding the following recurrence for the total space needed by the oracle: $S(n) = S(A) + S(B) + O(n^{1.5})$. This is exactly the same recurrence as the one for the running time of building the oracle. Thus, the amount of space needed to store the oracle is $O(n^{1.5})$.

3.2 Querying the Oracle

To query the oracle, we essentially compute $\sum_{c \in C} P_{G'_c}[c, s]P_G[c, t]$ for each depth until (and including) s and t become separated by the separator. This can be done using the following algorithm:

Algorithm 3 Query the #PATH-DAG-ORACLE.

```
procedure QueryPaths(G, s, t)
    numPaths = 0
    while (s, t ∈ A) or (s, t ∈ B), where (A, B, C) is the stored separator of G do
        for c ∈ C do
            numPaths+ = P_{G'_c}[c, s]P_G[c, t]
        if s, t ∈ A then G = G[A]
        else G = G[B]
    for c ∈ C do
        numPaths+ = P_{G'_c}[c, s]P_G[c, t]
```

This algorithm relies on Lemmas 8 and 10. For any position of s and t, the paths from s to t can be split into two groups: those that intersect C and those that do not. The paths that intersect C contribute $\sum_{c \in C} |P_{G'_c}(c, s)||P_G(c, t)| = \sum_{c \in C} P_{G'_c}[s, c]P_G[t, c]$. Notice that this computation holds also in the case when $s \in C$ (in which case $|P_{G'_c}(c, s)| = 1$ for $c = s$ and $|P_{G'_c}(c, s)| = 0$ for every other c), or when $t \in C$ (in which case $|P_G(c, t)| = 1$ for $c = t$ and $|P_G(c, t)| = 0$ for every other c). The paths that do not intersect C, which occur only when s and t are either both in A or both in B, are entirely contained within $G[A]$ or $G[B]$. Hence, it suffices to recurse on the respective side of the graph.

It remains to analyze the running time of the query algorithm. Since the oracle has already been computed, the addition steps each take $O(1)$ time. The running time of this algorithm is bounded by the maximum depth before s and t are split by a separator and by the number of vertices in a separator at each depth. Since the separator is balanced, we have $T(n) \leq T(\alpha n) + O(\sqrt{n})$. With a simple application of the Master Theorem, the running time of a query is $O(\sqrt{n})$. This concludes the proof of Theorem 5.

4 Counting Oracle for Shortest Paths in Planar Graphs

We have shown that an oracle can be built for planar DAGs for counting the number of paths between any pair of vertices. In this section, we prove the existence of a similar data structure for shortest paths on any planar graph with positive edge weights. As noted in Observation 1, if all edge weights are positive, then shortest paths must be simple. We first prove shortest path versions of Lemmas 8 and 10. For notational convenience, we define two relevant sets of shortest paths:

▶ **Definition 11.** For a positively weighted graph $G = (V, E, w)$ with a separator (A, B, C) and vertices u and v, define:

- $Q_G(u, v)$ as the set of shortest paths from u to v in G, and
- $Q'_{G,C}(u, v, c)$ as the set of shortest paths from u to v in G with first C-intersecting vertex $c \in C$.

Note that the paths in $Q_G(u, v)$ are always simple by Observation 1. However, the paths in $Q'_{G,C}(u, v, c)$ do not have to be simple since they are required to pass through a specific vertex. Next we extend the notion of edge and path lengths to sets:

▶ **Definition 12.** Let S be a set of paths in G. We define $w(S)$ as the length of the shortest path in S. If $S = \emptyset$, $w(S) = \infty$.

In particular, $w(Q_G(u, v))$ is the length of the shortest u-v path in G, and $w(Q'_{G,C}(u, v, c))$ is the length of the shortest u-v path in G which has a first C-intersecting vertex c. We make a note that for some vertices $c \in C$, it may be the case that $w(Q'_{G,C}(u, v, c)) > w(Q_G(u, v))$. However, if $s \in A$ and $t \in B$, then there must be some $c \in C$ such that $w(Q'_{G,C}(u, v, c)) = w(Q_G(u, v))$ as any shortest path $p \in Q_G(u, v)$ must intersect separator C. There may be multiple such c, but at least one must always exist.

With these definitions in place, we now give a similar set of lemmas to compute $|Q_G(u, v)|$ by computing $|Q'_{G,C}(u, v, c)|$ for all paths that intersect the separator C. For the cases where $s \in C$ or $t \in C$, we note that the s-s and the t-t paths consisting of only one vertex have a weight of 0. As before, the remaining paths which lie entirely within A or B can be counted with recursion.

▶ **Lemma 13.** *For a graph G with separator (A, B, C) and a vertex $c \in C$, $|Q'_{G,C}(s, t, c)| = |Q'_{G,C}(s, c, c) \times Q_G(c, t)|$ and $w(Q'_{G,C}(s, t, c)) = w(Q'_{G,C}(s, c, c)) + w(Q_G(c, t))$.*

Proof. To prove the first part, we show a bijection between $Q'_{G,C}(s, t, c)$ and $Q'_{G,C}(s, c, c) \times Q_G(c, t)$. We map $p \in Q'_{G,C}(s, t, c)$ to a pair of paths $p_1 \in Q'_{G,C}(s, c, c)$ and $p_2 \in Q_G(c, t)$ as follows: let p_1 be the s-c sub-path of p and p_2 be the c-t sub-path of p. By Observation 1, both p_1 and p_2 must be shortest paths. Since we split p at c, p_1 only intersects the separator at c and thus $p_1 \in Q'_{G,C}(s, c, c)$. Since p_2 is a shortest path, $p_2 \in Q_G(c, t)$. Also, the map from p to (p_1, p_2) is injective by the same argument as in Lemma 8.

Conversely, let path $p_1 \in Q'_{G,C}(s, c, c)$ and path $p_2 \in Q_G(c, t)$. Since both p_1 and p_2 are shortest paths, it follows that the path p formed by concatenating p_1 and p_2 is a shortest path with respect to all s-t paths with first C-intersecting c. If a shorter s-t path with C-intersecting vertex c existed, then it either contains a shorter s-c or c-t sub-path than p_1 or p_2 respectively[2] which contradicts p_1 and p_2 being shortest paths. Thus for any pair

[2] This is not true for all s-t paths. There may be a shorter s-t path with a different first C crossing vertex. Such a case will be determined by a query by comparing path lengths across the vertices in the separator.

$p_1 \in Q'_{G,C}(s,c,c)$ and $p_2 \in Q_G(c,t)$, there is a corresponding path $p \in Q'_{G,C}(s,t,c)$ which maps to (p_1,p_2) by the above map. Thus the map is also surjective.

Since p was formed by concatenating p_1 with p_2, we have $w(p) = w(p_1) + w(p_2)$. Because all paths in each of Q and Q' have the same weight, $w(Q'_{G,C}(s,t,c)) = w(Q'_{G,C}(s,c,c)) + w(Q_G(c,t))$. ◀

We next show how to compute $|Q'_{G,C}(s,c,c)|$. For a graph G with edge weights w_G, construct G'_c according to Definition 9 and add weights $w_{G'_c}$ as follows: for $(u,v) \in E(G'_c)$ let $w_{G'_c}(u,v) = w_G(v,u)$. We get the following weighted version of Lemma 10:

▶ **Lemma 14.** *In a graph G with separator (A,B,C) and a vertex $c \in C$, $|Q'_{G,C}(s,c,c)| = |Q_{G'_c}(c,s)|$ and $w(Q'_{G,C}(s,c,c)) = w(Q_{G'_c}(c,s))$.*

The last piece we need in order to build an oracle for the number of shortest paths in a planar graph is a way to compute $|Q_G(u,v)|$ efficiently. As discussed in Section 2, in planar graphs we can compute both $|Q_G(u,v)|$ and $w(Q_G(u,v))$ in $O(n)$ time for all vertices $v \in V(G)$ and a source vertex $u \in V(G)$. Then, we can build an oracle for counting shortest paths by mimicking Algorithm 2 where the CountPaths calls get replaced with CountShortestPaths calls, see Algorithm 1. As before, both construction time for the oracle and the space needed are $O(n^{1.5})$ for planar graphs.

Algorithm 4 Build a Shortest Path Counting Oracle for graph G.

```
procedure ConstructShortestPathOracle(G)
    if |V(G)| = 0 then return
    find a separator (A, B, C) in G (and store it)
    for c ∈ C do
        construct graph G'_c (see Definition 9, plus add weights)
        call CountShortestPaths(G, c)
        call CountShortestPaths(G'_c, c)
        store the respective counts as Q_G[c, v] and Q_{G'_c}[c, v],
        store also the corresponding path lengths as w_G[c, v] and w_{G'_c}[c, v], respectively
    ConstructOracle(G[A])
    ConstructOracle(G[B])
```

Querying the oracle requires a few extra conditions, see Algorithm 5, but it can still be done in time $O(\sqrt{n})$. As we noted before, for some $c \in C$, $w(Q'_{G,C}(s,t,c))$ may be larger than $w(Q_G(s,t))$. We can detect this by comparing $w(Q'_{G,C}(s,c,c)) + w(Q'(c,t,G))$. Since at least one $c \in C$ must have $w(Q'_{G,C}(s,t,c)) = w(Q_G(s,t))$, we can have our query determine which c satisfy $\min w(Q'_{G,C}(s,c,c)) + w(Q_G(c,t)$ and only add counts from those c. We assume that the distance results from the SSSP are stored along with the number of shortest paths as before. We double the amount of space required, but this still falls within $O(n^{1.5})$ space.

This analysis leads to the following theorem:

▶ **Theorem 15.** *For any planar graph G, there exists an oracle for #SHORTPATH-ORACLE which takes $O(n^{1.5})$ space, takes $O(n^{1.5})$ time to construct, and for any pair of vertices $s,t \in V(G)$ the oracle can answer queries about the number of paths from s to t in $O(\sqrt{n})$ time.*

Algorithm 5 Querying the #SHORTPATH-ORACLE.

procedure QUERYPATHS(G, s, t)
  $numPaths = 0$
  $minDist = \infty$
  while $(s, t \in A)$ or $(s, t \in B)$, where (A, B, C) is the stored separator of G **do**
    for $c \in C$ **do**
      if $w_{G'_c}[c, s] + w_G[c, t] < minDist$ **then**
        $minDist = w_{G'_c}[c, s] + w_G[c, t]$
        $numPaths = Q_{G'_c}[c, s]Q_G[c, t]$
      else if $w_{G'_c}[c, s] + w_G[c, t] = minDist$ **then**
        $numPaths+ = Q_{G'_c}[c, s]Q_G[c, t]$
    if $s, t \in A$ **then** $G = G[A]$
    else $G = G[B]$
  for $c \in C$ **do**
    if $w_{G'_c}[c, s] + w_G[c, t] < minDist$ **then**
      $minDist = w_{G'_c}[c, s] + w_G[c, t]$
      $numPaths = Q_{G'_c}[c, s]Q_G[c, t]$
    else if $w_{G'_c}[c, s] + w_G[c, t] = minDist$ **then**
      $numPaths+ = Q_{G'_c}[c, s]Q_G[c, t]$

5 Generalizing the Oracle

In this section, we relax the constraints of the previous sections to generalize the oracle data structure. The constraints we require are as follows:

- G has positive edge weights, and
- G has an $(\alpha, f(n))$-balanced (A, B, C) separator which can be found in time $O(g(n))$.

Then, we can use Algorithms 1-5 (in fact, Algorithm 1 works for any graph and does not require separators) as stated and the proofs of correctness still hold. However, we need to rework the running time estimates and space bounds. The running time of the oracle construction is given by the following recurrence, where $T_{SSSP}(n)$ denotes the running time of SSSP:

$$T(n) \leq T(\alpha n) + T((1-\alpha)n) + O(T_{SSSP}(n)f(n)) + O(g(n)).$$

For simplicity, let us express the additive term as $\hat{f}(n)$. As before, this recurrence can be evaluated using the Akra-Bazzi Method. Again, the p value for which $\sum_i a_i b_i^p = 1$ is 1. (Take $a_0 = a_1 = 1$, $b_0 = \alpha$, and $b_1 = 1 - \alpha$.) With $p = 1$, the recurrence is evaluated as follows:

$$T(n) = \Theta\left(x^1\left(1 + \int_1^x \frac{\hat{f}(u)}{u^2}du\right)\right).$$

This splits nicely into three cases.

1. If $\hat{f}(n) = o(n)$, then $T(n) = \Theta(n)$.
2. If $\hat{f}(n) = \Theta(n \log^a n)$, then $T(n) = \Theta(n \log^{a+1} n)$.
3. If for every a we have $\hat{f}(n) = \omega(n \log^a n)$, then $T(n) = \Theta(\hat{f}(n))$.

Therefore, the running time of the oracle construction can be determined using $\hat{f}(n) = O(T_{SSSP}(n)f(n)) + O(g(n))$. The space requirement of this algorithm is $\hat{f}(n) = O(nf(n)+n)$ (this can be done by only storing which side of the separator a vertex lies in (A, B, or C), distances and numbers of paths to a vertex for each vertex in the separator.

In real applications of this oracle, case 1 will never occur as running SSSP takes $\Omega(n)$ time in any graph. However, case 2 may occur. In fact, for outerplanar graphs, which have $m = O(n)$ (since they are planar) but which also have $O(1)$ separators, case 2 applies, giving a running time of $O(n \log n)$ to build this oracle and a running time of $O(\log n)$ to query it.

The time to query this oracle data structure is given by the following recurrence: $T(n) = T(\alpha n) + 2f(n)$. As before, we only query A or B until the separator splits vertices s and t. This recurrence can be evaluated with the Master Theorem giving a query time as follows:

1. If $f(n) = o(\log n)$, then $T(n) = \Theta(\log n)$.
2. If $f(n) = \Omega(\log n)$, then $T(n) = \Theta(f(n))$.

Putting together all of this, we have the following theorem.

▶ **Theorem 16.** *Let G be a graph with $(\alpha, f(n))$ balanced separators which can be found in $g(n)$ time and let $T_{SSSP}(n)$ be the time needed to solve SSSP for G. Let $\hat{f}(n) = T_{SSSP}(n)f(n) + g(n)$. An oracle data structure for counting the number of shortest paths in G between any pair of vertices can be computed in the following time bounds:*

$\hat{f}(n)$	*Construction Time*
$o(n)$	$T(n) = \Theta(n)$
$\Theta(n \log^a n)$	$T(n) = \Theta(n \log^{a+1} n)$
$\omega(n \log^a n)$ *for all* a	$T(n) = \Theta(\hat{f}(n))$

$f(n)$	*Query Time*
$f(n) = o(\log n)$	$T(n) = \Theta(\log n)$
$f(n) = \Omega(\log n)$	$T(n) = \Theta(f(n))$

The space bounds required are $O(nf(n) + m)$.

For classes of graphs with small separators and fast SSSP algorithms (e. g. planar graphs and graphs of bounded genus) this oracle can improve the running time. We have seen that for planar graphs, the running time is bounded by $O(n^{1.5})$ and the query time for $O(k)$ pairs is given by $O(\sqrt{n}k)$. In graphs of bounded genus, SSSP can be done in linear time, in particular $h(g)n$ for some function h of the genus g and it is possible to find a balanced separator of size $O(\sqrt{gn})$. Thus these graphs have path counting oracles which can be found in $O(\sqrt{gn}h(g)n) = O(h(g)g^{0.5}n^{1.5})$ time, take $O(\sqrt{gn}n) = O(g^{0.5}n^{1.5})$ space, and answer queries in time $O(\sqrt{gn})$.

▶ **Application.** We conclude with mentioning how our oracle provides an improvement in the running time of the algorithm of Chambers, Fox, and Nayyeri [7] counting minimum (s,t)-cuts in graphs of bounded genus. Due to space constraints we do not reproduce their algorithm here but only discuss the parts that are relevant to our improvement of their original running time of $2^{O(g)}n^2$. The main component contributing to this running time (see Section 5.3 in [7]) is iterating through $2^{O(g)}$ "crossing sequences," which determine the different "shapes" of the possible minimum (s,t)-cuts. For each such crossing sequence a DAG embedded in a surface of the same genus is constructed, and in this DAG one needs to compute the number of paths between $O(n)$ pairs of vertices. Arithmetic operations (addition, multiplication) on these numbers then yield the desired number of minimum (s,t)-cuts. The original work simply bounded the running time needed for these cut-counts as $O(n^2)$, yielding an overall $2^{O(g)}n^2$ running time. Using our oracle approach, the running time becomes $O(\sqrt{gn})$ per query with $O(n)$ queries, totaling $2^{O(g)}\sqrt{g}n^{1.5}$ time, which includes the construction of the $2^{O(g)}$ oracles. The overall improved running time, including a maximum flow computation and a triangulation transformation of the input graph (both of which can be bounded by $O(n^2)$), is then $2^{O(g)}\sqrt{g}n^{1.5} + O(n^2)$.

References

1 Ittai Abraham, Shiri Chechik, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. On Dynamic Approximate Shortest Paths for Planar Graphs with Worst-case Costs. *SODA*, pages 740–753, 2016.

2 Mohamad Akra and Louay Bazzi. On the Solution of Linear Recurrence Equations. *Computational Optimization and Applications*, 10(2):195–210, 1998.

3 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 *Proceedings of Algorithms - ESA '96, Fourth Annual European Symposium*, pages 514–528, 1996.

4 Michael O. Ball and J. Scott Provan. Calculating Bounds on Reachability and Connectedness in Stochastic Networks. *Networks*, 13:253–278, 1983.

5 Ivona Bezáková and Adam J. Friedlander. Counting and Sampling Minimum (s, t)-Cuts in Weighted Planar Graphs in Polynomial Time. *Theor. Comp. Sci.*, 417:2–11, 2012.

6 Sergio Cabello. Many Distances in Planar Graphs. *Algorithmica*, 62(1-2):361–381, 2012.

7 Erin W. Chambers, Kyle Fox, and Amir Nayyeri. Counting and Sampling Minimum Cuts in Genus g Graphs. *Discrete & Computational Geometry*, 52(3):450–475, 2014.

8 Danny Z. Chen and Jinhui Xu. Shortest path queries in planar graphs. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 469–478, 2000.

9 Vincent Cohen-Addad, Søren Dahlgaard, and Christian Wulff-Nilsen. Fast and Compact Exact Distance Oracle for Planar Graphs. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 962–973, 2017.

10 Hristo Djidjev. On-Line Algorithms for Shortest Path Problems on Planar Digraphs. In *Proceedings of Graph-Theoretic Concepts in Computer Science, 22nd International Workshop, WG*, pages 151–165, 1996.

11 Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *J. Comput. Syst. Sci.*, 72(5):868–889, 2006.

12 Jörg Flum and Martin Grohe. The Parameterized Complexity of Counting Problems. *SIAM J. Comput.*, 33(4):892–922, 2004.

13 Pawel Gawrychowski, Shay Mozes, Oren Weimann, and Christian Wulff-Nilsen. Better Tradeoffs for Exact Distance Oracles in Planar Graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 515–529, 2018.

14 John R. Gilber, Joan P. Hutchinson, and Robert Endre Tarjan. A Separator Theorem for Graphs of Bounded Genus. *Journal of Algorithms*, 5(3):391–405, 1984.

15 Monika Rauch Henzinger, Philip N. Klein, Satish Rao, and Sairam Subramanian. Faster Shortest-Path Algorithms for Planar Graphs. *J. Comput. Syst. Sci.*, 55(1):3–23, 1997.

16 Richard J. Lipton and Robert E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979. doi:10.1137/0136016.

17 Matúš Mihalák, Rastislav Šrámek, and Peter Widmayer. Approximately Counting Approximately-Shortest Paths in Directed Acyclic Graphs. *Theory Comput. Syst.*, 58(1):45–59, 2016.

18 Shay Mozes and Christian Sommer. Exact distance oracles for planar graphs. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA,*, pages 209–222, 2012.

19 J. Scott Provan and Michael O. Ball. The Complexity of Counting Cuts and of Computing the Probability that a Graph is Connected. *SIAM J. Comput.*, 12(4):777–788, 1983.

20 Leslie G. Valiant. The Complexity of Enumeration and Reliability Problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

21 Masaki Yamamoto. Approximately counting paths and cycles in a graph. *Discrete Applied Mathematics*, 217:381–387, 2017.