UNIVERSITY OF COPENHAGEN
FACULTY OF SCIENCE

PhD thesis

On shortest path algorithms

Decremental and planar embedded graphs

Viktor Fredslund-Hansen

Advisor: Christian Wulff-Nilsen

Submitted: 18. juni 2022

This thesis has been submitted to the PhD School of The Faculty of Science, University of Copenhagen

Abstract

This thesis concerns itself with shortest path algorithms for decremental and planar embedded graphs, and constitutes a synopsis of four manuscripts:

Truly Subquadratic Exact Distance Oracles with Constant Query Time for Planar Graphs: We present a truly subquadratic size distance oracle for reporting, in *constant* time, the exact shortest-path distance between any pair of vertices of an undirected, unweighted planar graph G. For any $\varepsilon > 0$, our distance oracle requires $O(n^{5/3+\varepsilon})$ space and is capable of answering the distance between any pair of vertices of G in worst-case time $O(\log(1/\varepsilon))$. Previously no truly sub-quadratic size distance oracles with constant query time for answering exact shortest paths distance queries existed. This manuscript was accepted at ISAAC 2021.

Near-Optimal Distance Oracles for Vertex-Labeled Planar Graphs: Given an undirected n-vertex planar graph $G = (V, E, \omega)$ with non-negative edge weight function $\omega : E \to \mathbb{R}$ and given an assigned label to each vertex, a vertex-labeled distance oracle is a data structure which, for any query consisting of a vertex u and a label λ reports the distance from u to the nearest vertex with label λ. We show that if there is a distance oracle for undirected n-vertex planar graphs with non-negative edge weights using $s(n)$ space and with query time $q(n)$, then there is a vertex-labeled distance oracle with $\tilde{O}(s(n))$ space and $\tilde{O}(q(n))$ query time. Using the state-of-the-art distance oracle of Long and Pettie [LP21], our construction produces a vertex-labeled distance oracle of size $n^{1+o(1)}$ and with query time $\tilde{O}(1)$ at one extreme, and of size $\tilde{O}(n)$ and with $n^{o(1)}$ query time at the other extreme, as well as such oracles for the full tradeoff between space and query time obtained in their paper. To our knowledge, this is the first non-trivial exact vertex-labeled distance oracle for planar graphs and, for any interesting graph class other than trees. This manuscript was accepted at ISAAC 2021.

Decremental APSP in Directed Graphs: Given an unweighted digraph $G = (V, E)$, undergoing a sequence of edge deletions, with $m = |E|, n = |V|$, we consider the problem of maintaining all-pairs shortest paths (APSP). Whilst this problem has been studied in a long line of research [ACM'81, FOCS'99, FOCS'01, STOC'02, STOC'03, SWAT'04, STOC'13] and the problem of $(1 + \varepsilon)$-approximate, weighted APSP was solved to near-optimal update time $\tilde{O}(mn)$ by Bernstein [STOC'13], the problem has mainly been studied in the context of an *oblivious* adversary which fixes the update sequence before the algorithm is started. In this paper, we make significant progress on the problem for an adaptive adversary which can perform updates based on answers to previous queries:

- We first present a *deterministic* data structure that maintains the *exact* distances with total update time $\tilde{O}(n^3)$[1].
- We also present a *deterministic* data structure that maintains $(1+\varepsilon)$-approximate distance estimates with total update time $\tilde{O}(\sqrt{m}n^2/\varepsilon)$ which for sparse graphs is $\tilde{O}(n^{2+1/2}/\varepsilon)$.
- Finally, we present a randomized $(1 + \varepsilon)$-approximate data structure which works against an adaptive adversary; its total update time is $\tilde{O}(m^{2/3}n^{5/3} + n^{8/3}/(m^{1/3}\epsilon^2))$ which for sparse graphs is $\tilde{O}(n^{2+1/3}/\epsilon^2)$.

Our exact data structure matches the total update time of the best *randomized* data structure by Baswana et al. [STOC'02] and maintains the distance matrix in near-optimal time. Our approximate data structures improve upon the best data structures against an adaptive adversary which have $\tilde{O}(mn^2)$ total update time [JACM'81, STOC'03]. This manuscript was accepted at ICALP 2021.

[1] We use $\tilde{O}$-notation to hide logarithmic factors.

Degree of Convexity and Expected Distances in Simple Polygons: We present a simple algorithm for computing the so-called *Beer-index* of a simple polygon P in $O(n^2 \log n)$ time, where n is the number of corners of P. The Beer-index is the probability that two points chosen independently and uniformly at random in P can see each other. Given a finite set M of m points in P, we also show how the number of pairs in M that see each other can be computed in $O(n \log n + m \log^2 n + m \log n \log m)$ time, which is close to optimal. We likewise study the problem of computing the expected geodesic distance between two random points in P. We show how the expected L_1-distance can be computed in optimal $O(n)$ time by a conceptually very simple algorithm. We then describe an algorithm that outputs an expression for the expected L_2-distance in $O(n^2 \log^2 n)$ time. This manuscript is unpublished.

Resumé

Denne afhandling beskæftiger sig med korteste-vej algoritmer for dekrementelle og plane indlejrede grafer. Den er en synopsis af fire manuskripter:

Afstandsorakler med polynomielt subkvadratisk pladsforbrug og konstant forespørgselstid for plane grafer: Vi præsenterer et afstandsorakel af polynomiel subkvadratisk størrelse der tillader rapportering, i *konstant* tid, af den nøjagtige korteste-vej afstand mellem par af knuder af en ikke-orienteret, uvægtet plan graf G. For ethvert $\varepsilon > 0$ kræver vores afstandsorakel $O(n^{5/3+\varepsilon})$ plads og er i stand til at besvare afstandsforespørgsler om den korteste-vej for ethvert par af knuder på G i tid højst $O(\log(1/\varepsilon))$. Tidligere eksisterede der ingen afstandsorakler med polynomielt subkvadratisk størrelse og konstant forespørgselstid. Dette manuskript blev accepteret ved ISAAC 2021.

Næroptimale afstandsorakler for plane grafer med knudemærkater: Givet en uorienteret plan graf $G = (V, E, \omega)$ med n knuder og ikke-negativ kantvægtfunktion $\omega : E \mapsto \mathbb{R}$ samt en etiket tildelt hver knude, er et afstandsorakel for plane grafer med knudemærkater en datastruktur, der givet en forespørgsel bestående af en knude u og en etiket λ rapporterer den korteste vej afstand fra u til nærmeste knude med etiket λ. Vi viser, at hvis der er et afstandsorakel for ikke-orienterede plane grafer med n knuder, ikke-negative kantvægte, der har pladsforbrug $s(n)$ og forespørgselstid $q(n)$, så eksisterer der et afstandsorakel for plane grafer med knudemærkater, der bruger $\tilde{O}(s(n))$ plads og med $\tilde{O}(q(n))$ forespørgselstid. Ved brug af afstandsoraklet fra Long og Pettie [LP21], giver vores konstruktion et afstandsorakel for plane grafer med knudemærkater der bruger $n^{1+o(1)}$ plads og med $\tilde{O}(1)$ forespørgselstid i en ende af spektret og $\tilde{O}(n)$ pladsforbrug og $n^{o(1)}$ forespørgselstid i den anden ende af spektret. Dette er, så vidt vi ved, det første ikke-trivielle afstandsorakel for plane grafer med knudemærkater for en interessant grafklasse udover træer. Dette manuskript blev accepteret ved ISAAC 2021.

Dekrementel APSP in orienterede grafer imod en adaptiv modstander: Givet en orienteret graf $G = (V, E)$, der gennemgår en online sekvens af kantsletninger med m kanter i begyndelsen, og hvor $n = |V|$, kigger vi på vedligeholdelse af korteste veje i G. Vi studerer problemet med antagelse af en adaptiv modstander, hvor denne kan basere opdateringssekvensen baseret på outputtet af datastrukturforespørgsler. Vi præsenterer tre nye datastrukturer der virker under forskellige antagelser:

- Vi præsenterer først en *deterministisk* datastruktur, der vedligeholder *nøjagtige* afstande med samlet opdateringstid $\tilde{O}(n^3)$.
- Vi præsenterer også en *deterministisk* datastruktur, der vedligholder $(1+\varepsilon)$-approksimative afstandsestimater med samlet opdateringstid $\tilde{O}(\sqrt{m}n^2/\varepsilon)$ som for tilstrækkeligt små grafer svarer til $\tilde{O}(n^{2+1/2}/\varepsilon)$.
- Til sidst præsenterer vi en randomiseret $(1+\varepsilon)$-approksimativ datastruktur, som virker imod en adaptiv modstander; dennes samlede opdateringstid er $\tilde{O}(m^{2/3}n^{5/3} + n^{8/3}/(m^{1/3}\epsilon^2))$ som for tilstrækkeligt små grafer svarer til $\tilde{O}(n^{2+1/3})$.

Vores nøjagtige datastruktur matcher den samlede opdateringstid for den bedste *randomiserede* datastruktur af Baswana et al. og vedligeholder afstandsmatricen i nær-optimal tid. Vores approksimative datastrukturer forbedrer de bedste datastrukturer i der antager en adaptiv modstander; denne har $\tilde{O}(mn^2)$ samlet opdateringstid. Dette manuskript blev accepteret på ICALP 2021.

Konveksitetsgrad og forventede afstande i simple polygoner: Vi præsenterer en simpel algoritme til beregning af den såkaldte *Beer-indeks* i et simpelt polygon P i $O(n^2 \log n)$ tid, hvor n er antallet af hjørner af P. Beer-indekset er sandsynligheden

for, at to punkter valgt uafhængigt og uniformt tilfældigt i P kan se hinanden. Givet en endelig mængde M af m punkter i P, viser vi også, hvordan antallet af par i M, der kan se hinanden, kan beregnes i $O(n \log n + m \log^2 n + m \log n \log m)$ tid, som er tæt på optimalt. Vi studerer ligeledes problemet med at beregne den forventede geodætiske afstand mellem to tilfældigt valgte punkter i P. Vi viser hvordan den forventede L_1-afstand kan beregnes i optimal $O(n)$ tid ved hjælp af en begrebsmæssigt meget simpel algoritme. Vi beskriver derefter en algoritme, der udskriver et udtryk for den forventede L_2-afstand for to tilfældigt valgte punkter i P i $O(n^2 \log^2 n)$ tid. Dette manuscript er endnu ikke udgivet.

Preface

This thesis concerns itself with shortest-path algorithms for dynamic and planar embedded graphs. It provides a synopsis of four articles, of which the first three are published at conferences and where the last is an unpublished manuscript. In the body of this thesis an overview of each article and the results they convey is given, by defining the problems they attempt to solve, related work to put the results in a context and finally a brief technical overview might be given, hopefully providing the reader with some idea on the approaches taken. Some parts of this thesis was simply lifted from the manuscripts and altered where applicable to fit the pedagogic ambitions of this thesis, while others are original. The idea is to provide a context in which to place the results as well as to give high-level and hopefully somewhat intuitive overview of some techniques used and challenges encountered, while skipping over some of the tedious details presented in the manuscripts - albeit this is admittedly a highly subjective matter. Note that while the full manuscripts are included in appendices, they may not be be completely identical to the published versions. The topics covered in this thesis are: Distance oracles for planar grahs, vertex-labeled distance oracles for planar grahs, decremental dynamic APSP and measures polygons. The manuscripts considered are found in Appendices A, B, C and D.

Acknowledgments

Thank you!

To Christian, for being the best teacher and supervisor I could never imagine.

To my dear friends, coauthors and colleagues.

To my mother, sister and grandparents, I love you dearly.

To the great teachers I was blessed with throughout my life.

Special thanks to Joakim Blikstad, Jonas Conneryd, Andreas Høier and Frederik Peter Højte Poulsen for providing the errata addressed in this thesis, and for the corrections they so kindly provided.

♡

Contents

1 Preliminaries

Let $G = (V, E)$ be a graph. If G is edge-weighted, it furthermore has an assignment $\omega : E \mapsto D$ that maps edges to some range D, where typically $D \subset \mathbb{R}$. In this case we write $G = (V, E, \omega)$. For a graph H we denote by $V(H)$ and $E(H)$ the vertex and edge set of H, and overload this notation for faces of graphs as well. Thus if f is a face of some graph, $V(f)$ are the vertices on the face f. We denote by $u \rightsquigarrow_G v$ the shortest path from u to v in G and by $d_G(u, v)$ the length of this shortest path. When the context is clear, we will omit the subscripts and simply write $u \rightsquigarrow v$ and $d(u, v)$. We furthermore let $f(n) \in \tilde{O}(g(n))$ if there is some k s.t. $f(n) \in O(g(n) \log^k g(n))$, i.e. $\tilde{O}(g(n))$ hides polylog factors of $g(n)$.

1.1 Dynamic graph algoritms

A dynamic graph algorithm is an algorithm that maintains information about a graph that is subject to updates, for instance insertions and deletions of edges or vertices. We say that a dynamic graph problem, respectively an algorithm for solving it, is *decremental* if it only allows for deletions, *incremental* if it only allows for insertions and *fully-dynamic* if it allows for both in the dynamic graph. In the above cases, we shall simply refer to the graph as being decremental, incremental or fully dynamic. Incremental and decremental graphs are referred to as being *partially-dynamic*. A dynamic graph algorithm aims to efficiently process a sequence of online updates interspersed with queries about some property of the underlying dynamic graph, in our case shortest path distances.

Performance guarantees The performance of dynamic graph algorithms can be measured in various ways. For instance, in the incremental, respectively decremental case, where only edge deletions, respectively insertions, are allowed, it may seem natural to measure the performance in terms of the *total update time*, that is, the total number of steps used by the algorithm until the edge set becomes maximal, respectively empty. This provides an amortized guarantee. Other measure are then naturally the worst case update time, the expected update time, and the expected total running time.

Adversarial models Dynamic graph algorithms assume various *adversarial models*. This is in particular the case for the decremental algorithms of Section 4. The adversarial model defines the assumptions under which the sequence of updates and queries are assumed to be made to a dynamic algorithm by the user, referred to as the *adversary*. We say that a performance guarantee, e.g. on the total running time, of a dynamic algorithm holds against an *oblivious* adversary if the guarantee is stated with the assumption that the adversary must define the sequence of updates before the algorithm starts. In this case, the sequence of updates must therefore necessarily be independent of any random bits used by the algorithm. This is a weaker notion than that of an *adaptive* adversary, who is allowed to create the update sequence "on the go", e.g. based on answers to previous queries made to the algorithm. Depending on the data structure, these choices may not be independent of the random choices made, which may result in the data structure performing poorly. One key advantage of an algorithm that works against an adaptive adversary is that it can be used as a black box, regardless of whether the user

of it bases their sequence of updates based on the output of the algorithm. We point out that *deterministic* data structures always work against an adaptive adversary.

1.2 Machine models

In sections Section 2, Section 3 and Section 4 we assume the standard word-RAM model [Hag98] for computations. Section Section 5 assumes the *real RAM model* which can be thought of as an extension of the integer word-RAM model [Hoo22]. The main difference from the word-RAM model is that, it in addition has access to real registers. Numbers represented in real registers can be assumed to be real values (of infinite precision) and accordingly the model assumes certain operations on real values that each can be executed in unit time. One example is that this allows for the representation for a sum of square roots, which for instance could represent the distance of a path in Euclidean space. Some care has to be taken in the definition of this model, since for the model to be interesting, it turns out that certain types of operations must be disallowed, as it would otherwise imply a collapse of important complexity classes model.

2 Distance Oracles for Planar Graphs

In the following we consider the problem of resolving shortest path queries in planar graphs:

Problem 2.1. *Given a graph $G = (V, E)$ describe a compact data structure which efficiently supports queries $u, v \in V$ that report the shortest path distance from u to v.*

A *distance oracle* is a compact data structure that given a pair of vertices is capable of efficiently reporting the length of the shortest path going between them. By "efficiently", we usually mean answering distance queries in constant or sub poly-logarithmic time in the size of the input graph, and by compact, we refer to keeping the space usage as low as possible.

2.1 Related work

A naive approaches to solving the problem would be to represent the distances by an $n \times n$ distance matrix, as this allows for answering distance queries in constant time by looking up the entry corresponding to the pair. The obvious drawback of this approach, however, is the large space requirement of $\Theta(n^2)$ which is impractical for larger graphs. At the other extreme, one could imagine to to lazily compute the shortest path distance from scratch with each query, for instance with Dijkstra's algorithm. This approach only uses memory for storing the input graph and running the shortest path algorithm, and has no requirement for preprocessing. However, the high query time makes this approach infeasible for many applications where a small query time is of importance.

Lower bounds It is well known that there are graphs for which no distance oracle with $o(n^2)$ bits of space and $O(1)$ query time exists. More generally, Thorup and Zwick showed that for any $k \in \mathbb{N}$, there are dense graphs for which any distance oracle with *stretch* at most $2k - 1$ and constant query time requires $\omega(n^{1+1/k})$ bits of space[TZ05];

this is conditioned on the widely believed *girth conjecture* of Erdős [Erd65] which has been proven for small values of k. In the context of distance oracles, *stretch* refers to the multiplicative approximation error of the reported distances, that is, a distance oracle which reports an estimate $\tilde{d}_G(u,v)$ for which $\tilde{d}_G(u,v) \leq t \cdot d_G(u,v)$ is said to have stretch t. We refer to a distance oracle with stretch $t = 1$ as an *exact distance oracle* and as an *approximate distance oracle* otherwise. Furthermore Pătraşcu and Roditty [PR14] showed that there are sparse graphs on $O(n \operatorname{polylog} n)$ edges for which constant query-time distance oracles with stretch less than 2 must be of size $\Omega(n^2 \operatorname{polylog} n)$, assuming the set intersection conjecture.

Planar graphs The lower bounds mentioned above provide evidence that a trivial lookup table is essentially the best possible data structure for constant query time in general graphs. It is therefore natural to consider the study of distance oracles in more restricted settings, for instance for certain classes of graphs or when we relax the requirement that exact distances be reported. This section of the thesis concerns itself with the special case of exact distance oracles for planar graphs. To the knowledge of the authors there are no non-trivial lower bounds for distance oracles for planar graphs (with the exception of conditional lower bounds), and thus the "holy grail" for distance oracles for planar graphs would be a distance oracle of near-linear size with constant query time. In the following we provide notable as well as recent developments for subquadratic space exact distance oracles for planar graphs:

Many constructions in the literature of distance oracles for planar graphs provide tradeoffs between query time and size [Dji96; Ari+96; Cab12; CX00; FR06; MS12; Nus11]. However, all of these distance oracles use essentially quadratic space for poly-logarithmic query time. As a matter of fact, it was, until recently, a major open problem whether a size $O(n^{2-\varepsilon})$ exact distance oracle with poly-logarithmic query-time could be constructed for an n-vertex planar graph where $\varepsilon > 0$ is a constant. This was answered in the affirmative by Cohen-Addad et al. [CDW17] who presented a construction of size $O(n^{5/3})$ distance oracle with query time $O(\log n)$, as well as a space/query-time tradeoff allowing for construction of size S distance oracles with query time $O\left((n^{5/2}(S^{3/2}) \log n\right)$ for $S \geq n^{3/2}$. Their results were inspired by the, then novel, *abstract Voronoi diagram* techniques introduced by Cabello [Cab18] who gave the first truly sub-quadratic time algorithm for computing the diameter of planar graphs. By devising an elegant point-location structure for the abstract Voronoi diagrams, Gawrychowski et al. [Gaw+18c] managed to further improve on this bound: They presented a size $O(n^{3/2})$ distance oracle with $O(\log n)$ query time, as well as a tradeoff allowing for size $O(S)$ distance oracles with query time $O(\max\left\{1, n^{3/2} \operatorname{polylog} n/S\right\})$ for $n \leq S \leq n^2$. Recently, Charalampopoulos et al. [Cha+19] presented constructions that are near-optimal. They described the following oracles for edge-weighted planar digraphs; one of size $O(n^{1+\varepsilon} \operatorname{polylog} n)$ with $O(\log^{1/\varepsilon} n)$ query time, another of size $O(n \operatorname{polylog} n)$ with $O(n^{\varepsilon} \operatorname{polylog} n)$ query time and one of size $n^{1+o(1)}$ with query-time $n^{o(1)}$ for constant $\varepsilon > 0$. They achieved this by combining the very same point location structure of [Gaw+18c] with a clever recursion scheme. This was improved slightly by Long and Pettie et al. [LP21], their construction being the state-of-the-art at the time of the writing.

It should be emphasized that all of the truly subquadratic-size exact distance oracles so far require at least poly-logarithmic query time. Considering oracles with constant

query time, the biggest improvement in space over the trivial look-up result is less than a $\log n$-factor even when restricted to unweighted undirected planar graphs [Wul13; Cha12; Wul10]. For a less recent, but more comprehensive treatment of developments and techniques employed in the field of distance oracles, we refer to the survey of Sommer [Som14]. We finally note that, despite the large body of work on distance oracles for planar graphs, it has remained an open question to determine whether an exact distance oracle with of size $O(n^{2-\varepsilon})$ with *constant* query-time can be constructed for some constant $\varepsilon > 0$. We answer this in the affirmative.

2.2 Contributions

Our result is summarized in the following theorem

Theorem 2.2. *Let $G = (V, E)$ be an undirected unweighted n-vertex planar graph. For any $\varepsilon > 0$ there exists a data structure requiring $O(n^{5/3+\varepsilon})$ space that, for any $s, t \in V$, reports the shortest path distance between s and t in G in time $O(\log 1/\varepsilon)$.*

We remark that this differs from the result as stated here slightly differs from published version, due to an erratum, see Section 2.4.

We also present another (denser) subquadratic distance oracle of size $O(n^{7/4})$ and remark that it can be distributed into a distance labeling scheme with size $O(n^{3/4})$ per label, such that the distance between any two vertices s, t can be computed in $O(1)$ time given just the labels of s and t.

2.3 Technical overview

In the following we give a brief overview of the techniques used in our constructions. We first introduce the decomposition framework for graphs used and introduce the notion of *patterns* along with some properties, which will be useful for characterizing shortest paths that cross separators in the decomposition framework. We then describe the query scheme used for the (denser) construction.

r-divisions It is well known that any planar graph admits a decomposition into $O(n/r)$ connected subgraphs, called regions, each of size $O(r)$, where the boundary of each region (i.e., the vertices that have neighbors outside the region in G) is a cycle h, with only a constant number of such cycles. To readers familiar with the concept, this is just an r-division with few holes, but *without* the important property that each region has just $O(\sqrt{r})$ boundary vertices. This is because one cannot triangulate unweighted graphs without changing the distances, as triangulating to preserve distances would entail adding edges with infinite weights. A hole of a subgraph R of G is simply a face of R which is not a face of G. For a hole h of G, we denote by $V(h)$ the vertices of G that are embedded in the non-strict interior of the cycle h in G and by $b_0^h, b_1^h, \ldots$ the vertices of h as they are encountered when walking along the edge of h in the clockwise direction. Here, we refer to b_0^h as the *canonical vertex* of h, simply by the merit that it is a distinguished vertex from which we start the walk of h.

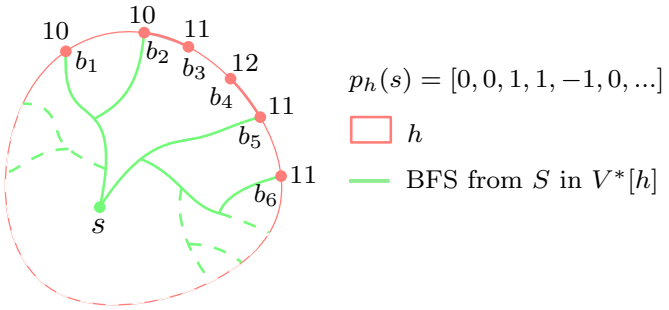


Figure 1: Illustration of a pattern.

Patterns The concept of patterns were first used by [Wul13] and later by by [LP19] (here it was referred to as a "distance tuple"). Consider a vector storing the distances from a vertex u to the vertices of a (perhaps non-simple) C. The pattern of u w.r.t. C is simply the discrete derivative of this vector. That is, the sequence obtained by taking the difference between every pair of consecutive values. This notion is illustrated in Section 2.3.

Definition 2.3. *(Pattern) Let H be a graph, h any hole of H and $u \in V(H)$. The pattern of u (w.r.t. h in H) is a vector $p_{h,H}(u)$ satisfying $p_{h,H}(u)[i] = d_H(u, b_i^h) - d_H(u, b_{i-1}^h)$ for all i.*

We note that when G is unweighted, $p_{h,H}(u)[i] \in \{0, -1, 1\}$. A simple but useful observation is that distances from $s \in V(h)$ to any vertex in H are completely determined by the pattern of s w.r.t. h, since h is a separator. This follows from a simple telescoping sums-argument when taking the prefix sum of $p_{h,H}$. Hence, we can represent the distances from any such vertex s to any vertex of h by just storing the distance from s to the canonical vertex and a pointer to the pattern of s with respect to h.

Parter [LP19] used this technique to achieve improved bounds for diameter computation for planar graphs by showing that in unweighted undirected planar graphs the number of patterns is actually quite small. More precisely, they show that the VC-dimension of a set corresponding to all patterns is at most 3. By the Sauer-Shelah lemma [Sau72], this implies that the number of distinct patterns w.r.t. a face (hole) f (h) is $O(|V(f)|^3)$ ($O(|V(h)|^3)$. Notably this is independent of the size of the graph. We next wish to outline how this observation can be used to break the quadratic space barrier. We first introduce the notion of distance from a vertex to a pattern:

Definition 2.4. *(pattern to vertex distance) Let R be a region in a graph G. Let h be a hole of R and $b = b_0^h, b_1^h, \ldots, b_k^h$ be the vertices encountered when performing the walk of h. Let p be any pattern w.r.t. h. For a vertex $v \in R$ we define $d_G(p, v)$, the distance between p and v, to be $\min_{i=0}^{k} \left\{ d_G(b_i^h, v) + \sum_{j=0}^{i} p[j] \right\}$.*

While this definition is simple, it is somewhat unnatural because the distance from a pattern to a vertex does not necessarily correspond to the length of any specific path in the graph. However, the distance between s and any vertex $t \in R$ turns out to be the sum of the distance between s and the canonical vertex v_h and the distance from the pattern of s with respect to h to t, which is apparent from this lemma:

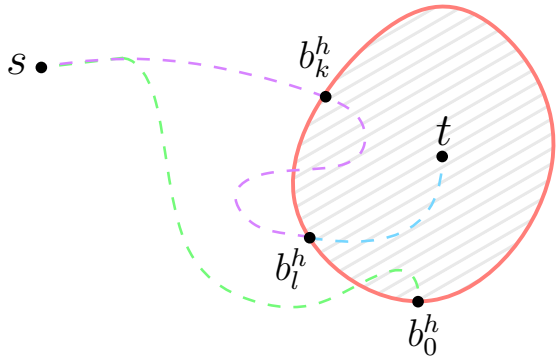


Figure 2: Illustration Lemma 2.5 and a simple query and for $s, t \in V$. If s knows its distance to b_0^h and the pattern it induces w.r.t. h, and t knows its distance to h w.r.t. the pattern, we can apply Lemma 2.5 to recover $d(s,t)$.

Lemma 2.5. *Let R be a region of a graph G. Let h be a hole of R. For every $u \in V(h)$ and every $v \in R$, $d_G(u,v) = d_G(u,b_0) + d_G(p_{h,G}(u),v))$.*

Proof. By definition of pattern and by a telescoping sum, for every $0 \leq i \leq k$, $d(u,b_i) = d(u,b_0) + \sum_{j=0}^{i} p[j]$. Let b_ℓ be any vertex of $w(h)$ on a shortest u-to-v path (b_ℓ exists since $u \in V(h)$ and $v \in R$). By choice of b_ℓ,

$$\begin{aligned} d(u,v) &= d(u,b_\ell) + d(b_\ell,v) \\ &= \min_{0\leq i\leq k} \{d(u,b_i) + d(b_i,v)\} \\ &= \min_{0\leq i\leq k} \left\{ d(u,b_0) + \sum_{j=0}^{i} p[j] + d(b_i,v) \right\} \\ &= d(u,b_0) + d(p,v) \end{aligned}$$

. □

This provides a means of recovering distances for shortest paths that cross the separator corresponding to h in G.

Simple query scheme In this section we outline the query scheme for the denser construction. Consider some r-division $R_1, R_2, \ldots$ of the input graph, and consider query vertices of $u, v \in V(G)$. We can characterize $d(u,v)$ in terms of the r-divison by noting that $u \in R_i$, $v \in R_j$ where either $i = j$ or $i \neq j$. In the case in which $i = j$ we can afford to store all pairwise distances and resolve the query by a tabular lookup. Otherwise, w.l.o.g. v is in $V(h)$ where h is a hole of R_i. Since h separates u and v in G, we get that $d(u,v) = \min_i \left\{ d_G(u,b_i^h) + d_G(b_i^h,v) \right\}$. If we assume we stored the vertex to b_0^h and a pointer to the pattern of u w.r.t. h in G as well as the distance from the pattern to v, it then follows by Lemma 2.5, that we can resolve the query in constant time. The query is illustrated in. Figure 2. Storing information about which region vertices belong to uses $O(n)$ overall space. Storing distances to patterns of all holes in a region uses $O(n/r \cdot r^4) = O(nr^3)$ space. Storing distances to canonical vertices as well as pointers

to the pattern for each vertex of each region uses $O(n/r \cdot n) = O(n^2/r)$ space. Finally, storing pairwise distances for vertices in all regions uses $O(n/r \cdot r^2)$ space. Most of these terms are subsumed by others, so overall the distance oracle requires $O(n^2/r + nr^3)$. These ideas alone already imply an oracle with space $\tilde{O}(n^{7/4})$ and constant query time. Combining these ideas with recursion and the notion of distances between patterns yields the improved space bound of Theorem 2.2.

Differences from previous oracles As we previously mentioned, breaking the quadratic space barrier for constant query time has remained a long standing open question and can therefore be considered an important result in its own right. The main difference between the approach taken in our recursive oracle and the approaches used in all existing distance oracles we are aware of, are that all existing distance oracles, both exact and approximate, and both for general graphs and planar graphs, recover the distance from s to t by identifying a vertex or vertices on some (possibly approximate) shortest path between s and t, for which distances have been stored in the preprocessing stage. These vertices are usually referred to as landmarks, portals, hubs, beacons, seeds, or transit nodes (cf. [Som14]). Our oracle, on the other hand, reports the exact shortest path without identifying vertices on the shortest path from s to t.

2.4 Errata

We remark the since the time of publishing, colleagues of ours have made us aware of two errors in this result; none of which, however, seem to be of a critical nature.

The first error pertains to the balancing of parameters in the recursive construction; specifically we were made aware that the space analysis provided in the published version should be $O(kn^{5/3+\varepsilon})$ for any $\varepsilon > 0$ instead of the claimed $O(n^{5/3+\varepsilon})$ in the published version; here ε is a parameter which controls the recursion depth (and fanout) of the decomposition. They furthermore provided a new analysis for the space used by the construction, achieving the claimed space usage $O(n^{5/3+\varepsilon})$ and $O(\log 1/\varepsilon)$ query time, which Theorem 2.2 does reflect.

The second error pertains to the choice of graph to which we apply the lemma of [LP19] as to bound the number of patterns. Consider a region R and a hole h of R. We wish to bound the number of distinct patterns for vertices of $V(h)$. To do so, we consider the graph $G-(R-h)$, but a problem arises since it is claimed that h is a face of $G-(R-h)$. This is not always the case, however; consider e.g. a case in which $R = h$. The solution is to consider the union of the portion of G corresponding to the interior of G contained in the hole h as well as h itself. Thus h may not be a simple face, but this is not a problem when defining the pattern of a vertex w.r.t. by considering the sequence $b_0^h, b_0^h, \ldots$, in which case the vertices that make h non-simple are simply encountered at most twice.

The colleagues who made us aware of this are thanked in the acknowledments; corrections will be made accordingly, and venues notified.

2.5 Future work

Future directions of research would first and foremost be the search for a construction for distance oracles of size $\tilde{O}(n^{5/3-\rho})$ for $0 < \rho \leq 2/3$ and constant query time. Since

our construction inherently relies on the notion of patterns and the bound provided by Li and Parter [LP19], showing any bound $O(r^{3-\rho})$ for $0 < \rho$ would immediately yield an improvement to our construction. An alternative proof for the number of patterns that go beyond the VC-dimension argument is given by Mozes et al. [Wal22]. Their proof instead relies on the cut-cycle duality. They furthermore show tight bounds, i.e. of $\Theta(r^2)$ patterns, for restricted families of planar graphs. We remark that there are currently no lower bounds for any family of planar graphs that go beyond $\Theta(r^2)$, and constructing examples that admit this number is quite an easy task (consider a grid graph).

Approaching the problem from the other direction, i.e. by showing any super-linear lower bound on the space required by such a construction, is also of interest.

Finally, one could consider more general or restricted families of graphs, by e.g. considering undirected or weighted graphs, or by generalizing the current construction to, say, $K_{3,3}$ or K_5-minor-free graphs.

3 Vertex-Labeled Distance Oracles for Planar Graphs

Continuing in the vein of Section 2, we now turn to a more general problem of resolving shortest path queries, where in addition to the input graph, each vertex is furthermore assigned a label:

Problem 3.1. *Given an undirected n-vertex planar graph $G = (V, E, \omega)$ and a labeling $l : V \to L$, describe a compact data structure which given a query consisting of $(u, \lambda) \in V \times L$ efficiently reports the shortest path distance from u to the nearest vertex with label λ in G.*

A data structure resolving the above problem is referred to as a *vertex-labeled distance oracle*. To give some practical motivation, the graph could represent a road network, the vertex the location of the hungry querier who is traveling by car and the label of interest being "restaurant".

3.1 Related work

We note that above problem is a generalization of that of Problem 2.1 since vertex-to-vertex distance queries can be answered by a vertex-labeled distance oracle if each vertex is given its own unique label. A trivial solution to Problem 3.1 which offers constant query time is that of a look-up table that simply stores the answers to all possible queries. This approach uses space $O(n|L|)$, which may be quadratic in n. To our knowledge, there were previously no non-trivial upper bound for vertex-labeled distance oracles for any interesting graph classes other than trees [Gaw+18b; Tsu18].

Approximations Vertex-labeled distance oracles have received considerably more attention in the approximate setting. With $(1 + \varepsilon)$ multiplicative approximation, it is known how to get $\tilde{O}(n)$ space and $\tilde{O}(1)$ query time both for undirected [LMN13] and directed planar graphs [MS18] and it has been shown how oracles with such guarantees can be maintained dynamically under label changes to vertices using $\tilde{O}(1)$ time per vertex relabel. For general graphs, vertex-labeled distance oracles with constant approximation have been presented [Her+11; Che12; Pro18a] with state of the art being an oracle with

$O(kn|L|^{1/k})$ space, $4k-5$ multiplicative approximation, and $O(\log k)$ query time, for any $k \in \mathbb{N}$.

Lower bounds The lower bounds described in Section 2.1 also apply here. For the more general case considered here, Probst Gutenberg [Pro18b] showed that for undirected, unweighted graphs, any $O(\text{polylog}\, n)$ stretch distance oracle of size S has query time at least $\Omega\left(\log \frac{\log L}{\log(S/n)+\log\log n}\right)$.

3.2 Contributions

Our contribution, is that for undirected edge-weighted planar graphs, the vertex-labeled distance oracle problem can be reduced to the distance oracle problem in the sense that (up to log-factors), any space/query time tradeoff for distance oracles also holds for vertex-labeled distance oracles. We now state our reduction and its corollary:

Theorem 3.2. *If there is an distance oracle for n-vertex undirected edge-weighted planar graphs with $s(n)$ space, $q(n)$ query time, and $t(n)$ preprocessing time, then there is an exact vertex-labeled distance oracle for such graphs using $s(n) + O(n \log^2 n)$ space, $O(q(n) \log n + \log^3 n)$ time for queries, and $t(n) + \text{poly}(n)$ time for preprocessing.*

Plugging in the distance oracle of Long and Pettie et al. [LP21] gives the following corollary which can be seen as a generalization of their result:

Corollary 3.3. *For n-vertex undirected edge-weighted planar graphs, there exist exact vertex-labeled distance oracles with the following tradeoffs between space and query time:*

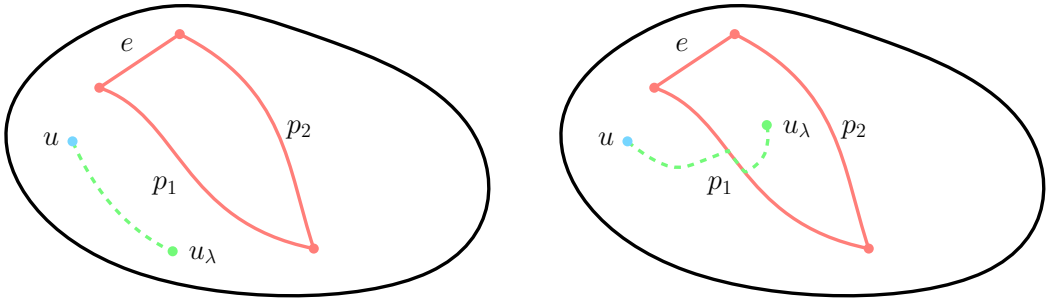
1. *$n^{1+o(1)}$ space and $\tilde{O}(1)$ query time,*
2. *$\tilde{O}(n)$ space and $n^{o(1)}$ query time.*

All oracles have preprocessing time polynomial in n.

Up to logarithmic factors, the full tradeoff between space and query time in their paper similarly extends to vertex-labeled distance oracles in undirected edge-weighted planar graphs. We note that a strength of our result being a reduction is that any future progress on distance oracles in undirected planar graphs immediately translates to vertex-labeled distance oracles.

3.3 Technical overview

Consider any fundamental cycle $C = p_1 \cup p_2 \cup \{e\}$ of G and a query pair $(u, \lambda) \in V \times L$. Denote by u_λ the vertex with label λ closest to u. The overall idea behind the construction is to efficiently identify a small (constant-size) subset W of witnesses on $V(C)$ s.t. if $s = u \rightsquigarrow u_\lambda$ intersects C, then there is some vertex $w \in W$ for which $d(u \rightsquigarrow u_\lambda) = d(u \rightsquigarrow w) + d(w \rightsquigarrow u_\lambda)$. Otherwise s is fully contained in the component corresponding to either the non-strict interior or non-strict exterior of C containing u, in which case we can apply the approach recursively to this component. This is illustrated in Figure 3. Across all levels of recursion, this yields a logarithmic-sized set W_λ, of witnesses, granted that the fundamental cycle separator balances the components



(a) $C \cap s = \emptyset$ (b) $C \cap s \neq \emptyset$

Figure 3: Fundamental cycle C and s.

in which they live across all levels. We note that it is well-known how to efficiently compute such a decomposition using fundamental cycle-separators. The closest vertex is then given by $\arg\min_{w \in W_\lambda} \{d(u, W_\lambda)\}$, which now reduces to an instance of Problem 2.1, since queries of this type can be resolved by a regular distance oracle. The above now becomes one, where given $u \in V$ and $\lambda \in L$, we wish to identify witnesses in $V(C)$ s.t. one of these certify the shortest path from u to any vertex with label λ, s.t. it intersects C. Specifically, we describe a data structure $O_{G,p}$ which given a simple path p in G does this for a path and not a cycle separator of G, and so we may simply query O_{G,p_1} and O_{G,p_2} (the paths that constitute C) and treat their union of outputs as the set of witnesses:

Lemma 3.4. *Let $G = (V, E, \omega)$ be an undirected, planar embedded, edge-weighted graph with labeling $l : V \to L$ and let p be a shortest path in G. There is a data structure $O_{G,p}$ using $O(|V| \log |V|)$ space which given $u \in V$ and $\lambda \in L$ returns a subset $C \subset V$ of constant size, s.t. if v is the vertex with label λ closest to u and $v \rightsquigarrow u$ intersects p, then $v \in C$. Each such query takes time at most $O(\log^2 |V|)$.*

Define by $S_\lambda = \{v \in V \mid l(v) = \lambda\}$ and consider a simple path p in G. In fact, the data structure of Lemma 3.4 is a union of (sub)data structures $O_{G,p,\lambda}$ that provide the same guarantees, but concern itself only with S_λ. A query (u, λ) is then simply forwarded to the appropriate data structure.

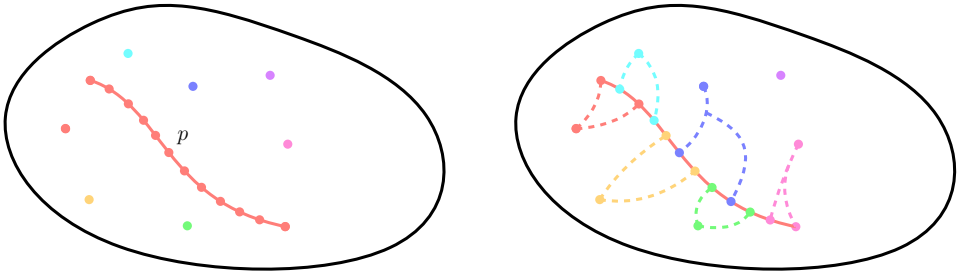
Label sequences

From a high-level point-of-view, we therefore are interested in a labeling of p according to the following definition:

Definition 3.5. *Let $G = (V, E)$ be a graph, $p = p_1, \ldots, p_k$ a sequence of vertices and $S \subseteq V$. The label-sequence of p w.r.t. S is a sequence $M_{G,S,p} \in S^k$ satisfying $M_{G,S,p}(i) = \arg\min_{s \in S} dist_G(s, p_i)$. The alternation number on p w.r.t. S in G is defined as $|M_{G,S,p}| = \sum_{i=1}^{k-1} [M_{G,S,p}(i) \neq M_{G,S,p}(i+1)]$.*

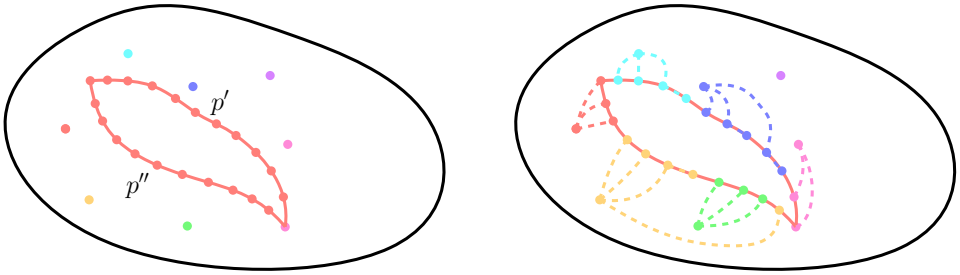
Let us consider the label-sequence of $p = p_1, \ldots$ w.r.t. S_λ. $M_{G,S_\lambda,p}$ maps indices of p to its closest vertex of label λ. On one hand, this is helpful if we can now efficiently identify a set of witnesses containing p_{i^*} where

$$i^* = \arg \min_{i \in \{1, \ldots, |p|\}} \{d(u, v) + d(v, M_{G,S_\lambda,p}(i))\}$$



(a) (b)

Figure 4: (a) Abstract representation of the path p and vertices S_λ; each color corresponds to a vertex with label λ. b) The color of a vertex of p indicates the vertex of S_λ it maps to when applying $M_{G,S_\lambda,p}$. The dotted lines indicate the shortest paths from a vertex of S_λ to vertices corresponding to its pre-image under $M_{G,S_\lambda,p}$.



(a) (b)

Figure 5: (a) Abstract representation of the paths p_1 and p_2 after incision, as well as vertices S_λ. b) The color of a vertex of p indicates the vertex of S_λ it maps to when applying M_{G,S_λ,p_1} and M_{G,S_λ,p_2}. Note that the non-crossing property entails that vertices of S_λ enclosed by shortest paths of others can not "escape", which restricts the number of alternations on p.

as then we are almost done; but there too is the issue of label sequences having alternation numbers as high as $\Theta(n)$, since the fundamental cycle separator may be proportional to the size of V. This is illustrated in Figure 4. This is a problem since each data (sub)strucure, $O_{G,p,\lambda}$, of Lemma 3.4 conceptually works by identifying p_{i^*} in a data structure whose size is proportional to the alternation number of the sequence. Storing this at even one level therefore may use as much as $\Theta(n|L|)$ space which is no better than using the naive approach. One part of our solution to this is performing an "incision" in G along p and exploiting the non-crossing property of shortest paths. The incision step corresponds to replacing p with two copies p' and p'' of p, s.t. they constitute a single new face of G. We refer to the graph G with the incision along p by G_p. Each edge incident to some vertex of p in G, prior to the incision, is now connected to the copy of that vertex, corresponding to the "side" of p from which the edge was emanating. The behaviour of label sequences of p' and p'' after the incision is illustrated in Figure 5. Intuitively this imposes an ordering on vertices of S_λ w.r.t. their shortest paths to $V(p_1) \cup V(p_2)$. We show that this ordering actually implies that the sequence $M_{G,S_\lambda,p}$ (when repeatedly replacing maximal substrings of identical symbols by a single occurence) is a *Davenport*

Schinzel-sequence or order 2, which in our case implies the following corollary:

Corollary 3.6. *Let G be an undirected, weighted planar graph, $S \subseteq V$ and p be a simple path whose vertices belong to the same face of G. Then $|M_{G,S,p}| = O(|S|)$.*

Since $S_\lambda \cap S_{\lambda'} = \emptyset$ for $\lambda \neq \lambda' \in L$, we have that $\sum_\lambda |M_{G,S_\lambda,p}| = O(n)$ over all data structures, which is an improvement by a factor of $|L|$.

The point location data structure

We finally provide an overview of how to we determine the subset of $V(p)$ containing p_{i^*}. Our point location structure is quite similar to one found in [Gaw+18c] but with some modifications to improve space usage in our setting; these will not be described here, but they involve using standard persistence tricks of [Dri+89] and top-trees due to [Als+05]. We furthermore perform some modifications to the query procedure, due to the aforementioned changes. The main idea is to use point location in Voronoi diagrams, and the following definitions in this section will largely be made in a manner similar to those of of [Gaw+18c], but are included as they are essential for understanding the point location data structure of Lemma 3.4.

Voronoi Diagrams Given a planar graph $G = (V, E, \omega)$, $S \subseteq V$, the *Voronoi diagram of S in G*, denoted by $VD(S)$ in G is a partition of V into disjoint sets, $Vor(u)$, referred to as *Voronoi cells*, with one such set for each $u \in S$. The set $Vor(u)$ is defined to be

$$\left\{v \in V \mid d(u,v) < d(u',v) \text{ for all } u' \in S \setminus \{u\}\right\},$$

that is the set of vertices that are closer to u than any other site in terms of $d(\cdot,\cdot)$.

Duals of Voronoi Diagrams It will also be useful to work with a dual representation of Voronoi diagrams. Let VD_0^* be the subgraph of G^* s.t. $E(VD_0^*)$ is the subset of edges of G^* where $uv^* \in VD_0^*$ iff u and v belong to different Voronoi cells in VD. Let VD_1^* be the graph obtained by repeatedly contracting edges of VD_0^* incident to degree 2 vertices until no such vertex remains[2]. We refer to the vertices of VD_1^* as *Voronoi vertices*, and each face of the resulting graph VD_1^* can be thought of as corresponding to some Voronoi cell in the sense that its edges enclose exactly the vertices of some Voronoi cell in the embedding of the primal. We shall restrict ourself to the case in which all vertices of S lie on a single face h. In particular, h^* is a Voronoi vertex, since each site is a vertex on the boundary of h in the primal. Finally, let VD^* be the graph obtained by replacing h^* with multiple copies, one for each edge. We note that since there are $|S|$ Voronoi sites (and thus faces in VD^*), the number of Voronoi vertices in VD^* is $O(|S|)$ due to Euler's formula. Furthermore, [Gaw+18c] show that when assuming unique shortest paths and a triangulated input graph, VD^* is a ternary tree. It follows that the primal face corresponding to a Voronoi vertex f^* consists of exactly three vertices, each belonging to different Voronoi cells. We refer to the number of sites in a Voronoi diagram as its *complexity*.

[2]Formally, given a degree 2 vertex v with incident edges vw, vw', we replace these edges by ww', concatenate their arcs and embed ww' using this arc in the embedding.

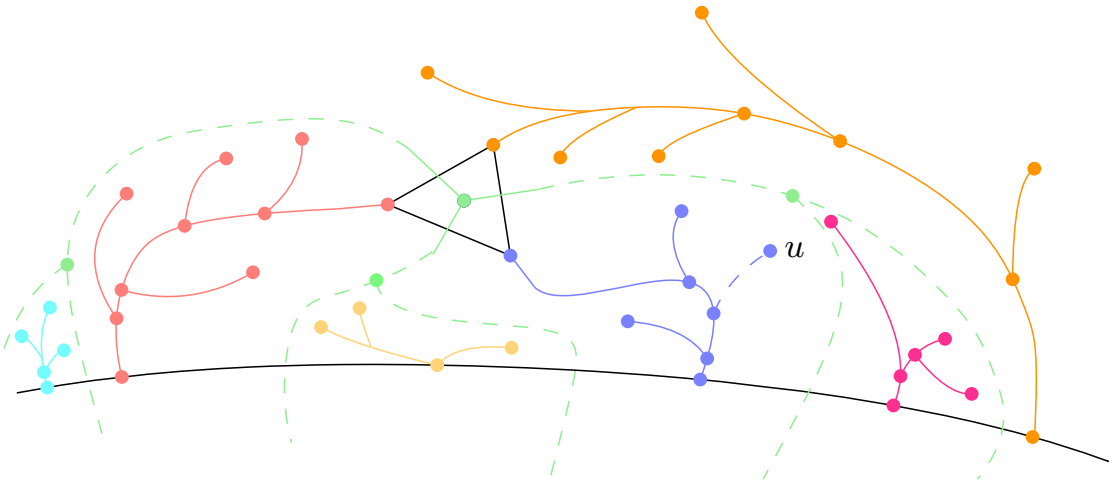


Figure 6: Abstract representation of a centroid and a point-location query u, belonging to the Voronoi cell containing the purple subtree.

Centroid decompositions Finally a *centroid decomposition*, T^*, can be computed from VD^* s.t. each node of T^* corresponds to a Voronoi vertex f^* and the children of f^* in T^* correspond to the subtrees resulting from splitting the tree at f^*, and s.t. the number of vertices of each child is at most a constant fraction of that of the parent. We remark that $VD^*(S)$ can be computed by connecting all sites to a super-source and running a single-source shortest paths algorithm, and its centroid decomposition in time proportional to $|V(VD^*(S))|$. Point location in Voronoi diagrams is then the task of determining $Vor(u)$, which can be done by recursively identifying to which subtree of the centroid decomposition the dual cell containing u belongs and recursing on the subtree until a leaf node corresponding to $Vor(u)$ is found. Such a step is illustrated in Figure 6. As the centroid decomposition is a balanced tree, this can be done in $O(\log S)$ steps. In the case of [Gaw+18c], they can afford to store many centroid decompositions partially due to using sparse separators, using as their set of sites the separator vertices (that is, after removing the strict interior or exterior of the separator from G, in which case the separator vertices now belong to the same face (hole) of the resulting graph), but as we already argued this will not suffice for our purposes. Instead we wish to use the fact that the alternation number is small (by Corollary 3.6) and show that it is sufficient to consider a small number, $O(S_\lambda)$, of sites.

Handling a query We note that the shortest path u to u_λ in G may enter p from either side, with one side corresponding to p' and the other p''. If we let p_i be the first intersection vertex, we thus either have: $d_G(u, u_\lambda) = d_{G_p}(u, p'_i) + d_{G_p}(p''_i, u_\lambda)$ or $d_G(u, u_\lambda) = d_{G_p}(u, p''_i) + d_{G_p}(p'_i, u_\lambda)$

We w.l.o.g. show how to handle the case in which the second first holds, as the other case is symmetric. We create an augmented version of G_p, by inserting at most $|S_\lambda|$ extra vertices to the face whose vertices are $V(p_1) \cup V(p_2)$, with one such for each maximal subsequence consisting of the same symbol of $M_{G_p,S_\lambda,p''}$. Let r be the vertex added for the maximal subsequence with indices $k \leq l$, then for each j s.t. $k \leq j \leq l$, we then

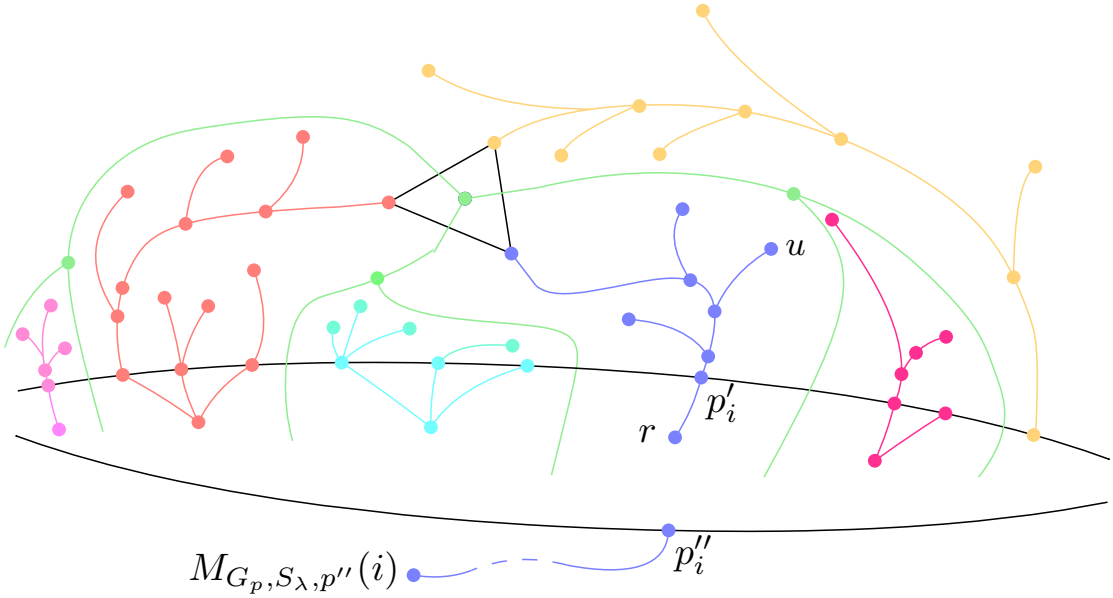


Figure 7: Abstract representation of a centroid decomposition for handling the case where $u \rightsquigarrow u_\lambda$ enters p at the side corresponding to p' and leaves at the side corresponding to p''. The Voronoi sites form a path of length $O(|S_\lambda|)$ on the same face. Here r a vertex corresponding to $M_{G_p,S_\lambda,p''}(i)$, which is a maximal subsequence of length 1, with $\omega(r, p'_i) = d_{G_p}(M_{G_p,S_\lambda,p''}(i), p''_i)$. Here, u belongs to the Voronoi cell of the site r and $d_{G_p}(u, p'_i) + \omega(r, p'_i) = d_{G_p}(u, p'_i) + d_{G_p}(p''_i, M_{G_p,S_\lambda}) = d_G(u, p_i) + d_G(p_i, M_{G,S_\lambda}) = d_G(u, M_{G,S_\lambda}) = d_G(u, u_\lambda)$.

add an edge from r to p'_j with the weight $d_{G_p}(p''_j, M_{G_p,S_\lambda,p''}(j))$. This creates a fan-like topology for each such r, which is illustrated in Figure 7. Denote by R the set of added vertices. They all lie on a common face f, see again Figure 7; we compute and store the centroid decomposition from $R \cup (V(p') \cap V(f))$ the size of which is $O(S_\lambda)$. Now, to identify i s.t. $d_{G_p}(u, p'_i) + d_{G_p}(p'_i, u_\lambda)$ is minimized, we can simply query for u in the centroid decomposition, which is also illustrated in Figure 7.

The preprocessing is done for each $\lambda \in L$ and for both types of crossings, and upon a query (u, λ), point location is done in the two centroid decompositions corresponding to λ. If $u \rightsquigarrow u_\lambda$ intersects p one of the witnesses will be p^*_i. We use $\sum_\lambda O(|S_\lambda|) = O(n)$ space to represent all such centroid decompositions. As aforementioned, the point location in centroid decompositions use $O(\log |S_\lambda|)$ steps. We can resolve each such step in $O(\log |V|)$ time, so the point location query spends $O(\log^2 |V|)$ overall.

4 Decremental Dynamic APSP

We now consider the problem of maintaining *decremental all-pairs shortest-paths* where the goal is to efficiently maintain shortest paths and distances between all pairs of vertices in a decremental directed, unweighted graph.

Problem 4.1. *Given an directed unweighthed graph $G = (V, E)$, efficiently maintain G subject to the following operations:*

- DIST(u, v): *report the shortest path distance* $d_G(u, v)$ *from* u *to* v *in the current version of* G,
- DELETE(u, v): *deletes an edge* (u, v) *from* E *if it is present.*

We furthermore consider Problem 4.1 also in a relaxed version where we only aim to maintain *approximate* distance estimates. We denote by $\tilde{d}_G(u, v)$ a distance estimate for the distance from u to v and we say that an APSP algorithm is t-approximate or has an *approximation ratio* (or *stretch*) of $t > 1$ if for any $u, v \in V$, we have that $d_G(u, v) \leq \tilde{d}_G(u, v) \leq t \cdot d_G(u, v)$. This thesis will be concerned with both the exact and the $(1 + \varepsilon)$-approximate version of the problem.

4.1 Related work

The naive approach towards maintaining APSP dynamically is to recompute the shortest path distances for all pairs of vertices in G after each update using the best static algorithm. We can then maintain distances explicitly, s.t. the query time is constant, and so, the time for a single update becomes $\tilde{O}(mn)$ for APSP and $\tilde{O}(m)$ for SSSP. At the other end of the spectrum one could achieve optimal update time by simply updating the input graph and only running the algorithm whenever a query is processed. Running a static algorithm each time, however, fails to reuse any information between updates and therefore often yields a high query time, motivating more efficient dynamic approaches that do this. In the following, we outline a few of these:

In 1981, Even and Shiloach [ES81] gave a deterministic data structure for maintaining a shortest path tree to a given depth d in undirected, unweighted decremental graphs in total time $O(md)$. Henzinger and King [HK95] and King [Kin99] later adapted this to directed graphs with integer weights. By using their structure for each vertex one can maintain decremental all-pairs shortest paths in total time $O(mn^2W)$, where the edge weights are integers in the range $[1, W]$. They subsequently improved this to total update time $\tilde{O}(mn^{2.5}\sqrt{W})$ [Kin99] for $W = \omega(n)$. Demetrescu and Italiano [DI06] made slight progress by showing that the restriction to integral edge weights could be dropped. Finally, the same authors presented a data structure with total update time $\tilde{O}(mn^2)$ which is the state-of-the-art for any data structure against an adaptive adversary up to today [DI04]. In fact, their algorithm extends to a fully-dynamic algorithm with $\tilde{O}(n^2)$ amortized update time. It is also capable of handling insertions and deletions of vertices with up to $n - 1$ incident edges. The data structure was later simplified and generalized by Thorup [Tho04].

Baswana, Hariharan, and Sen gave an *oblivious* Monte-Carlo construction for maintaining decremental APSP with total update time $\tilde{O}(n^3)$ for *unweighted* graphs [BHS02]. They further showed that their data structure could be adapted to maintain approximate APSP in *weighted* graphs with total update time of $\tilde{O}(\sqrt{m}n^2/\varepsilon)$ and stretch $(1 + \varepsilon)$.

Bernstein described a $(1 + \varepsilon)$-approximate algorithm with total running time $\tilde{O}(mn \log(W)/\varepsilon)$ by using a clever approach of shortcutting paths [Ber16]. Whilst his algorithm achieves near-optimal running time, again, this algorithm has the issue of assuming an oblivious adversary. Karczmarz and Łącki [KŁ20] showed how to maintain approximate APSP deterministically in decremental graphs with stretch $(1 + \varepsilon)$ in total time $\tilde{O}(n^3 \log(W)/\varepsilon)$. They also presented the first non-trivial algorithm for incremental

graphs, achieving total update time $\tilde{O}(mn^{4/3}\log(W)/\varepsilon)$ [KL19]. Very recently Bernstein, Probst Gutenberg and Saranurak [BGS22] described a deterministic data-structure which maintains $(1+\varepsilon)$-approximate decremental SSSP in near-linear time, implying a algorithm for maintaining APSP decrementally with $nm^{1+o(1)}$ total update time.

4.2 Contributions

We present three data structures. The first being a *deterministic* data structure for the exact variant of the problem with near-optimal $\tilde{O}(n^3)$ total update time. It matches the best *randomized* algorithm by Baswana et al. [BHS02] and improves the previous best bound of $\tilde{O}(mn^2)$ obtained by running an ES-tree from every source [ES81] or using the data structure of [DI04] (this, however, also works in weighted graphs) and improves over all but the sparsest graph domains. Our data structure is near-optimal in the sense that we also show an $\Omega(n^3)$ lower bound on the total update time of any decremental data structure that explicitly maintains the distance matrix:

Theorem 4.2. *Let G be an unweighted directed graph. There exists a deterministic data structure for maintaining decremental APSP in total time of $O(n^3\log^3 n)$. At any point, a shortest path distance query can be answered in constant time and the shortest path between any query pair can be reported in time proportional to the length of the path.*

Our second data structure deterministically maintains APSP with stretch $(1+\varepsilon)$. This constitutes the first deterministic data structure that solves the problem in subcubic time with small approximation error (except for graphs that are not extremely dense):

Theorem 4.3. *Let G be an unweighted directed graph. For any $\varepsilon > 0$, there exists a deterministic data structure for maintaining decremental approximate APSP in total time $O(\sqrt{m}n^2\log^2(n)/\varepsilon)$. At any point, a $(1+\varepsilon)$-approximate shortest path distance query can be answered in constant time and a $(1+\varepsilon)$-approximate shortest path between the query pair can be reported in time proportional to the length of the path.*

The third data structure achievies a better time bound but uses randomization. However, the data structure can assume an adaptive adversary:

Theorem 4.4. *Let G be an unweighted directed graph. For any $\varepsilon > 0$, there exists a data structure for maintaining decremental approximate APSP in expected total time $\tilde{O}(m^{2/3}n^{5/3}/\varepsilon + n^{8/3}/(m^{1/3}\varepsilon^2))$. This bound holds w.h.p. and and assuming an adaptive adversary. At any point, a $(1+\varepsilon)$-approximate shortest path distance query can be answered in constant time.*

Note however, that the third structure does not allow for path-reporting. We summarize our results as well as previous state-of-the-art results in Table 1.

4.3 Technical overview

Our overall approach for the deterministic data structures is similar to that of Baswana et al. [BHS02] but with a key difference that allows us to avoid using a randomized hitting set and instead rely on deterministic separators. The idea of the construction by Baswana et al. relies on a well-known result which says that if we sample a subset H_i^ρ

Time	Stretch	Adversary	Reference
$O(mn^2)$	exact	deterministic	[ES81; DI04]
$\tilde{O}(n^3)$	exact	deterministic	**Appendix C**
$\tilde{O}(n^3)$	exact	adaptive	[BHS02]

Table 1: Our results and previous state-of-the-art results for decremental APSP in the exact setting.

Time	Stretch	Adversary	Reference
$\tilde{O}(\sqrt{m}n^2/\varepsilon)$	$(1+\varepsilon)$	deterministic	**Appendix C**
$\tilde{O}(m^{2/3}n^{5/3}/\varepsilon + n^{8/3}/(m^{1/3}\varepsilon^2))$	$(1+\varepsilon)$	adaptive	**Appendix C**
$\tilde{O}(\sqrt{m}n^2/\varepsilon)$	$(1+\varepsilon)$	oblivious	[BHS02]
$\tilde{O}(nm)$	$(1+\varepsilon)$	oblivious	[Ber16]
$nm^{1+o(1)}$	$(1+\varepsilon)$	deterministic	[BGS22]

Table 2: Our results and previous state-of-the-art results for decremental APSP in the approximate setting.

of the vertices of size $\tilde{O}(n/\rho^i)$ (where ρ is some constant strictly larger than 1), each with uniform probability, then, w.h.p. we "hit" each shortest-path of length $[\rho^i, \rho^{i+1})$ between any pair of vertices in any version of the graph G.

Phrased differently, given vertices $u, v \in V$, we have that if a shortest path from u to v is of length $\ell \in [\rho^i, \rho^{i+1})$, then there is some vertex $w \in H_i^\rho$, such that the concatenation of a shortest path from u to w and a shortest path from w to v is of length ℓ. For each such w, we say w is a *witness* for the tuple (u, v) for distance ℓ.

Now for each $u, v \in V$, if the initial distance from u, v was $\ell \in [\rho^i, \rho^{i+1})$, we can check H_i^ρ to find a witness w. If the length of the path from u to w to v is increased, we can continue our scanning of H_i^ρ to see whether another witness exists. If there is no witness $w \in H_i^\rho$ left at some stage, we know that there is no path of length ℓ left in G w.h.p. and increase our guess by setting $\ell \mapsto \ell + 1$.

Sampling initially a hitting set H_i^ρ for every $i \in [0, \log_\rho n]$, we can find the "right" hitting set for each distance ℓ. Observe now that for each tuple $(u, v) \in V^2$, we have to scan a hitting set of size $\tilde{O}(n/\rho^i)$ for $\rho^{i+1} - \rho^i \sim \rho^{i+1}$ levels before the hitting set index i is increased which only occurs $O(\log n)$ times, thus we only spend time $\tilde{O}(n)$ for each vertex tuple (u, v). Thus, the total running time of the searches for witnesses can be bound by $\tilde{O}(n^3)$.

Overview of Theorem 4.2

Our construction is similar in the sense that we maintain witnesses for each distance scale $[\rho^i, \rho^{i+1})$ for every $i \in [0, \log_\rho n]$ such that each distance ℓ is in one such distance scale. The key difference is that instead of using a randomized *global* hitting set H_i^ρ

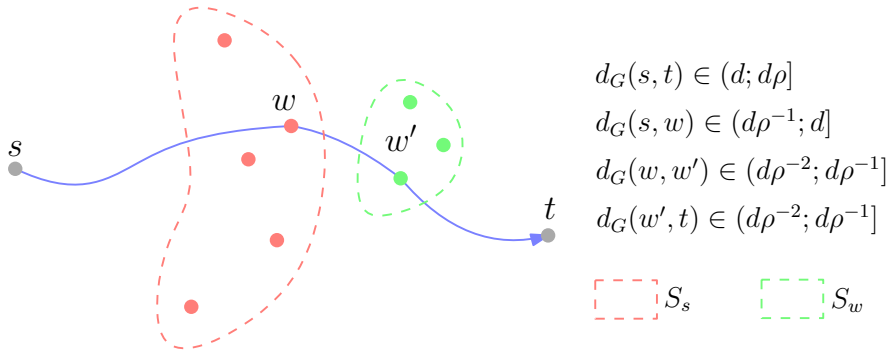


Figure 8: Illustration of separators and path "hierarchy". Here S_s is part of the local separator for s and S_w for w. The path $u \rightsquigarrow v$ travels through a witness vertex w, and $w \rightsquigarrow v$ travels through w'. If the length of the path $w' \rightsquigarrow v$ is increased by Δ, the distance estimates of all 2-hop-paths that use $w' \rightsquigarrow v$ as a sub-path are increased by that amount. In this case, the estimate for $w \rightsquigarrow w' \rightsquigarrow v$ is increased and is propagated to the next level where subsequently the estimate for $s \rightsquigarrow w \rightsquigarrow v$ is increased.

for a distance scale $[\rho^i, \rho^{i+1})$, our construction relies on deterministically maintaining a small local vertex separator $S_i(u)$ for every vertex $u \in V$ of size $\tilde{O}(n/\rho^i)$ separating all shortest paths starting in u with a distance in $[\rho^i, \rho^{i+1})$.

More precisely, for each distance scale $[\rho^i, \rho^{i+1})$ and vertex $u \in V$, we maintain a separator $S_i(u)$ that satisfies the invariant that every shortest path from u to a vertex v at distance at least ρ^i is intersected by a vertex in $S_i(u)$. If this invariant is violated after an adversarial update, then we find such a vertex v and need to add additional vertices to $S_i(u)$ during the time step. The challenge is to take these additional separator vertices such that the total size of $S_i(u)$ is not increased beyond $\tilde{O}(n/\rho^i)$. The separator procedure makes use of sparse layers of BFS trees and here is where we rely on the assumption that the graph is unweighted. We defer the details of the separator procedure to a later section and continue our discussion of the APSP data structure.

Since we need to detect whether vertices have distance less than ρ^i from u or not in G, we further have to use a bottom-up approach to compute distances after an edge deletion, i.e. we start with the smallest possible distance range and update all distances in this range and then update larger distances using the information already computed. This issue did not arise in Baswana et al. [BHS02] but can be handled by a careful approach. The distances computed for one distance scale include all distances to and from witnesses for the next larger distance scale.

It is now easy to see that the scanning for witnesses can be implemented in the same time as in the analysis sketched above by scanning the list of local separator vertices which serve as witnesses instead of the hitting set. Further, we can maintain local vertex separators using careful arguments in total time $\tilde{O}(mn)$ giving our result in Theorem 4.2.

Overview of Theorem 4.3

In order to improve the running time for sparse graphs, we can further focus on only considering distances that are roughly at a $(1+\epsilon)$-multiplicative factor from each other. More concretely, instead of increasing the expected distance from ℓ to $\ell+1$ when we

cannot find a witness for some path from u to v for distance ℓ, we can increase the next expected distance level ℓ' to $\sim (1+\epsilon)\ell$ and consider every vertex w a witness if there is a path $u \rightsquigarrow w \rightsquigarrow v$ of length at most ℓ'. Thus, we handle fewer distances and can thereby reduce the time to maintain distances that are at least d in total time $\tilde{O}(n^3/d + mn)$. Again, a careful approach is necessary to ensure that approximations do not add up over distance scales.

This is no faster than the data structure for exact distances when d is small so in order to get Theorem 4.3, we use the $O(mnd)$ data structure of Even and Shiloach [ES81] to maintain distances up to d. Picking d such that $mnd = n^3/d$ gives the result of Theorem 4.3 (the term $\tilde{O}(mn)$ vanishes since it is subsumed by the two other terms, also we assumed $\epsilon > 0$ to be a constant to simplify the presentation).

Maintaining Separators

We now sketch how to deterministically maintain the "small" local separator for a vertex $s \in V$ with some useful invariants. Let S be the local separator for s. The first invariant that will be useful is that any vertex $t \in V$ that is reachable from s in $G \setminus S$, is "close" to s or roughly within distance d. As edges are deleted from G, the distances from s to such vertices t may increase. If a vertex t moves too far away from s, the invariant is re-established by growing BFS trees in parallel, one layer at a time, from s in $G \setminus S$ and from t in the graph obtained from $G \setminus S$ by reversing the orientations of all edges. The search halts when a layer (corresponding to the leaves of the BFS tree at the current iteration) that is "thin" is found, and its vertices are added to S; vertices that are on the opposite side of the separator than s are cut off as they must all be too far away from s. Here, "thin" refers to a BFS layer such that the number of vertices added to the separator is only a factor $\tilde{O}(1/d)$ times the number of vertices cut off. It is well known that such a layer exists. Summing up, it follows that $|S| = \tilde{O}(n/d)$ at all times. By marking vertices as they are searched (according to the side of the BFS layer on which they are found), the vertices that are "cut off" from s by the augmented separator will never be searched again, and the cost of searching the edges of either side of the search can be charged to sum of the degree of these vertices, for a total update time of $O(m)$. This is illustrated in Figure 9. For our randomized data structure, we need an additional property that essentially allows us to take a snapshot of the current separator and use it in later updates rather than having to repeatedly update the separator. This will be key to getting an improved randomized time bound.

Overview of Theorem 4.4

The randomized approximate data structure of Theorem 4.4 follows the same overall approach but is technically more involved. Instead of keeping track of all 2-hop paths $u \rightsquigarrow s \rightsquigarrow v$[3] for every $s \in S_i(u)$, the randomized data structure samples a subset of these by picking each vertex of $S_i(u)$ independently with some probability p. It only keeps track of approximate shortest path distances going through this subset rather

[3]Note that such a path may have more than one intermediate vertex, but it is useful to think of it as a path of two weighted edges/hops (u, s) and (s, v) since this is what is maintained by the data structure.

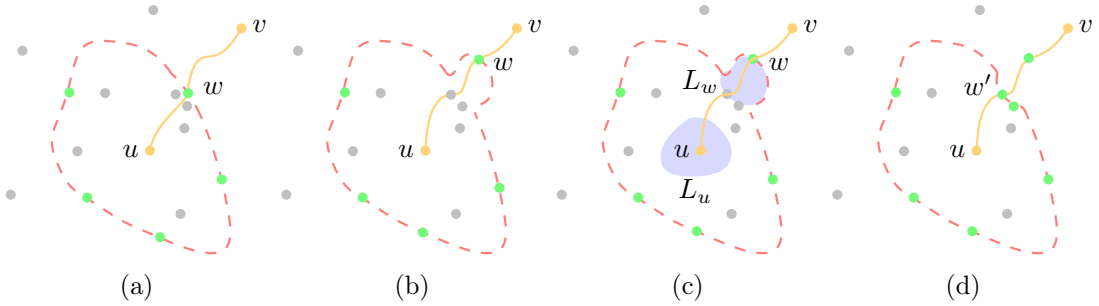


Figure 9: Maintenance of separators. The green vertices are vertices that have been searched, and the vertices on the dotted line are in the separator. (a) w is a witness for $u \rightsquigarrow v$. (b) w moves too far away from u. (c) The parallel search is initiated from u and w and is highlighted in magenta with L_u and L_w corresponding to the BFS layers. (d) w' is the new witness for $u \rightsquigarrow v$.

than the full set $S_i(u)$. This will speed up the above since the subset of the separator we need to scan is smaller by a factor p. However, this approach fails once no short 2-hop path intersects the sampled subset. At this point, w.h.p. there should only be short 2-hop paths through $O(\log n/p)$ vertices of $S_i(u)$ so also in this case, the subset can be kept small. However, scanning linearly through $S_i(u)$ to find this small subset will take $\tilde{O}(n/d)$ time and happen over all pairs (u, v).

Our solution is roughly the following. Suppose no sampled vertex certifies an approximate short path from u to v. Then v scans linearly through $S_i(u)$ to find the small size $O(\log n/p)$ subset $S'_i(u)$. Consider the set W of vertices w such that $d_G(w, v)$ is small compared to d, i.e., $d_G(w, v) \leq \epsilon d$ for some small constant $\epsilon > 0$. Then we show that the small subset $S'_i(u)$ found for v can also be used for each vertex $w \in W$. The intuition is that for any vertex $s \in S_i(u) \setminus S'_i(u)$, the approximate shortest path distance from u to w through s must be large since otherwise we get a short path $u \rightsquigarrow s \rightsquigarrow w \rightsquigarrow v$ from u to v through s, contradicting that $s \notin S'_i(u)$.

It follows that if $|W|$ is large, the $\tilde{O}(n/d)$ cost of scanning $S_i(u)$ can be distributed among a large number of vertices of W. Dealing with the case where $|W|$ is small is more technical so we omit it here.

The way we deal with an adaptive adversary is roughly as follows. Consider a deterministic data structure that behaves like the randomized data structure above, except that it maintains 2-hop paths $u \rightsquigarrow s \rightsquigarrow v$ for all $S_i(u)$ rather than only through a sampled subset. The slack from the approximation allows us to round up all "short" approximate distances to the same value. Hence, as long as the randomized data structure has short 2-hop paths, it maintains exactly the same approximate distances as the deterministic structure and hence the approximate distances output to the adversary is independent of the random bits used.

5 Degree of Convexity and Expected Distances in Polygons

We study real-valued functions defined on pairs of points in a simple polygon P, and want to describe algorithms for computing the expected value when picking a pair of

points uniformly and independently at random from some subset of P.

Problem 5.1. *Given a simple polygon P, $M \subseteq M$ and $f : P \times P \mapsto \mathbb{R}$, descibe an algorithm wich computes the expected value of f when applied to a pair of points of chosen uniformly and independently at random from M.*

Here, M can either be a given finite set of points in P or all points in P, i.e., $M = P$. For a pair of points $p, q \in P$, the functions we will consider are:

- the indicator $[pq \subset P] \in \{0, 1\}$, i.e., whether p and q can see each other in P, and
- the length of the geodesic shortest path from p to q in P in the L_1- or L_2-metric.

We also touch upon the problem of counting numbers of visible pairs and the number of edges in visibility graphs.

5.1 Related work and contributions

Beer index Letting $M = P$ and $f(p, q) = [pq \subset P]$ be the visibility indicator w.r.t. P, we obtain the probability, $B(P)$, that two points p, q, chosen uniformly and independently at random in P, see each other,

$$B(P) = \frac{1}{|P|^2} \int_{p \in P} \int_{q \in P} [pq \subset P] dq \, dp,$$

where $|\cdot|$ denotes the area. This is referred to as the *Beer index*, and is one of the of the well-known characterizations for the degree of convexity a polygon P. The problem of partitioning a polygon into components that are close to convex is useful in many practical settings, as such partitions provide similar benefits as convex partitions, while the number of components can be significantly smaller when the components are allowed to be slightly non-convex [Gho+13; LA06]. This motivates ways to quantify the degree of convexity and algorithms for computing these measures.

Buchin, Kostitsyna, Löffler and Silveira [Buc+19] described an algorithm that outputs $B(P)$, again for a given polygon P which may have holes. The claimed running time of the algorithm is $O(n^2)$, but as it is described in the paper [Buc+19], it may output $B(P)$ as a sum of $\Omega(n^4)$ closed-form expressions and thus likewise have a running time of $\Omega(n^4)$. An erratum has been sent to the authors who are aware, and we refer to the appendix of our unpublished manuscript where we give such a counter-example.

For a given simple polygon P with n corners, we describe an algorithm outputting $B(P)$ as a sum of $O(n^2)$ closed-form expressions using $O(n^2 \log n)$ time. The algorithm is a divide-and-conquer algorithm, and the main idea is to split P along a diagonal uv into two parts P_1 and P_2 with roughly equally many corners. We compute the contribution to $B(P)$ from pairs of points that see each other across the diagonal uv and then recursively add the contributions of pairs of points that are either both in P_1 or both in P_2. The algorithm is arguably simpler than the algorithm from [Buc+19] and presumably also the unpublished, faster algorithm by the same authors, both of which rely on geometric duality, whereas our algorithm deals exclusively with primitive objects in the usual primal space.

Theorem 5.2. *Given a simple polygon P with n corners, there is an algorithm that returns the Beer-index $B(P)$ as a sum of $O(n^2)$ closed-form expressions. The algorithm runs in time $O(n^2 \log n)$ and uses $O(n^2)$ space.*

Counting visible pairs of points Very recently, Buchin, Custers, van der Hoog, Löffler, Popov, Roeloffzen and Staals [Buc+22] studied the problem of computing the number of visible pairs among m points in a simple polygon P with n corners. If m is not much larger than n, they suggest to use a data structure by Hershberger and Suri [HS95] to test if each pair is visible in $O(\log n)$ time. This results in an algorithm with running time $O(n+m^2 \log n)$. For the case that m is large, they describe algorithms with running times $O(nm^{3/2} + m^{3/2} \log m)$ and $O(n + m^{3/2+\varepsilon} \log n \log m)$ for any $\varepsilon > 0$, respectively. We present an algorithm for counting the number of visible pairs among m points in $O(n \log n + m \log^2 n + m \log n \log m)$ time, so it is superior to the algorithms from [Buc+22] in all cases.

Theorem 5.3. *Given a simple polygon P with n corners and a set M of m point in P, the number of pairs of points in M that can see each other can be computed in $O(n \log n + m \log^2 n + m \log n \log m)$ time.*

Finding the edges of the visibility graph of a given polygon is one of the classical problems in computational geometry. Lee [Lee78] described a simple and well-known algorithm with running time $O(n^2 \log n)$ already in 1979, where n is the number of corners. The algorithm works by performing a rotational sweep around all corners. The same algorithm was described by Sharir and Schorr [SS86], and it also appears in the book [Ber+08]. Asano, Asano, Guibas, Hershberger and Imai [Asa+86] and Welzl [Wel85] gave algorithms with running time $O(n^2)$. Hershberger [Her89] gave an output-sensitive algorithm using $O(k)$ time to compute the visibility graph of a triangulated polygon, where k is the number of edges in the visibility graph. Together with Chazelle's algorithm for triangulating a polygon in $O(n)$ time, this yields an optimal algorithm with time $O(n+k)$. When the set M of points in P are the corners of P, our algorithm from Theorem 5.3 returns the *size* of the visibility graph of a simple polygon in $O(n \log^2 n)$ time. To the best of our knowledge, this is the first algorithm that counts the number of visibility edges faster than the size of the graph. Note that the number $k/\binom{n}{2} \in [0, 1]$ can be considered a discrete variant of the Beer-index.

Corollary 5.4. *Given a simple polygon P, the number k of edges in the visibility graph of P can be computed in $O(n \log^2 n)$ time.*

Expected distance The problem of determining the expected distance between two points picked independently and uniformly at random from a given domain has a long history. Czuber's book from 1884 [Czu84] contains calculations of the values for equilateral triangles, squares and rectangles. Bäsel [Bäs21] recently derived formulas for the expected distance, as well as higher moments, in regular n-gons for $n = 3, 4, 5, 6, 8, 10, 12$. The paper likewise contains more historical information about these problems. In another recent paper, Bonnet, Gusakova, Thäle and Zaporozhets [Bon+21] proved that the expected distance between two points in a convex body in the plane with perimeter 1 is between $7/60$ and $1/6$, and that these bounds are tight. They also provide bounds for higher dimensional convex bodies.

As a warmup for computing the expected distances in simple polygons, we consider the conceptually simpler problem of computing the sum of L_1-distances between all pairs of points from a finite set $M \subset \mathbb{R}^d$, i.e., with no polygon involved. We give a very simple algorithm computing the sum in $O(d \cdot n \log n)$ time.

Problem	Time	Space	Reference
Beer-index	$O(n^2 \log n)$	$O(n^2)$	**Appendix D**
L_1-distance	$O(n)$	$O(n)$	**Appendix D**
L_2-distance	$O(n^2 \log^2 n)$	$O(n^2)$	**Appendix D**

Table 3: Our results for computing expected distances and the Beer index in a simple polygon.

Theorem 5.5. *Given a set M of n points in $\mathbb{R}^d$, the sum of pairwise distances in the L_1-metric between points in M can be computed in $O(d \cdot n \log n)$ time.*

Hsu [Hsu90] studied the algorithmic problems of computing the expected distance between two random points in a given polygon. He gave a $O(n^2)$ time algorithm for the expected geodesic L_1-distance in a simple polygon and a $O(n^3)$ time algorithm for the expected L_2-distance in a *convex* polygon. In this paper, we describe a very simple algorithm for computing the expected geodesic L_1-distance in $O(n)$ time and the L_2-distance in $O(n^2 \log^2 n)$ time in a simple polygon. To the best of our knowledge, no algorithm has been described before for computing the expected L_2-distance in simple polygons in general. Table 3 summarizes our results on computing the Beer- and Wiener-index of a simple polygon.

Theorem 5.6. *Given a simple polygon P with n corners, there is an algorithm for computing the expected geodesic L_1-distance between two random points in P using $O(n)$ time and space.*

Theorem 5.7. *Let P be a simple polygon. There is an algorithm which outputs an expression representing the expected geodesic L_2 distance of P in $O(n^2 \log^2 n)$ time using $O(n^2)$ space. Each term of the expression is a simple integral of constant size. The number of terms in the expression is $O(n^2)$.*

This characterization can then be used to approximate the expected geodesic L_2-distance. This relies simply on approximating the certain kind of "simple" integrals that correspond to each term in the expression of Theorem 5.7. It is unknown to the authors whether the terms admit a closed solution, however. This is captured in the following corollary:

Corollary 5.8. *Let P be a simple polygon and assume that there is a procedure O that d-approximates simple integrals in time $O(t(d))$ pr. integral. Then there is an algorithm which d-approximates the expected geodesic L_2-distance in time $O(n^2 \log^2 n \cdot t(d))$.*

More attention has been given to the problem of computing the average distance between two vertices in a graph G. This known as the *Wiener-index* of G, and it is a fundamental measure with important applications in mathematical chemestry and appears in thousands of publications. Note that the Wiener-index is equivalent to the sum of pairwise distances. For more information on the problem, see the papers [Cab19; Gaw+18a].

5.2 Technical overview

In the following we give a brief overview of the ideas for computing the Beer index and Geodesic shortest path distances. Consider a polygon P with n corners, and some diagonal uv splitting P into "upper" and "lower" polygons P_1 and P_2 across some diagonal uv. If we let X and Y be stochastic variables that assumes points of P uniformly at random. We then have that

$$E(d(X,Y)) = \int_{x\in P}\int_{y\in P} f(x,y)dy\ dx \tag{1}$$

$$= E(d(X_{P_1},Y_{P_1})) + E(d(X_{P_2},Y_{P_2})) + \int_{x\in P_1}\int_{y\in P_2} d(x,y)dy\ dx. \tag{2}$$

for some appropriate choice of f. Choosing the diagonal s.t. the number of corners are split into roughly the same number in each subpolygon yields a recursion depth of $O(\log n)$. For $p, q \in P$, we denote in this section by $p \rightsquigarrow q$ the unique shortest path contained in P from p to q. We can thus consider the following three cases of paths:

Observation 5.9. *Given $p_1 \in P_1$ and $p_2 \in P_2$, there is an unique intersection point with $p_1 \rightsquigarrow p_2$ and the diagonal. The segment of this shortest path intersects the diagonal at some unique angle. Furthermore, either*

1) $p_1 \rightsquigarrow p_2$ is a single segment crossing the diagonal, or

2) $p_1 \rightsquigarrow p_2$ consists of two or more segments and either has the form

2a) $p_1 \rightsquigarrow rt \rightsquigarrow p_2$ where rt is the segment crossing the diagonal and $p_1 \neq r$ and $p_2 \neq t$, or

2b) $p_1 \rightsquigarrow rp_2$ where rp_2 is the segment crossing the diagonal and $p_1 \neq r$, or

2c) $p_1 r \rightsquigarrow p_2$ where $p_1 r$ is the segment crossing the diagonal and $p_2 \neq r$.

These cases are illustrated in Figure 10. We will use this to characterize the third term of Equation (2).

Beer index In the case of the Beer index, i.e. when f is the visibility indicator function, we note that the only the points that satisfy case 1) of Observation 5.9 contribute to Equation (2). This case can be thought of as considering the set of all trapezoids defined by corners of P an an angle $\varphi \in [0;\pi]$. This is shown in Figure 11a.

Each trapezoid is defined by two corners g and h of P. Therefore trapezoids may appear and disappear as φ is increased according to when g and h becomes visible or invisible from uv at the angle ϕ; thus each trapezoid exists in some $\varphi \in [\varphi_1;\varphi_2] \subseteq [0;\pi]$. This is illustrated in Figure 11b. Our approach is then to characterize the contribution for each trapezoid by an integral parametrized by φ and providing closed forms for all trapezoids and to report the terms of their sum. Since a trapezoid is defined by the pair g, h of corners, there can be at most $O(n^2)$ of trapezoids over all choices of angles. This gives us Theorem 5.2.

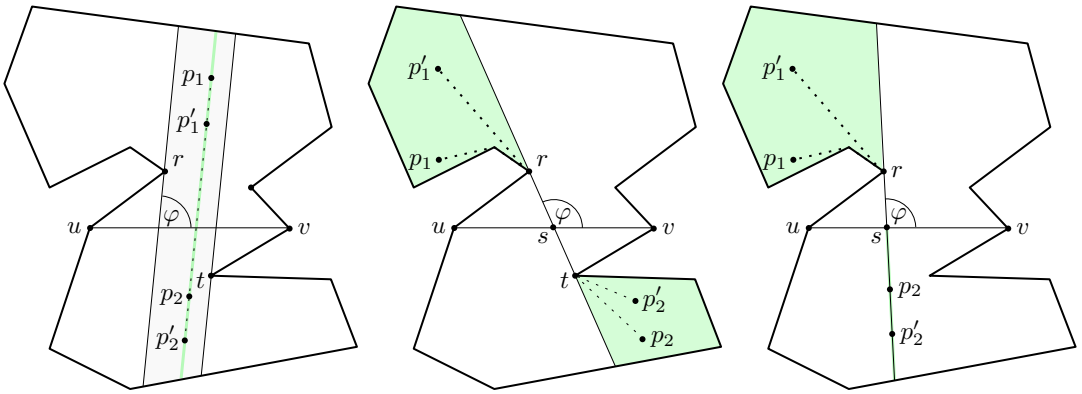


Figure 10: Visualization of cases, for pairs (p_1, p_2) and (p'_1, p'_2) where $(p_1, p_2) \sim (p'_1, p'_2)$ and $s \in uv$. (a) pairs of case 1) that see one another through s at an angle of φ, (b) pairs of case 2a) whose shortest path go through a diagonal rst, (c) pairs of case 2b) whose shortest path goes along rs but intersects no corner of P_2. Case 2c) is symmetric to case 2b) and omitted here.

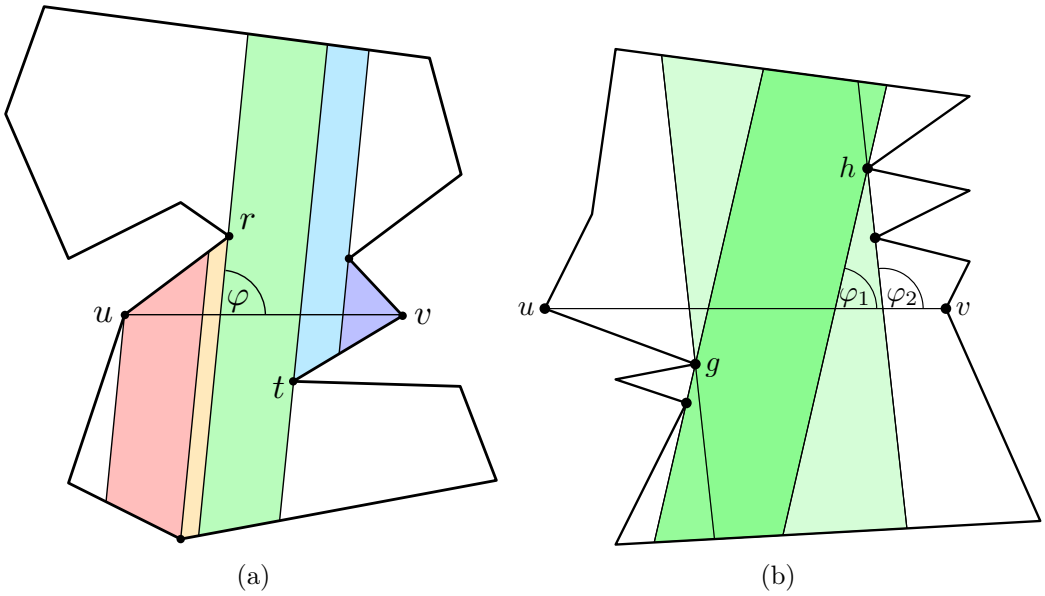


Figure 11: (a) All trapezoids crossing uv at an angle φ. (b) A trapezoid defined for corners g, h exists in some interval $[\varphi_1; \varphi_2]$ of angles.

Expected geodesic L_2 distance To compute the expected geodesic distance we can account for the contribution of case 1) by using the previous characterization but substituting the indicator function for the shortest geodesic L_2 distance, $d(p,q) = ||p \rightsquigarrow q||_2$. This is simply the sum of the L_2 norm of each segment of $p \rightsquigarrow q$. Accounting for the contributions for pairs corresponding to cases 2a), 2b) and 2c) then fully characterizes the expected distance. To this end, we use a shortest path decomposition which partitions P into simple regions, that then can be used to characterize the pairs of each case. Each of these regions can then be expressed as one of few "simple" integrals, whose sum (of $O(n^2)$ terms) completely characterizes the third term of Equation (2). At the time of writing it is unknown to the author whether these admit closed form expressions, for which reason the statement of Corollary 5.8 is added.

5.3 Future work

At the time of the writing, we believe that the characterization of Observation 5.9 implies that recursion is actually not necessary for our construction, and does not aid the description. We also believe we know how to extend a subset of the contributions to work for polygons with holes. An interesting direction is to determine whether the integrals of Theorem 5.7 admit a closed form, albeit we deem that it is unlikely that this is the case.

References

[Als+05] Stephen Alstrup, Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. „Maintaining information in fully dynamic trees with top trees“. In: *Acm Transactions on Algorithms (talg)* 1.2 (2005), pp. 243–264 (cit. on p. 12).

[Ari+96] Srinivasa Arikati, Danny Z. Chen, L. Paul Chew, Gautam Das, Michiel Smid, and Christos D. Zaroliagis. „Planar spanners and approximate shortest path queries among obstacles in the plane“. In: *Algorithms — ESA '96*. Ed. by Josep Diaz and Maria Serna. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 514–528. ISBN: 978-3-540-70667-0 (cit. on p. 3).

[Asa+86] Takao Asano, Tetsuo Asano, Leonidas J. Guibas, John Hershberger, and Hiroshi Imai. „Visibility of Disjoint Polygons“. In: *Algorithmica* 1.1 (1986), pp. 49–63 (cit. on p. 22).

[Bäs21] Uwe Bäsel. *The moments of the distance between two random points in a regular polygon*. 2021 (cit. on p. 22).

[Ber+08] Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008. ISBN: 9783540779735 (cit. on p. 22).

[Ber16] Aaron Bernstein. „Maintaining shortest paths under deletions in weighted directed graphs“. In: *SIAM Journal on Computing* 45.2 (2016), pp. 548–574 (cit. on pp. 15, 17).

[BGS22] Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. „Deterministic decremental sssp and approximate min-cost flow in almost-linear time“. In: *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE. 2022, pp. 1000–1008 (cit. on pp. 16, 17).

[BHS02] Surender Baswana, Ramesh Hariharan, and Sandeep Sen. „Improved decremental algorithms for maintaining transitive closure and all-pairs shortest paths“. In: *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. ACM. 2002, pp. 117–123 (cit. on pp. 15–18).

[Bon+21] Gilles Bonnet, Anna Gusakova, Christoph Thäle, and Dmitry Zaporozhets. „Sharp inequalities for the mean distance of random points in convex bodies“. In: *Advances in Mathematics* 386 (2021), pp. 1–27 (cit. on p. 22).

[Buc+19] Kevin Buchin, Irina Kostitsyna, Maarten Löffler, and Rodrigo I. Silveira. „Region-Based Approximation of Probability Distributions (for Visibility Between Imprecise Points Among Obstacles)“. In: *Algorithmica* 81.7 (2019), pp. 2682–2715 (cit. on p. 21).

[Buc+22] Kevin Buchin, Bram Custers, Ivor van der Hoog, Maarten Löffler, Aleksandr Popov, Marcel Roeloffzen, and Frank Staals. *Segment Visibility Counting Queries in Polygons*. 2022 (cit. on p. 22).

[Cab12] Sergio Cabello. „Many Distances in Planar Graphs“. In: *Algorithmica* 62.1-2 (Feb. 2012), pp. 361–381. ISSN: 0178-4617 (cit. on p. 3).

[Cab18] Sergio Cabello. „Subquadratic Algorithms for the Diameter and the Sum of Pairwise Distances in Planar Graphs“. In: *ACM Transactions on Algorithms* 15.2 (Dec. 2018), pp. 1–38. ISSN: 15496325 (cit. on p. 3).

[Cab19] Sergio Cabello. „Subquadratic Algorithms for the Diameter and the Sum of Pairwise Distances in Planar Graphs“. In: *ACM Trans. Algorithms* 15.2 (2019), 21:1–21:38 (cit. on p. 23).

[CDW17] Vincent Cohen-Addad, Soren Dahlgaard, and Christian Wulff-Nilsen. „Fast and Compact Exact Distance Oracle for Planar Graphs“. In: *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, Oct. 2017, pp. 962–973. ISBN: 978-1-5386-3464-6 (cit. on p. 3).

[Cha+19] Panagiotis Charalampopoulos, Paweł Gawrychowski, Shay Mozes, and Oren Weimann. „Almost Optimal Distance Oracles for Planar Graphs“. In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2019. Phoenix, AZ, USA: Association for Computing Machinery, 2019, pp. 138–151. ISBN: 9781450367059 (cit. on p. 3).

[Cha12] Timothy M. Chan. „All-pairs shortest paths for unweighted undirected graphs in $o(mn)$ time“. In: *ACM Trans. Algorithms* 8.4 (2012), 34:1–34:17 (cit. on p. 4).

[Che12] Shiri Chechik. „Improved Distance Oracles and Spanners for Vertex-Labeled Graphs“. In: *Algorithms – ESA 2012*. Ed. by Leah Epstein and Paolo Ferragina. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 325–336. ISBN: 978-3-642-33090-2 (cit. on p. 8).

[CX00] 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*. STOC 2000. Portland, Oregon, USA: Association for Computing Machinery, 2000, pp. 469–478. ISBN: 1581131844 (cit. on p. 3).

[Czu84] Emanuel Czuber. *Geometrische Wahrscheinlichkeiten und Mittelwerte*. 1884 (cit. on p. 22).

[DI04] Camil Demetrescu and Giuseppe F Italiano. „A new approach to dynamic all pairs shortest paths“. In: *Journal of the ACM (JACM)* 51.6 (2004), pp. 968–992 (cit. on pp. 15–17).

[DI06] Camil Demetrescu and Giuseppe F Italiano. „Fully dynamic all pairs shortest paths with real edge weights“. In: *Journal of Computer and System Sciences* 72.5 (2006), pp. 813–837 (cit. on p. 15).

[Dji96] Hristo Djidjev. „On-Line Algorithms for Shortest Path Problems on Planar Digraphs“. In: *Proceedings of the 22nd International Workshop on Graph-Theoretic Concepts in Computer Science*. WG 1996. Berlin, Heidelberg: Springer-Verlag, 1996, pp. 151–165. ISBN: 3540625593 (cit. on p. 3).

[Dri+89] James R. Driscoll, Neil Sarnak, Daniel D. Sleator, and Robert E. Tarjan. „Making data structures persistent“. In: *Journal of Computer and System Sciences* 38.1 (1989), pp. 86–124. ISSN: 0022-0000 (cit. on p. 12).

[Erd65] P Erdös. „On some extremal problems in graph theory“. In: *Israel Journal of Mathematics* 3.2 (June 1965), pp. 113–116. ISSN: 0021-2172 (cit. on p. 3).

[ES81] Shimon Even and Yossi Shiloach. „An on-line edge-deletion problem“. In: *Journal of the ACM (JACM)* 28.1 (1981), pp. 1–4 (cit. on pp. 15–17, 19).

[FR06] Jittat Fakcharoenphol and Satish Rao. „Planar graphs, negative weight edges, shortest paths, and near linear time“. In: *Journal of Computer and System Sciences* 72.5 (Aug. 2006), pp. 868–889. ISSN: 00220000 (cit. on p. 3).

[Gaw+18a] Pawel Gawrychowski, Haim Kaplan, Shay Mozes, Micha Sharir, and Oren Weimann. „Voronoi Diagrams on Planar Graphs, and Computing the Diameter in Deterministic $\tilde{O}(n^{5/3})$ Time“. In: *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2018)*. 2018, pp. 495–514 (cit. on p. 23).

[Gaw+18b] Paweł Gawrychowski, Gad M. Landau, Shay Mozes, and Oren Weimann. „The nearest colored node in a tree“. In: *Theoretical Computer Science* 710 (2018), pp. 66–73. ISSN: 0304-3975 (cit. on p. 8).

[Gaw+18c] Paweł 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 2018. New Orleans, Louisiana: Society for Industrial and Applied Mathematics, 2018, pp. 515–529. ISBN: 9781611975031 (cit. on pp. 3, 12, 13).

[Gho+13] Mukulika Ghosh, Nancy M. Amato, Yanyan Lu, and Jyh-Ming Lien. „Fast approximate convex decomposition using relative concavity“. In: *Comput. Aided Des.* 45.2 (2013), pp. 494–504 (cit. on p. 21).

[Hag98] Torben Hagerup. „Sorting and searching on the word RAM“. In: *Annual Symposium on Theoretical Aspects of Computer Science*. Springer. 1998, pp. 366–398 (cit. on p. 2).

[Her+11] Danny Hermelin, Avivit Levy, Oren Weimann, and Raphael Yuster. „Distance Oracles for Vertex-Labeled Graphs“. In: *Automata, Languages and Programming*. Ed. by Luca Aceto, Monika Henzinger, and Jiří Sgall. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 490–501. ISBN: 978-3-642-22012-8 (cit. on p. 8).

[Her89] John Hershberger. „An Optimal Visibility Graph Algorithm for Triangulated Simple Polygons“. In: *Algorithmica* 4.1 (1989), pp. 141–155 (cit. on p. 22).

[HK95] M.R. Henzinger and Valerie King. „Fully dynamic biconnectivity and transitive closure“. In: *Proceedings of IEEE 36th Annual Foundations of Computer Science*. IEEE Comput. Soc. Press, 1995, pp. 664–672. ISBN: 0-8186-7183-1 (cit. on p. 15).

[Hoo22] Ivor Djinn van der Hoog. „On Modelling, Structuring and Capturing Geometric Information“. PhD thesis. Utrecht University, 2022 (cit. on p. 2).

[HS95] John Hershberger and Subhash Suri. „A Pedestrian Approach to Ray Shooting: Shoot a Ray, Take a Walk“. In: *J. Algorithms* 18.3 (1995), pp. 403–431 (cit. on p. 22).

[Hsu90] Arthur C. Hsu. „The expected distance between two random points in a polygon“. MA thesis. Massachusetts Institute of Technology, 1990 (cit. on p. 23).

[Kin99] Valerie King. „Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs“. In: *Foundations of Computer Science, 1999. 40th Annual Symposium on*. IEEE. 1999, pp. 81–89 (cit. on p. 15).

[KL19] Adam Karczmarz and Jakub Lacki. „Reliable Hubs for Partially-Dynamic All-Pairs Shortest Paths in Directed Graphs“. In: *27th Annual European Symposium on Algorithms (ESA 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2019 (cit. on p. 16).

[KŁ20] Adam Karczmarz and Jakub Łącki. „Simple Label-Correcting Algorithms for Partially Dynamic Approximate Shortest Paths in Directed Graphs“. In: *Symposium on Simplicity in Algorithms*. SIAM. 2020, pp. 106–120 (cit. on p. 15).

[LA06] Jyh-Ming Lien and Nancy M. Amato. „Approximate convex decomposition of polygons“. In: *Comput. Geom.* 35.1-2 (2006), pp. 100–123 (cit. on p. 21).

[Lee78] Der-Tsai Lee. „Proximity and reachability in the plane“. PhD thesis. University of Illinois at Urbana-Champaign, 1978 (cit. on p. 22).

[LMN13] Mingfei Li, Chu Chung Christopher Ma, and Li Ning. „$(1+\epsilon)$-Distance Oracles for Vertex-Labeled Planar Graphs“. In: *Theory and Applications of Models of Computation*. Ed. by T-H. Hubert Chan, Lap Chi Lau, and Luca Trevisan. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 42–51. ISBN: 978-3-642-38236-9 (cit. on p. 8).

[LP19] Jason Li and Merav Parter. „Planar Diameter via Metric Compression“. In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2019. Phoenix, AZ, USA: Association for Computing Machinery, 2019, pp. 152–163. ISBN: 9781450367059 (cit. on pp. 5, 7, 8).

[LP21] Yaowei Long and Seth Pettie. „Planar Distance Oracles with Better Time-Space Tradeoffs“. In: *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. Philadelphia, PA: Society for Industrial and Applied Mathematics, Jan. 2021, pp. 2517–2537 (cit. on pp. i, iii, 3, 9).

[MS12] 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 2012. Kyoto, Japan: Society for Industrial and Applied Mathematics, 2012, pp. 209–222 (cit. on p. 3).

[MS18] Shay Mozes and Eyal E. Skop. „Efficient Vertex-Label Distance Oracles for Planar Graphs“. In: *Theory of Computing Systems* 62.2 (Feb. 2018), pp. 419–440. ISSN: 1432-4350 (cit. on p. 8).

[Nus11] Yahav Nussbaum. „Improved Distance Queries in Planar Graphs“. In: *Proceedings of the 12th International Conference on Algorithms and Data Structures*. WADS 2011. New York, NY: Springer-Verlag, 2011, pp. 642–653. ISBN: 9783642222993 (cit. on p. 3).

[PR14] Mihai Pătraşcu and Liam Roditty. „Distance Oracles beyond the Thorup–Zwick Bound“. In: *SIAM Journal on Computing* 43.1 (Jan. 2014), pp. 300–311. ISSN: 0097-5397 (cit. on p. 3).

[Pro18a] Maximilian Probst. „On the Complexity of the (Approximate) Nearest Colored Node Problem“. In: *26th Annual European Symposium on Algorithms (ESA 2018)*. Ed. by Yossi Azar, Hannah Bast, and Grzegorz Herman. Vol. 112. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 68:1–68:14. ISBN: 978-3-95977-081-1 (cit. on p. 8).

[Pro18b] Maximilian Probst. „On the complexity of the (approximate) nearest colored node problem“. In: *arXiv preprint arXiv:1807.03721* (2018) (cit. on p. 9).

[Sau72] N Sauer. „On the density of families of sets“. In: *Journal of Combinatorial Theory, Series A* 13.1 (July 1972), pp. 145–147. ISSN: 00973165 (cit. on p. 5).

[Som14] Christian Sommer. „Shortest-path queries in static networks“. In: *ACM Computing Surveys* 46.4 (Apr. 2014), pp. 1–31. ISSN: 0360-0300 (cit. on pp. 4, 7).

[SS86] Micha Sharir and Amir Schorr. „On Shortest Paths in Polyhedral Spaces“. In: *SIAM J. Comput.* 15.1 (1986), pp. 193–215 (cit. on p. 22).

[Tho04] Mikkel Thorup. „Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles“. In: *Scandinavian Workshop on Algorithm Theory*. Springer. 2004, pp. 384–396 (cit. on p. 15).

[Tsu18] Dekel Tsur. „Succinct data structures for nearest colored node in a tree“. In: *Information Processing Letters* 132 (2018), pp. 6–10. ISSN: 0020-0190 (cit. on p. 8).

[TZ05] Mikkel Thorup and Uri Zwick. „Approximate distance oracles“. In: *Journal of the ACM* 52.1 (2005), pp. 1–24. ISSN: 00045411 (cit. on p. 2).

[Wal22] Nathan Wallheimer. „Improved Compression of the Okamura-Seymour Metric“. PhD thesis. University of Haifa (Israel), 2022 (cit. on p. 8).

[Wel85] Emo Welzl. „Constructing the Visibility Graph for n-Line Segments in $O(n^2)$ Time“. In: *Inf. Process. Lett.* 20.4 (1985), pp. 167–171 (cit. on p. 22).

[Wul10] Christian Wulff-Nilsen. „Algorithms for Planar Graphs and Graphs in Metric Spaces.“ PhD thesis. 2010 (cit. on p. 4).

[Wul13] Christian Wulff-Nilsen. „Constant time distance queries in planar unweighted graphs with subquadratic preprocessing time“. In: *Computational Geometry* 46.7 (Oct. 2013), pp. 831–838. ISSN: 09257721 (cit. on pp. 4, 5).

Appendix A

Appendix B starts from the next page, and includes the ArXiV version of the ISAAC 2021 accept "Truly Subquadratic Exact Distance Oracles with Constant Query", see https://doi.org/10.48550/arXiv.2009.14716.

Truly Subquadratic Exact Distance Oracles with Constant Query Time for Planar Graphs

Viktor Fredslund-Hansen * Shay Mozes † Christian Wulff-Nilsen ‡

Abstract

Given an undirected, unweighted planar graph G with n vertices, we present a truly sub-quadratic size distance oracle for reporting exact shortest-path distances between any pair of vertices of G in constant time. For any $\varepsilon > 0$, our distance oracle takes up $O(n^{5/3+\varepsilon})$ space and is capable of answering shortest-path distance queries exactly for any pair of vertices of G in worst-case time $O(\log(1/\varepsilon))$. Previously no truly sub-quadratic size distance oracles with constant query time for answering exact all-pairs shortest paths distance queries existed.

*Department of Computer Science, University of Copenhagen, viha@di.ku.dk

†Efi Arazi school of Computer Science, IDC Herzliya, Israel. smozes@idc.ac.il https://cs.idc.ac.il/~smozes/. Partially supported by Israel Science Foundation grant no. 592/17.

‡Department of Computer Science, University of Copenhagen, koolooz@di.ku.dk, http://www.diku.dk/~koolooz/. This research is supported by the Starting Grant 7027-00050B from the Independent Research Fund Denmark under the Sapere Aude research career programme.

1 Introduction

Efficiently answering shortest path distance queries between pairs of vertices in a graph is a fundamental algorithmic problem with numerous important applications. Given an n-vertex graph $G = (V, E)$ a *distance oracle* is a compact data-structure capable of efficiently answering shortest path distance queries between pairs of vertices $u, v \in V$. Ideally one would like the data structure to be of linear size and the query time to be constant. However, it is well known that there are graphs for which no distance oracle with $o(n^2)$ bits of space and $O(1)$ query time exists. In fact, even resorting to approximation, Pătraşcu and Roditty [20] showed that there are sparse graphs on $O(n \operatorname{polylog} n)$ edges for which constant query-time distance oracles with stretch less than 2 must be of size $\Omega(n^2 \operatorname{polylog} n)$, assuming the set intersection conjecture. These impossibility results make it natural to consider the probelms in restricted classes of graphs.

In this paper we consider exact distance oracles for planar graphs. Distance oracles for planar graphs are well motivated by important real-world applications, notably in routing, navigation of road and sea maps as well as in the context of computational geometry. To the best of our knowledge there are no non-trivial lower bounds for (static) distance oracles for planar graphs, and thus achieving the "holy grail" of a linear-size distance oracle with constant query time may be possible. Indeed, there has been numerous works over at least three decades developing exact distance oracles for planar graph [10, 2, 3, 8, 11, 18, 19]. However, only recently Cohen-Addad et al. [9] gave the first oracle with truly subquadratic space and polylogarithmic query time. Their result was inspired by Cabello's [4] breakthrough result, who gave the first truly sub-quadratic time algorithm for computing the diameter of planar graphs by a novel use of Voronoi diagrams. The approach of [9] was subsequently improved by [12, 7], who gave an elegant point-location mechanism for Voronoi diagrams in planar graphs, and combined it with a clever recursive scheme to obtain exact distance oracles for directed weighted planar graphs with $O(n^{1+\varepsilon} \operatorname{polylog} n)$ space and $O(\log^{1/\varepsilon} n)$ query time for any small constant ε. We note that even though the oracle of [7] gets quite close to optimal, it remains wide open to support exact queries in *constant time* using truly subquadratic space, even in the most basic case of unweighted undirected planar graphs [25, 5, 24].

Allowing approximate answers does help in planar graphs. Many results reporting $(1+\varepsilon)$-approximate distances with various tradeoffs exist, all with (nearly) linear size and polylogarithmic, or even $O(1/\varepsilon)$ query-time [23, 15, 14, 26]. Gu and Xu[13] presented a size $O(n \operatorname{polylog} n)$ distance oracle capable of reporting $(1+\varepsilon)$-approximate distances in time $O(1)$. While their query time is a constant independent of ε, the preprocessing time and space are nearly linear, but with an exponential dependency on $(1/\varepsilon)$. This exponential dependency was recently improved to polynomial [6].

Thus, despite the large body of work on distance oracles for planar graphs, it has remained an open question to determine whether an exact distance oracle with of size $O(n^{2-\varepsilon})$ with *constant* query-time can be constructed for some constant $\varepsilon > 0$.

Our results and techniques. We answer this question in the affirmative. Our result is presented in the following theorem:

Theorem 1. *Let $G = (V, E)$ be an undirected unweighted n-vertex planar graph. For any $\varepsilon > 0$ there exists a data-structure requiring $O(n^{5/3+\varepsilon})$ space that, for any $s, t \in V$, reports the shortest path distance between s and t in G in time $O(\log(1/\varepsilon))$.*

We remark that the simple distance oracle we present in Section 4 can be distributed into a distance labeling scheme with size $O(n^{3/4})$ per label, such that the distance between any two vertices s, t can be computed in $O(1)$ time given just the labels of s and t.

The main concept we use to obtain our result is that of a *pattern* capturing distances between a vertex and a cycle. This concept was used by [25] , and weas called "distance tuple" by and [17]. Consider a vector storing the distances from a vertex u to the vertices of a cycle β in their cyclic order. The pattern of u w.r.t. β is simply the discrete derivative of this vector. That is, the vector obtained by taking the difference between every pair of consecutive values. Li and Parter [17] observed that when the input graph is planar, the number of different patterns w.r.t. a face with r vertices is $O(r^3)$ regardless of the size of the graph. We next outline how this observation can be used to break the quadratic space barrier.

Roughly speaking, any planar graph can be decomposed into $O(n/r)$ subgraphs, called regions, of size r each, where the boundary of each region (i.e., the vertices that have neighbors outside the region) is a single cycle h.[1] Applying Li and Parter's observation in this setting, the number of different patterns for the hole of each region R is $O(r^3)$. Hence, we can represent the distances from any vertex $s \notin R$ to h by just storing the distance from s to an arbitrarily chosen canonical vertex v_h of h, and a pointer to the pattern of s with respect to h. This requires just $O(n)$ space plus $O(r^3 \cdot r)$ for storing all the patterns for h. Summing over all $O(n/r)$ regions, the space required is $O(n^2/r + nr^3)$. We define the notion of distance from a pattern to a vertex (see Definition 2). While this definition is simple, it is somewhat unnatural because the distance from a pattern to a vertex does not necessarily correspond to the length of any specific path in the graph! However, the distance between s and any vertex $t \in R$ is just the sum of the distance between s and the canonical vertex v_h and the distance from the pattern of s with respect to h to t.

We therefore store the distances from each of the $O(r^3)$ possible patterns of R to each vertex of R. This requires $O(r^3 \cdot r)$ space per region, so $O(nr^3)$ space overall. This way we can report the distance between s and t in constant time by (i) retrieving the pattern p of s w.r.t. h, and (ii) adding the distance from s to the canonical vertex v_h of h and the distance from the pattern p to t. These ideas alone already imply an oracle with space $\tilde{O}(n^{7/4})$ and constant query time. Combining these ideas with recursion yields the improved space bound of Theorem 1.

As we argued in the introduction, breaking the quadratic space barrier for constant query time is important and significant result in its own right. We highlight the following difference between the approach taken in our recursive oracle and the approaches used in all existing distance oracles we are aware of. To the best of our knowledge, all existing distance oracles, both exact and approximate, and both for general graphs and planar graphs, recover the distance from s to t by identifying a vertex or vertices on some (possibly approximate) shortest path between s and t, for which distances have been stored in the preprocessing stage. These vertices are usually referred to as landmarks, portals, hubs, beacons, seeds, or transit nodes [22]. Our oracle, on the other hand, reports the exact shortest path without identifying vertices on the shortest path from s to t. Instead, it "zooms in" on t by recovering distances to the canonical vertices of a sequence of subgraphs of decreasing size that terminates at a constant size graph containing t. We emphasize that *none of these canonical vertices necessarily lies on a shortest path* from s to t. This property may be viewed as a disadvantage if we also want to report the shortest path, but when reporting

[1] In fact, a constant humber of cycles. To readers familiar with the concept, this is just an r-division with a few holes, but *without* the important feature that each region has just $O(\sqrt{r})$ boundary vertices. This is because one cannot triangulate unweighted graphs without changing the distances.

multiple edges on long paths, constant query time is no longer relevant. On the other hand, it may be that just reporting the distance is easier than also reporting an internal vertex on a shortest path. Hence, it may be that developing oracles based on this new approach may lead to further advances on the way to linear size distance oracles for planar graphs with constant query time, and in other related problems.

2 Preliminaries

Let G be a graph. We denote by $V(G)$ and $E(G)$ the vertex and edge-set of G, and denote by $n = |V(G)|$ the number of vertices of G. For a subset S of edges or vertices we denote by $G[S]$ the subgraph of G induced on S. We denote by $u \rightsquigarrow_H v$ a *shortest path* from u to v in the subgraph H, by $\mathbf{d}_H(u, v)$ the length of $u \rightsquigarrow_H v$, and define $u \rightsquigarrow v \equiv u \rightsquigarrow_G v$.

The following definitions will be useful when talking about decompositions of G. A *region* R of G is an edge-induced subgraph of G, and its *boundary* ∂R is the vertices of R that are adjacent to some vertex of $V(G) \setminus V(R)$ in G. Vertices of $V(R) \setminus \partial R$ are called *interior* vertices of R. Observe that for a region R and for $u \in R$ and $v \in V \setminus V(R)$, any path from u to v in G must intersect ∂R.

It will be useful to assume some global strict order on a vertex set V s.t. for any $U \subseteq V$ there is a minimum vertex $\min U \in U$ w.r.t this order. We refer to this as the *canonical vertex* of U.

Faces and holes: We assume the reader is familiar with the the basic definitions of planarity and of planar embeddings. The edges of a plane graph induce maximal open portion of the plane that do not intersect any edges. A *face* of the graph is the closure of one such portion of the plane. We refer to the edges bounding a face as the *boundary* of that face. Given a face f, $V(f)$ is the set of vertices on the boundary of f. We denote by $w(f)$ the *facial walk* of f which is the sequence of vertices encountered when walking along f starting at $\min V(f)$ and going in the clockwise direction. Note that f may be non-simple, so some vertices may appear multiple times in $w(f)$.

A *hole* h in a region R of a graph G is a face of R which is not a face in G. We say that a vertex $u \in V(G) \setminus V(R)$ is *inside* hole h if u lies in the region of the plane corresponding to the face h of R. We denote by $V^\circ(h) = \{u \in V(G) \mid u \text{ is inside } h\}$ all the vertices that are inside h.

Decompositions of unweighted planar graphs. An r-division is a widely used decomposition of planar graphs into regions with small boundary. We use the r-divisions with a few holes as studied in [16], which works for triangulated biconnected graphs:

Lemma 1. *(r-division with few holes for triangulated graphs [16]) Let G be a biconnected, triangulated n-vertex planar embedded graph, and let $0 < r \leq n$. G can be decomposed into $\Theta(n/r)$ connected regions, each of which with $O(r)$ vertices and $O(\sqrt{r})$ boundary vertices. Each region has a constant number of holes. Every boundary vertex lies on some hole, and each hole has $O(\sqrt{r})$ vertices.*

The fact that the boundaries of regions are small (only $O(\sqrt{r})$ boundary vertices for a region with r vertices) is the basis for many efficient algorithms and data structures for planar graphs. Unweighted planar graphs posses additional structure (in comparison to weighted planar graphs), which may also be useful algorithmically. See for example the unit-Monge property in [1], or the limited number of patterns [25, 17], which we use in this work. However, exploiting such additional structure in conjunction with a decomposition into regions with small boundaries has been elusive

because of the seemingly technical requirement in Lemma 1 that the graph be triangulated and biconnected.

Any graph can be triangulated and biconnected by adding to each face f an artificial vertex and infinitely weighted artificial edges from the artificial vertex to each vertex of $V(f)$. This transformation preserves planarity and shortest paths, and ensures that the graph consists only of simple faces of size 3. However, the graph is no longer unweighted. We refer to an *artificial* vertex (edge) of G as a vertex (edge) which was added in the triangulation step, and a *natural* vertex (edge) of G as a vertex (edge) which is not artificial. In order to exploit the structure of the unweighted input graph we will remove the artificial edges and vertices after computing the decomposition using Lemma 1. On the one hand the graph is again unweighted. On the other hand, while the number of boundary vertices in each region remains $O(\sqrt{r})$, the holes may now contain new non-boundary vertices, and the total size of the holes in each region may be $\Theta(r)$. We note, however, that the deletion of artificial edges and vertices does not disconnect regions [16]. We therefore restate the decomposition lemma for unweighted graphs that are not necessarily triangulated or biconnected.

Lemma 2. *(r-division with few holes for non-triangulated graphs) Let G be a n-vertex planar embedded graph G, and let $0 < r \leq n$. G can be decomposed into $\Theta(n/r)$ connected regions, each of which with $O(r)$ vertices and $O(\sqrt{r})$ boundary vertices. Each region has a constant number of holes, and each boundary vertex lies on some hole.*

Recursive r-divisions. Our second construction relies on a *recursive r-division* which is a recursive decomposition of G into r-divisions for varying values of r. Specifically, for a decreasing sequence $\mathbf{r} = r_1, r_2, \ldots$, where $n \geq r_1 > r_2 > \ldots \geq 1$, we want r_i-divisions for all $i = 1, 2, \ldots$, such that each region in the r_i division is the union of regions in the r_{i+1}-division on the next level. We associate with the recursive r-division a decomposition tree, $\mathcal{T}_{\mathbf{r}}$, which is a rooted tree whose nodes correspond to the regions of the recursive decomposition of G. We will refer to nodes and their corresponding regions interchangeably. The root node corresponds to all of G. A node x of $\mathcal{T}_{\mathbf{r}}$ at depth i corresponds to a region of the r_i-division, and its children are the regions of the r_{i+1}-division whose union is the region corresponding to x. We denote by $\mathcal{T}^i_{\mathbf{r}}$ all the nodes at level i. It was shown in [16] that recursive r-divisions can be computed efficiently:

Lemma 3. *(Recursive r-division) Given a biconnected, triangulated n-vertex planar graph G and an exponentially decreasing sequence $\mathbf{r} = n \geq r_1, r_2, \ldots \geq 1$, a decomposition tree, $\mathcal{T}_{\mathbf{r}}$ can be computed in linear time s.t $\mathcal{T}^i_{\mathbf{r}}$ corresponds to an r_i-division of G with few holes for each i.*

3 Patterns

Both [25] and [17] introduce a notion of a "distance tuple" which can be thought of as a vector of shortest-path distances from a vertex to consecutive vertices of some hole. We introduce the following similar notion of a *pattern* (See Figure 1 for an illustration):

Definition 1. *(Pattern) Let G be a graph. Let H be a subgraph of G. Let u be a vertex in H, and let $\beta = b_0, b_1, \ldots, b_k$ be a path in H. The pattern of u (w.r.t. β in H) is a vector $p_{\beta,H}(u) \in \{-1, 0, 1\}^k$ satisfying $p_{\beta,H}(u)[i] = \mathbf{d}_H(u, b_i) - \mathbf{d}_H(u, b_{i-1})$ for $1 \leq i \leq k$. For a region R in G, a hole h of R, and a vertex $u \in V^\circ(h)$, we write $p_{h,G}(u)$ instead of $p_{w(h),G}(u)$.*

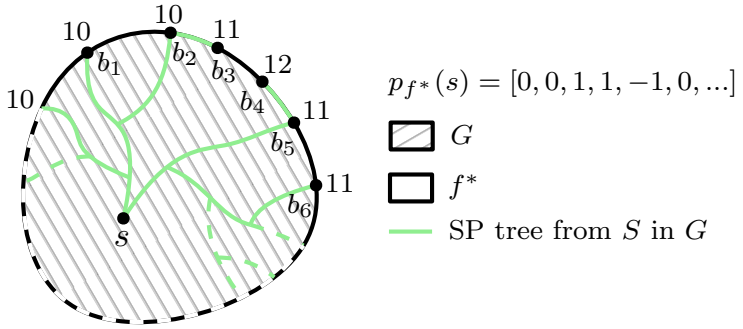


Figure 1: Illustration of the pattern of the vertex s w.r.t. f^* in an undirected graph. In this case f^* is the external face of the embedding. The numeric labels indicate the shortest path distances from s to each b_i where $b_i \in V(f^*)$ for $1 \leq i \leq 6$.

Definition 2. *(pattern to vertex distance) Let R be a region in a graph G. Let h be a hole of R. Let $b_0, b_1, \ldots, b_k$ be the vertices of $w(h)$ in their cyclic order. Let p be some pattern w.r.t. h (i.e., $p = p_h(u)$ for some $u \in V^\circ(h)$). For a vertex $v \in R$ we define $\mathbf{d}_G(p, v)$ the distance between p and v to be $\min_{i=0}^{k} \left\{ \mathbf{d}_G(b_i, v) + \sum_{j=0}^{i} p[j] \right\}$.*

Lemma 4. *Let R be a region of a graph G. Let h be a hole of R. For every $u \in V^\circ(h)$ and every $v \in R$, $\mathbf{d}_G(u, v) = \mathbf{d}_G(u, b_0) + \mathbf{d}_G(p_h(u), v))$.*

Proof. By definition of pattern and by a telescoping sum, for every $0 \leq i \leq k$, $\mathbf{d}_G(u, b_i) = \mathbf{d}_G(u, b_0) + \sum_{j=0}^{i} p[j]$. Let b_ℓ be any vertex of $w(h)$ on a shortest u-to-v path (b_ℓ exists since $u \in V^\circ(h)$ and $v \in R$). By choice of b_ℓ, $\mathbf{d}_G(u, v) = \mathbf{d}_G(u, b_\ell) + \mathbf{d}_G(b_\ell, v) = \min_{0 \leq i \leq k} \{\mathbf{d}_G(u, b_i) + \mathbf{d}_G(b_i, v)\} = \min_{0 \leq i \leq k} \left\{ \mathbf{d}_G(u, b_0) + \sum_{j=0}^{i} p[j] + \mathbf{d}_G(b_i, v) \right\} = \mathbf{d}_G(u, b_0) + \mathbf{d}_G(p, v)$. □

Bounding the number of patterns As mentioned, In a recent paper, Li and Parter [17] achieve improved bounds for diameter computation for planar graphs by showing that in unweighted undirected planar graphs the number of patterns is quite small. More specifically, they show that the VC-dimension of a set corresponding to all patterns is at most 3. By the Sauer-Shelah lemma [21], this implies that the number of distinct patterns w.r.t. a face f is in $O(|S|^3)$. Their result is stated in the following lemma:

Lemma 5. *(Pattern compression) [17] Let $G' = (V, E)$ be an n-vertex unweighted undirected planar graph, let f be a face in G', and let S be a set of consecutive vertices on f. Then the number of distinct patterns w.r.t. S, $| \cup_{u \in V} \{p_{S,G'}(u)\} |$, is bounded by $O(|S|^3)$.*

We observe that the bound of Lemma 5 also holds for patterns w.r.t. the entire set of vertices on a hole h of a region R even when distances are defined in the entire graph G.

Corollary 1. *Let R be a region in an n-vertex unweighted undirected planar graph G, and let h be a hole of R. Then the number of distinct patterns w.r.t. h, $| \cup_{u \in V^\circ[h]} \{p_{h,G}(u)\} |$, is bounded by $O(|h|^3)$.*

Proof. Since h is a hole of R, h is a face of $G-(R-h)$. By Lemma 5, $|\cup_{v\in V^\circ[h]}\{p_{h,G-(R-h)}(v)\}| = O(|h|^3)$. The corollary follows since for every two vertices $u, u' \in V^\circ(h)$, $p_{h,G-(R-h)}(u) = p_{h,G-(R-h)}(u')$ implies $p_{h,G}(u) = p_{h,G}(u')$. □

For the remainder of the paper we only deal with distances in G and with patterns in G, so we will omit the subscript G, and write $\mathbf{d}(\cdot,\cdot)$ and $p_h(\cdot)$ instead of $\mathbf{d}(\cdot,\cdot)$ and $p_{h,G}(\cdot)$.

4 $O(n^{7/4})$ space distance oracle

Before presenting our main result, we describe a simpler construction which yields a distance oracle with a larger space requirement of $O(n^{7/4})$ and $O(1)$ query time:

Preprocessing. The preprocessing consists of computing an r-division $\mathcal{R}$ of G with a parameter r to be determined later. For every vertex v of G and every region R of $\mathcal{R}$, we store the hole h of R s.t. v is in $V^\circ(h)$. This requires $O(n \cdot n/r) = O(n^2/r)$ space.

For every region $R \in \mathcal{R}$, for every hole h of R, we maintain the $O(r^3)$ patterns of the vertices in $V^\circ(h)$ w.r.t. h as follows. Let k denote the size of the boundary walk $w(h)$ of h. let v_h be the canonical (i.e., first) vertex of $w(h)$. We maintain the patterns seen so far in a ternary tree $\mathcal{A}$ whose edges are labeled by $\{-1, 0, 1\}$. The depth of $\mathcal{A}$ is $k-1$, and the labels along each root-to-leaf path correspond to a unique pattern, which we associate with that leaf. For every vertex $v \in V^\circ(h)$, we compute the pattern $p_h(v)$ and we make sure that $p_h(v)$ is represented in the tree $\mathcal{A}$ by adding the corresponding labeled edges that are not yet present in $\mathcal{A}$. After all the vertices in $V^\circ(h)$ were handled, the tree $\mathcal{A}$ has $O(r^3)$ leaves. For each leaf of $\mathcal{A}$ with an associated pattern p, we compute and store (i) the distance from p to each vertex of R. This requires $O(r^4)$ time and space for all leaves of $\mathcal{A}$, so a total of $O(n/r \cdot r^4) = O(nr^3)$ space for storing all this information over all regions.

For each vertex $v \in V^\circ(h)$ we store (ii) a pointer to (the leaf of $\mathcal{A}$ that is associated with) the pattern $p_{h,G}(v)$, as well as (iii) the distance $\mathbf{d}(v, v_h)$ between v and the canonical vertex of h. The total space required to store all these pointers and distances is $O(n \cdot n/r) = O(n^2/r)$.

To complete the preprocessing we also store (iv) for each region $R \in \mathcal{R}$, the distance $\mathbf{d}(u, v)$ for all pairs of vertices $u, v \in R$. This takes $O(n/r \cdot r^2)$ additional space, which is dominated by the above terms.

The total space required by the oracle is thus $O(n^2/r)+O(nr^3)$. This is minimized for $r = n^{1/4}$, resulting in an $O(n^{7/4})$-space data structure.

We note that once this information has been computed we no longer need to store the entire tree $\mathcal{A}$. Rather, it suffices to only store just the list of leaves of $\mathcal{A}$ and the distances stored with each of them. In particular, we no longer need to remember what is the actual pattern associated with each leaf, we only need to know the distances from this pattern to the vertices of the region R. In the current scheme this has no asymptotic effect on the size of the data structure, since each pattern is of size $O(r)$, and we anyway store the $O(r)$ distances from each pattern to all vertices of R. However, in the recursive scheme in the next section this observation will become useful.

Query. To answer a query for the distance between vertices s and t we proceed as follows. If s and t are in the same regions, we simply return the distance $\mathbf{d}(s, t)$ stored in item (iv). Otherwise, let R be the region containing t, and let h be the hole of R such that $s \in V^\circ(h)$. Let v_h be the canonical vertex of h. We return $\mathbf{d}(s, v_h) + \mathbf{d}(p_{h,G}(s), t)$. The correctness is immediate from

Lemma 4. We note that $\mathbf{d}(s, v_h)$ is stored in item (iii), a pointer to $p_{h,G}(s)$ is stored in item (ii), and $\mathbf{d}(p_{h,G}(s), t)$ is stored in item (i). The query is illustrated in Figure 4.

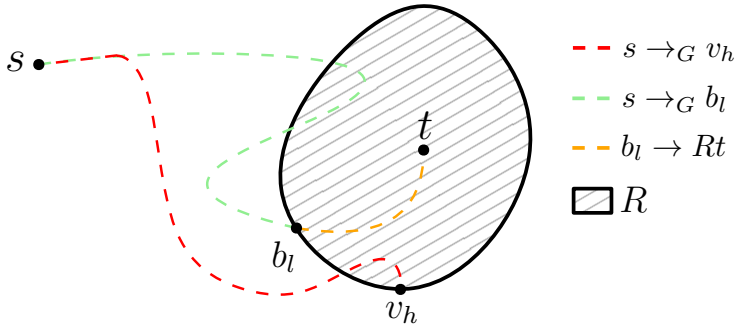


Figure 2: Illustration of the query in Section 4. By Lemma 4 the query returns $\mathbf{d}(s, b_0) + \mathbf{d}_R(p, t) = \mathbf{d}(s, b_\ell) + \mathbf{d}_R(b_\ell, t) = \mathbf{d}(s, t)$ where b_ℓ is some boundary vertex of R on $s \rightsquigarrow t$.

As aforementioned this oracle can be distributed into a distance labeling of size $O(n^{3/4})$ per label such that the distance between any two vertices s, t can be computed in $O(1)$ time given just the labels of s and t.

5 $O(n^{5/3+\varepsilon})$ space distance oracle

A bottleneck in the above approach comes from having to store, for each pattern p of a hole h of a region R, the distances from p to all vertices of R. Instead, we use a recursive r-division, in which we store for p, only the distances to the canonical vertex of a hole h' of each child region R' of R instead of all the vertices in the region. For this information to be useful we also store the pattern induced by p on the hole h', which is defined as follows.

Definition 3. *(Pattern induced by a pattern) Let R be a region in a graph G. Let h be a hole of R and p_h be a pattern of h (w.r.t. a vertex or another pattern). Let R' be a child region of R. Let $v_{h'} = b_0, b_1, \ldots, b_k$ be the vertices of the boundary walk of a hole of h' of R'. The pattern induced by p_h on h' is the vector $p_{h'}$ satisfying $p_{h'}[i] = \mathbf{d}(p, b_i) - \mathbf{d}(p, b_{i-1})$ for $1 \le i \le k$.*

Lemma 6. *Consider the settings of Definition 3. If $p_h = p_h(u)$ for some $u \in V^\circ(h)$, then $p_{h'} = p_{h'}(u)$.*

Proof. By Lemma 4, for every $0 \le i \le k$, $\mathbf{d}(u, b_i) - \mathbf{d}(u, v_h) = \mathbf{d}(p, b_i)$. Hence for all $1 \le i \le k$, $p_{h'}[i] = \mathbf{d}(p, b_i) - \mathbf{d}(p, b_{i-1}) = \mathbf{d}(u, b_i) - \mathbf{d}(u, v_h) - (\mathbf{d}(u, b_{i-1}) - \mathbf{d}(u, v_h)) = \mathbf{d}(u, b_i) - \mathbf{d}(u, b_{i-1})$, which is, by definition, $p_{h'}(u)[i]$. □

Preprocessing. We first compute a $\mathbf{r} = (r_0, r_1, \ldots, r_k, r_{k+1})$-division of G for $\mathbf{r}$ to be determined later, and denote by $\mathcal{T}_\mathbf{r}$ the associated decomposition tree. For convenience, we let $r_0 = n$, $r_{k+1} = 1$ and define $C(R) = \{R' \mid R' \text{ is a child of } R \text{ in } \mathcal{T}_\mathbf{r}\}$. In the following we let P_h denote the set $\{p_h(u) : u \in G\}$. We store the following:

1. For each $u \in V(G)$ we store a list of regions $R_0 \supset R_1 \supset \cdots \supset R_k$ containing u, where $R_i \in \mathcal{T}^i_\mathbf{r}$. (Recall that $\mathcal{T}^i_\mathbf{r}$ is the set of all nodes of $T_\mathbf{r}$ at level i).

2. For each $u \in G$, for each $0 \leq i \leq k-1$, for each region $R \in \mathcal{T}_{\mathbf{r}}^i$ containing u, for each child region $R' \subset R$ at level-$(i+1)$, let h be the hole of R' such that $u \in V^\circ(h)$. We associate with the pair (u, R') (i) a pointer to $p_h(u)$, (ii) the canonical vertex v_h, and (ii) the distance $\mathbf{d}(u, v_h)$).

3. For each $1 \leq i \leq k$, for each $R \in \mathcal{T}_{\mathbf{r}}^i$, for each hole h in R, for each $p \in P_h$ and for each $R' \in C(R)$, let h' be the hole of R' such that $v_h \in V^\circ(h')$. We associate with the pair (p, R') (i) a pointer to the pattern $p_{h'}(p)$ induced by p on h', (ii) the canonical vertex $v_{h'}$, and (iii) the distance $\mathbf{d}(p, v_{h'})$).

Space analysis. Storing 1 requires space $O(kn)$. To bound the space for item 2, we note that the number of regions at level i to which a vertes u belongs is bounded by the degree of u. Since the average vertex degree in a planar graph is at most 6, the average number of regions at level i to which u belongs is at most 6. Each such region has r_i/r_{i+1} subregions at level-$(i+1)$, so storing 2 requires space $O(n\sum_{i=0}^{k-1} r_i/r_{i+1}) = O(n^2/r) + O(n\sum_{i=1}^{k-1} r_i/r_{i+1})$. Storing 3 requires space $O(\sum_{i=1}^{k}(n/r_i)\cdot r_i^3 \cdot r_i/r_{i+1}) = O(n\sum_{i=1}^{k} r_i^3/r_{i+1})$. The total space is thus, $O(nk + n^2/r + n\sum_{i=1}^{k} r_i^3/r_{i+1})$.

Query. Algorithm 1 show pseudocode describing the query procedure. To process a query $\mathbf{d}(s,t)$ the query procedure first determines the largest value i for which s and t belong to the same region in $\mathcal{T}_{\mathbf{r}}^i$. Note that such a region must always exists as the root of $\mathcal{T}_{\mathbf{r}}$ is all of G. This level can be found in $O(k)$ time by traversing $\mathcal{T}_{\mathbf{r}}$, starting from a leaf region containing s and a leaf region containing t.

Let R_t be the level-$(i+1)$ region stored for t in item 1. Note that, $t \in R_t$, and, by choice of i, $s \notin R_t$. Hence, s is in some hole h of R_t. We retrieve the pattern $p_h(s)$ and the distance $\mathbf{d}(s, v_h)$ associated with (s, R_t) in item 2. We then proceed iteratively "zooming" into increasingly smaller regions containing t.

We show that the algorithm maintains the invariant that, at the beginning of each iteration, we have a level-i region R_t containing t, the variable d stores $\mathbf{d}(s, v_h)$, where h is the hole of R_t such that $s \in V^\circ(h)$, and the variable p stores (a pointer) to the pattern $p_h(t)$. Thus, when we reach the singleton region containing t, the variable d stores $\mathbf{d}(s,t)$.

Algorithm 1 Query procedure for the $O(n^{5/3+\varepsilon})$ construction.

```
1:  procedure QUERY(s, t)
2:      i ← the largest i s.t. the region R_i stored in item 1 for t contains both s and t
3:      R_t ← level (i + 1) region stored in item 1 for t
4:      (p, d) ← the tuple associated with (s, R_t)
5:      i ← i + 1
6:      while i ≤ k do
7:          R'_t ← level (i + 1) subregion of R_t stored in item 1 for t
8:          (p', d') ← the tuple associated with (p, R'_t)
9:          d ← d + d' ; p ← p' ; R_t ← R'_t ; i ← i + 1
10:     return d
```

We have already established that the invariant is maintained just before the loop is entered for the first time. In each iteration of the loop we retrieve R'_t, a level-$(i+1)$ subregion or R_t containing t (available in item 1), and retrieve $d' \leftarrow \mathbf{d}(p, v_{h'})$ and $p' \leftarrow p_{h'}(p)$ (associated with the pair (p, R'_t) in item 3). By Lemma 4, $d + d' = \mathbf{d}(s, v_h) + \mathbf{d}(p_h(u), v_{h'}) = \mathbf{d}(s, v_{h'})$. By Lemma 6, $p' = p_{h'}(t)$. Hence, after the assignements in Line 9, the invariant is restored.

The time complexity of the query is clearly $O(k)$.

Choosing parameters: Recall that the space requirement is $O(nk + n^2/r + n\sum_{i=1}^{k} r_i^3/r_{i+1})$. Picking each r_i s.t. $r_i/r_{i+1} = r_1^{\varepsilon}$ results in $r_k = \Theta(1)$ when $k = \Theta(1/\varepsilon)$, and in a query time of $O(1/\varepsilon)$. Choosing $r_1 = n^{1/3+\varepsilon}$, the total space used becomes

$$O\left(n\sum_{i=1}^{k} r_i^3/r_{i+1}\right) = O(nr_1^2 r_1^{\varepsilon}) = O(n^{1+2/3+2\varepsilon+\varepsilon/3+\varepsilon^2}) = O(n^{5/3+\varepsilon'})$$

for a suitable choice of ε'.

One can decrease the sizes of regions more aggressively to get the query time of $k = O(\log(1/\varepsilon))$ of Theorem 1. To this end we choose $\mathbf{r}$ such that $r_i^3/r_{i+1} = n^{2/3+\varepsilon}$, and $r_1 = n^{1/3}$. Then the space requirement is $O(n^{5/3} + nkn^{2/3+\varepsilon}) = O(kn^{5/3+\varepsilon})$. It is not hard to verify that one gets $r_i = O(n^{1/3-\varepsilon\frac{3^{i-2}-1}{2}})$, so $r_k = O(1)$ with $k = O(\log(1/\varepsilon))$.

As a last remark we note that the smallest interesting choice of ε in Theorem 1 is $\Theta(1/\log n)$, giving $O(n^{5/3})$ space and $O(\log\log n)$ query-time, which is a faster query-time than was previously known for this amount of space [9, 7].

References

[1] A. Abboud, P. Gawrychowski, S. Mozes, and O. Weimann. Near-Optimal Compression for the Planar Graph Metric. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 530–549. Society for Industrial and Applied Mathematics, Philadelphia, PA, jan 2018.

[2] S. Arikati, D. Z. Chen, L. P. Chew, G. Das, M. Smid, and C. D. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. In J. Diaz and M. Serna, editors, *Algorithms — ESA '96*, pages 514–528, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[3] S. Cabello. Many Distances in Planar Graphs. *Algorithmica*, 62(1-2):361–381, feb 2012.

[4] S. Cabello. Subquadratic Algorithms for the Diameter and the Sum of Pairwise Distances in Planar Graphs. *ACM Transactions on Algorithms*, 15(2):1–38, dec 2018.

[5] T. M. Chan. All-pairs shortest paths for unweighted undirected graphs in $o(mn)$ time. *ACM Trans. Algorithms*, 8(4):34:1–34:17, 2012.

[6] T. M. Chan and D. Skrepetos. Faster Approximate Diameter and Distance Oracles in Planar Graphs. *Algorithmica*, 81(8):3075–3098, aug 2019.

[7] P. Charalampopoulos, P. Gawrychowski, S. Mozes, and O. Weimann. Almost optimal distance oracles for planar graphs. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, pages 138–151, New York, NY, USA, 2019. Association for Computing Machinery.

[8] D. Z. Chen and J. Xu. Shortest path queries in planar graphs. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, STOC 2000, pages 469–478, New York, NY, USA, 2000. Association for Computing Machinery.

[9] V. Cohen-Addad, S. Dahlgaard, and C. Wulff-Nilsen. Fast and Compact Exact Distance Oracle for Planar Graphs. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 962–973. IEEE, oct 2017.

[10] H. Djidjev. On-line algorithms for shortest path problems on planar digraphs. In *Proceedings of the 22nd International Workshop on Graph-Theoretic Concepts in Computer Science*, WG 1996, pages 151–165, Berlin, Heidelberg, 1996. Springer-Verlag.

[11] J. Fakcharoenphol and S. Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *Journal of Computer and System Sciences*, 72(5):868–889, aug 2006.

[12] P. Gawrychowski, S. Mozes, O. Weimann, and C. 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 2018, pages 515–529, USA, 2018. Society for Industrial and Applied Mathematics.

[13] Q. Gu and G. Xu. Constant query time $(1+\epsilon)$ -approximate distance oracle for planar graphs. In *26th ISAAC*, pages 625–636, 2015.

[14] K.-i. Kawarabayashi, P. N. Klein, and C. Sommer. Linear-Space Approximate Distance Oracles for Planar, Bounded-Genus, and Minor-Free Graphs. apr 2011.

[15] P. Klein. Preprocessing an undirected planar network to enable fast approximate distance queries. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '02, pages 820–827, USA, 2002. Society for Industrial and Applied Mathematics.

[16] P. N. Klein, S. Mozes, and C. Sommer. Structured recursive separator decompositions for planar graphs in linear time. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing - STOC '13*, page 505, New York, New York, USA, 2013. ACM Press.

[17] J. Li and M. Parter. Planar diameter via metric compression. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, pages 152–163, New York, NY, USA, 2019. Association for Computing Machinery.

[18] S. Mozes and C. Sommer. Exact distance oracles for planar graphs. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 2012, pages 209–222, USA, 2012. Society for Industrial and Applied Mathematics.

[19] Y. Nussbaum. Improved distance queries in planar graphs. In *Proceedings of the 12th International Conference on Algorithms and Data Structures*, WADS 2011, pages 642–653, Berlin, Heidelberg, 2011. Springer-Verlag.

[20] M. Pătraşcu and L. Roditty. Distance Oracles beyond the Thorup–Zwick Bound. *SIAM Journal on Computing*, 43(1):300–311, jan 2014.

[21] N. Sauer. On the density of families of sets. *Journal of Combinatorial Theory, Series A*, 13(1):145–147, jul 1972.

[22] C. Sommer. Shortest-path queries in static networks. *ACM Computing Surveys*, 46(4):1–31, apr 2014.

[23] M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *Journal of the ACM (JACM)*, 51(6):993–1024, nov 2004.

[24] C. Wulff-Nilsen. *Algorithms for Planar Graphs and Graphs in Metric Spaces.* PhD thesis, 2010.

[25] C. Wulff-Nilsen. Constant time distance queries in planar unweighted graphs with subquadratic preprocessing time. *Computational Geometry*, 46(7):831–838, oct 2013.

[26] C. Wulff-Nilsen. Approximate distance oracles for planar graphs with improved query time-space tradeoff. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 351–362, USA, 2016. Society for Industrial and Applied Mathematics.

Appendix B

Appendix C starts from the next page, and includes the ArXiV version of the ISAAC 2021 accept "Near-Optimal Distance Oracles for Vertex-Labeled Planar Graphs", see https://doi.org/10.48550/arXiv.2110.00074.

Near-Optimal Distance Oracles for Vertex-Labeled Planar Graphs

Jacob Evald, Viktor Fredslund-Hansen*, Christian Wulff-Nilsen†

October 4, 2021

Abstract

Given an undirected n-vertex planar graph $G = (V, E, \omega)$ with non-negative edge weight function $\omega : E \to \mathbb{R}$ and given an assigned label to each vertex, a vertex-labeled distance oracle is a data structure which for any query consisting of a vertex u and a label λ reports the shortest path distance from u to the nearest vertex with label λ. We show that if there is a distance oracle for undirected n-vertex planar graphs with non-negative edge weights using $s(n)$ space and with query time $q(n)$, then there is a vertex-labeled distance oracle with $\tilde{O}(s(n))$[1] space and $\tilde{O}(q(n))$ query time. Using the state-of-the-art distance oracle of Long and Pettie [12], our construction produces a vertex-labeled distance oracle using $n^{1+o(1)}$ space and query time $\tilde{O}(1)$ at one extreme, $\tilde{O}(n)$ space and $n^{o(1)}$ query time at the other extreme, as well as such oracles for the full tradeoff between space and query time obtained in their paper. This is the first non-trivial exact vertex-labeled distance oracle for planar graphs and, to our knowledge, for any interesting graph class other than trees.

*Department of Computer Science, University of Copenhagen, viha@di.ku.dk. This research is supported by the Starting Grant 7027-00050B from the Independent Research Fund Denmark under the Sapere Aude research career programme.

†Department of Computer Science, University of Copenhagen, koolooz@di.ku.dk, http://www.diku.dk/koolooz/. This research is supported by the Starting Grant 7027-00050B from the Independent Research Fund Denmark under the Sapere Aude research career programme.

[1]We use $\tilde{O}$-notation to suppress $\text{poly}(\log n)$-factors.

1 Introduction

Efficiently answering shortest path distance queries between pairs of vertices in a graph is a fundamental algorithmic problem with a wide range of applications. An algorithm like Dijkstra's can answer such a query in near-linear time in the size of the graph. If we allow for precomputations, we can break this bound, for instance by simply storing the answers to all possible queries in a look-up table. However, a fast query time should preferably not come at the cost of a large space requirement. A *distance oracle* is a compact data structure that can answer a shortest path distance query in constant or close to constant time.

A lot of research has focused on approximate distance oracles which allow for some approximation in the distances output. This is reasonable since there are graphs for which the trivial look-up table approach is the best possible for exact distances. However, for restricted classes of graphs, it may be possible to obtain exact oracles with a much better tradeoff between space and query time. Indeed, for any planar n-vertex digraph, there is an exact distance oracle with space close to linear in n and query time close to constant [7, 2, 12].

A related problem is that of obtaining a vertex-labeled distance oracle. Here, we are given a graph with each vertex assigned a label. A query consists of a pair (u, λ) of a vertex u and a label λ and the output should be the distance from u to the nearest vertex with label λ. Each vertex is given only one label but the same label may be assigned to multiple vertices. To give some practical motivation, if the graph represents a road network, a label λ could represent supermarkets and the output of query (u, λ) gives the distance to the nearest supermarket from the location represented by u.

Note that this is a generalization of the distance oracle problem since vertex-to-vertex distance queries can be answered by a vertex-labeled distance oracle if each vertex is given its own unique label. If L is the set of labels, a trivial vertex-labeled distance oracle with constant query time is a look-up table that simply stores the answers to all possible queries, requiring space $O(n|L|)$. This bound can be as high as quadratic in n.

Our main result, which we shall state formally later in this section, is that for undirected edge-weighted planar graphs, the vertex-labeled distance oracle problem can be reduced to the more restricted distance oracle problem in the sense that up to $\log n$-factors, any space/query time tradeoff for distance oracles also holds for vertex-labeled distance oracles. Hence, the tradeoff from [12] translates to vertex-labeled distance oracles, assuming that the planar graph is undirected. To the best of our knowledge, this is the first non-trivial upper bound for vertex-labeled distance oracles in any interesting graph class other than trees [8, 15]. A strength of our result is that any future progress on distance oracles in undirected planar graphs immediately translates to vertex-labeled distance oracles.

1.1 Related work on vertex-labeled distance oracles

Vertex-labeled distance oracles have received considerably more attention in the approximate setting. With $(1+\epsilon)$ multiplicative approximation, it is known how to get $\tilde{O}(n)$ space and $\tilde{O}(1)$ query time both for undirected [11] and directed planar graphs [13] and it has been shown how oracles with such guarantees can be maintained dynamically under label changes to vertices using $\tilde{O}(1)$ time per vertex relabel.

For general graphs, vertex-labeled distance oracles with constant approximation have been presented [9, 3, 14] with state of the art being an oracle with $O(kn|L|^{1/k})$ space, $4k-5$ multiplicative approximation, and $O(\log k)$ query time, for any $k \in \mathbb{N}$.

1.2 Our contributions

We now state our reduction and its corollary:

Theorem 1. *If there is an exact distance oracle for n-vertex undirected edge-weighted planar graphs with $s(n)$ space, $q(n)$ query time, and $t(n)$ preprocessing time, then there exists an exact vertex-labeled distance oracle for such graphs with $s(n) + O(n \log^2 n)$ space, and with $O(q(n) \log n + \log^3 n)$ query time, and $t(n) + \text{poly}(n)$ preprocessing time.*

Plugging in the distance oracle of Long and Pettie et al. [12] gives the following corollary which can be seen as a generalization of their result:

Corollary 1. *For n-vertex undirected edge-weighted planar graphs, there exist exact vertex-labeled distance oracles with the following tradeoffs between space and query time:*

1. *$n^{1+o(1)}$ space and $\tilde{O}(1)$ query time,*
2. *$\tilde{O}(n)$ space and $n^{o(1)}$ query time.*

All oracles have preprocessing time polynomial in n.

Up to logarithmic factors, the full tradeoff between space and query time in their paper similarly extends to vertex-labeled distance oracles in undirected edge-weighted planar graphs.

The rest of the paper is organized as follows. In Section 2, we introduce basic definitions and notation and present tools from the literature that we will need for our oracle. In Section 3 we state the key lemmas but defer their proofs until later sections, and thus immediately present our reduction by describing how to obtain a vertex-labeled distance oracle given a distance oracle as a black box. In Section 4, we present a point location structure similar to [7] but with some important modifications to improve space in our setting.

2 Preliminaries

Let $G = (V, E, \omega)$ be a graph with edge weight function $\omega : E \to \mathbb{R} \cup \{\infty\}$. We denote by $V(G) = V$ and $E(G) = E$ the vertex and edge-set of G, respectively, and by $n = |V(G)|$ the number of vertices of G. A graph G' is said to be a subgraph of G if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. We denote by $u \rightsquigarrow_G v$ a *shortest path* from u to v in G, by $\mathbf{d}_G(u, v)$ the weight of $u \rightsquigarrow_G v$, and write $u \rightsquigarrow v = u \rightsquigarrow_G v$ and $\mathbf{d}(u, v) = \mathbf{d}_G(u, v)$ when G is clear from context. For a shortest path $p = u \rightsquigarrow v = (u = p_1), p_2, \ldots, (p_k = v)$ we define vertex p_i to occur *before* p_j on p if $i < j$ and similarly for edges $p_i p_{i+1}$ and $p_j p_{j+1}$. Thus statements such as "the first/last vertex/edge on p satisfying some property P" will always be made w.r.t. this ordering. We also write $p \circ p'$ to denote the concatenation of paths (or edges) p and p', assuming the last vertex of p equals the first vertex of p'. Given $u, v, v' \in V$; we say that v is *closer* than v' to u in G if $\mathbf{d}_G(u, v) < \mathbf{d}_G(u, v')$ or $\mathbf{d}_G(u, v) = \mathbf{d}_G(u, v)$ and $v < v'$, assuming some lexicographic ordering on vertices. We denote by $V(p)$, respectively $E(p)$, the set of vertices, respectively edges, on a path p.

Assume in the following that G is undirected. G is said to be connected, respectively bi-connected, if any pair of vertices are connected by at least one, respectively two, vertex-disjoint paths. For a rooted spanning tree T in G and for any edge $e = uv$ not in T, we define *the fundamental cycle* of uv w.r.t. T as the cycle obtained as the concatenation of uv and the two paths of T from the root to u and v, respectively.

2.1 Planar graphs and embeddings

An *embedding* of a planar graph G assigns to each vertex a point in the plane and to each edge a simple arc such that its endpoints coincide with those of the points assigned to its vertices. A *planar embedding* of G is an embedding such that no two vertices are assigned the same point and such that no pair of arcs coincide in points other than those corresponding to vertices they share. A graph is said to be *planar* if it admits a planar embedding. When we talk about a planar graph we assume that it is *planar embedded* and hence some implicit, underlying planar embedding of the graph. When it is clear from the context we shall refer interchangeably to a planar graph and its embedding, its edges and arcs and its vertices and points. Thus the term graph can refer to its embedding, an edge to its corresponding arc and a vertex to its corresponding point in the embedding.

Assumptions about the input Unless stated otherwise, we shall always assume that G refers to a graph which is weighted, undirected and planar with some underlying embedding. Furthermore, we shall make the structural assumption that G is triangulated. Triangulation can be achieved by standard techniques, i.e. adding to each face f an artificial vertex and artificial edges from the artificial vertex to each vertex of $V(f)$ with infinite weight. This transformation preserves planarity, shortest paths and ensures that the input graph consists only of simple faces. We also assume that shortest paths in the input graph are unique; this can be ensured for any input graph by either randomly perturbing edge weights or with e.g. the deterministic approach in [6] which gives only an $O(1)$-factor overhead in running time. Finally, it will be useful to state the following lemma when talking about separators in a graph with unique shortest paths:

Lemma 1. *Let $u, v, x, y \in V(G)$. Then $u \rightsquigarrow v$ and $x \rightsquigarrow y$ share at most one edge-maximal subpath.*

Proof. Assume that $x \rightsquigarrow y$ intersects $u \rightsquigarrow v$ and let a resp. b be the first resp. last intersection along $u \rightsquigarrow v$. Since G is undirected, uniqueness of shortest paths implies that $a \rightsquigarrow b$ is a subpath of $u \rightsquigarrow v$ shared by $x \rightsquigarrow y$. □

Edge orderings, path turns and path intersections For an edge $e = uv$ of a planar embedded graph H, we let $<_e^H$ be the clockwise ordering of edges of H incident to v starting at e (ignoring edge orientations). Hence $<_e^H$ is a strict total order of these edges and e is the first edge in this order.

For vertices $u, v \in V(H)$, $x \in V(u \rightsquigarrow v) \setminus \{u, v\}$ and $y \in V(H) \setminus V(u \rightsquigarrow v)$, let pq be the last edge shared by $u \rightsquigarrow v$ and $x \rightsquigarrow y$. Furthermore let qz resp. qz' be the edge following pq in the traversal of $u \rightsquigarrow v$ and $x \rightsquigarrow y$, respectively. We say that $x \rightsquigarrow y$ *emanates from the left* of $u \rightsquigarrow v$ if $qz' <_{pq}^H qz$, and otherwise it *emanates from the right*. We dually say that $y \rightsquigarrow x$ *intersects $u \rightsquigarrow v$ from the left (right)* if $x \rightsquigarrow y$ *emanates from the left (right)*.

Given a face f of H, vertices $u \in V(f)$, and $v, v' \in V$, let H'_f be a copy of H with an artificial vertex f^* embedded in the interior of f along with an additional edge f^*u. Define the paths $p_v = f^*u \circ u \rightsquigarrow_{H'_f} v$ and $p_{v'} = f^*u \circ u \rightsquigarrow_{H'_f} v'$ and assume that neither path is a prefix of the other. By assumption and Lemma 1, p_v and $p_{v'}$ share exactly one edge-maximal subpath $f^* \rightsquigarrow x$. We say that $u \rightsquigarrow_H v$ makes a *left turn* w.r.t $u \rightsquigarrow_H v'$ from f if $x \rightsquigarrow v$ emanates from the left of p_v, and otherwise it makes a *right turn*; we will omit mention of f when the context is clear. Note that the notion of a turn is symmetric in the sense that $u \rightsquigarrow_H v$ makes a left turn w.r.t $u \rightsquigarrow_H v'$ iff $u \rightsquigarrow_H v'$ makes a right turn w.r.t $u \rightsquigarrow_H v$.

2.2 Voronoi Diagrams

The definitions in this subsection will largely be made in a manner identical to those of [7], but are included as they are essential to a point location structure which will be presented in Section 4. Given a planar graph $G = (V, E, \omega)$, $S \subseteq V$, the *Voronoi diagram of S in G*, denoted by $VD(S)$ in G is a partition of V into disjoint sets, $Vor(u)$, referred to as *Voronoi cells*, with one such set for each $u \in S$. The set $Vor(u)$ is defined to be $\{v \in V \mid d(u, v) < d(u', v)$ *for all* $u' \in S \setminus \{u\}\}$, that is the set of vertices that are closer to u than any other site in terms of $d(\cdot, \cdot)$. We shall simply write VD when the context is clear.

It will also be useful to work with a dual representation of Voronoi diagrams. Let VD_0^* be the subgraph of G^* s.t. $E(VD_0^*)$ is the subset of edges of G^* where $uv^* \in VD_0^*$ iff u and v belong to different Voronoi cells in VD. Let VD_1^* be the graph obtained by repeatedly contracting edges of VD_0^* incident to degree 2 vertices until no such vertex remains[2]. We refer to the vertices of VD_1^* as *Voronoi vertices*, and each face of the resulting graph VD_1^* can be thought of as corresponding to some Voronoi cell in the sense that its edges enclose exactly the vertices of some Voronoi cell in the embedding of the primal. We shall restrict ourself to the case in which all vertices of S lie on a single face h. In particular, h^* is a Voronoi vertex, since each site is a vertex on the boundary

[2]Formally, given a degree 2 vertex v with incident edges vw, vw', we replace these edges by ww', concatenate their arcs and embed ww' using this arc in the embedding.

of h in the primal. Finally, let VD^* be the graph obtained by replacing h^* with multiple copies, one for each edge. We note that since there are $|S|$ Voronoi sites (and thus faces in VD^*), the number of Voronoi vertices in VD^* is $O(|S|)$ due to Euler's formula. Furthermore, [7] show that when assuming unique shortest paths and a triangulated input graph, VD^* is a ternary tree. It follows that the primal face corresponding to a Voronoi vertex f^* consists of exactly three vertices, each belonging to different Voronoi cells. We refer to the number of sites in a Voronoi diagram as its *complexity*.

Finally, they also note that a *centroid decomposition*, T^*, can be computed from VD^* s.t. each node of T^* corresponds to a Voronoi vertex f^* and the children of f^* in T^* correspond to the subtrees resulting from splitting the tree at f^*, and s.t. the number of vertices of each child is at most a constant fraction of that of the parent. We remark that $VD^*(S)$ can be computed by connecting all sites to a super-source and running a single-source shortest paths algorithm, and its centroid decomposition in time proportional to $|V(VD^*(S))|$.

2.3 Separators and decompositions

In the following, we will outline the graph decomposition framework used by our construction. As part of the preprocessing step, we will recursively partition the input graph using balanced fundamental cycle separators until the resulting graphs are of constant size. We shall associate with the recursive decomposition of G a binary decomposition tree, $\mathcal{T}$, which is a rooted tree whose nodes correspond to the regions of the recursive decomposition of G. We will refer to nodes and their corresponding regions interchangeably. The root node of $\mathcal{T}$ corresponds to all of G. The following lemma states the invariants of the decomposition that will be used in our construction:

Lemma 2. *Let $G = (V, E, \omega)$ be an undirected, planar embedded, edge-weighted, triangulated graph and let T be a spanning tree[3] of G. Then there is an $\tilde{O}(n)$ time algorithm that returns a binary decomposition tree $\mathcal{T}$ of G s.t.*

1. *for any non-leaf node $G' \in \mathcal{T}$, its children G'_l, respectively G'_r corresponds to the non-strict interior, respectively non-strict exterior of some fundamental cycle in G' w.r.t. T,*
2. *for any child node, it contains at most a constant fraction of the faces of its parent,*
3. *for any leaf node it contains a constant number of faces of G,*
4. *for all nodes at depth i, $\mathcal{T}_i$, $\sum_{G' \in \mathcal{T}_i} |V(G')| = O(n)$*

Properties 1-3 follow from recursively applying a classic linear time algorithm for finding fundamental cycles. Property 4 follows from employing standard techniques that involve contracting degree-two vertices of the separators found at each level of recursion and weighting the resulting edges accordingly. This transformation results in a decomposition where the sum of faces of all regions at any level is preserved. We stress that our construction does not rely on the usual sparse simple cycle separators (of size $O(\sqrt{n})$) but rather fundamental cycle separators of size $O(n)$.

3 The vertex labeled distance oracle

In this section we describe our reduction which shows our main result. The reduction can be described assuming Lemma 2 and the existence of the point location structure which we will state in the following lemma, the proof of which is deferred to Section 4:

Lemma 3. *Let $G = (V, E, \omega)$ be an undirected, planar embedded, edge-weighted graph with labeling $l : V \to L$ and let p be a shortest path in G. There is a data structure $O_{G,p}$ with $O(|V| \log |V|)$ space which given $u \in V$ and $\lambda \in L$ returns a subset $C \subset V$ of constant size, s.t. if v is the vertex with label λ closest to u and $v \rightsquigarrow u$ intersects p, then $v \in C$. Each such query takes time at most $O(\log^2 |V|)$.*

[3] For our purposes, the spanning tree will be a shortest path tree.

3.1 Preprocessing

Given the input graph $G = (V, E, \omega)$, the preprocessing phase initially computes the decomposition tree, $\mathcal{T}$, of Lemma 2. Associated with each non-leaf node $G' \in \mathcal{T}$ is a fundamental cycle separator of $ab \in E(G')$ w.r.t. the shortest path tree T rooted at some $c \in V(G')$. For such a G' we shall refer to $S_1(G') = c \rightsquigarrow_{G'} a$ and $S_2(G') = c \rightsquigarrow_{G'} b$. Thus the fundamental cycle separator is given by $S_1(G') \circ ab \circ S_2(G')$. The preprocessing phase proceeds as follows: For all non-leaf nodes $G' \in \mathcal{T}$, compute and store data structures $O_{G',S_1(G')}$ and $O_{G',S_2(G')}$ of Lemma 3. Finally, a distance oracle D with $O(s(|V|))$ space capable of reporting vertex-to-vertex shortest path distances in time $O(t(|V|))$ is computed for G and stored alongside the decomposition tree and the point location structures.

Space complexity The decomposition tree $\mathcal{T}$ can be represented with $O(|V| \log |V|)$ space and D with $O(s(n))$ space. For each node $G' \in \mathcal{T}$, we store data structures $S_1(G')$ and $S_2(G')$, so by Lemma 2 and 3, we get

$$\sum_{i=0}^{\infty} \sum_{G' \in \mathcal{T}_i} |V(G')| \log |V(G')| = \sum_{i=0}^{c \log n} O(|V| \log |V|) = O(|V| \log^2 |V|)$$

for a total space complexity of $O(s(|V|) + |V| \log^2 |V|)$.

3.2 Query

Let $G' \in \mathcal{T}$ and consider the query $\mathbf{d}_{G'}(u, \lambda)$. If G' is a leaf node, the query is resolved in time $O(t(n))$ by querying D once for each vertex of G'. If G' is a non-leaf node, the query is handled as follows: First, data structures $O_{G',S_1(G')}$ and $O_{G',S_2(G')}$ are queried with u and λ, resulting in two "candidate sets", C_1 and C_2, one for each query. By Lemma 3, $C_1 \cup C_2$ contains the nearest vertex with label λ for which $u \rightsquigarrow_{G'} v'$ intersects either S_1 or S_2 if such a vertex exists. Compute $d_{G'} = \min \{\mathbf{d}_G(u, c) \mid c \in C_1 \cup C_2\} \cup \{\infty\}$ by querying D once for each vertex of $C_1 \cup C_2$. The query then recursively resolves $d_{G''} = \mathbf{d}_{G''}(u, \lambda)$ where G'' is a child of G' in $\mathcal{T}$ containing u. Finally, the query returns $\min \{d_{G'}, d_{G''}\}$.

Algorithm 1 Query procedure for the distance oracle.

```
1: procedure QUERY(u, λ, G')
2:     if G' is a leaf node in T then
3:         return min {d_{G'}(u, v) | v ∈ V(G') and l(v) = λ}
4:     else
5:         C_1 ← O_{G',S_1(G')}(u, λ); C_2 ← O_{G',S_2(G')}(u, λ)
6:         G'' ← A child of G' in T containing u
7:         d_{G'} ← min {d_G(u, c) | c ∈ C_1 ∪ C_2} ∪ {∞}
8:         d_{G''} ← QUERY(u, λ, G'')
9:         return min {d_{G'}, d_{G''}}
```

Correctness Denote by v the vertex of G with label λ nearest to u in G', and consider the case in which $u \rightsquigarrow_{G'} v$ intersects $S_1(G')$ or $S_2(G')$. In this case, $v \in C$ by Lemma 3 and $C \neq \emptyset$, so

$$d_{G'} = \min \{\mathbf{d}_G(u, c) \mid c \in C\} = \mathbf{d}_G(u, v) = \mathbf{d}_{G'}(u, v) = \mathbf{d}_{G'}(u, \lambda) \leq d_{G''}$$

with the inequality following from definition of v. Note that in case u is a vertex of either S_1 or S_2, the correct estimate is returned at the current level, but for a simpler description, the recursion proceeds anyways. Otherwise $u \rightsquigarrow'_G v$ intersects neither S_1 and S_2 in which case, the path must be fully contained in the (unique) child node, G'', of G' containing u. In this case, the query reports $d_{G''} = \mathbf{d}_{G''}(u, \lambda) = \mathbf{d}_{G''}(u, v) \leq d_{G'}$, showing the correctness.

Time complexity At each level of the recursion, $O_{G',S_1(G')}$ and $O_{G',S_2(G')}$ are queried in time $O(\log^2|V|)$. Furthermore, D is queried $|C| = O(1)$ times in total time $O(1) \cdot O(t(|V|)) = O(t(|V|))$. By Lemma 2, the query is recursively resolved on a problem instance which is a constant fraction smaller at each level of recursion, giving rise to the recurrence relation $T(n) = T(n/a) + O(t(n) + \log^2 n)$. When G' is a leaf node, then by Lemma 2, G' consists of a constant number of faces, described by the base case $T(n) = O(t(n))$ when $n \leq b$ for sufficiently small b. It is easily verified that a solution to the recurrence is bounded by $O(\log^3 n + t(n)\log n)$. This shows the main theorem, and the rest of this paper is devoted to proving Lemma 3.

4 The point location data structure

Our point-location data-structure uses techniques similar to those of [7] for point location in additively weighted Voronoi diagrams, but with some crucial differences in order to save space.

Both structures rely on being able to determine left/right turns of shortest paths in shortest path trees rooted at sites in G, but to facilitate this, the data structure of [7] explicitly stores an (augmented) shortest path tree rooted at each site as well as a data structure for answering least common ancestor (LCA) queries. The point location structure thus requires $\Theta(|S|n)$ space where S is the number of sites, and since S may be large as $\Theta(\sqrt{n})$ (corresponding to the size of a sparse balanced separator in a planar graph), this translates to $\Theta(n^{3/2})$ space for their problem. This will not work in our case since the number of sites can in fact be as high as $\Theta(n)$, leading to a quadratic space bound.

The second issue with applying the techniques from [7] directly to our setting, is that it requires us to store a Voronoi diagram for each label, for each shortest path. Each vertex of the path separator would then be a site in the stored Voronoi diagram but as each separator may be large, i.e. $\Theta(n)$, we may use as much as $\Theta(n|L|)$ space over *all* labels of L for a single separator. What we need is for the number of sites involved for a label λ to be proportional to the number of vertices with label λ; this would give a near-linear bound on the number of sites when summing over all $\lambda \in L$ across all levels of the recursive decomposition. We address these issues in the following sections.

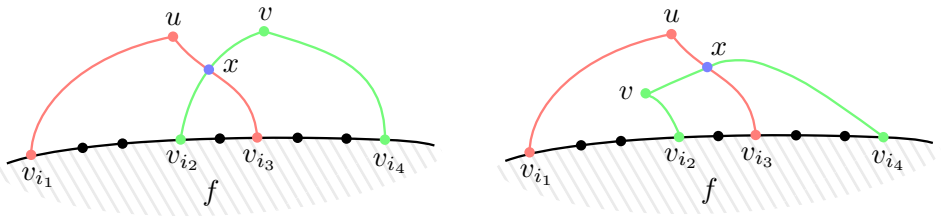
4.1 MSSP structure

To compactly represent shortest path trees, our point location structure uses an augmented version of the multiple-source shortest-path (MSSP) data structure of Klein [10]. It cleverly uses the persistence techniques of [5] in conjunction with top trees [1] to obtain an implicit representation of shortest path trees rooted at each site. Top-trees allow for shortest path distance queries and least-common ancestor (LCA) queries in time $O(\log n)$ per query while using $O(n \log n)$ space, and can easily be augmented to support turn queries, as we shall see shortly. To be used as a black box, the MSSP structure relies on being initialized from a face of G. In our construction, we wish to use it for querying left/right turns of paths and distances from vertices residing on shortest paths of fundamental cycle separators, and thus some further preprocessing is required. The guarantees of the augmented MSSP structure used for the point location structure are summarized in the following lemma:

Lemma 4. *Let $G = (V, E, \omega)$ be an edge-weighted planar graph, f be a face of G and let T_u denote the shortest path tree rooted at u. Then there exists a data structure MSSP(G, f) with $O(n \log n)$ space which given $u \in V(f)$ and $c, v \in V$ supports queries*

1. DIST(u, v)*: report* $\mathbf{d}_G(u, v)$*,*
2. LCA(u, c, v)*: report the least common ancestor of v, c in T_u,*
3. TURN(u, c, v)*: report whether $u \rightsquigarrow_{T_u} c$ makes a left or right turn w.r.t. $u \rightsquigarrow_{T_u} v$ or if one is a prefix of the other.*

in time $O(\log n)$ per query. The data structure can be preprocessed in $O(n \log n)$ time.



(a) Vertex v not contained in C.

(b) Vertex v contained in C.

Figure 1: Illustration of the proof of Lemma 6. The concatenation of $u \rightsquigarrow v_{i_1}$, $v_{i_1} \rightsquigarrow v_{i_3}$, and the reverse of $u \rightsquigarrow v_{i_3}$ forms a cycle C and $v \rightsquigarrow v_{i_2}$ intersects $u \rightsquigarrow v_{i_3}$ in x. Note that w.l.o.g. C is not necessarily simple, that $v \rightsquigarrow v_{i_2}$ may intersect the cycle more than once and that v may be a vertex of C.

Descriptions of Dist and LCA are available in [10] and [1], and a description of how to implement Turn is provided in Appendix 5 in terms of the vocabulary and interface specified in [1] for completeness. A top-tree representing any shortest path tree rooted at a vertex on f can be accessed in time $O(\log n)$ by using persistence in the MSSP structure. Lemma 4 then readily follows from applying the bound of Lemma 8 in Appendix 5.

4.2 Label sequences

To address the second issue, we first need to make the following definition and state some of its properties when applied in the context of planar graphs:

Definition 1. *Let $G = (V, E)$ be a graph, $p = p_1, \ldots, p_k$ a sequence of vertices and $S \subseteq V$. The label-sequence of p w.r.t. S is a sequence $M_{G,S,p} \in S^k$ satisfying $M_{G,S,p}(i) = \arg\min_{s \in S} dist_G(s, p_i)$. The alternation number on p w.r.t. S in G is defined as $|M_{G,S,p}| = \sum_{i=1}^{k-1} [M_{G,S,p}(i) \neq M_{G,S,p}(i+1)]$.*

When G, S and p are clear from the context, we shall simply write M, and also note that the sequence is well-defined due the tie-breaking scheme chosen in the preliminaries. The alternation number can be thought of as the number of times consecutive vertices on p change which vertex they are closest to among S when "moving along" p.

When G is an undirected planar graph and p is a shortest path in G, it can be observed[4] that M is essentially a Davenport-Schinzel sequence of order 2, and it immediately follows that the alternation number is "small" in the sense it is proportional to S while being agnostic towards the length of p altogether.

Definition 2 (Davenport-Schinzel [4])**.** *A sequence $u_1, u_2, \ldots, u_k$ over an alphabet Σ on n symbols is said to be a (n, s)-Davenport-Schinzel sequence if*

1. *$u_i \neq u_{i+1}$ for all $1 \leq i < k$ and*
2. *There do not exist $s + 2$ indices $1 \leq i_1 < i_2 < \ldots < i_{s+2} \leq k$ for which $u_{i_1} = u_{i_3} = \ldots = u_{i_{s+1}} = u$ and $u_{i_2} = u_{i_4} = \ldots = u_{i_{s+2}} = v$ for $u \neq v \in \Sigma$.*

Lemma 5 (Davenport-Schinzel [4])**.** *Let U be a $(n, 2)$-Davenport-Schinzel sequence of length m. Then $|U| \leq 2n - 1$.*

For a sequence of $\mathcal{S} = u_1, u_2, \ldots, u_k$ over an alphabet Σ, the *contraction* of $\mathcal{S}$ is the subsequence obtained from $\mathcal{S}$ by replacing every maximal substring $s, s, \ldots, s$ of $\mathcal{S}$ consisting of identical symbols s by a single occurence of s. As an example with $\Sigma = \{0, 1, 2\}$, the contraction of $0, 0, 1, 2, 2, 1, 1, 0, 1, 0, 0, 2$ is $0, 1, 2, 1, 0, 1, 0, 2$.

[4]We thank the anonymous reviewer for this observation which saved a tedious proof.

Lemma 6. *Let G be an undirected, weighted planar graph, let $S \subseteq V$, and let p be a shortest path of G contained in (the boundary of) a face of G. Then the contraction of M is a $(|S|, 2)$-Davenport-Schinzel sequence.*

Proof. Define $v_1, \ldots, v_k = p$ and assume for the sake of contradiction that for some $1 \leq i_1 < i_2 < i_3 < i_4 \leq k$ and $u, v \in S$ with $u \neq v$, it holds that $u = M(i_1) = M(i_3)$ and $v = M(i_2) = M(i_4)$. Then the concatenation of $u \rightsquigarrow v_{i_1}$, $v_{i_1} \rightsquigarrow v_{i_3}$, and the reverse of $u \rightsquigarrow v_{i_3}$ forms a cycle. Thus, either $v \rightsquigarrow v_{i_2}$ intersects $u \rightsquigarrow v_{i_1} \cup u \rightsquigarrow v_{i_3}$ or $v \rightsquigarrow v_{i_4}$ intersects $u \rightsquigarrow v_{i_1} \cup u \rightsquigarrow v_{i_3}$; see Figure 1a and 1b. By symmetry, we only need to consider the former case. If $v \rightsquigarrow v_{i_2}$ intersects $u \rightsquigarrow v_{i_3}$ in some vertex x then $v \rightsquigarrow x$ has the same weight as $u \rightsquigarrow x$. By the "closer than"-relation, $u = M(i_3) = M(i_2) = v$, contradicting our assumption that $u \neq v$. A similar contradiction is obtained if $v \rightsquigarrow v_{i_2}$ intersects $u \rightsquigarrow v_{i_1}$. □

Corollary 2. *Let G, S and p be as in Lemma 6. Then $|M_{G,S,p}| = O(|S|)$.*

We remark that M can be readily computed in polynomial time by adding a super-source connected to each vertex of S and running an SSSP algorithm. Furthermore M can be represented with $O(S)$ space, by storing only the indices for which $M(i) \neq M(i+1)$ and $M(i)$ for each such index.

We will now describe how to achieve $O(n)$ space for storing Voronoi diagrams for all labels $\lambda \in L$ at any level of the recursive decomposition. We do so by modifying the preprocessing steps and query scheme of [7] in a manner suitable for application of Lemma 6 and Corollary 2.

4.3 Preprocessing

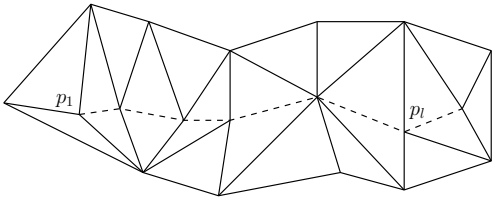
Let us briefly recall the statement of Lemma 3; that is, we let G be an undirected, edge-weighted, planar embedded graph with associated labeling $l : V \to L$ and let $p = p_1 \rightsquigarrow p_k$ be a shortest path in G. Given a query $(u, \lambda) \in V \times L$, we want to identify a small "candidate" set of vertices $C \subseteq V$ such that if v is the vertex with label λ closest to u and $u \rightsquigarrow v$ intersects p, then $v \in C$.

Here, we first describe how to compute a data structure which provides the guarantees of Lemma 3, but restricts itself to the case only where $u \rightsquigarrow v$ intersects p *from the left*. The description of the data structure for handling paths that intersect p from the right is symmetric (e.g. by swapping the endpoints of p). Lemma 3 thus readily follows from the existence of such structures.

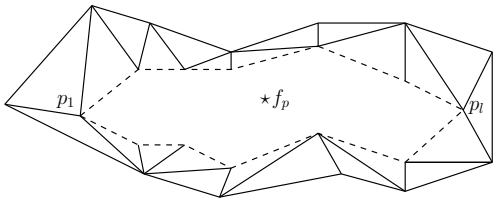
First, a copy, G_p, of G is stored and an incision is added along p in G_p. This results in a planar embedded graph G_p, which has exactly one more face than G. Define by $p' = p'_1, \ldots, p'_l$ and $p'' = p''_1, \ldots, p''_l$ the resulting paths along the incision, where $p'_1 = p''_1$ and $p'_l = p''_l$. We denote by f_p the face whose boundary vertices are $V(p') \cup V(p'')$. An illustration of this is provided in Figure 2a and 2b.

Next, the MSSP data structure of Lemma 4, $\mathit{MSSP}(G_p, f_p)$, is computed and stored as part of the point-location data structure. This structure will be used for the point location query.

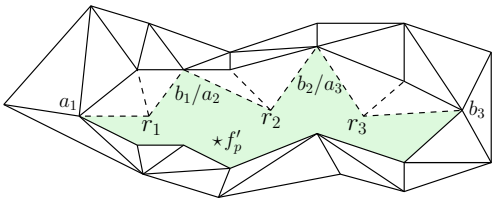
Centroid decompositions of Voronoi diagrams The following preprocessing is now done for *each* label $\lambda \in L$: First a copy, G_p^λ, of G_p is made. Next, $M_{G_p, S_\lambda, p'}$ is computed for $S_\lambda = \{v \in V \mid l(v) = \lambda\}$. For convenience we assume that $M(0) = \textsc{nil}$. The preprocessing phase now consists of modifying G_p^λ before computing the Voronoi diagram and the associated centroid decomposition associated with λ as follows: For $i \leftarrow 1, \ldots, l$, whenever $M(i) \neq M(i-1)$, a new vertex is added to G_p^λ and embedded in f_p along the curve formed by the deleted arc of the embedding of p. Denoting by r_i the most recently added vertex after iteration i, edge $p'_i r_i$ with $\omega(p'_i r_i) = d_G(M(i), p'_i)$ is added to G_p^λ and embedded for all i. Once again, it is fairly easy to see that G_p^λ is planar embedded. See Figure 2c for an illustration of this. Denote by a_i and b_i the endpoints of the first and last edges added incident to r_i (w.r.t. the order in which they were added). Denote by f'_p that has $\{r_i, a_i, b_i \mid 1 \leq i \leq l\} \cup V(p'')$ as its boundary vertices. Now the Voronoi diagram, its dual and subsequently its corresponding centroid decomposition, $T^*_{p,\lambda}$, is computed in (the now modified) G_p^λ using $R = \{r_i \mid 1 \leq i \leq l\}$ as Voronoi sites, see Figure 2d. The intuition is that each site in R corresponds to a contiguous subsequence $M(k), \ldots, M(l)$



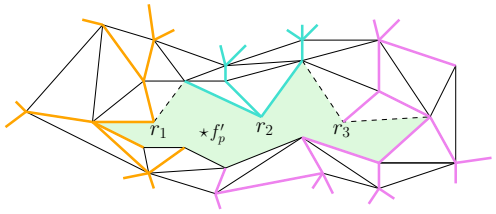
(a) G before incision. Here p as indicated by the dashed line.



(b) The resulting graph G_p after the incision; p is replaced by two paths p' and p'' that enclose the face f_p.



(c) The face f'_p resulting from adding edges to r_i. Here $b_1 = a_2$ and $b_2 = a_3$



(d) The Voronoi diagram is represented by a colored shortest-path tree for each site r_i.

Figure 2: Preprocessing steps for the point-location structure.

of M for which $M(j) = M(j+1) = v$ where v is the vertex with label λ closest to p'_j for $k \leq j < l$. This implies that the number of sites is proportional to the number of vertices with label λ instead of the length of the original separator p: By Lemma 2, $|M| = O(|S_\lambda|)$ and since $|R| = |M|$ it follows that $|R| = O(|S_\lambda|)$ bounds the complexity of $T^*_{p,\lambda}$, which is stored as part of the data structure. As aforementioned, each centroid $c \in T^*_{p,\lambda}$ corresponds to some degree three Voronoi vertex, f^*_c, with vertices, $\{x_1, x_2, x_3\}$ in the corresponding primal face f_c s.t. each x_j belongs to a different Voronoi site r_{i_j} for $j \in \{1, 2, 3\}$. For each such j, the centroid c stores a pointer to its corresponding face f_c, the first vertex p_{k_j} of p' on $r_{i_j} \rightsquigarrow_{G'_p} x_j$ and the weight $\omega(p_{k_j} r_{i_j})$.

Space complexity The space used for storing the MSSP structure is $O(|V| \log |V|)$ by Lemma 4. For each centroid, we store a constant amount of data, so the space required for storing the centroid decompositions is

$$\sum_{\lambda \in L} O(1) \cdot |T^*_{p',\lambda}| = \sum_{\lambda \in L} O(|S_\lambda|) = O(V)$$

since $S_\lambda \cap S_{\lambda'} = \emptyset$ for $\lambda \neq \lambda' \in L$ as each vertex has exactly one label. The total space used is thus $O(|V| \log |V|)$.

4.4 Handling a point location query

We now show how to handle a point location query. Note that in the following, we can assume that the vertices of p' appear in increasing order of their indices when traversing the boundary of f_p in a clockwise direction. Recall that given $u \in V$ and $\lambda \in L$, we wish to find a subset $C \subset V$ of constant size, s.t. if v is the closest vertex with label λ where $u \rightsquigarrow v$ intersects p from the left, then $v \in C$. The query works by identifying a subset $P \subseteq V(p')$ s.t. for some $p'_k \in P$ it holds that $M_{G_p, S_\lambda, p'}(k) = v$. We first show how to identify the subset by recursively querying the centroid decomposition $T^*_{p,\lambda}$ according to the following lemma, which we note is modified version of the query in [7]:

Lemma 7. *Given a query $(u, \lambda) \in V \times L$, consider the centroid decomposition tree $T^*_{p,\lambda}$ computed from G^λ_p in the preprocessing. Let c be a centroid $c \in T^*_{p,\lambda}$ corresponding to some Voronoi vertex, f^*_c, with associated primal triangle containing vertices $\{x_0, x_1, x_2\} = V(f_c)$ where x_j belongs*

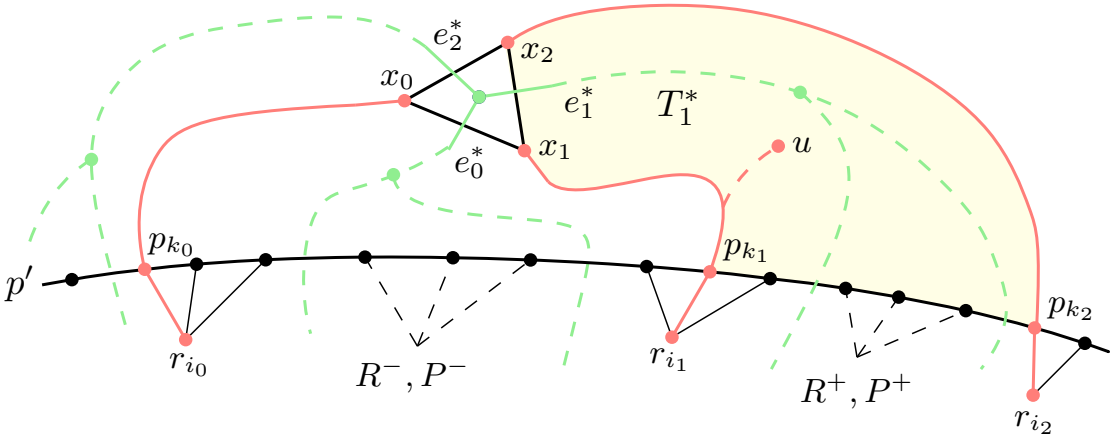


Figure 3: Illustration of Lemma 7. The dashed green lines represent Voronoi edges in the centroid decomposition and the red lines the primal shortest paths to the primal vertices of the centroid. In this case, $j^* = 1$, so u is contained in primal faces spanned by the subtree T_1^* contained in the region highlighted in yellow.

to the Voronoi cell of r_{i_j} for $j \in \{0,1,2\}$ and $i_0 < i_1 < i_2$. Furthermore let e_j^* be the dual edge incident to f_c^*, s.t. $e_j = x_j x_{(j+1) \bmod 3}$, let p_{k_j} be the successor of r_{i_j} on $r_{i_j} \rightsquigarrow_{G_p^\lambda} x_j$, let $P_j = p_{k_j} \rightsquigarrow_{G_p} u$, and let T_j^* be the subtree of $T_{p,\lambda}^*$ attached to c by e_j^* for $j \in \{0,1,2\}$. Finally, let $j^* = \arg\min_{j\in\{0,1,2\}}\{\mathbf{d}_{G_p}(p_{k_j}, u) + \omega(r_{i_j} p_{k_j})\} = \arg\min_{j\in\{0,1,2\}}\{\mathbf{d}_{G_p^\lambda}(r_{i_j}, u)\}$. Then

1. *If $p_{k_{j^*}} \rightsquigarrow_{G_p} u$ emanates from the left of P_{j^*} or $u \in P_{j^*}$, then the site closest to u in G_p^λ belongs to $R^- = \{r_{i_{(j^*-1) \bmod 3}}, \ldots, r_{i_{j^*}}\}$ and the second vertex on the shortest path from that site to u in G_p^λ belongs to $P^- = \{p_{i_{(j^*-1) \bmod 3}}, \ldots, p_{i_{j^*}}\}$; furthermore, $T_{j^*}^*$ is the centroid decomposition tree for G_p^λ when restricted to shortest paths from sites in R^- through successors in P^-.*

2. *otherwise, the site closest to u in G_p^λ belongs to $R^+ = \{r_{i_{j^*}}, \ldots, r_{i_{(j^*+1) \bmod 3}}\}$ and the second vertex on the shortest path from that site to u belongs to $P^+ = \{p_{i_{j^*}}, \ldots, p_{i_{(j^*+1) \bmod 3}}\}$; furthermore, $T_{(j^*+1) \bmod 3}^*$ is the centroid decomposition tree for G_p^λ when restricted to shortest paths from sites in R^+ through successors in P^+.*

Proof. By symmetry, we only consider the first case since the second case occurs when $p_{k_{j^*}} \rightsquigarrow_{G_p} u$ emanates from the right of P_{j^*} and the first case also includes the case where $u \in P_{j^*}$.

By the choice of j^*, $p_{k_{j^*}} \rightsquigarrow_{G_p} u$ cannot intersect any of the paths $P_{j'}$ with $j' \in \{(j^*-1) \bmod 3, (j^*+1) \bmod 3\}$ since these two paths are subpaths of shortest paths from sites $r_{i_{j'}} \neq r_{i_{j^*}}$ in G_p^λ and we assume unique shortest paths. Let x be the first vertex of $p_{k_{j^*}} \rightsquigarrow_{G_p} u$ such that either $x = u$ or the path emanates from the left of P_{j^*} at x. The rest of $p_{k_{j^*}} \rightsquigarrow_{G_p} u$ following x will not intersect P_{j^*} again due to uniqueness of shortest paths. Thus u belongs to the region of the plane enclosed by paths $P_{(j^*-1) \bmod 3}$, P_{j^*}, edge $e_{(j^*-1) \bmod 3}$, and path $p_{k_{(j^*-1) \bmod 3}}, \ldots, p_{k_{j^*}}$. Note that $T_{j^*}^*$ is the subtree of $T_{p',\lambda}^*$ spanning the primal faces contained in this region. Hence, $T_{j^*}^*$ is the centroid decomposition tree for G_p^λ when restricted to shortest paths from sites in R^- through successors in P^-. □

For an illustration of Lemma 7, see Figure 3. The Lemma implies a fast recursive point location scheme. On query (u, λ), obtain centroid c from $T_{p',\lambda}^*$. Since weights of edges from sites have been precomputed, $MSSP(G_p, f_p)$ is applied to find j^*. $MSSP(G_p, f_p)$ is also used to determine if $p_{k_{j^*}} \rightsquigarrow_{G_p} u$ emanates from the left of P_{j^*} and hence whether the first or second case of the lemma applies. The point location structure now recurses on a subset of sites and vertices of p' and on a subtree of $T_{p',\lambda}^*$, depending on which case applies.

The recursion stops when reaching a subtree corresponding to a bisector for two sites. The vertices of V with label λ corresponding to these two sites are reported as the set C, yielding the desired bound.

Time complexity The $O(\log^2 |V|)$ query time bound of Lemma 3 follows since there are $O(\log |V|)$ recursion levels and in each step, a constant number of queries to $MSSP(G_p, f_p)$ are executed, each taking $O(\log |V|)$ time.

5 Acknowledgments

We are grateful for the thorough and useful feedback provided by the reviewers.

References

[1] Stephen Alstrup, Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. Maintaining information in fully dynamic trees with top trees. *ACM Trans. Algorithms*, 1(2):243–264, October 2005. doi:10.1145/1103963.1103966.

[2] Panagiotis Charalampopoulos, Paweł Gawrychowski, Shay Mozes, and Oren Weimann. Almost optimal distance oracles for planar graphs. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, pages 138–151, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3313276.3316316.

[3] Shiri Chechik. Improved distance oracles and spanners for vertex-labeled graphs. In Leah Epstein and Paolo Ferragina, editors, *Algorithms – ESA 2012*, pages 325–336, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[4] H. Davenport and A. Schinzel. A Combinatorial Problem Connected with Differential Equations. *American Journal of Mathematics*, 87(3):684, jul 1965. URL: https://www.jstor.org/stable/2373068?origin=crossref, doi:10.2307/2373068.

[5] James R. Driscoll, Neil Sarnak, Daniel D. Sleator, and Robert E. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38(1):86–124, 1989. URL: https://www.sciencedirect.com/science/article/pii/0022000089900342, doi:https://doi.org/10.1016/0022-0000(89)90034-2.

[6] Jeff Erickson, Kyle Fox, and Luvsandondov Lkhamsuren. Holiest minimum-cost paths and flows in surface graphs. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 1319–1332. ACM, 2018. doi:10.1145/3188745.3188904.

[7] Paweł 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 2018, pages 515–529, USA, 2018. Society for Industrial and Applied Mathematics.

[8] Paweł Gawrychowski, Gad M. Landau, Shay Mozes, and Oren Weimann. The nearest colored node in a tree. *Theoretical Computer Science*, 710:66–73, 2018. URL: https://www.sciencedirect.com/science/article/pii/S0304397517306308, doi:https://doi.org/10.1016/j.tcs.2017.08.021.

[9] Danny Hermelin, Avivit Levy, Oren Weimann, and Raphael Yuster. Distance oracles for vertex-labeled graphs. In Luca Aceto, Monika Henzinger, and Jiří Sgall, editors, *Automata, Languages and Programming*, pages 490–501, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[10] Philip Klein. Multiple-source shortest paths in planar graphs. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 146–155, 01 2005.

[11] Mingfei Li, Chu Chung Christopher Ma, and Li Ning. $(1+\epsilon)$-distance oracles for vertex-labeled planar graphs. In T-H. Hubert Chan, Lap Chi Lau, and Luca Trevisan, editors, *Theory and Applications of Models of Computation*, pages 42–51, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[12] Yaowei Long and Seth Pettie. Planar Distance Oracles with Better Time-Space Tradeoffs. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2517–2537. Society for Industrial and Applied Mathematics, Philadelphia, PA, jan 2021. URL: https://epubs.siam.org/doi/10.1137/1.9781611976465.149, doi:10.1137/1.9781611976465.149.

[13] Shay Mozes and Eyal E. Skop. Efficient Vertex-Label Distance Oracles for Planar Graphs. *Theory of Computing Systems*, 62(2):419–440, feb 2018. URL: http://link.springer.com/10.1007/s00224-017-9827-0, doi:10.1007/s00224-017-9827-0.

[14] Maximilian Probst. On the Complexity of the (Approximate) Nearest Colored Node Problem. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms (ESA 2018)*, volume 112 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 68:1–68:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: http://drops.dagstuhl.de/opus/volltexte/2018/9531, doi:10.4230/LIPIcs.ESA.2018.68.

[15] Dekel Tsur. Succinct data structures for nearest colored node in a tree. *Information Processing Letters*, 132:6–10, 2018. URL: https://www.sciencedirect.com/science/article/pii/S0020019017301710, doi:https://doi.org/10.1016/j.ipl.2017.10.001.

Appendix A

A description of how to implement TURN in Lemma 4 is provided here in terms of the terminology and the interface specified in [1]. Readers that are not familiar with the terminology and interface pertaining to top-trees are referred to [1].

Lemma 8. *Let $G = (V, E, \omega)$ be a weighted planar graph, and let $\mathcal{T}$ be the root-cluster of a top-tree corresponding to some tree T in G. Then in addition to* DIST *and* LCA*, we can support* TURN *queries: For $u, c, v \in V$, report whether $u \rightsquigarrow c$ makes a left or right turn w.r.t. $u \rightsquigarrow_T v$, or if one is a prefix of the other in time $O(\log n)$ per query.*

Proof. A description of how to perform LCA queries is found in [1]. Assume that $u, c, v \in T$ and w.l.o.g. that u is strictly more rootward in T than v. First use $\mathcal{T}$ to determine the LCA c' of (v, c). If c' is on both $u \rightsquigarrow c$ and $u \rightsquigarrow v$ one path is a prefix of the other. Otherwise invoke EXPOSE(u, c') and traverse $\mathcal{T}$ until a leaf of $\mathcal{T}$ corresponding to the edge, $e_v \in E(T)$ is reached, which connects c' to the subtree containing v in T. This can be done in time $O(\log n)$. The same is done for (u, c') and (c, c'), exposing edges $e_u, e_c \in T$. Now, if $e_c = e_v$ or $e_c = e_u$, c must be on $u \rightsquigarrow_T v$. Otherwise it is easily checked, by maintaining a cyclic order of edges in the adjacency list of c', in constant time, whether e_c emanates to the left or right of the subpath $e_u e_v$ and hence $u \rightsquigarrow_T v$. The total time spent is $O(\log n)$. □

Appendix C

Appendix C starts from the next page, and includes the version of the ICALP 2021 accept "Decremental APSP in Directed Graphs Versus an Adaptive Adversary" in which the LIPICS typesetting options have been disabled to improve the readability, see https://doi.org/10.4230/LIPIcs.ICALP.2021.64.

Decremental APSP in Unweighted Digraphs Versus an Adaptive Adversary

Jacob Evald* Viktor Fredslund-Hansen† Maximilian Probst Gutenberg‡
Christian Wulff-Nilsen§

June 18, 2022

Abstract

Given an unweighted digraph $G = (V, E)$, undergoing a sequence of edge deletions, with $m = |E|, n = |V|$, we consider the problem of maintaining all-pairs shortest paths (APSP). Whilst this problem has been studied in a long line of research [ACM'81, FOCS'99, FOCS'01, STOC'02, STOC'03, SWAT'04, STOC'13] and the problem of $(1+\epsilon)$-approximate, weighted APSP was solved to near-optimal update time $\tilde{O}(mn)$ by Bernstein [STOC'13], the problem has mainly been studied in the context of an *oblivious* adversary which fixes the update sequence before the algorithm is started. In this paper, we make significant progress on the problem for an adaptive adversary which can perform updates based on answers to previous queries:

- We first present a *deterministic* data structure that maintains the *exact* distances with total update time $\tilde{O}(n^3)$[1].
- We also present a *deterministic* data structure that maintains $(1+\epsilon)$-approximate distance estimates with total update time $\tilde{O}(\sqrt{m}n^2/\epsilon)$ which for sparse graphs is $\tilde{O}(n^{2+1/2}/\epsilon)$.
- Finally, we present a randomized $(1+\epsilon)$-approximate data structure which works against an adaptive adversary; its total update time is $\tilde{O}(m^{2/3}n^{5/3}+n^{8/3}/(m^{1/3}\epsilon^2))$ which for sparse graphs is $\tilde{O}(n^{2+1/3}/\epsilon^2)$.

Our exact data structure matches the total update time of the best *randomized* data structure by Baswana et al. [STOC'02] and maintains the distance matrix in near-optimal time. Our approximate data structures improve upon the best data structures against an adaptive adversary which have $\tilde{O}(mn^2)$ total update time [JACM'81, STOC'03].

*`jeav@di.ku.dk`. The author is supported by the Starting Grant 7027-00050B from the Independent Research Fund Denmark under the Sapere Aude research career programme.

†`viha@di.ku.dk`. The author is supported by the Starting Grant 7027-00050B from the Independent Research Fund Denmark under the Sapere Aude research career programme.

‡`probst@di.ku.dk`. The author is supported by Basic Algorithms Research Copenhagen (BARC), supported by Thorup's Investigator Grant from the Villum Foundation under Grant No. 16582.

§`koolooz@di.ku.dk`, `http://www.diku.dk/~koolooz/`. The author is supported by the Starting Grant 7027-00050B from the Independent Research Fund Denmark under the Sapere Aude research career programme.

[1]We use $\tilde{O}$-notation to hide logarithmic factors.

1 Introduction

Shortest paths is a classical algorithmic problem dating back to the 1950s. The two main variants are the all-pairs shortest paths (APSP) problem and the single-source shortest paths (SSSP) problem, both of which have been extensively studied in various models, including the partially and fully-dynamic setting.

A dynamic graph algorithm is an algorithm that maintains information about a graph that is subject to updates such as insertions and deletions of edges or vertices. Such a graph can model real-world networks that change over time, such as road networks where traffic changes and roads are blocked from time to time. We say that a dynamic graph problem is *decremental* if it only allows deletions, *incremental* if it only allows insertions and *fully-dynamic* if it allows both. Incremental and decremental graphs are referred to as being *partially-dynamic*. A dynamic graph algorithm aims to efficiently process a sequence of online updates interspersed with queries about some property of the underlying dynamic graph.

1.1 Problem Definition

In this paper, we consider the *decremental all-pairs shortest-paths* problem where the goal is to efficiently maintain shortest path distances between all pairs of vertices in a decremental directed graph $G = (V, E)$. We shall restrict our attention to the case where G is unweighted. Letting m denote the initial number of edges and $n = |V|$, we want a data-structure which for any $u, v \in V$ supports the following operations:

- $\textsc{Dist}(u, v)$: reports the distance $d_G(u, v)$ from u to v in the current version of G,
- $\textsc{Delete}(u, v)$: deletes an edge (u, v) from E.

We furthermore consider the problem also in its relaxed version where we only aim to maintain *approximate* distance estimates which can then be queried. We denote by $\tilde{d}_G(u, v)$ a distance estimate for the distance from u to v and we say that an APSP algorithm has an *approximation ratio* (or *stretch*) of $t > 1$ if for any $u, v \in V$, we have that $d_G(u, v) \leq \tilde{d}_G(u, v) \leq t \cdot d_G(u, v)$. This paper will be concerned with both the exact and the $(1 + \epsilon)$-approximate version of the problem.

Another focus of this article is the *adversarial model*; the adversarial model defines the model under which the sequence of updates and queries are assumed to be made by an *adversary*. We say that a performance guarantee of an algorithm works against an *oblivious* adversary if the adversary must define the sequence of updates before the algorithm starts for the guarantee to hold. Thus the sequence of updates is independent of any random bits used by the algorithm. This is opposed to algorithms that work against an *adaptive* adversary, where the adversary is allowed to create the update sequence "on the go", e.g. based on answers to previous queries made to the data structure. Depending on the data structure, these choices may not be independent of the random choices made, which may result in the data structure performing poorly. One key advantage of a data structure that works against an adaptive adversary is that it can be used inside an algorithm as a black box, regardless of whether that algorithm adapts its updates to answers to queries. We point out that *deterministic* data structures always work against an adaptive adversary.

The performance of a *partially-dynamic* algorithm is usually measured in terms of the *total update time*. That is, the accumulated time it takes to process all updates (edge deletions). The *query time*, on the other hand, is the time to answer a single distance query. A natural goal is to minimize the total update time while keeping the stretch and query time small. Since all the structures presented in this paper explicitly maintain a distance matrix, the query time is constant.

1.2 Prior Work

The naive approach to dynamic APSP is to recompute the shortest path distances after each update using the best static algorithm. The query time is then constant and the time for a single update is $\tilde{O}(mn)$ for APSP and $\tilde{O}(m)$ for SSSP. At the other end of the spectrum one could achieve optimal update time by simply updating the input graph and only running an SSSP algorithm whenever a query is processed. Running a static algorithm each time, however, fails to reuse any information between updates whatsoever and gives a high query time, motivating more efficient dynamic approaches that do this.

In 1981, Even and Shiloach [6] gave a deterministic data-structure for maintaining a shortest path tree to given depth d in an undirected, unweighted decremental graph in total time $O(md)$. Henzinger and King [7] and King [10] later adapted this to directed graphs with integer weights. Running their structure for each vertex solves the decremental all-pairs shortest paths problem in $O(mn^2W)$ time, where edge weights are integers in $[1, W]$.

Henzinger and King were the first to improve upon this bound, giving an algorithm with total update time $\tilde{O}(mn^{2.5}\sqrt{W})$ [10] which is an improvement for $W = \omega(n)$. Demetrescu and Italiano [4] improved this data structure slightly and showed that the restriction to integral edge weights can be removed. Finally, the same authors [3] presented a data structure with total update time $\tilde{O}(mn^2)$ which is the state of the art for any data structure against an adaptive adversary up to today. In fact, their algorithm can be extended to a fully-dynamic algorithm with $\tilde{O}(n^2)$ amortized update time and which can handle vertex updates[2]. We also point out that this data structure was later simplified and generalized by Thorup [11].

Around the same time Baswana, Hariharan, and Sen [1] gave an *oblivious* Monte-Carlo construction with total update time $\tilde{O}(n^3)$ for *unweighted* graphs. Further, they showed that their data structure could be adapted to give an $(1 + \epsilon)$-approximate APSP algorithm for *weighted* graphs with total update time of $\tilde{O}(\sqrt{m}n^2/\varepsilon)$. In the exact setting, the oblivious adversary assumption is only required when paths are to be reported rather than just shortest path distances which are unique. Finally, Bernstein presented a $(1 + \epsilon)$-approximate algorithm with total running time $\tilde{O}(mn \log(W)/\epsilon)$ by using a clever approach of shortcutting paths [2]. Whilst his algorithm achieves near-optimal running time, again, the algorithm has to assume an oblivious adversary.

More recently, Karczmarz and Łącki [9] gave a deterministic $(1+\epsilon)$-approximate APSP algorithm for decremental graphs that runs in total time $\tilde{O}(n^3 \log(W)/\epsilon)$. They also presented the first non-trivial algorithm for incremental graphs [8] achieving total update time $\tilde{O}(mn^{4/3} \log(W)/\epsilon)$.

We refer the reader to the full version of the paper [5] for a more comprehensive treatment

[2]In this case, vertex updates refers to insertions or deletions of vertices with up to $n - 1$ incident edges.

of related work which also includes algorithms for *undirected* graphs and algorithms with larger stretch.

1.3 Our Contributions

In this paper, we present three new data structures for the all-pairs shortest paths problem. Our first theorem gives a *deterministic* data structure for the exact variant of the problem with near-optimal $\tilde{O}(n^3)$ total update time. It also matches the best *randomized* algorithm by Baswana et al. [1] and constitutes a significant improvement over the previous best deterministic bound of $\tilde{O}(mn^2)$ which is obtained by running an ES-tree [6] from every source or by the data structure Italiano et al. [3] (that also works in weighted graphs) and improves over all but the sparsest graph densities.

Our exact data structure is near-optimal as there is an $\Omega(n^3)$ lower bound on the total update time of any decremental data structure that explicitly maintains the distance matrix. The lower bound follows by considering an initial undirected, unweighted graph consisting of a simple path $v_0, v_1, v_2, \ldots, v_{n-1}$ plus additional edges (v_i, v_{i+2}) for each even $i \leq n-3$. Deleting these additional edges in any order creates $\Omega(n^3)$ distance matrix changes in total.

Theorem 1. *Let G be an unweighted directed graph with n vertices and initially m edges. Then there exists a deterministic data structure which maintains all-pairs shortest path distances in G undergoing an online sequence of edge deletions using a total time of $O(n^3 \log^3 n)$. The $n \times n$ distance matrix is explicitly maintained so that at any point, a shortest path distance query can be answered in constant time. The data structure can report a shortest path between any query pair in time proportional to the length of the path.*

Our second result is concerned with maintaining $(1+\epsilon)$-approximate all-pairs shortest path distances. This constitutes the first deterministic data structure that solves the problem in subcubic time with small approximation error (except for graphs that are not extremely dense).

Theorem 2. *Let G be an unweighted directed graph with n vertices and initially m edges. Then given $\epsilon > 0$, there exists a deterministic data structure that maintains all-pairs $(1+\epsilon)$-approximate shortest path distances in G undergoing an online sequence of edge deletions using a total time of $O(\sqrt{m}n^2 \log^2(n)/\epsilon)$. At any point, a $(1+\epsilon)$-approximate shortest path distance query can be answered in constant time and a $(1+\epsilon)$-approximate shortest path between the query pair can be reported in time proportional to the length of the path.*

Our third result gives a data structure achieving a better time bound. While we use randomization to achieve the improved time bound, our algorithm again works against an adaptive adversary.

Theorem 3. *Let G be an unweighted directed graph with n vertices and initially m edges. Then given any $\epsilon > 0$, there exists a Las Vegas data structure that maintains all-pairs $(1+\epsilon)$-approximate shortest path distances in G under an online sequence of edge deletions using a total expected time of $\tilde{O}(m^{2/3}n^{5/3}/\epsilon + n^{8/3}/(m^{1/3}\epsilon^2))$. This bound holds w.h.p. and the data structure works against an adaptive adversary. At any point, a $(1+\epsilon)$-approximate shortest path distance query can be answered in constant time.*

We summarize our results as well as previous state-of-the-art results in Table 1.

Time	Approximation	Adversary/ Deterministic	Reference
$O(mn^2)$	exact	deterministic	[6, 3]
$\tilde{O}(n^3)$	exact	deterministic	**New Result**
$\tilde{O}(n^3)$	exact	adaptive	[1]
$\tilde{O}(\sqrt{m}n^2/\epsilon)$	$(1+\epsilon)$	deterministic	**New Result**
$\tilde{O}(m^{2/3}n^{5/3}/\epsilon + n^{8/3}/(m^{1/3}\epsilon^2))$	$(1+\epsilon)$	adaptive	**New Result**
$\tilde{O}(\sqrt{m}n^2/\epsilon)$	$(1+\epsilon)$	oblivious	[1]
$\tilde{O}(nm)$	$(1+\epsilon)$	oblivious	[2]

Table 1: Our results and previous state-of-the-art results for decremental APSP.

1.4 Overview

Our overall approach for the deterministic data structures is similar to that of Baswana et al. [1] but with a key difference that allows us to avoid using a randomized hitting set and instead rely on deterministic separators. The idea of the construction by Baswana et al. relies on a well-known result which says that if we sample a subset H_i^ρ of the vertices of size $\tilde{O}(n/\rho^i)$ (where ρ is some constant strictly larger than 1), each with uniform probability, then, w.h.p. we "hit" each shortest-path of length $[\rho^i, \rho^{i+1})$ between any pair of vertices in any version of the graph G.

Phrased differently, given vertices $u, v \in V$, we have that if a shortest path from u to v is of length $\ell \in [\rho^i, \rho^{i+1})$, then there is some vertex $w \in H_i^\rho$, such that the concatenation of a shortest path from u to w and a shortest path from w to v is of length ℓ. For each such w, we say w is a *witness* for the tuple (u, v) for distance ℓ.

Now for each $u, v \in V$, if the initial distance from u, v was $\ell \in [\rho^i, \rho^{i+1})$, we can check H_i^ρ to find a witness w. If the length of the path from u to w to v is increased, we can continue our scanning of H_i^ρ to see whether another witness exists. If there is no witness $w \in H_i^\rho$ left at some stage, we know that there is no path of length ℓ left in G w.h.p. and increase our guess by setting $\ell \mapsto \ell + 1$.

Sampling initially a hitting set H_i^ρ for every $i \in [0, \log_\rho n]$, we can find the "right" hitting set for each distance ℓ. Observe now that for each tuple $(u, v) \in V^2$, we have to scan a hitting set of size $\tilde{O}(n/\rho^i)$ for $\rho^{i+1} - \rho^i \sim \rho^{i+1}$ levels before the hitting set index i is increased which only occurs $O(\log n)$ times, thus we only spend time $\tilde{O}(n)$ for each vertex tuple (u, v). Thus, the total running time of the searches for witnesses can be bound by $\tilde{O}(n^3)$.

The Deterministic Exact Data Structure. Our construction is similar in the sense that we maintain witnesses for each distance scale $[\rho^i, \rho^{i+1})$ for every $i \in [0, \log_\rho n]$ such that each distance ℓ is in one such distance scale. The key difference is that instead of using a randomized *global* hitting set H_i^ρ for a distance scale $[\rho^i, \rho^{i+1})$, our construction relies on deterministically maintaining a small local vertex separator $S_i(u)$ for every vertex $u \in V$ of size $\tilde{O}(n/\rho^i)$ separating all shortest paths starting in u with a distance in $[\rho^i, \rho^{i+1})$.

More precisely, for each distance scale $[\rho^i, \rho^{i+1})$ and vertex $u \in V$, we maintain a sep-

arator $S_i(u)$ that satisfies the invariant that every shortest path from u to a vertex v at distance at least ρ^i is intersected by a vertex in $S_i(u)$. If this invariant is violated after an adversarial update, then we find such a vertex v and need to add additional vertices to $S_i(u)$ during the time step. The challenge is to take these additional separator vertices such that the total size of $S_i(u)$ is not increased beyond $\tilde{O}(n/\rho^i)$. The separator procedure makes use of sparse layers of BFS trees and here is where we rely on the assumption that the graph is unweighted. We defer the details of the separator procedure to a later section and continue our discussion of the APSP data structure.

Since we need to detect whether vertices have distance less than ρ^i from u or not in G, we further have to use a bottom-up approach to compute distances after an edge deletion, i.e. we start with the smallest possible distance range and update all distances in this range and then update larger distances using the information already computed. This issue did not arise in Baswana et al. [1] but can be handled by a careful approach. The distances computed for one distance scale include all distances to and from witnesses for the next larger distance scale.

It is now easy to see that the scanning for witnesses can be implemented in the same time as in the analysis sketched above by scanning the list of local separator vertices which serve as witnesses instead of the hitting set. Further, we can maintain local vertex separators using careful arguments in total time $\tilde{O}(mn)$ giving our result in Theorem 1.

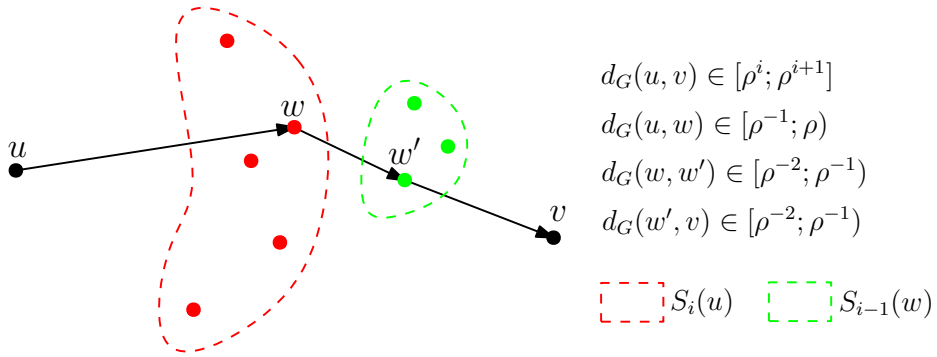


Figure 1: Illustration of separators and path "hierarchy". Here $u \rightsquigarrow v$ goes through a witness w, and $w \rightsquigarrow v$ goes through w'. If the length of the path $w' \rightsquigarrow v$ is increased by Δ, the distance estimates of all 2-hop-paths that use $w' \rightsquigarrow v$ as a sub-path are increased by that amount. In this case, the estimate for $w \rightsquigarrow w' \rightsquigarrow v$ is increased and is propagated to the next level where subsequently the estimate for $u \rightsquigarrow w \rightsquigarrow v$ is possibly increased.

The Deterministic Approximate Data Structure. In order to improve the running time for sparse graphs, we can further focus on only considering distances that are roughly at a $(1+\epsilon)$-multiplicative factor from each other. More concretely, instead of increasing the expected distance from ℓ to $\ell+1$ when we cannot find a witness for some path from u to v for distance ℓ, we can increase the next expected distance level ℓ' to $\sim(1+\epsilon)\ell$ and consider every vertex w a witness if there is a path $u \rightsquigarrow w \rightsquigarrow v$ of length at most ℓ'. Thus, we handle fewer distances and can thereby reduce the time to maintain distances that are at least d in total time $\tilde{O}(n^3/d + mn)$. Again, a careful approach is necessary to ensure that approximations do not add up over distance scales.

This is no faster than the data structure for exact distances when d is small so in order to get Theorem 2, we use the $O(mnd)$ data structure of Even and Shiloach [6] to maintain distances up to d. Picking d such that $mnd = n^3/d$ gives the result of Theorem 2 (the term $\tilde{O}(mn)$ vanishes since it is subsumed by the two other terms, also we assumed $\epsilon > 0$ to be a constant to simplify the presentation).

Maintaining Separators. We now sketch how to deterministically maintain a "small" local separator for a vertex $s \in V$ with some useful invariants.

Let S be the local separator for s. The first invariant that will be useful is that any vertex $t \in V$ that is reachable from s in $G \setminus S$, is "close" to s or roughly within distance d. As edges are deleted from G, the distances from s to such vertices t may increase. If a vertex t moves too far away from s, the invariant is re-established by growing BFS trees in parallel, one layer at a time, from s in $G \setminus S$ and from t in the graph obtained from $G \setminus S$ by reversing the orientations of all edges. The search halts when a layer (corresponding to the leaves of the BFS tree at the current iteration) that is "thin" is found, and its vertices are added to S; vertices that are on the opposite side of the separator than s are cut off as they must all be too far away from s. Here, "thin" refers to a BFS layer such that the number of vertices added to the separator is only a factor $\tilde{O}(1/d)$ times the number of vertices cut off. It is well known that such a layer exists (cfr. Lemma 2 for the details). Summing up, it follows that $|S| = \tilde{O}(n/d)$ at all times. By marking vertices as they are searched (according to the side of the BFS layer on which they are found), the vertices that are "cut off" from s by the augmented separator will never be searched again, and the cost of searching the edges of either side of the search can be charged to sum of the degree of these vertices, for a total update time of $O(m)$.

For our randomized data structure, we need an additional property that essentially allows us to take a snapshot of the current separator and use it in later updates rather than having to repeatedly update the separator. This will be key to getting an improved randomized time bound. Details can be found in Lemma 3 which states our separator result.

The Randomized Approximate Data Structure. The randomized approximate data structure of Theorem 3 follows the same overall approach but is technically more involved. Instead of keeping track of all 2-hop paths $u \rightsquigarrow s \rightsquigarrow v$[3] for every $s \in S_i(u)$, the randomized data structure samples a subset of these by picking each vertex of $S_i(u)$ independently with some probability p. It only keeps track of approximate shortest path distances going through this subset rather than the full set $S_i(u)$. This will speed up the above since the subset of the separator we need to scan is smaller by a factor p. However, this approach fails once no short 2-hop path intersects the sampled subset. At this point, w.h.p. there should only be short 2-hop paths through $O(\log n/p)$ vertices of $S_i(u)$ so also in this case, the subset can be kept small. However, scanning linearly through $S_i(u)$ to find this small subset will take $\tilde{O}(n/d)$ time and happen over all pairs (u, v).

Our solution is roughly the following. Suppose no sampled vertex certifies an approximate short path from u to v. Then v scans linearly through $S_i(u)$ to find the small size $O(\log n/p)$ subset $S_i'(u)$. Consider the set W of vertices w such that $d_G(w, v)$ is small com-

[3]Note that such a path may have more than one intermediate vertex, but it is useful to think of it as a path of two weighted edges/hops (u, s) and (s, v) since this is what is maintained by the data structure.

pared to d, i.e., $d_G(w,v) \leq \epsilon d$ for some small constant $\epsilon > 0$. Then we show that the small subset $S'_i(u)$ found for v can also be used for each vertex $w \in W$. The intuition is that for any vertex $s \in S_i(u) \setminus S'_i(u)$, the approximate shortest path distance from u to w through s must be large since otherwise we get a short path $u \rightsquigarrow s \rightsquigarrow w \rightsquigarrow v$ from u to v through s, contradicting that $s \notin S'_i(u)$.

It follows that if $|W|$ is large, the $\tilde{O}(n/d)$ cost of scanning $S_i(u)$ can be distributed among a large number of vertices of W. Dealing with the case where $|W|$ is small is more technical so we omit it here.

The way we deal with an adaptive adversary is roughly as follows. Consider a deterministic data structure that behaves like the randomized data structure above, except that it maintains 2-hop paths $u \rightsquigarrow s \rightsquigarrow v$ for all $S_i(u)$ rather than only through a sampled subset. The slack from the approximation allows us to round up all "short" approximate distances to the same value. Hence, as long as the randomized data structure has short 2-hop paths, it maintains exactly the same approximate distances as the deterministic structure and hence the approximate distances output to the adversary is independent of the random bits used.

2 Definitions and Notation

In the following, let $G = (V, E)$ be a directed unweighted graph. The graph $G_{\mathbf{rev}}$ is obtained from G by reversing the orientation of each edge. For any two vertices $u, v \in V$, we denote by $u \rightsquigarrow v$ a shortest path from u to v in G and let $d_G(u,v)$ denote the length of such a path. We extend this notation to sets so that, e.g., $d_G(u, V') = \min\{d_G(u,v) | v \in V'\}$ for $V' \subseteq V$.

We define a BFS-layer to mean the set of nodes at some fixed distance from some v in G. An *in-tree* in G is a BFS tree in $G_{\mathbf{rev}}$.

We will need notation to refer to dynamically changing data at specific points in time. Consider a sequence of updates to some object X where each update takes place at a time step $t \in \mathbb{N}$. We denote by $X^{(t)}$ the object just after update t. Here, X could be a graph, a shortest path distance, etc.

For handling small distances, we rely on the data-structure of Even and Shiloach [6], the properties of which we will state in the following lemma:

Lemma 1 ([6])**.** *Given a directed unweighted graph G undergoing a sequence of edge deletions, a source vertex $s \in V$, and $d > 0$, a shortest path tree in G rooted at s can be maintained up to distance d in total time $O(md)$. The structure requires $O(m)$ space and can be constructed in time $O(m+n)$.*

3 Maintaining Separators

Lemma 3 below provides a key tool used in all of our data structures. It gives an efficient data structure that maintains a growing separator set S of small size in a decremental graph G. To prove it, we need the following well-known result.

Lemma 2. *Given a directed unweighted n-vertex graph $G = (V, E)$, given $d_1, d_2 \in \mathbb{N}_0$ with $d_2 - d_1 + 1 \geq \lg n$, and given vertices $u, v \in V$ with $d_G(u,v) \geq d_2$, a BFS tree in G with root u contains a layer $L \subseteq V$ with $d_1 \leq d_G(u,L) \leq d_2$ and $|L| \leq |L_-| \lg n/(d_2 - d_1 + 1)$ where $L_- = \{w \in V | d_G(u,w) < d_G(u,L)\}$ is the union of layers closer to u than L.*

Proof. Denote by L_i the ith layer of the BFS tree from u. For each i, let $L_{<i} = \cup_{j<i} L_j$. Let $q = (d_2 - d_1 + 1)/\lg n$. Assume for contradiction that L does not exist. Then for $i = d_1, \ldots, d_2$, $|L_i| > |L_{<i}|/q$ so $|L_{<i+1}| = |L_i| + |L_{<i}| > (1 + 1/q)|L_{<i}|$. Since $q \geq 1$, we have $(1 + 1/q)^q \geq 2$ so

$$|L_{<d_2+1}| > (1 + 1/q)^{d_2-d_1+1}|L_{<d_1}| \geq 2^{(d_2-d_1+1)/q} = n,$$

contradicting that there are only n vertices in G. □

We now state and prove Lemma 3. It gives an efficient data structure that maintains a growing separator set S of small size in a decremental graph G with the following guarantees. Let s be a fixed vertex and let d be some given threshold distance. Then at every time step, vertices reachable from s in $G \setminus S$ are of distance slightly less than d from s in G. Conversely, for vertices v not reachable from s in $G \setminus S$, we have $d_G(s, v) = \Omega(d)$; furthermore, if $d_G(s, v)$ is larger than d by some small constant factor then any shortest path $s \rightsquigarrow v$ in G can be decomposed into $s \rightsquigarrow w \rightsquigarrow v$ such that $w \in S$, $d_G(s, w) \leq d$, and $d_G(w, v) \leq d$. In fact, the lemma states that w can be chosen in S^{t_0} where t_0 is the first time step in which $d_G(s, v)$ became (slightly) larger than d; note that this is a stronger statement since S is growing over time.

Lemma 3. *separatorLemma Let $G = (V, E)$ be an n-vertex unweighted digraph undergoing a sequence of edge deletions, let $s \in V$ be a source, and let $d \in \mathbb{N}$ with $d > 33 \lg n$. Let $\mathcal{O}$ be a data structure that maintains for each $v \in V$ a distance estimate $\tilde{d}(s, v) \geq d_G(s, v)$ such that if $d_G(s, v) \leq d$ then $\tilde{d}(s, v) \leq \frac{4}{3} d_G(s, v)$. Whenever an estimate $\tilde{d}(s, v)$ grows to a value of at least $\frac{32}{33}d$, $\mathcal{O}$ outputs v. Then there is a data structure $\mathcal{S}$ with access to $\mathcal{O}$ which maintains a growing set $S \subseteq V$ such that for each $v \in V$,*

1. *if v is reachable from s in $G \setminus S$ then $d_G(s, v) < \frac{32}{33}d$ and otherwise $d_G(s, v) > \frac{2}{3}d$,*
2. *if t_0 is a time step in which $d < d_G^{(t_0)}(s, v) \leq \frac{34}{33}d$ then for every time step $t_1 \geq t_0$ in which $d_G^{(t_1)}(s, v) \leq \frac{34}{33}d$, any shortest s-to-v path P in $G^{(t_1)}$ intersects $S^{(t_0)}$ and for the first such intersection vertex w along P, $d_G^{(t_1)}(s, w) \leq d$, and $d_G^{(t_1)}(w, v) \leq d$.*

At any time, $|S| = O(n \log n/d)$ and $\mathcal{S}$ has total update time $O(m)$, excluding the time spent by $\mathcal{O}$.

Proof. Let $\epsilon = \frac{1}{33}$. For each $v \in V$, let $\hat{d}(v)$ be obtained from the degree of v in the initial graph G by rounding up to the nearest multiple of $\Delta = \lceil m/n \rceil$. In the description of $\mathcal{S}$ below, processing one edge takes at most one unit of time.

Data structure $\mathcal{S}$ initializes $S = \emptyset$ and unmarks all vertices of V. Whenever $\mathcal{O}$ outputs an unmarked vertex v (marked output vertices are ignored), $\mathcal{S}$ runs a modified BFS from s in $G_S = G \setminus S$ which for each vertex w spends $\hat{d}(w)$ time to process its outgoing edges; this can always be achieved by busy-waiting at w if needed. In parallel, $\mathcal{S}$ runs a similar modified BFS from v in $G'_S = (G \setminus S)_{\mathbf{rev}}$[4]. The search from s halts if a layer L_s is found such that $\frac{2}{3}d < d_{G_S}(s, L_s) \leq (\frac{2}{3} + \epsilon)d$ and $|L_s| = O((x \log n)/d)$ (for a suitable hidden

[4]By "parallel", we mean that $\mathcal{S}$ alternates between spending one unit of time in one search, then one unit of time in the other search, and so on.

constant to be specified) where x is the number of vertices visited by the search, excluding L_s. Similarly, the search from v halts if a layer L_v is found such that $d_{G'_S}(v, L_v) < \epsilon d$ and $|L_v| = O((y \log n)/d)$ (again, for a suitable hidden constant) where y is the number of vertices visited by the search excluding L_v. Let L be the first of the two layers found. $\mathcal{S}$ halts both searches when L is found. Then L is added to S and all vertices visited by the search from v in G'_S are marked. The hidden constants are chosen such that the existence of L follows from Lemma 2; this lemma applies since by assumption, $\epsilon d > \lg n$.

Observe that when $\mathcal{O}$ outputs v, we have $d_G(s, v) \geq (1 - \epsilon)d/(4/3) = (\frac{2}{3} + 2\epsilon)d$ as otherwise we have $d_G(s, v) \leq d$ and hence $\tilde{d}(s, v) \leq \frac{4}{3} d_G(s, v) < (1 - \epsilon)d = \frac{32}{33} d$. This shows the existence of L_s and L_v and that no vertex or edge is visited by both searches. We have $d_{G_S}(s, v) \geq d_G(s, v) \geq (\frac{2}{3} + 2\epsilon)d$ and $d_{G_S}(s, L_s) > \frac{2}{3} d$ and for every $w \in L_v$,

$$d_{G_S}(s, w) \geq d_{G_S}(s, v) - d_{G_S}(w, v) \geq \left(\frac{2}{3} + 2\epsilon\right) d - d_{G'_S}(v, w) = \left(\frac{2}{3} + 2\epsilon\right) d - d_{G'_S}(v, L_v)$$
$$> \left(\frac{2}{3} + \epsilon\right) d,$$

implying that $d_{G_S}(s, L_v) > (\frac{2}{3} + \epsilon)d$. Hence $d_{G_S}(s, L) \geq \min\{d_{G_S}(s, L_s), d_{G_S}(s, L_v)\} > \frac{2}{3} d$.
Showing part 1: Let $v \in V$ and consider any point during the sequence of updates. Assume first that v is reachable from s in G_S. Then $\mathcal{O}$ has not yet output v. For suppose otherwise. If v was unmarked when it was output by $\mathcal{O}$, the above procedure would separate v from s with S, making v unreachable from s in G_S. Conversely, if v was marked, v would already be unreachable from s in G_S since a vertex is only marked when it is separated from s in G_S. In both cases, we have a contradiction. It follows that $\mathcal{O}$ did not output v so $d_G(s, v) \leq \tilde{d}(s, v) < \frac{32}{33} d$, as desired.

Now, assume that v is not reachable from s in G_S. We may assume that there is a shortest path P from s to v in G since otherwise $d_G(s, v) = \infty > \frac{2}{3} d$. Let w be the first vertex of S along P. It suffices to show that for the prefix P' of P from s to w, $|P'| > \frac{2}{3} d$. At some earlier point in time, the procedure added w to S; just prior to this, P' was contained in G_S so from the above, $|P'| > \frac{2}{3} d$, as desired.

Showing part 2: Let $t_0 \leq t_1$ satisfy the second part of the lemma. Since $d_G^{(t_0)}(s, v) > d$ by assumption, the first part of the lemma implies that v is not reachable from s in $G_S^{(t_0)}$ and hence v is also not reachable from s in $G_S^{(t_1)}$.

Let P be a shortest path from s to v in $G^{(t_1)}$. From what we have just shown, P must intersect $S^{(t_0)}$. Let w be the first vertex of $S^{(t_0)}$ along P. Then clearly, $d_G^{(t_1)}(s, v) = d_G^{(t_1)}(s, w) + d_G^{(t_1)}(w, v)$. Since the vertex w' preceding w on P is reachable from s in $G_S^{(t_0)}$, the first part of the lemma implies that $d_G^{(t_0)}(s, w) \leq d_G^{(t_0)}(s, w') + 1 < \frac{32}{33} d + 1$ and $d_G^{(t_0)}(s, w) > \frac{2}{3} d$. The latter implies that $d_G^{(t_1)}(w, v) = d_G^{(t_1)}(s, v) - d_G^{(t_1)}(s, w) \leq \frac{34}{33} d - d_G^{(t_0)}(s, w) < \frac{34}{33} d - \frac{2}{3} d < d$, showing one of the two inequalities in the second part of the lemma.

We show the other inequality by contradiction so assume that $d_G^{(t_1)}(s, w) > d$. Then $d_G^{(t_1)}(s, w) \geq d + 1$ so by the above $d_G(s, w)$ would have increased by more than $d + 1 - (\frac{32}{33} d + 1) = \frac{1}{33} d$ from time step t_0 to t_1. Combining this with $d_G^{(t_1)}(s, v) = d_G^{(t_1)}(s, w) + d_G^{(t_1)}(w, v)$, $d_G^{(t_0)}(w, v) \leq d_G^{(t_1)}(w, v)$, and the triangle inequality, we get

$$d_G^{(t_1)}(s, v) - d_G^{(t_0)}(s, v) \geq d_G^{(t_1)}(s, w) + d_G^{(t_1)}(w, v) - (d_G^{(t_0)}(s, w) + d_G^{(t_0)}(w, v)) > \frac{1}{33} d$$

This contradicts the assumption $d < d_G^{(t_0)}(s,v) \leq d_G^{(t_1)}(s,v) \leq \frac{34}{33}d$. We conclude that $d_G^{(t_1)}(s,w) \leq d$ and $d_G^{(t_1)}(w,v) \leq d$ which shows the second part of the lemma.

Bounding $|S|$ and running time: To bound, $|S|$, consider the two parallel searches from s and from v, respectively, in some update. As argued earlier, there cannot be an edge or vertex visited by both searches. Let X resp. Y be the set of vertices visited by the BFS from s resp. v, excluding L_s resp. L_v and let $x = |X|$ and $y = |Y|$.

Assume first that $L = L_s$. Then all vertices in $Y \cup L_v$ become unreachable in G_S once L has been added to S. For each $w \in V$, $\hat{d}(w)/\Delta \geq 1$. Since each BFS spends $\hat{d}(w)$ time to process edges incident to w and since the two searches run in parallel, we have

$$|L| = O((x \log n)/d) = O\left(\frac{\log n}{d} \sum_{w \in X} \frac{\hat{d}(w)}{\Delta}\right) = O\left(\frac{\log n}{d} \sum_{w \in Y \cup L_v} \frac{\hat{d}(w)}{\Delta}\right)$$

Now, assume that $L = L_v$. Then all vertices of $Y \cup L_v$ become unreachable in G_S once L has been added to S so again,

$$|L| = O((y \log n/d) = O\left(\frac{\log n}{d} \sum_{w \in Y \cup L_v} \frac{\hat{d}(w)}{\Delta}\right)$$

In both cases, the cost $|L|$ of adding L to S can be paid for by charging each vertex w no longer reachable from s in G_S a cost of $O(\frac{\log n}{d}\hat{d}(w)/\Delta)$. Since a vertex is only charged once during the course of the algorithm, we get that for the final separator S (and hence for each intermediate separator),

$$\begin{aligned}|S| &= O\left(\frac{\log n}{d} \sum_{w \in V} \frac{\hat{d}(w)}{\Delta}\right) = O\left(\frac{\log n}{d} \sum_{w \in V} \frac{d(w) + \Delta}{\Delta}\right) = O\left(\frac{\log n(m + n\lceil m/n \rceil)}{d\lceil m/n \rceil}\right) \\ &= O\left(\frac{n \log n}{d}\right)\end{aligned}$$

where the last bound follows since we may assume that all vertices are initially reachable from s in G, implying $m \geq n - 1$ and hence $\lceil m/n \rceil = \Theta(m/n)$. This shows the desired bound on $|S|$.

The running time cost of any two parallel searches can be charged to the total degree of the vertices that get marked since they all become unreachable in G_S (this is the vertex set Y above). Since a marked vertex is never visited again by a BFS search, the total running time of parallel searches over all updates is $O(m)$, as desired. □

The lemma is somewhat technical and its full strength is only needed for the randomized data structure. For the deterministic data structures, the second part of the lemma will only be applied to the current time step $t_1 = t_0$ so it can be simplified to:

2. if $d < d_G(s,v) \leq \frac{34}{33}d$ then any shortest s-to-v path P in G intersects S and for the first such intersection vertex w along P, $d_G(s,w) \leq d$, and $d_G(w,v) \leq d$.

4 Deterministic Decremental APSP

In this section, we present our deterministic data structures for the exact resp. $(1+\epsilon)$-approximate decremental APSP problem and show Theorems 1 and 2. In the following, let $G = (V, E)$ denote the decremental graph.

4.1 Exact distances

Let $\rho = \frac{34}{33}$ and $D_i = \rho^i$ for $i = 0, \ldots, \lfloor \log_\rho n \rfloor$. For each i and each $u \in V$, we give a data structure $\mathcal{D}_i(u)$ which for any query vertex v maintains a value $\tilde{d}_i(u,v) \geq d_G(u,v)$ with equality if $d_G(u,v) \in (D_i, D_{i+1}]$. In each update, these data structures will be updated in order of increasing i.

Handling all-pairs shortest path distances up to at most $33 \lg n$ can be done in $O(mn \log n)$ time using the data structure of Even and Shiloach so we only consider i such that $D_i > 33 \lg n$. This allows us to apply Lemma 3. Consider such an i and assume that we already have data structures for all values smaller than i.

Data structure $\mathcal{D}_i(u)$ maintains a separator set $S_i(u)$ using an instance $\mathcal{S}_i(u)$ of the data structure of Lemma 3 with $s = u$ and $d = D_i$; inductively, we have an exact data structure for distances smaller than d and this data structure plays the role of $\mathcal{O}$ with $\tilde{d} = d_G$. At the beginning of each update, $\mathcal{S}_i(u)$ updates $S_i(u)$. Then for each v, if $\mathcal{O}$ reports that $\tilde{d}(u,v)$ has increased from a value of at most D_i to a value strictly greater than D_i, $\mathcal{D}_i(u)$ initializes $S_i(u,v)$ to be the current separator set $S_i(u)$; $\mathcal{D}_i(u)$ then initializes a priority queue $Q_i(u,v)$ where elements are all $s \in S_i(u,v)$ with corresponding keys $\tilde{d}_{i-1}(u,s) + \tilde{d}_{i-1}(s,v)$. During updates, whenever $\mathcal{D}_{i-1}(u)$ resp. $\mathcal{D}_{i-1}(s)$ reports that $\tilde{d}_{i-1}(u,s)$ resp. $\tilde{d}_{i-1}(s,v)$ increases, the key value of s in $Q_i(u,v)$ increases by the same amount. Note that after initialization, $S_i(u,v)$ remains fixed and so does $Q_i(u,v)$ (except for key value changes).

For each vertex v, $\mathcal{D}_i(u)$ maintains $\tilde{d}_i(u,v)$ as the min key value in $Q_i(u,v)$ after $Q_i(u,v)$ has been initialized; prior to this, $\tilde{d}_i(u,v) = \infty$. This completes the description of each $\mathcal{D}_i(u)$.

The overall data structure $\mathcal{D}$ maintains a priority queue $Q(u,v)$ for each vertex pair (u,v) with an element for each i of key value $\tilde{d}_i(u,v)$. For i in increasing order, $\mathcal{D}$ updates $\mathcal{D}_i(u)$ for each u. Whenever a data structure $\mathcal{D}_i(u)$ increases a value $\tilde{d}_i(u,v)$, the corresponding key in $Q(u,v)$ is increased accordingly. On a query (u,v), $\mathcal{D}$ reports the min key value in $Q(u,v)$.

Correctness: We prove that for each i, each time step t_1, and each vertex pair (u,v), $\tilde{d}_i(u,v) \geq d_G(u,v)$ with equality if $d_G(u,v) \in (D_i, D_{i+1}]$. The inequality is clear since every estimate corresponds to the length of some path in the current graph. The equality part is shown by induction on i.

The base cases where $D_i < 33 \lg n$ are clear so pick i with $D_i \geq 33 \lg n$ and $d_G^{(t_1)}(u,v) \in (D_i, D_{i+1}]$ and assume that correctness holds for all vertex pairs and time steps for $i-1$. Let $t_0 \leq t_1$ be the first time step such that $d_G^{(t_0)}(u,v) \in (D_i, D_{i+1}]$. Note that $S_i(u,v) = S_i(u)^{(t_0)}$. The induction hypothesis and the second part of Lemma 3 combined with the observation that no key value in $Q_i(u,v)$ is below $d_G(u,v)$, it follows that the min key value in $Q_i(u,v)$ equals $d_G^{(t_1)}(u,v)$. This shows correctness.

Running time: Consider an $i \in \{0, \ldots, \lfloor \log_\rho n \rfloor\}$ with $D_i \geq 33 \lg n$ and a vertex $u \in V$. We will show that maintaining $\mathcal{D}_i(u)$ takes $O(n^2 \log^2 n)$ time using a standard binary heap. Total time over all i and u will thus be $O(n^3 \log^3 n)$. This dominates the $O(n^3 \log^2 n)$ time to maintain priority queues $Q(u,v)$ and the $O(mn \log n)$ time for the data structure of Even and Shiloach for small values of i.

Maintaining $S_i(u)$ takes a total of $O(m)$ time by Lemma 3. The total number of elements

in priority queues $Q_i(u,v)$ over all $v \in S_i(u,v)$ is $O(n^2 \log n / D_i)$, again by Lemma 3. The number of increase-key operations for a single priority queue element s of $Q_i(u,v)$ is $O(D_i)$ which takes a total of $O(D_i \log n)$ time. Over all elements of priority queues $Q_i(u,v)$, this is $O(n^2 \log^2 n)$.

Reporting paths: It is easy to extend our data structure to efficiently answer queries for shortest paths (rather than only shortest path distances) between any vertex pair (u,v). Associated with the min element of $Q(u,v)$ is a vertex s such that for the associated index i, $\tilde{d}_i(u,v) = d_G(u,v) = d_G(u,s) + d_G(s,v)$, $\tilde{d}_{i-1}(u,s) = d_G(u,s)$, and $\tilde{d}_{i-1}(s,v) = d_G(s,v)$. Hence, by recursively querying for pairs (u,s) and (s,v), a shortest u-to-v path in G is reported in time proportional to its length.

4.2 Approximate distances

Let $\epsilon > 0$ be given. We now present our deterministic data structure for the $(1+\epsilon)$-approximate variant of the problem.

The data structure is quite similar to the one for the exact variant so we only describe the changes needed. For $i > 0$ and $u \in V$, we describe data structure $\mathcal{D}_i(u)$ and assume that we have data structures for values less than i. As before, we only consider i with $D_i \geq 33 \lg n$.

Let $\epsilon' > 0$ be a value depending on ϵ such that $(1+\epsilon')^c = \rho$ for some $c \in \mathbb{N}$; we will specify ϵ' later. For $j = 0, \ldots, c = \log_{1+\epsilon'} \rho$, let $d_{i,j} = D_i(1+\epsilon')^j$. This partitions each interval $(D_i, D_{i+1}]$ into c sub-intervals $(D_i(1+\epsilon')^j, D_i(1+\epsilon')^{j+1}]$ for $j = 0, \ldots, c-1$.

$\mathcal{D}_i(u)$ maintains $S_i(u)$ as in the exact version. For each $v \in V$, $\mathcal{D}_i(u)$ maintains an initially empty set $S_i(u,v)$. Once $\mathcal{D}_{i-1}(u)$ reports that $\tilde{d}_{i-1}(u,v)$ increased from a value of at most $D_i(1+\epsilon')^i$ to a value strictly greater than $D_i(1+\epsilon')^i$, $\mathcal{D}_i(u)$ sets $S_i(u,v)$ equal to the current set $S_i(u)$.

For each $j = 0, \ldots, c-1$, a data structure $\mathcal{D}_{i,j}(u)$ maintains the following set for each vertex v:

$$Q_{i,j}(u,v) = \left\{ s \in S_i(u,v) \mid \tilde{d}_{i-1}(u,s) + \tilde{d}_{i-1}(s,v) \leq (1+\epsilon')^i d_{i,j} \right\}.$$

$Q_{i,j}(u,v)$ is maintained by $\mathcal{D}_{i,j}(u)$ as a queue in which every $s \in Q_{i,j}(u,v)$ has key $\tilde{d}_{i-1}(u,s) + \tilde{d}_{i-1}(s,v)$ and where s is removed from $Q_{i,j}(u,v)$ (or increased to ∞) when this value exceeds $(1+\epsilon')^i d_{i,j}$.

For each vertex v, define $\tilde{d}_{i,j}(u,v) = (1+\epsilon')^i d_{i,j}$ if $Q_{i,j}(u,v)$ contains at least one element and otherwise $\tilde{d}_{i,j}(u,v) = \infty$.

Data structure $\mathcal{D}_i(u)$ maintains a min-priority queue $Q_i(u,v)$ for each vertex v with an element of key value $\tilde{d}_{i,j}(u,v)$ for each j. On query v, it outputs $\tilde{d}_i(u,v) = \min\{k, \tilde{d}_{i-1}(u,v)\}$ where k is the min-key value of this queue, i.e., $\tilde{d}_i(u,v) = \min\{\tilde{d}_{i-1}(u,v), \min_j \tilde{d}_{i,j}(u,v)\}$.

The overall data structure $\mathcal{D}$ works in the same manner as for the exact data structure.

Correctness: Consider any point during the sequence of edge deletions. We will show that for suitable choice of ϵ', the estimate $\tilde{d}(u,v)$ that $\mathcal{D}$ outputs satisfies $d_G(u,v) \leq \tilde{d}(u,v) \leq (1+\epsilon) d_G(u,v)$ for every vertex pair (u,v).

We first show that $d_G(u,v) \leq \tilde{d}(u,v)$. It suffices to prove by induction on $i \geq 0$ that $d_G(u,v) \leq \tilde{d}_i(u,v)$. The proof holds for small i such that $D_i < 33 \lg n$ since then we use the data structure of Even and Shiloach, implying $\tilde{d}_i(u,v) = d_G(u,v)$. Now, consider an i such that $D_i \geq 33 \lg n$ and assume that the claim holds for smaller values than i. We have $\tilde{d}_i(u,v) = \min\{\tilde{d}_{i-1}(u,v), \min_j \tilde{d}_{i,j}(u,v)\}$ and $\tilde{d}_{i,j}(u,v) \geq (1+\epsilon')^i d_{i,j} \geq \tilde{d}_{i-1}(u,s)+\tilde{d}_{i-1}(s,v)$ for each $s \in Q_{i,j}(u,v)$. Additionally, if $Q_{i,j}(v) = \emptyset$ then $\tilde{d}_{i,j}(u,v) = \infty$. The induction hypothesis now implies $\tilde{d}_i(u,v) \geq d_G(u,v)$, showing the induction step. Thus, $d_G(u,v) \leq \tilde{d}(u,v)$.

To show that $\tilde{d}(u,v) \leq (1+\epsilon) d_G(u,v)$, we prove by induction on $i \geq 0$ that during all updates and for all vertex pairs (u,v), if $d_G(u,v) \in (0, D_{i+1}]$ then $\tilde{d}_i(u,v) \leq (1+\epsilon')^i d_G(u,v)$. If we can show this then picking $\epsilon' \leq \ln(1+\epsilon)/(\lfloor \log_\rho n \rfloor)$ gives $\tilde{d}(u,v) \leq (1+\epsilon')^{\lfloor \log_\rho n \rfloor} d_G(u,v) \leq e^{\epsilon' \lfloor \log_\rho n \rfloor} d_G(u,v) \leq (1+\epsilon) d_G(u,v)$ for every vertex pair (u,v).

We only need to consider i with $D_i \geq 33 \lg n$ since otherwise, we use the data structure of Even and Shiloach. Assume inductively that the claim holds for values less than i.

Let t_1 be the current time step and consider a vertex pair (u,v) with $d_G^{(t_1)}(u,v) \in (0, D_{i+1}]$. By the induction hypothesis, we may assume that $d_G^{(t_1)}(u,v) \in (D_i, D_{i+1}]$. We may further assume that $\tilde{d}_{i-1}^{(t_1)}(u,v) > D_i(1+\epsilon')^i$ since otherwise,

$$\tilde{d}_i^{(t_1)}(u,v) \leq \tilde{d}_{i-1}^{(t_1)}(u,v) \leq D_i(1+\epsilon')^i < (1+\epsilon')^i d_G^{(t_1)}(u,v).$$

Let $t_0 \leq t_1$ be the first time step where $\tilde{d}_{i-1}^{(t_0)}(u,v) > D_i(1+\epsilon')^i$. We must have $d_G^{(t_0)}(u,v) > D_i$ since otherwise, the induction hypothesis would imply $\tilde{d}_{i-1}^{(t_0)}(u,v) \leq d_G^{(t_0)}(u,v)(1+\epsilon')^{i-1} \leq D_i(1+\epsilon')^{i-1}$, contradicting the choice of t_0. Since also $d_G^{(t_0)}(u,v) \leq d_G^{(t_1)}(u,v) \leq D_{i+1}$, Lemma 3 implies that there is a vertex $s \in S_i^{(t_0)}(u) = S_i^{(t_0)}(u,v) = S_i^{(t_1)}(u,v)$ such that $d_G^{(t_1)}(u,v) = d_G^{(t_1)}(u,s) + d_G^{(t_1)}(s,v)$, $d_G^{(t_1)}(u,s) \leq D_i$, and $d_G^{(t_1)}(s,v) \leq D_i$.

Pick j such that $d_G^{(t_1)}(u,v) \in (d_{i,j}, d_{i,j+1}]$. By the induction hypothesis,

$$\tilde{d}_{i-1}^{(t_1)}(u,s) + \tilde{d}_{i-1}^{(t_1)}(s,v) \leq (1+\epsilon')^{i-1} d_G^{(t_1)}(u,v) \leq (1+\epsilon')^{i-1} d_{i,j+1} = (1+\epsilon')^i d_{i,j}.$$

Hence, $Q_{i,j}(u,v)$ is non-empty at time step t_1 so $\tilde{d}_i^{(t_1)}(u,v) \leq \tilde{d}_{i,j}^{(t_1)}(u,v) = (1+\epsilon')^i d_{i,j} \leq (1+\epsilon')^i d_G^{(t_1)}(u,v)$. This shows the induction step.

Running time: The analysis is similar to the one for exact distances. Pick an $i \in \{0, \ldots, \lfloor \log_\rho n \rfloor\}$ with $D_i \geq 33 \lg n$. The total time to maintain $S_i(u)$ over all u is $O(mn)$.

Observe that each approximate distance $\tilde{d}_{i-1}(u_1,u_2)$ is of the form $(1+\epsilon')^{i'} d_{i',j}$ for $i' \leq i-1$. Since each element s in a queue $Q_{i,j}(u,v)$ has key value $\tilde{d}_{i-1}(u,s)+\tilde{d}_{i-1}(s,v)$, it follows that the number of increase-key operations applied to s in $Q_{i,j}(u,v)$ is $O(\log_{1+\epsilon'} D_i) = O(\log D_i/\epsilon') = O(\log n/\epsilon')$. For our purpose, a simplified queue $Q_{i,j}(u,v)$ suffices which keeps a counter of the number of elements of key value at most $(1+\epsilon')^i d_{i,j}$; this follows since the min key value is at most $(1+\epsilon')^i d_{i,j}$ if and only if the counter is strictly greater than 0. Every queue operation for $Q_{i,j}(u,v)$ can then be supported in $O(1)$ time. The number of elements in $Q_{i,j}(u,v)$ over all u, v, and j is $O(cn^3 \log n/D_i) = O(n^3 \log n/(D_i\epsilon'))$ by Lemma 3. This gives a total time bound of $O(mn + n^3 \log^2 n/(D_i(\epsilon')^2))$. This dominates the time spent on maintaining priority queues $Q_i(u,v)$.

Recall from above that $\epsilon' \leq \ln(1+\epsilon)/(\lfloor \log_\rho n \rfloor)$. The only additional constraint on ϵ' is that $(1+\epsilon')^c = \rho$ for some $c \in \mathbb{N}$. This can be achieved with $\epsilon' = \Theta(\ln(1+\epsilon)/(\lfloor \log_\rho n \rfloor))$. Hence, we get a time bound of $O(mn + n^3 \log^4 n/(D_i \epsilon^2))$.

Note that this bound is no better than the exact data structure for small D_i. We thus consider a hybrid data structure that only applies our data structure when D_i is above some distance threshold d and otherwise applies the data structure of Even and Shiloach which takes a total of $O(mnd)$ time. Summing over all $D_i > d$ and applying a geometric sums argument, the total time for our hybrid data structure is

$$O(mnd + \sum_{i:D_i>d} n^3 \log^4 n/(D_i\epsilon^2))) = O(mnd + n^3 \log^4 n/(d\epsilon^2)))$$

Setting $d = n \log^2 n/(\epsilon\sqrt{m})$ gives Theorem 2. Showing the bound for reporting approximate shortest paths in the theorem is done in the same way as in Section 4.1.

5 Randomized Decremental APSP

In this section, we provide a high-level overview of the randomized $(1+\epsilon)$-approximate data structure and analysis to achieve the result presented in Theorem 3. Building on this overview we will then prove the theorem.

Let us start by focusing on maintaining approximate distances close to the value D_i from a single vertex u and for now we assume an oblivious adversary.

Maintaining a sampled separator subset: Instead of maintaining each separator $S_{i,j}(u,v)$ (with associated with priority queue $Q_{i,j}(u,v)$) as the full vertex separator $S_i(u)$, we obtain a speed-up by only maintaining a sampled subset of $S_i(u)$. As long as this sampled subset certifies that there is a short two-hop path from u to v, the data structure proceeds as in the previous section. When this is no longer the case, there might still be a short two-hop path from u to v through a non-sampled vertex s in the full separator set $S_i(u)$. However, since there are no more sampled candidates, the expected number of vertices of $S_i(u)$ that provide a short two-hop path is small and we can update $S_{i,j}(u,v)$ to be this small subset. It follows that in expectation, $S_{i,j}(u,v)$ can be kept small at all times, which is needed to give a speed-up.

A speed-up using shallow in-trees: The problem with the data structure sketched above is that the entire set $S_i(u)$ had to be scanned in order to update $S_{i,j}(u,v)$ which means that the data structure will not be faster than our deterministic structure from the previous section. To deal with this, consider the following modification. The set $S_{i,j}(u,v)$ is updated as before by scanning over the entire set $S_i(u)$. Now, an in-tree $T(v)$ is grown from v of radius at most $\epsilon' D_i$. Each vertex v' in $T(v)$ then inherits the set of v, i.e., $S_{i,j}(u,v')$ is updated to the set $S_{i,j}(u,v)$ and this update is fast since $S_{i,j}(u,v)$ is small in expectation. This works since v is a proxy for v' in the sense that a short two-hop path from u to v' via $S_{i,j}(u,v)$ can be extended with a short suffix from $T(v)$, giving a short two-hop path from u to v via $S_{i,j}(u,v)$ (as $T(v)$ is an in-tree of small radius). Now, the time spent on the single scan of $S_i(u)$ can be distributed among all vertices of $T(v)$ and the number of such vertices must be at least $\epsilon' D_i + 1$ (if not, v would be within distance $\epsilon' D_i$ from u).

Unfortunately, the time analysis for the above procedure breaks down if the in-trees grown during the sequence of updates overlap too much. We now sketch how to deal with this. Mark vertices of each in-tree grown so far. When the BFS procedure grows a new in-tree $T(v)$, this procedure is modified by having it backtrack at previously marked vertices which thus become leaves of $T(v)$; this set of marked leaves will be referred to as L in the detailed description below.

Case 1, dealing with a large in-tree: If the number of unmarked vertices visited in $T(v)$ is greater than $\epsilon' D_i$, the above procedure and analysis can be applied; this is referred to as Case 1 in the detailed description below.

Case 2, dealing with a small in-tree: Otherwise, we are in Case 2; here we recall that $T(v)$ has small radius and observe that the only way to enter $T(v)$ from $G \setminus T(v)$ is through L. Hence, for every vertex s in the union $\cup_{v' \in L} S_{i,j}(u, v')$, there is a good two-hop path from u to v through s. But since we know that there is only a small number of such vertices left (in expectation), this union must be small. Furthermore, the union must contain a good separator for every vertex in $T(v)$ (again because $T(v)$ has small radius and because $T(v)$ must be entered through L) and we thus have an efficient way to update $S_{i,j}(u, w)$ for all $w \in T(v)$.

Handling an adaptive adversary: Above we assumed an oblivious adversary. When the adversary is adaptive, we need to be more careful since the approximate distances reported might reveal information about which vertices have been sampled. To deal with this, we round up every two-hop distance on a given distance scale to the same upper bound value (this will only increase the weight of each two-hop path by a small factor so that the output to a query will still be $(1+\epsilon)$-approximate). Hence, the rounded up approximate weight of a two-hop path $u \rightsquigarrow s \rightsquigarrow v$ is the same for every choice of "good" separator vertex s regardless of whether it was sampled or not. It follows that our randomized structure outputs the same distance estimates as a slower deterministic algorithm that maintains the full separator sets. Hence, our randomized algorithm works against an adaptive adversary, as desired.

5.1 The data structure

We now make the the overview formal. First, redefine $\rho = \frac{34-\frac{1}{2}}{33} = \frac{67}{66}$ and pick ϵ' such that $(1+\epsilon')^c = \rho$ for some $c \in \mathbb{N}$ and such that $\rho(1+\epsilon') \leq \frac{34}{33}$. For each u and i such that $D_i \geq 33 \lg n$, a separator $S_i(u)$ is maintained with a data structure $\mathcal{S}_i(u)$ as in Section 4.

We extend the range of index j by 1 so that $j \in \{0, \ldots, c+1\}$. Each structure $\mathcal{D}_{i,j}(u)$ maintains a growing set $M_{i,j}(u)$ of marked vertices; this set is initially empty. In the following, let $U_{i,j}(u) = V \setminus M_{i,j}(u)$ denote the set of unmarked vertices and let $G_{U_{i,j}(u)}$ denote the graph with vertex set V and containing the edges of G having at least one unmarked endpoint.

In each update, $\mathcal{D}_{i,j}(u)$ maintains $S_{i,j}(u, v)$ and $Q_{i,j}(u, v)$ for $v \in V$ in the following way.

For each $v \in V$ and every vertex s added to $S_i(u)$ in the current update, s is added to $S_{i,j}(u, v)$ with some probability p to be fixed later. Note that only vertices v for which s is actually added to $S_{i,j}(u, v)$ need to be processed. In the full version of the paper [5], we

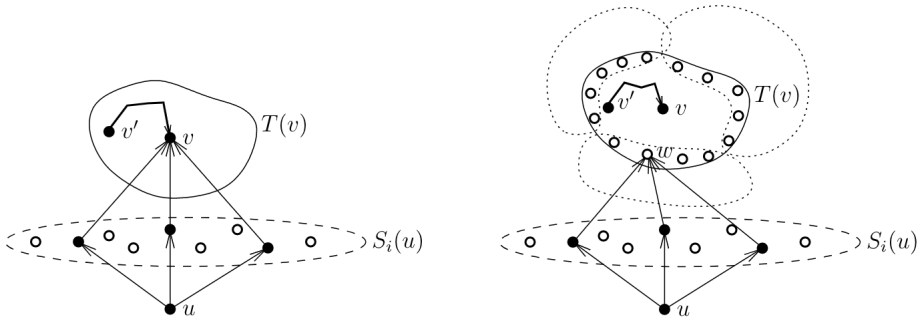


Figure 2: The two cases in the description of the randomized algorithm. Case 1: black vertices of $S_i(u)$ form the subset of vertices s with $\tilde{d}_{i-1}(u,s) + \tilde{d}_{i-1}(s,v) \leq d_{i,j}(1+\epsilon')^{2i}$. For each $v' \in V(T(v))$, $Q_{i,j}(u,v')$ is set to be this subset (with key values adjusted). 2-hop paths from u to v through the subset are shown. Case 2: Vertices of L are shown in white inside $T(v)$. Dotted regions are trees touching $T(v)$. For a $w \in L$, black vertices of $S_i(u)$ form the set $Q_{i,j}(u,w)$ and 2-hop paths from u to w through this set are shown. Q is the union of these sets over all $w \in L$. For each $v' \in V(T(v)) \setminus L$, $Q_{i,j}(u,v')$ is a subset of Q.

employ a different sampling scheme that avoids having to flip a coin for every vertex $v \in V$ in every update.

For vertices v such that $v \in M_{i,j}(u)$ or such that both $v \in U_{i,j}(u)$ and $\tilde{d}_{i-1}(u,v) \leq D_i(1+\epsilon')^{2i}$, no further processing is done.

Now, assume that $v \in U_{i,j}(u)$ and that $\tilde{d}_{i-1}(u,v) > D_i(1+\epsilon')^{2i}$. If this inequality did not hold in the previous update, each (sampled) vertex of $S_{i,j}(u,v)$ is added to a new min-queue $Q_{i,j}(u,v)$ with key values as in the previous section. Conversely, if the inequality did hold in the previous update, each new (sampled) vertex added to $S_{i,j}(u,v)$ in the current update is added to $Q_{i,j}(u,v)$.

If the min key value of $Q_{i,j}(u,v)$ is greater than $d_{i,j}(1+\epsilon')^{2i}$, $\mathcal{D}_{i,j}(u)$ grows an in-tree $T(v)$ from v in $G_{U_{i,j}(u)}$ up to radius $\epsilon' D_i$.

There are now two cases (see Figure 2): $|V(T(v)) \setminus M_{i,j}(u)| > \epsilon' D_i$ and $|V(T(v)) \setminus M_{i,j}(u)| \leq \epsilon' D_i$.

Case 1: If $|V(T(v)) \setminus M_{i,j}(u)| > \epsilon' D_i$ then $\mathcal{D}_{i,j}(u)$ scans once over $S_i(u)$ to find the subset of vertices $s \in S_i(u)$ for which $\tilde{d}_{i-1}(u,s) + \tilde{d}_{i-1}(s,v) \leq d_{i,j}(1+\epsilon')^{2i}$. For each $v' \in V(T(v))$, $Q_{i,j}(u,v')$ is set to contain exactly this subset of vertices s but with key value $\tilde{d}_{i-1}(u,s) + \tilde{d}_{i-1}(s,v')$.

Case 2: If $|V(T(v)) \setminus M_{i,j}(u)| \leq \epsilon' D_i$ then let $L = V(T(v)) \cap M_{i,j}(u)$ and let $Q = \cup_{v' \in L} Q_{i,j}(u,v')$. For each $v' \in V(T(v)) \setminus L$, $\mathcal{D}_{i,j}(u,v)$ sets $Q_{i,j}(u,v')$ to contain the elements $s \in Q$ with $\tilde{d}_{i-1}(u,s) + \tilde{d}_{i-1}(s,v) \leq d_{i,j}(1+\epsilon')^{2i}$; their key values are $\tilde{d}_{i-1}(u,s) + \tilde{d}_{i-1}(s,v')$.

In both cases, $\mathcal{D}_{i,j}(u)$ then marks all vertices of $T(v)$, i.e., $M_{i,j}(u) \leftarrow M_{i,j}(u) \cup V(T(v))$.

Approximate distances $\tilde{d}_{i,j}(u,v)$ are maintained by $\mathcal{D}_{i,j}(u)$ in a way similar to that in Section 4.2: $\tilde{d}_{i,j}(u,v) = (1+\epsilon')^{2i} d_{i,j}$ if the min key value of $Q_{i,j}(u,v)$ is at most $(1+\epsilon')^{2i} d_{i,j}$ and otherwise $\tilde{d}_{i,j}(u,v) = \infty$.

Data structures $\mathcal{D}_i(u)$ as well as the overall data structure $\mathcal{D}$ work exactly as in Section 4.2.

5.2 Correctness

Consider any point during the sequence of edge deletions. We will show that for suitable choice of ϵ', we have $d_G(u,v) \leq \tilde{d}(u,v) \leq (1+\epsilon) d_G(u,v)$.

We do this by proving that during all updates and for all vertex pairs (u,v), if $d_G(u,v) \in (0, D_{i+1}(1+\epsilon')]$ then $\tilde{d}_i(u,v) \leq (1+\epsilon')^{2i} d_G(u,v)$. By picking $\epsilon' = \ln(1+\epsilon)/(2\lfloor \log_\rho n \rfloor)$, this will give $d_G(u,v) \leq \tilde{d}_G(u,v) \leq (1+\epsilon')^{2\lfloor \log_\rho n \rfloor} d_G(u,v) \leq e^{2\lfloor \log_\rho n \rfloor \epsilon'} d_G(u,v) \leq (1+\epsilon) d_G(u,v)$, as desired.

The proof is by induction on i. The claim is clear for i with $D_i < 33 \lg n$ since then we use the data structure of Even and Shiloach. Now, consider an i with $D_i \geq 33 \lg n$ and assume that the claim holds for values less than i. By the induction hypothesis, we only need to consider pairs (u,v) with $d_G(u,v) \in (D_i(1+\epsilon'), D_{i+1}(1+\epsilon')]$, i.e., $d_G(u,v) \in (d_{i,j}, d_{i,j+1}]$ with $j > 0$.

We first show the following invariant for marked vertices that holds prior to each update over the entire sequence of updates:

Invariant 1. *At the end of each update, for every $w \in M_{i,j}(u)$ with $d_G(u,w) \in (d_{i,j}, d_{i,j+1}]$, each shortest u-to-w path in G intersects a vertex $s \in Q_{i,j}(u,w)$ such that $d_G(u,s) \leq D_i$ and $d_G(s,w) \leq D_i$.*

Proof. The invariant is shown by induction on the rank of w in the order in which vertices are marked. Note that this is a proof by induction inside a step of the main proof by induction on i; in addition to the induction hypothesis stated above, we may thus assume that the invariant holds for values less than i. Additionally, for the current value of i, we may assume by induction that the invariant holds for vertices of lower rank than w.

Let t_1 be a time step with $w \in M_{i,j}(u)^{(t_1)}$ and $d_G^{(t_1)}(u,w) \in (d_{i,j}, d_{i,j+1}]$, let $t_0 \leq t_1$ be the time step in which w was marked, and let r be the vertex from which an in-tree $T(r) \ni w$ was grown in time step t_0. Let P be a shortest u-to-w path in $G^{(t_1)}$.

We must have $\tilde{d}_{i-1}^{(t_0)}(u,r) > D_i(1+\epsilon')^{2i}$ since otherwise, no processing would be done for r in time step t_0, contradicting that $T(r)$ is grown in that time step. We also have $d_G^{(t_0)}(u,r) > D_i(1+\epsilon')$ since otherwise the induction hypothesis would give the contradiction $D_i(1+\epsilon') \geq d_G^{(t_0)}(u,r) \geq \tilde{d}_{i-1}^{(t_0)}(u,r)/(1+\epsilon')^{2(i-1)} > D_i(1+\epsilon')^{2i-2(i-1)} = D_i(1+\epsilon')^2$.

By the triangle inequality and the fact that $w \in T(r)$ and $T(r)$ has radius at most $\epsilon' D_i$, we get $d_G^{(t_0)}(u,w) \geq d_G^{(t_0)}(u,r) - d_G^{(t_0)}(w,r) > D_i(1+\epsilon') - \epsilon' D_i = D_i$. Hence, $D_i < d_G^{(t_0)}(u,w) \leq d_G^{(t_1)}(u,w) \leq d_{i,j+1} \leq \frac{34}{33} D_i$ so by Lemma 3, P intersects $S_i^{(t_0)}(u)$ and for the first such intersection vertex s along P, $d_G^{(t_0)}(u,s) \leq d_G^{(t_1)}(u,s) \leq D_i$ and $d_G^{(t_0)}(s,w) \leq d_G^{(t_1)}(s,w) \leq D_i$. We consider the two cases in the description of $\mathcal{D}_{i,j}(u)$ (see Figure 3): **Case 1:** It suffices to show that $s \in Q_{i,j}^{(t_1)}(u,w)$. We have $d_G^{(t_0)}(s,r) \leq d_G^{(t_0)}(s,w) + d_G^{(t_0)}(w,r) \leq$

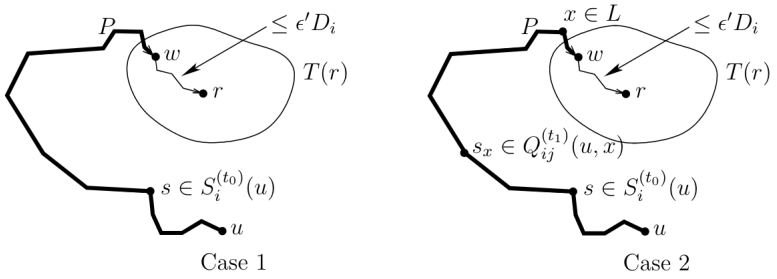


Figure 3: The two cases in the proof of Invariant 1. The path P is marked in bold.

$(1+\epsilon')D_i$. By the induction hypothesis,

$$\begin{aligned}\tilde{d}_{i-1}^{(t_0)}(u,s) + \tilde{d}_{i-1}^{(t_0)}(s,r) &\leq (1+\epsilon')^{2(i-1)} d_G^{(t_0)}(u,r) \\ &\leq (1+\epsilon')^{2i-2}(d_G^{(t_0)}(u,w) + \epsilon' D_i) \\ &\leq (1+\epsilon')^{2i-2}(d_G^{(t_1)}(u,w) + \epsilon' D_i) \\ &\leq (1+\epsilon')^{2i-2}(d_{i,j+1} + \epsilon' d_{i,j+1}) \\ &= (1+\epsilon')^{2i} d_{i,j},\end{aligned}$$

so $s \in Q_{i,j}^{(t_0)}(u,w) = Q_{i,j}^{(t_1)}(u,w)$, showing maintenance of the invariant.
Case 2: We first show that P must intersect the set L formed when growing $T(r)$ in time step t_0. Since we are in Case 2, every leaf of $T(r)$ either belongs to L or has no ingoing edges from vertices not in $T(r)$; otherwise, $T(r)$ would contain more than $\epsilon' D_i$ vertices since it is grown up to radius $\epsilon' D_i$. Hence, the only way that P could not intersect L would be if P were fully contained in $T(r)$. But this is not possible since then $T(r)$ would contain at least $|P| + 1 \geq d_{i,j} + 1 > D_i \geq \epsilon' D_i$ unmarked vertices at the beginning of time step t_0, contradicting that we are in Case 2.

Thus, P intersects L and we have $w \notin L$ since w was an unmarked vertex of $T(r)$ when growing this tree. Let x be the last vertex of P belonging to L. Since x was marked earlier than w, the induction hypothesis implies that the subpath of P from u to x intersects $Q_{i,j}^{(t_1)}(u,x) = Q_{i,j}^{(t_0)}(u,x)$ in a vertex s_x such that $d_G^{(t_0)}(u,s_x) \leq d_G^{(t_1)}(u,s_x) \leq D_i$ and $d_G^{(t_0)}(s_x,x) \leq d_G^{(t_1)}(s_x,x) \leq D_i$. The latter implies $d_G^{(t_0)}(s_x,r) \leq (1+\epsilon')D_i$. By the induction hypothesis, $\tilde{d}_{i-1}^{(t_0)}(u,s_x) + \tilde{d}_{i-1}^{(t_0)}(s_x,r) \leq (1+\epsilon')^{2(i-1)}(d_G^{(t_0)}(u,s_x) + d_G^{(t_0)}(s_x,r)) = (1+\epsilon')^{2(i-1)} d_G^{(t_0)}(u,r)$ which by the same calculations as in Case 1 is at most $(1+\epsilon')^{2i} d_{i,j}$. Inspecting the execution of $\mathcal{D}_{i,j}(u)$ in Case 2, it follows that $s_x \in Q_{i,j}^{(t_0)}(u,w) = Q_{i,j}^{(t_1)}(u,w)$. We have $s_x \in Q_{i,j}^{(t_0)}(u,x) \subseteq S_i^{(t_0)}(u)$. Since s is the first vertex of $S_i^{(t_0)}(u)$ along P, P can thus be decomposed into $u \rightsquigarrow s \rightsquigarrow s_x \rightsquigarrow x \rightsquigarrow w$ and we get $d_G^{(t_1)}(u,s_x) \leq D_i$ (as shown above) and $d_G^{(t_1)}(s_x,w) \leq d_G^{(t_1)}(s,w) \leq D_i$. This shows maintenance of the invariant with s_x in place of s. □

Now, we continue with our proof by induction on i. Consider any vertex pair (u,v) at the end of an update with $d_G(u,v) \in (d_{i,j}, d_{i,j+1}]$ and $j > 0$. If $v \notin M_{i,j}(u)$ and

$\tilde{d}_{i-1}(u,v) \leq (1+\epsilon')^{2i} D_i$ then $d_G(u,v) \leq \tilde{d}_{i,j}(u,v) \leq (1+\epsilon')^{2i} D_i < (1+\epsilon')^{2i} d_G(u,v)$, as desired.

Now assume that $v \notin M_{i,j}(u)$ and $\tilde{d}_{i-1}(u,v) > (1+\epsilon')^{2i} D_i$. Since v was not marked in the current update, the min key value of $Q_{i,j}(u,v)$ at the end of the update is at most $d_{i,j}(1+\epsilon')^{2i}$ so $d_G(u,v) \leq \tilde{d}_{i,j}(u,v) \leq (1+\epsilon')^{2i} d_{i,j} < (1+\epsilon')^{2i} d_G(u,v)$, as desired.

Finally assume that $v \in M_{i,j}(u)$. By Invariant 1, there is an $s \in Q_{i,j}(u,v)$ such that $d_G(u,v) = d_G(u,s) + d_G(s,v)$, $d_G(u,s) \leq D_i$, and $d_G(s,v) \leq D_i$. By the induction hypothesis, $d_G(u,v) \leq \tilde{d}_i(u,v) \leq \tilde{d}_{i-1}(u,s) + \tilde{d}_{i-1}(s,v) \leq (1+\epsilon')^{2(i-1)} d_G(u,v)$, as desired. This completes the inductive proof and correctness follows.

5.3 Running time

Maintaining separators $S_i(u)$ over all u and i takes $O(mn \log_\rho n) = O(mn \log n)$ time by Lemma 3. For the remaining time analysis, we focus on a single data structure $\mathcal{D}_{i,j}(u)$. It is useful in the following to regard this structure as handling an adversarial sequence of updates consisting of changes to approximate distances maintained by structures $\mathcal{D}_{i'}(v)$ for $i' < i$ and $v \in V$. We will give an expected time bound for $\mathcal{D}_{i,j}(u)$ and we shall rely on the following key lemma; the proof can be found in the full version of the paper [5].

Lemma 4. *Let $r \in V$. If at some point in the sequence of updates, $\mathcal{D}_{i,j}(u)$ grows an in-tree from r then at the end of that update, the expected number of vertices $s \in S_i(u)$ satisfying $\tilde{d}_{i-1}(u,s) + \tilde{d}_{i-1}(s,r) \leq D_i(1+\epsilon')^{2i}$ is $O(\ln n/p)$. This bound holds against an adaptive adversary.*

Corollary 1. *When a vertex v is marked, $E[|Q_{i,j}(u,v)|] = O(\ln n/p)$ and this bound holds against an adaptive adversary.*

Proof. Consider the update in which v is marked and let r be the root of the in-tree $T(r)$ containing v. If $|T(r)| \geq \epsilon' D_i$ then $Q_{i,j}(u,v) = Q_{i,j}(u,r) \subseteq S_i(u)$ and all $s \in Q_{i,j}(u,r)$ satisfy the inequality of Lemma 4. In the case where $|T(r)| < \epsilon' D_i$, let Q be as defined in the description of the data structure. Then vertices $s \in Q \subseteq S_i(u)$ are only added to $Q_{i,j}(u,v)$ if they satisfy the inequality of Lemma 4. The corollary now follows. □

Now, we can bound the time spent by $\mathcal{D}_{i,j}(u)$. The total time spent on growing in-trees is $O(m)$ since every edge (w_1, w_2) visited must have $w_1 \notin M_{i,j}(u)$ at the beginning of the BFS search and $w_1 \in M_{i,j}(u)$ immediately afterwards and a vertex can never be unmarked. This also bounds the time spent on marking vertices.

The total expected number of sampled vertices added to $Q_{i,j}(u,v)$ prior to v being marked is at most xp where x is the size of the set $S_i(u)$ after the final update. By Lemma 3, $x = O(n \log n/D_i)$. By Corollary 1, the expected size of $Q_{i,j}(u,v)$ after v is marked is $O(\ln n/p)$. Using the same argument as in the running time analysis of Section 4.2, the number of increase-key operations applied to a single element of $Q_{i,j}(u,v)$ is $O(\log n/\epsilon')$. Hence, the total expected time spent on operations on $Q_{i,j}(u,v)$ is $O((n \log n \cdot p/D_i + \log n/p) \log^3 n/\epsilon)$.

Whenever $\mathcal{D}_{i,j}(u)$ grows an in-tree $T(r)$ with $|V(T(r)) \setminus M_{i,j}(u,v)| > \epsilon' D_i$, scanning $S_i(u)$ takes $O(n \log n/D_i)$ time by Lemma 3. Since all vertices of $V(T(r)) \setminus M_{i,j}(u,v)$ are marked just after $T(r)$ is grown and since vertices are never unmarked, the number of such

trees over the course of the updates is at most $n/(\epsilon' D_i)$ so the total time for all these scans is $O(n^2 \log^2 n/(\epsilon D_i^2))$.

Whenever $\mathcal{D}_{i,j}(u)$ grows an in-tree $T(r)$ with $|V(T(r)) \setminus M_{i,j}(u,v)| \leq \epsilon' D_i$, the set $\cup_{x \in L} Q_{i,j}(u,x)$ needs to be computed. Note that for each $x \in L$, $E[|Q_{i,j}(u,x)|] = O(\log n/p)$ by Corollary 1. At least one edge (y,x) ingoing to x belongs to $T(r)$ and this edge is not part of any later grown in-tree since x is marked immediately after $T(r)$ is grown. We charge a cost of $O(\log n/p)$ to (y,x) for computing $Q_{i,j}(u,x)$. Over all $x \in L$, this pays for computing $\cup_{x \in L} Q_{i,j}(u,x)$ and we get a total expected time bound for this part of $O(m \log n/p)$.

Summing the above over all u, v, i, and j, we get a total expected time bound for our data structure of

$$\tilde{O}(mn/\epsilon + \sum_i \sum_j (n^3 \cdot p/(D_i \epsilon) + n^2/(p\epsilon) + n^3/(\epsilon D_i^2) + mn/p).$$

Since this bound is only fast for sufficiently large i, we pick a distance threshold d and apply the algorithm of Even and Shiloach for distances of at most d and our data structure for distances above d. By a geometric sums argument, our hybrid algorithm has a expected total time bound of

$$\begin{aligned}&\tilde{O}(mnd + mn/\epsilon + n^3 \cdot p/(d\epsilon^2) + n^2/(p\epsilon^2) + n^3/(\epsilon^2 d^2) + mn/(p\epsilon))\\ &= \tilde{O}(mnd + n^3 \cdot p/(d\epsilon^2) + n^2/(p\epsilon^2) + n^3/(d^2\epsilon^2) + mn/(p\epsilon))\end{aligned}$$

Setting the second and fifth terms equal to each other, we get $p = \tilde{\Theta}(\sqrt{m\epsilon d}/n)$ and the time bound simplifies to

$$\tilde{O}(mnd + \sqrt{m} n^2/(\sqrt{d}\epsilon^{3/2}) + n^3/(\sqrt{md}\epsilon^{5/2}) + n^3/(d^2\epsilon^2)).$$

We balance the first two terms by setting $d = \tilde{\Theta}(n^{2/3}/(m^{1/3}\epsilon))$ and we get a time bound of

$$\tilde{O}(m^{2/3} n^{5/3} n/\epsilon + n^{8/3}/(m^{1/3}\epsilon^2)),$$

which shows the time bound of Theorem 3.

Acknowledgements

We thank anonymous reviewers for their comments and remarks that helped improve the presentation of the paper.

References

[1] Surender Baswana, Ramesh Hariharan, and Sandeep Sen. Improved decremental algorithms for maintaining transitive closure and all-pairs shortest paths. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 117–123. ACM, 2002.

[2] Aaron Bernstein. Maintaining shortest paths under deletions in weighted directed graphs. *SIAM Journal on Computing*, 45(2):548–574, 2016.

[3] Camil Demetrescu and Giuseppe F Italiano. A new approach to dynamic all pairs shortest paths. *Journal of the ACM (JACM)*, 51(6):968–992, 2004.

[4] Camil Demetrescu and Giuseppe F Italiano. Fully dynamic all pairs shortest paths with real edge weights. *Journal of Computer and System Sciences*, 72(5):813–837, 2006.

[5] Jacob Evald, Viktor Fredslund-Hansen, Maximilian Probst Gutenberg, and Christian Wulff-Nilsen. Decremental APSP in directed graphs versus an adaptive adversary. *CoRR*, abs/2010.00937, 2020. URL: `https://arxiv.org/abs/2010.00937`, http://arxiv.org/abs/2010.00937 `arXiv:2010.00937`.

[6] Shimon Even and Yossi Shiloach. An on-line edge-deletion problem. *Journal of the ACM (JACM)*, 28(1):1–4, 1981.

[7] M.R. Henzinger and Valerie King. Fully dynamic biconnectivity and transitive closure. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 664–672. IEEE Comput. Soc. Press, 1995. URL: `http://ieeexplore.ieee.org/document/492668/`, https://doi.org/10.1109/SFCS.1995.492668 `doi:10.1109/SFCS.1995.492668`.

[8] Adam Karczmarz and Jakub Lacki. Reliable hubs for partially-dynamic all-pairs shortest paths in directed graphs. In *27th Annual European Symposium on Algorithms (ESA 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[9] Adam Karczmarz and Jakub Łącki. Simple label-correcting algorithms for partially dynamic approximate shortest paths in directed graphs. In *Symposium on Simplicity in Algorithms*, pages 106–120. SIAM, 2020.

[10] Valerie King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 81–89. IEEE, 1999.

[11] Mikkel Thorup. Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles. In *Scandinavian Workshop on Algorithm Theory*, pages 384–396. Springer, 2004.

Appendix D

Appendix D starts from the next page, and includes the unpublished manuscript "Degree of Convexity and Expected Distances in a Simple Polygon".

Degree of Convexity and Expected Distances in Simple Polygons

Mikkel Abrahamsen Viktor Fredslund-Hansen

June 2022

Abstract

We present a simple algorithm for computing the so-called *Beer-index* of a simple polygon P in $O(n^2 \log n)$ time, where n is the number of corners of P. The Beer-index is the probability that two points chosen independently and uniformly at random in P can see each other. Given a finite set M of m points in P, we also show how the number of pairs in M that see each other can be computed in $O(n \log n + m \log^2 n + m \log n \log m)$ time, which is close to optimal.

We likewise study the problem of computing the expected geodesic distance between two random points in P. We show how the expected L_1-distance can be computed in optimal $O(n)$ time by a conceptually very simple algorithm. We then describe an algorithm that outputs an expression for the expected L_2-distance in $O(n^2 \log^2 n)$ time.

1 Introduction

In this paper, we consider a function defined for pairs of points in a simple polygon P, and we want to compute the expected value of the function when choosing two points uniformly and independently at random from some set $M \subset P$. Here, M can either be a given finite set of points in P or all points in P, i.e., $M = P$. For a pair of points $p, q \in P$, the functions we will consider are:

- the indicator $[pq \subset P] \in \{0, 1\}$, i.e., whether p and q can see each other in P, and
- the length of the geodesic shortest path from p to q in P in the L_1- or L_2-metric.

Beer-index. With $M = P$ and the first function $[pq \subset P]$, we obtain the probability $B(P)$ that two points p, q, chosen uniformly and independently at random in P, see each other,

$$B(P) := \frac{1}{|P|^2} \int_{p \in P} \int_{q \in P} [pq \subset P] \, \mathrm{d}q \, \mathrm{d}p,$$

where $|\cdot|$ denotes the area. Throughout the paper, our integrals are defined with respect to Lebesgue measure.

One of the well-known characterizations of a polygon P being convex is that all pairs of points in P can see each other. Thus, the number $B(P) \in [0, 1]$ quantifies the degree to which this holds, so it is a natural measure for the degree of convexity of P. The problem of partitioning a polygon into components that are close to convex is useful in many practical settings, as such partitions provide similar benefits as convex partitions, while the number of components can be significantly smaller when the components are allowed to be slightly non-convex [12, 19]. This motivates ways to quatify the degree of convexity and algorithms for computing these measures. Stern [25] was the first to introduced the measure $B(P)$, using the equivalent form

$$B(P) = \frac{1}{|P|^2} \int_{p \in P} |\mathrm{vis}_P(p)| \, \mathrm{d}p,$$

where $\mathrm{vis}_P(p)$ is the *visibility polygon* of the point p, i.e., the region of points in P that p can see. Stern suggested that $B(P)$ be estimated by measuring $|\mathrm{vis}_P(p)|$ for every grid point p in P on a rectangular grid.

Rote [23] showed that $B(P)$ can be expressed as a sum of $O(n^9)$ closed-form expressions. Here, P is a polygon which may have holes, and n is the number of corners. Using a bound from [17], Rote later observed that the number is in fact bounded by $O(n^7)$.

Buchin, Kostitsyna, Löffler and Silveira [5] described an algorithm that outputs $B(P)$, again for a given polygon P which may have holes. They claimed the running time of the algorithm to be $O(n^2)$. Unfortunately, as the algorithm is described in the paper [5], it may output $B(P)$ as a sum of $\Omega(n^4)$ closed-form expressions and thus likewise have a running time of $\Omega(n^4)$. In Appendix A, we show that even when P is a simple polygon, i.e., when P has no holes, the running time of the algorithm may be $\Omega(n^4)$. We have made the authors aware of this mistake, and they have informally described a different algorithm to us in private communication which seems to have a running time of $O(n^2 \log n)$. However, at the time of writing, the algorithm has apparently not been written down in detail.

For a given simple polygon P with n corners, we describe an algorithm outputting $B(P)$ as a sum of $O(n^2)$ closed-form expressions using $O(n^2 \log n)$ time. The algorithm is a divide-and-conquer algorithm, and the main idea is to split P along a diagonal uv into two parts P_1 and P_2 with roughly equally many corners. We compute the contribution to $B(P)$ from pairs of points that see each other across the diagonal uv and then recursively add the contributions of pairs of points that are either both in P_1 or both in P_2. The algorithm is arguably simpler than the algorithm from [5] and presumably also the unpublished, faster algorithm by the same authors, both of which rely on geometric duality, whereas our algorithm deals exclusively with primitive objects in the usual primal space.

Theorem 1. *Given a simple polygon P with n corners, there is an algorithm that returns the Beer-index $B(P)$ as a sum of $O(n^2)$ closed-form expressions. The algorithm runs in time $O(n^2 \log n)$ and uses $O(n^2)$ space.*

Counting visible pairs of points. Very recently, Buchin, Custers, van der Hoog, Löffler, Popov, Roeloffzen and Staals [4] studied the problem of computing the number of visible pairs among m points in a simple polygon P with n corners. If m is not much larger than n, they suggest to use a data structure by Hershberger and Suri [15] to test if each pair is visible in $O(\log n)$ time. This results in an algorithm with running time $O(n + m^2 \log n)$. For the case that m is large, they describe algorithms with running times $O(nm^{3/2} + m^{3/2} \log m)$ and $O(n + m^{3/2+\varepsilon} \log n \log m)$ for any $\varepsilon > 0$, respectively. We present an algorithm for counting the number of visible pairs among m points in $O(n \log n + m \log^2 n + m \log n \log m)$ time, so it is superior to the algorithms from [4] in all cases.

Theorem 2. *Given a simple polygon P with n corners and a set M of m point in P, the number of pairs of points in M that can see each other can be computed in $O(n \log n + m \log^2 n + m \log n \log m)$ time.*

Finding the edges of the visibility graph of a given polygon is one of the classical problems in computational geometry. Lee [18] described a simple and well-known algorithm with running time $O(n^2 \log n)$ already in 1979, where n is the number of corners. The algorithm works by performing a rotational sweep around all corners. The same algorithm was described by Sharir and Schorr [24], and it also appears in the book [2]. Asano, Asano, Guibas, Hershberger and Imai [1] and Welzl [26] gave algorithms with running time $O(n^2)$. Hershberger [14] gave an output-sensitive algorithm using $O(k)$ time to compute the visibility graph of a triangulated polygon, where k is the number of edges in the visibility graph. Together with Chazelle's algorithm for triangulating a polygon in $O(n)$ time, this yields an optimal algorithm with time $O(n + k)$. When the set M of points in P are the corners of P, our algorithm from Theorem 2 returns the *size* of the visibility graph of a simple polygon in $O(n \log^2 n)$ time. To the best of our knowledge, this is the first algorithm that counts the number of visibility edges faster than the size of the graph. Note that the number $k/\binom{n}{2} \in [0, 1]$ can be considered a discrete variant of the Beer-index.

Corollary 3. *Given a simple polygon P, the number k of edges in the visibility graph of P can be computed in $O(n \log^2 n)$ time.*

Problem	Time	Space	Reference
Beer-index	$O(n^2 \log n)$	$O(n^2)$	Section 3
L_1-distance	$O(n)$	$O(n)$	Section 2
L_2-distance	$O(n^2 \log^2 n)$	$O(n^2)$	Section 5

Table 1: Our results for computing expected distances and the Beer index in a simple polygon.

Expected distances. The problem of determining the expected distance between two points picked independently and uniformly at random from a given domain has a long history. Czuber's book from 1884 [10] contains calculations of the values for equilateral triangles, squares and rectangles. Bäsel [6] recently derived formulas for the expected distance, as well as higher moments, in regular n-gons for $n = 3, 4, 5, 6, 8, 10, 12$. The paper likewise contains more historical information about these problems. In another recent paper, Bonnet, Gusakova, Thäle and Zaporozhets [3] proved that the expected distance between two points in a convex body in the plane with perimeter 1 is between $7/60$ and $1/6$, and that these bounds are tight. They also provide bounds for higher dimensional convex bodies.

As a warmup for computing the expected distances in simple polygons, we consider the conceptually simpler problem of computing the sum of L_1-distances between all pairs of points from a finite set $M \subset \mathbb{R}^d$, i.e., with no polygon involved. We give a very simple algorithm computing the sum in $O(d \cdot n \log n)$ time.

Theorem 4. *Given a set M of n points in $\mathbb{R}^d$, the sum of pairwise distances in the L_1-metric between points in M can be computed in $O(d \cdot n \log n)$ time.*

Hsu [16] studied the algorithmic problems of computing the expected distance between two random points in a given polygon. He gave a $O(n^2)$ time algorithm for the expected geodesic L_1-distance in a simple polygon and a $O(n^3)$ time algorithm for the expected L_2-distance in a *convex* polygon. In this paper, we describe a very simple algorithm for computing the expected geodesic L_1-distance in $O(n)$ time and the L_2-distance in $O(n^2 \log^2 n)$ time in a simple polygon. To the best of our knowledge, no algorithm has been described before for computing the expected L_2-distance in simple polygons in general. Table 1 summarizes our results on computing the Beer- and Wiener-index of a simple polygon.

Theorem 5. *Given a simple polygon P with n corners, there is an algorithm for computing the expected geodesic L_1-distance between two random points in P using $O(n)$ time and space.*

Theorem 6. *Let P be a simple polygon. There is an algorithm which outputs an expression representing the expected geodesic L_2 distance of P in $O(n^2 \log^2 n)$ time using $O(n^2)$ space. Each term of the expression is a simple integral of constant size. The number of terms in the expression is $O(n^2)$.*

This characterization can then be used to approximate the expected geodesic L_2-distance. This relies simply on approximating the certain kind of "simple" integrals that correspond to each term in the expression of Theorem 6. It is unknown to the authors whether the terms admit a closed solution, however. This is captured in the following corollary:

Corollary 7. *Let P be a simple polygon and assume that there is a procedure O that d-approximates simple integrals in time $O(t(d))$ pr. integral. Then there is an algorithm which d-approximates the expected geodesic L_2-distance in time $O(n^2 \log^2 n \cdot t(d))$.*

More attention has been given to the problem of computing the average distance between two vertices in a graph G. This known as the *Wiener-index* of G, and it is a fundamental measure with important applications in mathematical chemestry and appears in thousands of publications. Note that the Wiener-index is equivalent to the sum of pairwise distances. For more information on the problem, see the papers [7, 11].

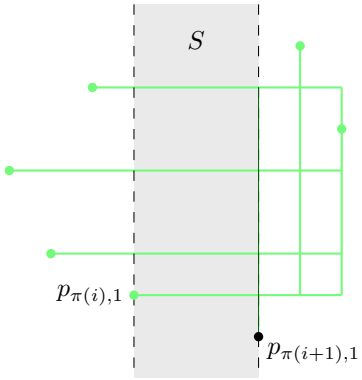


Figure 1: The number of shortest paths that cross the slab S is $(n-i)\cdot i$.

2 Expected distances in the L_1-metric

2.1 Finite set of points in $\mathbb{R}^d$

As a warmup, consider n points $M = \{p_1, p_2, \ldots, p_n\} \subseteq \mathbb{R}^d$. Let π_k be a permutation of $\langle 1, 2, \cdots, n\rangle$ satisfying $p_{\pi_k(i),k} \leq p_{\pi_k(i+1),k}$ for all $1 \leq i < n$, where $p_{i,k}$ denotes the kth coordinate of p_i. Then it is easily seen that for $1 \leq i \leq n$, we have

$$\sum_{j=1}^{i-1} |p_{\pi(i),k} - p_{\pi(j),k}| = \sum_{j=1}^{i-1} j \cdot (p_{\pi(j+1),k} - p_{\pi(j),k}),$$

and it follows that

$$\begin{aligned}\sum_{i=1}^{n}\sum_{j=1}^{i-1} |p_{\pi(i),k} - p_{\pi(j),k}| &= \sum_{i=1}^{n}\sum_{j=1}^{i-1} j \cdot (p_{\pi(j+1),k} - p_{\pi(j),k}) \\ &= \sum_{i=1}^{n-1} (n-i)\cdot i \cdot (p_{\pi(i+1),k} - p_{\pi(i),k}). \end{aligned} \tag{1}$$

In the planar case $d = 2$, this formula has the following geometric interpretation. Let us consider the slab S between the vertical lines through $p_{\pi(i),1}$ and $p_{\pi(i+1),1}$ and count how many pairs of points are separated by S; see Figure 1. As there are i points to the left of S and $n-i$ points to the right, there are $(n-i)\cdot i$ such pairs, and the shortest path between all these pairs cross S. Summing over all values of i, we get the total difference of x-coordinates over all pairs of points. This can easily be generalized to an arbitrary dimension d.

By definition of the L_1 norm, and by replacing the sum of (2) with (1) in the following, we get

$$\begin{aligned}\sum_{i=1}^{n}\sum_{j=1}^{i-1} ||p_i - p_j||_1 &= \sum_{i=1}^{n}\sum_{j=1}^{i-1}\sum_{k=1}^{d} |p_{i,k} - p_{j,k}| \\ &= \sum_{k=1}^{d}\sum_{i=1}^{n}\sum_{j=1}^{i-1} |p_{i,k} - p_{j,k}| \\ &= \sum_{k=1}^{d}\sum_{i=1}^{n-1} (n-i)\cdot i \cdot (p_{\pi(i+1),k} - p_{\pi(i),k}). \end{aligned} \tag{2}$$

The above implies that the sum of all pairwise distances may be computed in time $O(d \cdot n \log n)$ by sorting the coordinates according to their kth coordinate and evaluating the inner sum for each k, which establishes Theorem 4.

2.2 Expected L_1-distance in a polygon

We are given a simple polygon P with n corners. Let us assume for simplicity that no two corners of P have the same x-coordinates or the same y-coordinates. Note that shortest paths in the L_1-metric can be realised as a union of horizontal and vertical segments. For points $a, b \in P$, denote by $a \rightsquigarrow b$ such a shortest path from a to b. As in our warmup-case of points in $\mathbb{R}^d$ from Section 2, we find the expected distance by computing the contribution from horizontal and vertical segments separately.

Let $\Delta_1(p)$ (resp. $\Delta_2(p)$) be the segments of a path p that are horizontal (resp. vertical) and define $\Delta_i^+(a,b) = \sum_{s \in \Delta_i(a \rightsquigarrow b)} \|s\|_1$. Define for $A, B \subset P$ the notation

$$D_i(A,B) = \int_{(p,p') \in A \times B} \Delta_i^+(p,p') \, \mathrm{d}(p,p').$$

We now observe

$$\begin{aligned} \int_{(p,p') \in P^2} |p \rightsquigarrow p'| \, \mathrm{d}(p,p') &= \int_{(p,p') \in P^2} \left(\Delta_1^+(p,p') + \Delta_2^+(p,p') \right) \mathrm{d}(p,p') \\ &= D_1(P,P) + D_2(P,P). \end{aligned}$$

We explain how to compute $D_1(P,P)$ in $O(n)$ time, and $D_2(P,P)$ can be computed in an analogous way, leading to an algorithm for computing the expected L_1-distance with linear running time. We first construct a vertical trapezoidation of P: For each corner c of P, we add the maximal vertical segment contained in P and containing c; see Figure 2. These vertical segments partition P into trapezoids, each which has a pair of vertical edges. Some trapezoids degenerate into triangles. The trapezoids induce a tree T, where the vertices are the trapezoids and two vertices are connected by an edge if the trapezoids share a vertical edge. We choose an arbitrary root r of T, which induce parent-child relationships among neighbouring pairs of trapezoids in T. Define $p(t)$ to be the parent of t and $c(t)$ to be the set of children of t.

For each trapezoid t which is not the root, let $l(t)$ be vertical edge of t that separates t from the parent $p(t)$. Furthermore, let $P[t] \subset P$ be the region consisting of t and all decendants of t. Define

$$L_i(t) = \int_{p \in P[t]} \Delta_i^+(p, l(t)) \, \mathrm{d}p.$$

We now show how we can compute the numbers $D_1(P[t],P[t])$, $L_1(t)$ and $|P[t]|$ for all trapezoids t in $O(n)$ time in total. We have then in particular computed $D_1(P,P) = D_1(P[r],P[r])$.

It is trivial to evaluate each of the numbers $D_1(P[t],P[t])$, $L_1(t)$ and $|P[t]| = |t|$ for a leaf t of T in $O(1)$ time, since t has complexity $O(1)$, so suppose now that t is not a leaf.

By our general position assumption, a trapezoid t has at most four children. Assuming that the values $D_1(P[t'],P[t'])$, $L_1(t')$ and $|P[t']|$ have been computed for each of the children t' of t, we show how $D_1(P[t],P[t])$, $L_1(t)$ and $|P[t]|$ can be computed in $O(1)$ time. The claim that we can compute the numbers for all t in $O(n)$ time then follows.

The area $|P[t]|$ can be simply computed as

$$|P[t]| = |t| + \sum_{t' \in c(t)} |P[t']|.$$

We now explain how to exaluate $L_1(t)$, so suppose that $t \neq r$. We have

$$L_1(t) = \int_{p \in t} \Delta_1^+(p, l(t)) \, \mathrm{d}p + \sum_{t' \in c(t)} \int_{p \in P[t']} \Delta_1^+(p, l(t)) \, \mathrm{d}p.$$

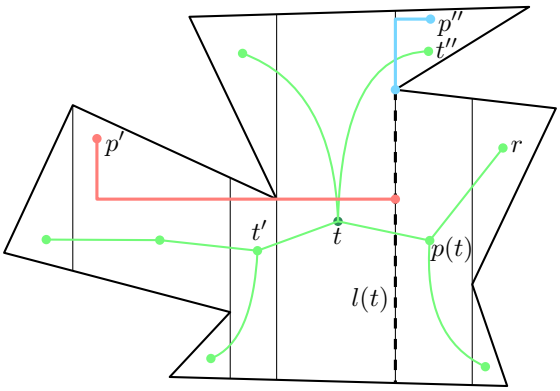


Figure 2: A vertical trapezoidation of a simple polygon P and the tree it induces. Case (i) and (ii) in the computation of $L_1(t)$ are shown as shortest paths from points $p' \in P[t']$ and $p'' \in P[t'']$ for children t' and t'' of t, respectively, to the segment $l(t)$.

Clearly, the first term can be evaluated in $O(1)$ time, as the trapezoid t has complexity $O(1)$. There are two cases when evaluating the integral in the sum in the second term: (i) t' and $p(t)$ are on the same side of t, and (ii) t' and $p(t)$ are on different sides of t; see Figure 2. In case (i), the shortest path from from a point $p \in P[t']$ to $l(t)$ first follow a shortest path to $l(t')$ and then follow a vertical segment to $l(t)$, so we have

$$\int_{p \in P[t']} \Delta_1^+(p, l(t))\,\mathrm{d}p = \int_{p \in P[t']} \Delta_1^+(p, l(t'))\,\mathrm{d}p = L_1(t').$$

In case (ii), we have

$$\int_{p \in P[t']} \Delta_1^+(p, l(t))\,\mathrm{d}p = \int_{p \in P[t']} (\Delta_1^+(p, l(t')) + w(t))\,\mathrm{d}p = L_1(t') + |P[t']|\,w(t),$$

where $w(t)$ is the width of t.

In order to evaluate $D_1(P[t], P[t])$, we observe

$$D_1(P[t], P[t]) = D_1(t, t) + \sum_{t' \in c(t)} D_1(P[t'], t) + \sum_{t', t'' \in c(t)} D_1(P[t'], P[t'']).$$

Again, the first term takes $O(1)$ to evaluate as t has complexity $O(1)$. To evaluate the sum in the second term, we observe

$$\begin{aligned} D_1(P[t'], t) &= \int_{(p', p) \in P[t'] \times t} \Delta_1^+(p', p)\,\mathrm{d}(p', p) \\ &= \int_{(p', p) \in P[t'] \times t} (\Delta_1^+(p', l(t')) + \Delta_1^+(p, l(t')))\,\mathrm{d}(p', p) \\ &= L_1(t')\,|t| + |P[t']| \int_{p \in t} \Delta_1^+(p, l(t'))\,\mathrm{d}p. \end{aligned}$$

Again, the integral in the final expression can be evaluated in $O(1)$ time, so the same holds for $D_1(P[t'], t)$.

Consider now a term of the form $D_1(P[t'], P[t''])$. If $t' = t''$, the number $D_1(P[t'], P[t''])$ has already been computed, so suppose $t' \neq t''$. We have two cases: (i) t' and t'' are on the same side of t and (ii) t' and

t'' are on different sides. In case (i), we get

$$\begin{aligned} D_1(P[t'], P[t'']) &= \int_{(p',p'')\in P[t']\times P[t'']} \Delta_1^+(p', p'')\,\mathrm{d}(p', p'') \\ &= \int_{(p',p'')\in P[t']\times P[t'']} (\Delta_1^+(p', l(t')) + \Delta_1^+(p'', l(t'')))\,\mathrm{d}(p', p'') \\ &= |P[t']|\,L_1(t'') + L_1(t')\,|P[t'']|\,. \end{aligned}$$

In case (ii), we get

$$\begin{aligned} D_1(P[t'], P[t'']) &= \int_{(p',p'')\in P[t']\times P[t'']} \Delta_1^+(p', p'')\,\mathrm{d}(p', p'') \\ &= \int_{(p',p'')\in P[t']\times P[t'']} (\Delta_1^+(p', l(t')) + w(t) + \Delta_1^+(p'', l(t'')))\,\mathrm{d}(p', p'') \\ &= |P[t']|\,L_1(t'') + L_1(t')\,|P[t'']| + |P[t']|\,|P[t'']|\,w(t). \end{aligned}$$

Since a trapezoidation of P can be computed in $O(n)$ time using Chazelle's algorithm [8], we get Theorem 5. Instead of Chazelle's algorithm, which is known for being complicated, we can also use a simpler sweep-line algorithm [2] and obtain an algorithm running in time $O(n \log n)$.

3 Beer-index

Our algorithms for computing Beer and Wiener indices in a simple polygon P are divide-and-conquer algorithms. Assume without loss of generality that P has area 1 and let n be the number of corners of P. We first triangulate P, for instance using the simple algorithm from [2] running in $O(n \log n)$ time, or the more complicated algorithm from [8] running in $O(n)$ time. It is well known that there exists a diagonal d in the triangulation that splits P into two sub-polygons P_1 and P_2, each consisting of at least $\lfloor \frac{n-1}{3} \rfloor$ and at most $\lfloor \frac{2n-5}{3} \rfloor$ triangles. By partitioning P_1 and P_2 recursively, we obtain a balanced hierarchical decomposition of P representated as a balanced binary tree T. Here, the root of T represents the diagonal d, the left and right subtrees represent P_1 and P_2, respectively, and the leafs represent the individual triangles. The contribution from a single triangle Δ is simply

$$\int_{p\in\Delta}\int_{q\in\Delta} \mathrm{d}q\,\mathrm{d}p = |\Delta|^2,$$

i.e. the squared area of Δ.

The triangulation of P induces a dual graph G which is a tree with maximum degree 3, and the diagonal d can be chosen as incident to the triangle corresponding to the centroid of G. We can therefore find d in $O(n)$ time by a simple traversal of G, which yields an algorithm for constructing the tree T in $O(n \log n)$ time. By a more refined approach, T can also be computed in $O(n)$ time [13].

Once the diagonal d has been chosen, we proceed as follows. Note that if p and q are points in P that see each other, then either (i) the segment pq intersects d, (ii) pq is contained in P_1, or (iii) pq is contained in P_2. We therefore first compute the contribution to the Beer index of P from pairs of points seeing each other across d. We then recursively compute the contribution from pairs of points on each side of d, i.e., in each of the sub-polygons P_1 and P_2. The base case is when we reach a triangle of area x, where we add x^2 to the Beer index. As we will see, we can compute the contribution of points seeing each other across d in $O(n^2)$ time, resulting in the running time recurrence $t(n) = O(n^2) + t(k) + t(n-k)$ for $k \in [n/3, 2n/3]$, which gives the bound $t(n) = O(n^2 \log n)$.

Recall that d is a diagonal splitting P into two polygons P_1 and P_2 with roughly equally many corners. Assume without loss of generality that the diagonal d is horizontal. Let P_1 be the part below d and P_2 the part above d. We want to compute the contribution to the Beer index of the pairs of points that see each other across d and then recurse on P_1 and P_2. Let $d = uv$, so that u and v are the left and right endpoints

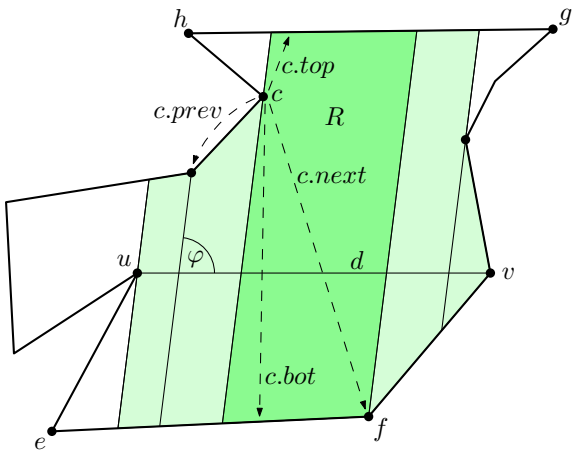


Figure 3: The polygon $\mathrm{vis}(d, \varphi)$ shown in gray with a trapezoid R in darker gray. The pointers stored with the corner c are shown. The left and right pivots of the trapezoid R are c and f, respectively, and the top and bottom edges are gh and ef, respectively.

of d, respectively. We define the *angle* of a segment pq to be the counterclockwise angle from d to pq, which we express as a number in the interval $[0, \pi)$. We say that two points p, q *look across* d if p and q see each other and $d \cap pq \neq \emptyset$. We will compute the contribution B_d to the Beer index of all pairs of points p, q that look across d where $p \in P_1$ and $q \in P_2$. The total contribution from pairs of points looking across d is then $2B_d$, because we should also account for the symmetric case that $p \in P_2$ and $q \in P_1$.

Rotational sweep. In order to compute this contribution, B_d, we perform a rotational sweep from all points on the entire segment d simultaneously. That is, we let an angle φ run from 0 to π and as φ increases, we compute the contribution to the Beer-index of the pairs of points $p \in P_1$ and $q \in P_2$ that look across d and such that the angle of pq is φ. A point y in P is said to be *visible* from d at an angle of φ if there exists a point x on d such that x and y see each other and the angle of xy is φ.

Define $\mathrm{vis}_P(d, \varphi) = \mathrm{vis}(d, \varphi)$ to be the points of P visible from d at an angle of φ. We now partition $\mathrm{vis}(d, \varphi)$ into trapezoids, as demonstrated in Figure 3: For each corner c of P visible from d at an angle of φ, we cut $\mathrm{vis}(d, \varphi)$ along the maximal line segment containing c, contained in $\mathrm{vis}(d, \varphi)$ and with angle φ. A trapezoid R has a pair of parallel edges with angle φ, each of which contains a corner of P. These corners are called the *left* and *right pivot* of R, respectively. The other edges of R are contained in edges of P, and these are called the *top* and *bottom* edges of R, respectively, so that the bottom edge is in P_1 and the top edge is in P_2. As indicated in Figure 3, we will represent the trapezoids by storing certain pointers together with the corners of P.

Events. The events of the rotational sweep are the angles φ at which the combinatorial structure of the trapezoids of $\mathrm{vis}(d, \varphi)$ change. This can happen because a corner becomes visible or invisible from d, or because two corners on opposite sides of d are visible from the same point on d at an angle of φ. Whenever a change happens, we will compute a contribution to the Beer index from the trapezoids that stop existing at the event. It then follows that we compute the contribution B_d of all pairs that see each other across d, since every pair appears in a trapezoid that will eventually stop existing.

When a corner of P becomes visible from d, an old trapezoid will be replaced by two new ones, and we call it an *appearance* event. Likewise, when a corner of P becomes invisible from d, two old trapezoids will be replaced by one new, and we call it a *disappearance* event.

A third type of event occurs when the angle φ coincides with the angle of a diagonal ab crossing d. Here, both corners a and b keep being visible before and after the event, but swap order, and the incident trapezoids must be updated. Figure 4 shows examples of appearance, disappearance and swap events.

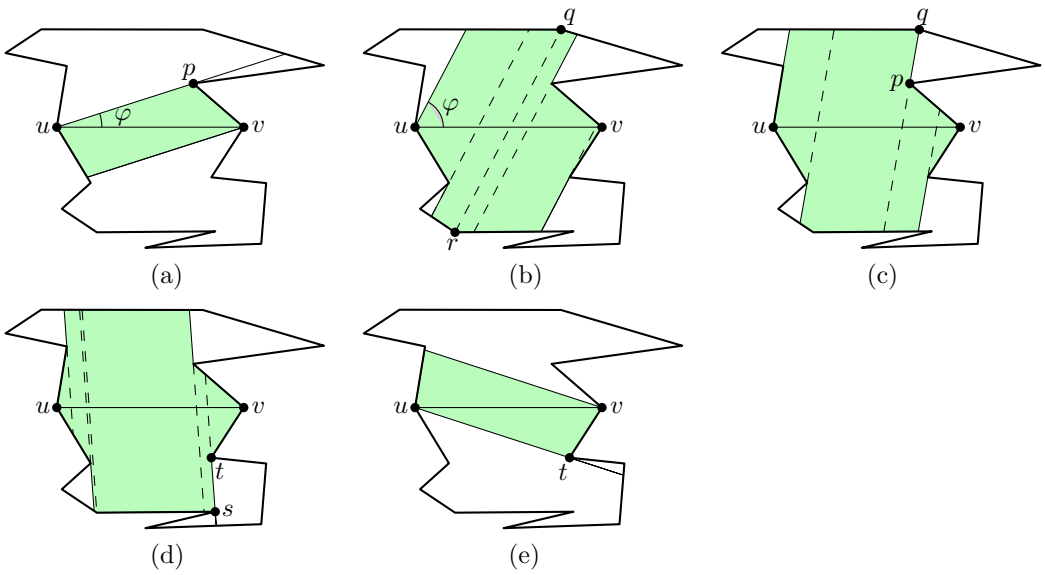


Figure 4: Different types of events in order of φ. (a) Appearance of p. (b) Swap of r and q. (c) Disappearance of q. (d) Appearance of s. (e) Disappearance of t.

The angles $\varphi = 0$ and $\varphi = \pi$ are exceptional in that for $\varphi = 0$, the first trapezoid is created and for $\varphi = \pi$, the last trapezoid is removed.

Note that when a corner c becomes visible, it is because an edge e stops blocking c from being visible. Likewise, when a corner c stops being visible, it is because an edge e starts blocking c from being visible. Therefore, appearance and disappearance events happen when c and an endpoint c' of e are both visible at an angle of φ from the same point on and to the same side of d. Likewise, the swap events happen when two corners c and c' are both visible at an angle of φ from the same point on d, but this time c and c' are on opposite sides of d. We conclude that each event corresponds to a diagonal of P (recall that the edges of P are also diagonals by definition).

Computing diagonals. In order to keep track of the events, we compute the set $\mathcal{D}$ of all diagonals of P before starting the divide-and-conquer process, and we sort the segments $\mathcal{D}$ by angle. We can compute the diagonals by running a simple rotating sweep-line algorithm centered at each corner [2], which takes $O(n^2 \log n)$ time in total, including sorting all the diagonals in the end.

For each diagonal ab, we will also need to know the edges we see from a when looking towards b and vice versa. Let $\eta(a, b)$ be the edge of P we see immediately to the left of b when looking from a; see Figure 5 for an illustration. In the special case that ab is an edge of P, we define $\eta(a, b) := ab$ if P is to the right of ab. We keep track of the two edges $\eta(a, b)$ and $\eta(b, a)$ for each diagonal ab, and we call them the *η-edges* of the diagonal. We can easily define the η-edges while finding the diagonals with a rotational sweep around each corner.

We construct three subsets $\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_d \subseteq \mathcal{D}$ such that $\mathcal{D}_1$ and $\mathcal{D}_2$ are the diagonals with both endpoints in P_1 and P_2, respectively. When computing the contributions to the Beer index in the polygons P_1 and P_2 recursively, we then pass on the sets $\mathcal{D}_1$ and $\mathcal{D}_2$ as arguments in the recursive calls. We define $\mathcal{D}_d$ as the diagonals whose extensions intersect d. Note that the extension of a diagonal ℓ intersects d if and only if the η-edges of ℓ are on different sides of d or ℓ shares an endpoint with d.

For each of the diagonals in $\mathcal{D}_1$ in P_1, one of the η-edges can be an edge in the other polygon P_2 (which, as noted before, can happen when the extension of the diagonal crosses d). In that case, we update the η-edge to be d, so that the η-edges are correctly defined with respect to the smaller polygon P_1. We do the same for the diagonals $\mathcal{D}_2$, so that the η-edges are correct within P_2.

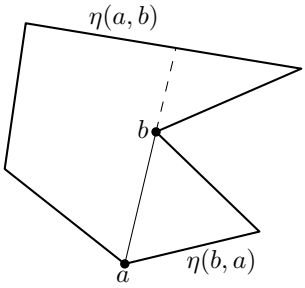


Figure 5: The η-edges for a diagonal ab.

Handling events. When we set out to compute the contribution B_d, we know all events $e_1, \dots, e_k$. Here, e_i is defined by two corners $e_i = a_i b_i$, and the events are sorted according to angle, where the first event e_1 is the diagonal d. Furthermore, $a_1 = b_k = u$ and $a_k = b_1 = v$, so that the diagonal d itself corresponds to two special events that start and stop the rotational sweep. Note that these events happen for the angles $\varphi = 0$ and $\varphi = \pi$, respectively.

As the angle φ runs from 0 to π, we maintain the trapezoids of the decomposition of $\mathrm{vis}(d, \varphi)$ using a linked list L; see Figure 3. Each entry in L is a corner c of P which is visible from d at the angle φ. We store pointers $c.next$ and $c.prev$ to the neighbouring corners, as described in the following. Before the sweep starts, these pointers are all set to `nil` and L will be empty. When handling the first event $e_1 = uv$, we set $L.head := u$, $L.tail := v$, $u.next := v$ and $v.prev := u$, and until the last event e_k, it will be the case that $L.head = u$ and $L.tail = v$. To define the pointer $next$ and $prev$ in general, consider any trapezoid R of $\mathrm{vis}(d, \varphi)$ and let the left and right pivots be g and h, respectively. Then we will have $g.next = h$ and $h.prev = g$. At the last event e_k, the list L becomes empty again, as the last trapezoid disappears.

With each corner c in L, we furthermore store pointers to the top and bottom edges of the trapezoid to the right of c (which has c as the left pivot). We denote these pointers $c.top$ and $c.bot$, respectively.

The pointers stored with the corners must be updated at each event. This happens according to a few cases, as shown in Figure 4. Consider an appearance event where a corner s becomes visible since φ reaches the angle of a diagonal st. Suppose that st is contained in P_1, as in Figure 4 (d). Then a new trapezoid is created with s and t as left and right pivots, and $\eta(s,t)$ and $\eta(t,s)$ as top and bottom edges, respectively. The trapezoid that had t as right pivot before the event is replaced by a similar trapezoid, but with s as left pivot instead of t. The case where st is contained in P_2 is analogous, but with 'right' and 'left' swapped, as well as $\eta(s,t)$ and $\eta(t,s)$ swapped.

Consider a disappearance event, where a corner q stops being visible since φ reaches the angle of a diagonal pq. Suppose that pq is contained in P_2, as in Figure 4 (c). A trapezoid with q and p as left and right pivots disappears, and the trapezoid that had q as right pivot before the event is replaced by a similar trapezoid with p as right pivot. Again, the case where pq is contained in P_1 is analogous.

In a swap event, φ reaches the angle of a diagonal rq, where r and q are corners of P_1 and P_2, respectively, as in Figure 4 (b). Here, a trapezoid with q and r as left and right pivots disappears and is replaced by a trapezoid with q and r replaced. Furthermore, there were trapezoids before the event with q and r as their right and left pivots, respectively, which are replaced by similar trapezoids, but now with r and q as pivots.

Any trapezoid R exists in some interval of angles $[\varphi_1, \varphi_2] \subseteq [0, \pi]$; see Figure 7. This means that R is created when $\varphi = \varphi_1$ and disappears again when $\varphi = \varphi_2$. We can then parameterize R by φ for $\varphi \in [\varphi_1, \varphi_2]$. We compute the contribution from all pairs of points in $R(\varphi)$ that look across d at the angle φ and integrate over the interval $\varphi \in [\varphi_1, \varphi_2]$. This will be explained in the following paragraph.

Contribution of one trapezoid. In order to evaluate the contribution of one trapezoid, we will use a transformation of coordinates. The contribution will then be an integral over the determinant of the associated Jacobian matrix. In the usual coordinate system, we evaluate the Beer index by integrating over quadruples (x_1, y_1, x_2, y_2), corresponding to pairs of points $(x_1, y_1), (x_2, y_2)$ that can see each other in P.

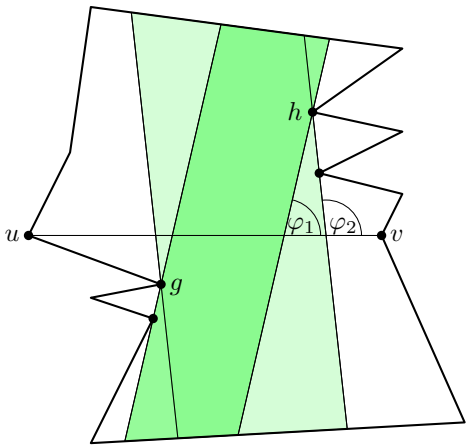


Figure 6: A trapezoid that appears at the angle φ_1 and disappears at φ_2.

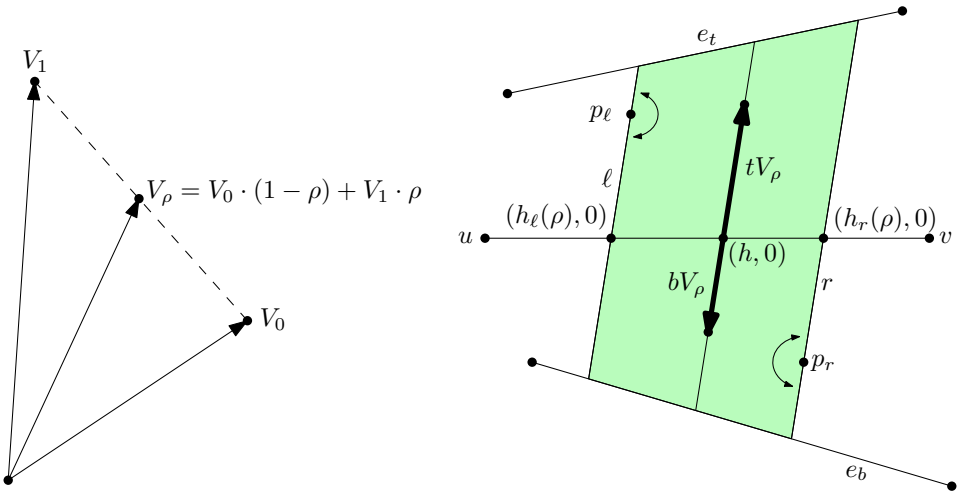


Figure 7: Left: Interpolation between the defining vectors V_0 and V_1. Right: For all points $(h, 0)$ on the segment uv in the trapezoid, we integrate over all points above and below in the direction of V_ρ.

Consider a trapezoid R and assume without loss of generality that the diagonal uv is on the x-axis. The trapezoid R exists in a range $[\varphi_0, \varphi_1]$ of angles that are defined by vectors V_0 and V_1, respectively. For $\rho \in [0,1]$, we define a corresponding interpolated vector $V_\rho = V_0 \cdot (1-\rho) + V_1 \cdot \rho$, which corresponds to the time where the left and right edges of R are parallel to V_ρ. Our new coordinates are (ρ, h, t, b), and the cooresponding usual coordinates are given by the transformation

$$F(\rho, h, t, b) = (h + t \cdot x(V_\rho), t \cdot y(V_\rho), h + b \cdot x(V_\rho), b \cdot y(V_\rho)),$$

where $x(\cdot)$ and $y(\cdot)$ denote the x- and y-coordinates, respectively.

We thus have the Jacobian

$$J_F = \begin{bmatrix} t \cdot x(V_1 - V_0) & 1 & x(V_\rho) & 0 \\ t \cdot y(V_1 - V_0) & 0 & y(V_\rho) & 0 \\ b \cdot x(V_1 - V_0) & 1 & 0 & x(V_\rho) \\ b \cdot y(V_1 - V_0) & 0 & 0 & y(V_\rho) \end{bmatrix}.$$

We now describe the bounds of the integral that we evaluate in order to compute the contribution to the Beer index of the trapezoid R. Let e_t and e_b be the top and bottom edges bounding R. Let $h_\ell(\rho)$ be the x-coordinate of the intersection of uv with the line through p_ℓ with direction V_ρ and define $h_r(\rho)$ analogously for the point p_r. For a value $h \in [h_\ell(\rho), h_r(\rho)]$, let $T(\rho, h) \geq 0$ be the number such that $(h, 0) + T(\rho, h) \cdot V_\rho$ is a point on e_t. Similarly, define $B(\rho, h) \leq 0$ such that $(h, 0) + B(\rho, h) \cdot V_\rho$ is on e_b. The contribution of R can then be expressed as

$$\int_0^1 \int_{h_\ell(\rho)}^{h_r(\rho)} \int_{B(\rho,h)}^{0} \int_0^{T(\rho,h)} |\det J_F| \, \mathrm{d}t \, \mathrm{d}b \, \mathrm{d}h \, \mathrm{d}\rho.$$

Using Maple [21], we have computed a closed-form formula for this integral depending on the coordinates of the endpoints of e_t, e_b, the pivots p_ℓ, p_r and corners defining the vectors V_0, V_1. Without loss of generality, we assumed that $x(p_\ell) = 0$. Unfortunately, the formula is very long: When output to a file, it takes up around 16 MB of space, so it seems quite impractical to use this formula directly. A more practical option is to compute a formula for the three innermost integrals, which has a much shorter form, and then use numerical integration to integrate over ρ to find an approximation of the value of the full integral.

An alternative transformation of coordinates, which is perhaps more appealing at first sight, is to use the angle φ as a parameter instead of the interpolation factor ρ, so that the integral has the form $\int_{\varphi_0}^{\varphi_1} \ldots \mathrm{d}\varphi$. This, however, has the downside that trigonometric functions will appear in the Jacobian, and square roots will appear in the bounds of the two innermost integrals. We used this parameterization in our first attempt to evaluate the contribution of a trapezoid, but were unable to find a formula using Maple.

4 Counting visible pairs

In this section, we describe an algorithm for computing the number of visible pairs among a set M of m points contained in a simple polygon P with n corners. Like in our algorithm for the Beer index, we choose a diagonal uv of P that separates P into two sub-polygons P_1 and P_2 with nearly equal numbers of corners. We then count the number k of pairs in M that see each other across uv and then recurse on P_1 and P_2. We are able to compute k in time $O(n + m \log n + m \log m)$. This results in an algorithm with running time $O(n \log n + m \log^2 n + m \log n \log m)$.

Our algorithm relies on the principle of geometric duality [2]. Here, a point $p = (a, b)$ is mapped to a line $p^* : y = ax - b$ and a line $\ell : y = cx + d$ is mapped to a point $\ell^* = (c, -d)$. Assume without loss of generality that the splitting diagonal uv is on the y-axis. For a segment s, denote by $l(s)$ the line containing s. For each point $p \in M$ and every point $q \in uv$ that p sees, we draw the point $l(pq)^*$ in the dual; see Figure 8. Since p sees an interval of uv and all the lines $l(pq)^*$ pass through the same point p, these dual points form a segment s_p. Our algorithm relies on the following observation:

Lemma 8. *If $p_1, p_2 \in M$ are on different sides of uv and the segments s_{p_1} and s_{p_2} intersect, then p_1 and p_2 see each other across uv.*

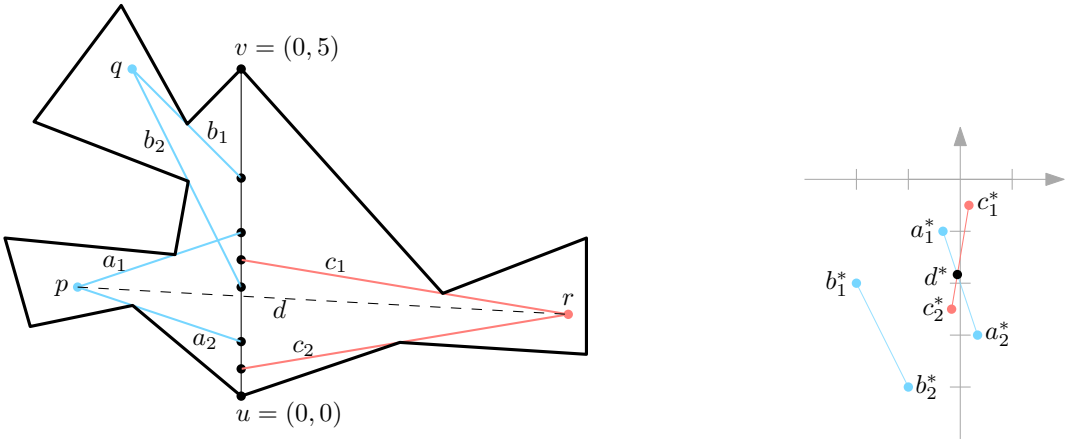


Figure 8: Left: We find the visible parts of the diagonal uv from the three points p, q, r. Right: We map the visibility segments to the dual space and identify the segment $d = pr$ as the intersection d^*.

Proof. The intersection point r^* of s_{p_1} and s_{p_2} corresponds to a line r in primal space that contains visibility segments from both p_1 and p_2 to uv. Therefore, p_1 and p_2 see each other across uv. □

When $p \in M$ is in P_1, we color the segment s_p blue. Otherwise, when p is in P_2, we color s_p red. Then the number of pairs that see each other across uv is exactly the number of intersections between a red and a blue segment in the dual space. Chazelle, Edelsbrunner, Guibas and Sharir [9] gave an algorithm for counting such bichromatic intersections in $O(m \log m)$ time. Palazzi and Snoeyink [22] and Mantler and Snoeyink [20] described simpler algorithms with the same running time.

In order to construct the segments s_p efficiently, we use a datastructure described by Guibas, Hershberger, Leven, Sharir and Tarjan [13]. Given the segment uv in P, we can preprocess P in $O(n)$ time so that given a query point p, we can compute the part of uv that is visible from p in $O(\log n)$ time. The authors assumed that P was given together with a triangulation, presumably because the paper appeared before it was known that a triangulation can be found in $O(n)$ time [8]. We therefore use $O(n + m \log n + m \log m)$ time on preprocessing, constructing the segments s_p for all $p \in M$, and counting the intersections. In order to handle the recursive calls, we need to find the sets $M_1 = M \cap P_1$ and $M_2 = M \cap P_2$. This can be done by creating a data structure supporting ray shooting queries in P in $O(\log n)$ time, as described by Hershberger and Suri [15]. For each point $m \in M$, we shoot a ray vertically up and check if the point where we hit the boundary is in P_1 or P_2. The data structure takes $O(n)$ time to construct. Taking the recursive calls into account, this results in an algorithm with total running time $O(n \log n + m \log^2 n + m \log n \log m)$, proving Theorem 2.

5 Expected distances in the L_2-metric

We now show how to compute the expected geodesic L_2 distance in a simple polygon P. For $x, y \in P$, we denote in this section by $x \rightsquigarrow y$ the unique shortest path in P from x to y in the L_2-metric. As before, consider a polygon P and the diagonal uv splitting P into upper and lower polygons P_1 and P_2. Let $d(x, y) = |x \rightsquigarrow y|_2$ be the length of the path. Furthermore, let X and Y be stochastic variables that assumes points of P uniformly at random. We then have that

$$E(d(X, Y)) = \int_{x \in P} \int_{y \in P} d(x, y) \, \mathrm{d}y \, \mathrm{d}x \tag{3}$$

$$= E(d(X_{P_1}, Y_{P_1})) + E(d(X_{P_2}, Y_{P_2})) + \int_{x \in P_1} \int_{y \in P_2} d(x, y) \, \mathrm{d}y \, \mathrm{d}x. \tag{4}$$

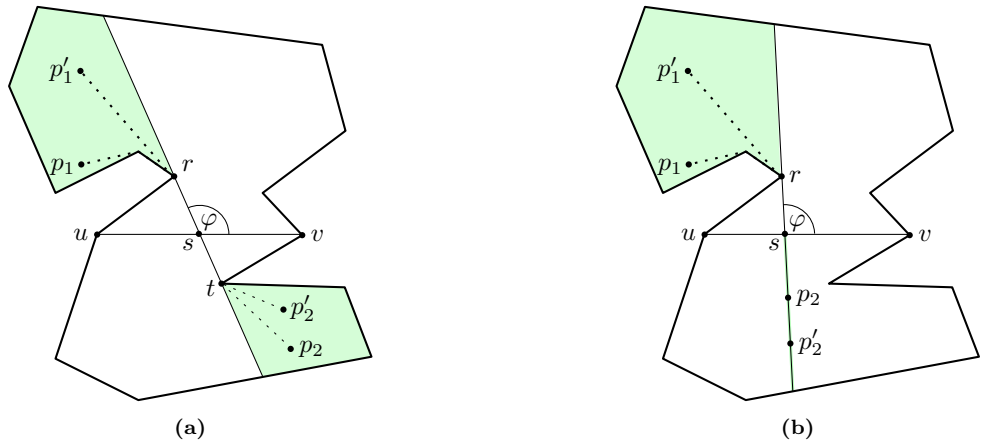


Figure 9: Visualization of cases and equivalence classes, where $(p_1, p_2) \sim (p_1', p_2')$. (a) Case 2a) where the areas highlighted in green correspond to $K^{r,t}$. (b) Case 2b) where the area and subpath highlighted in green corresponds to $K^{r,\varphi}$. Case 2c) is symmetric to case 2b) and omitted here.

The following observation will be useful in characterizing the second term of eq. (4); see Figure 9:

Observation 9. *Given $p_1 \in P_1$ and $p_2 \in P_2$, there is an unique intersection point with $p_1 \rightsquigarrow p_2$ and the diagonal. The segment of this shortest path intersects the diagonal at some unique angle. Furthermore, either*

1) $p_1 \rightsquigarrow p_2$ is a single segment crossing the diagonal, or

2) $p_1 \rightsquigarrow p_2$ consists of two or more segments and either has the form

2a) $p_1 \rightsquigarrow rt \rightsquigarrow p_2$ where rt is the segment crossing the diagonal and $p_1 \neq r$ and $p_2 \neq t$, or

2b) $p_1 \rightsquigarrow rp_2$ where rp_2 is the segment crossing the diagonal and $p_1 \neq r$, or

2c) $p_1 r \rightsquigarrow p_2$ where $p_1 r$ is the segment crossing the diagonal and $p_2 \neq r$.

Denote by $A \subseteq P_1 \times P_2$ and $B \subseteq P_1 \times P_2$ the pairs of vertices of P satisfying 1) and 2), respectively. We rewrite the third term of Equation (4) to obtain

$$\int_{(p_1,p_2)\in P_1\times P_2} d(p_1,p_2)\,\mathrm{d}(x,y) = \int_{(p_1,p_2)\in A} d(p_1,p_2)\,\mathrm{d}(p_1,p_2) \tag{5}$$

$$+ \int_{(p_1,p_2)\in B} d(p_1,p_2)\,\mathrm{d}(p_1,p_2) \tag{6}$$

Case 1. An expression for the first term in the RHS of Equation (5) can be computed using the algorithm of Section 3 but using the L_2-metric as the measurable map instead of the visibility indicator. Hence we need only account for the contribution that stems from pairs of points that are not visible from one another in P, that is the term of Equation (6). This will be accounted for in the following:

Case 2. Denote by $C, D, E \subseteq B$ the pairs satisfying 2a), 2b) and 2c) respectively. We then have

$$\int_{(p_1,p_2)\in B} d(p_1,p_2)\,\mathrm{d}(p_1,p_2) = \int_{(p_1,p_2)\in C} d(p_1,p_2)\,\mathrm{d}(p_1,p_2) \tag{7}$$

$$+ \int_{(p_1,p_2)\in D} d(p_1,p_2)\,\mathrm{d}(p_1,p_2) \tag{8}$$

$$+ \int_{(p_1,p_2)\in E} d(p_1,p_2)\,\mathrm{d}(p_1,p_2). \tag{9}$$

The following definition can be thought of as the set of points in P that go through c on their shortest path to s, and will be useful in characterizing integrals over the above domains:

Definition 10. *For any corner c and point s of P visible from c, we define $K_c^P(s) = \{p \in P \mid c \in p \rightsquigarrow s\}$.*

Case 2a. Now, if $(p_1, p_2) \in C$, we have that $p_1 \rightsquigarrow p_2 = p_1 \rightsquigarrow rt \rightsquigarrow p_2$ where r is a corner of P_1 and t of P_2. In other words, we can express C as a partition into equivalence classes $\{K^{r,t} \mid r \in V(P_1), t \in V(P_2)\}$, where $V(\cdot)$ denotes the corners of a polygon. The equivalence classes are defined so that $(p_1, p_2) \sim (p_1', p_2')$ if and only if the shortest paths $p_1 \rightsquigarrow p_2$ and $p_1' \rightsquigarrow p_2'$ both contain the diagonal rt, which crosses the splitting diagonal uv. We can now write $K^{r,t}$ as a cartesian product $K_r^P(t) \times K_t^P(r) \subseteq C$, so we may rewrite the RHS of Equation (7) as

$$\int_{(p_1,p_2)\in C} d(p_1, p_2)\, \mathrm{d}(p_1, p_2) \tag{10}$$

$$= \sum_{r\in V(P_1)} \sum_{t\in V(P_2)} \left(\int_{p_1\in K_r^P(t)} \int_{p_2\in K_t^P(r)} d(p_1, p_2)\, \mathrm{d}p_2\, \mathrm{d}p_1 \right). \tag{11}$$

Case 2b and 2c. Otherwise, consider the case in which $(p_1, p_2) \in D$. The case in which $(p_1, p_2) \in E$ is symmetric. Then $p_1 \rightsquigarrow p_2$ has the form $p_1 \rightsquigarrow rp_2$, where r is a corner of P_1 and rp_2 intersects uv. Let $s_\varphi r$ be the segment going from the diagonal uv at an angle of φ to r, where $s_\varphi \in uv$. Again, we can characterize D by a partition into equivalence classes $\{K^{t,\varphi} \mid \varphi \in (\pi; 2\pi), r \in V(P_1)\}$ where $K^{r,\varphi} = K_r^P(s_\varphi) \times K_{s_\varphi}^P(r) \subseteq D$ such that $(p_1, p_2) \sim (p_1', p_2')$ if and only if the last segment of $p_1 \rightsquigarrow p_2$ and $p_1' \rightsquigarrow p_2'$ start at the same corner, r, of P_1 and intersects the diagonal with the same angle φ. We hence get

$$\int_{(p_1,p_2)\in D} d(p_1, p_2)\, \mathrm{d}(p_1, p_2) \tag{12}$$

$$= \sum_{r\in V(P_1)} \left(\int_{\pi}^{2\pi} \int_{p_1\in K_r^P(s_\varphi)} \int_{p_2\in K_{s_\varphi}^P(r)} d(p_1, p_2)\, \mathrm{d}p_2\, \mathrm{d}p_1\, \mathrm{d}\varphi \right). \tag{13}$$

Continuing in this fashion, we note that for each choice of $r \in V(P_1)$, and edges $e_1 \in E(P_1)$ and $e_2 \in E(P_2)$, there is a unique, maximal interval I_{e_1,e_2}^r of angles such that the maximal line segment in P through r at angles in I_{e_1,e_2}^r has one endpoint lying on e_1 and the other on e_2. Specifically, I_{e_1,e_2}^r is the empty set when e_1 is not visible e_2 through r in P. Let $I_{e_1,e_2}^r = (\varphi_{e_1,e_2}^{1,r}; \varphi_{e_1,e_2}^{2,r})$ and note that the family $I^r = \left\{I_{e_1,e_2}^r \mid e_1 \in V(P_1), e_2 \in V(P_2)\right\}$ is a partition of $(\pi; 2\pi)$. So we can rewrite the RHS of eq. (13) as

$$\sum_{r\in V(P_1)} \sum_{I\in I^r} \left(\int_{\varphi\in I} \int_{K_r^P(s_\varphi)} \int_{K_{s_\varphi}^P(r)} d(p_1, p_2)\, \mathrm{d}p_2\, \mathrm{d}p_1\, \mathrm{d}\varphi \right). \tag{14}$$

We note that since P is simple, we have $|I^r| = O(|E(P_1)| + |E(P_2)|) = O(|E(P)|)$, which is easily seen as each segment appears and disappears at most once when performing a rotational sweep around r.

Shortest path decompositions. In this section we introduce notions and definitions that will be helpful for characterizing integrals of Equation (11) and Equation (14) in a manner that is is amenable for computation. We first introduce the notion of a shortest-path decomposition of P, which will partition the graph into triangles suitable for integration.

Consider the directed shortest path tree $T(s)$ rooted at s in P for some $s \in uv$. We observe that $T(s) = \cup_{w\in V(P)}(s \rightsquigarrow w)$, i.e. the tree is the union of the shortest paths to all corners of P from s. If we augment P with $T(s)$ producing $P \cup T(s)$, this splits P into regions with boundaries comprising one polygon edge and at least two shortest path-edges. If we furthermore augment $T(s)$ by adding, for each (directed) edge qr of $T(s)$, an edge starting at r such that it coincides with the infinite extension of qr in its

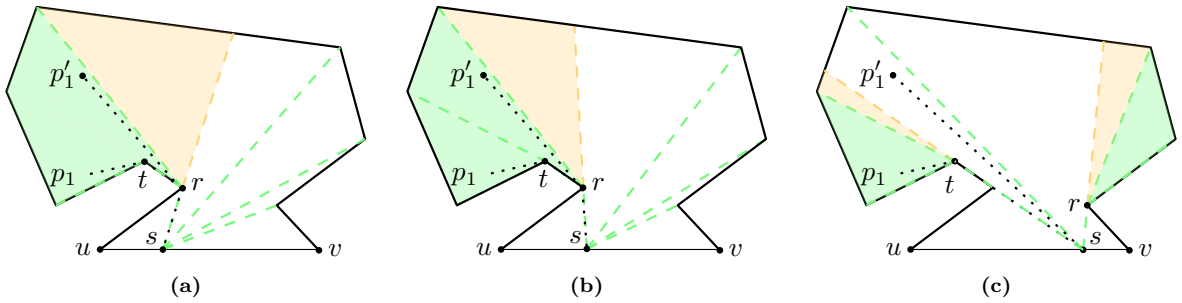


Figure 10: Shortest path tree $T(s)$ and its shortest path decomposition of P_1 for various choices of s.

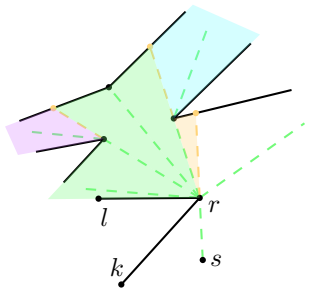


Figure 11: Triangles at the subtree at r rooted at s corresponding to $\kappa_s(sr)$.

out-direction, and going to its first intersection point with the boundary of P. This also adds an artificial corner to P at the intersection point. We refer to the union of an original edge qr and its extension rt, such that t is the artificial corner, as an *extended edge*. Note that there can be no crossings between extended edges due to the triangle inequality, so it follows that each region now forms a triangle whose sides comprise two extended edges and one edge of P. This is illustrated in fig. 10 for various choices of s. Now consider any edge qr of $T(s)$, that is, q is the predecessor of r. Denote by l and k be the corners of P adjacent to r. If l and k are on opposite sides of the infinite extension of qr in both directions, it follows from the triangle inequality that r is a leaf of $T(s)$. Otherwise, l and k are on the same side of the infinite extension, in which case, each adjacent pair of outgoing extended edges from r in $T(s)$ are sides of the same triangle. If they are on the left (right) side, we consider the ordering $e_i^{s,r}$ of the outgoing extended edges of r in $T(s)$ such that $e_i^{s,r}$ precedes $e_{i+1}^{s,r}$ in the (counter) clockwise cyclic order, with $e_1^{s,r}$ being the unique edge coinciding with a segment of P and $e_{|\mathcal{N}^+(r)|}^{s,r}$ being the extension from qr added in the augmentation step in the description of $T(s)$. We denote by $\Delta_i^{s,r}$ the triangle defined by the extended edges $e_i^{s,r}, e_{i+1}^{s,r}$ emanating from r in $T(s)$, and let $\Delta^{s,r} = \cup_i \Delta_i^{s,r}$. For any edge qr of $T(s)$, we now define the following set which can be thought of as the union of triangles associated with the subtree rooted at r in $T(s)$:

Definition 11. *Let P be a simple polygon, s be a point of P and qr be any edge of $T(s)$. We define*

$$\kappa_s(qr) = \begin{cases} \emptyset & \text{if } r \text{ is a leaf} \\ \Delta^{s,r} \cup (\cup_i \kappa_s(e_i^{s,r})) & \text{otherwise} \end{cases}$$

This notion is illustrated in Figure 11, and shall prove helpful when characterizing integrals of Equation (11) and Equation (14). We first prove the following lemma, which shows that $K_r^P(s)$ corresponds to $\kappa_s(sr)$:

Lemma 12. *Let P be a simple polygon, r be a corner of P and s be a point visible from r. Then $K_r^P(s) = \kappa_s(sr)$ μ-almost everywhere.*

Proof. If r is a leaf node of $T(s)$, then $K_r^P(s)$ and $\kappa_s(sr)$ both forms null sets, so assume without loss of generality that the edges rl and rk of P incident on r are both on the left of the infinite extension of $e_1^{s,r}$. The case in which they are on the right is symmetric. Now, by assumption $s \rightsquigarrow p$ has the form $sr \rightsquigarrow p$ where r is a corner of P_1. Denote by rr' the edge which follows sr on $s \rightsquigarrow p$ and by rr'' the edge of $T(s)$ which extends sr and consider the counter-clockwise cyclic ordering of edges incident on r in $P \cup T(s)$: We will show that rr' appears after rr'' and before both of rk and rl in the ordering. Since $s \rightsquigarrow p$ is a shortest path, by the triangle inequality, it can coincide with rr'' at most once, which happens at r. Since rr'' is a diagonal of $P_1 \cup V(T(s))$ and as P is simple, it follows that $rr' \rightsquigarrow p$ is fully contained in one of the polygonal pieces that result from splitting P_1 along rr'. Since $\kappa_s(sr)$ induces a decomposition of this polygonal piece, this will imply that $K_r^P(s) \subseteq \kappa_s(sr)$ which shows one direction of the equality.

Assume therefore, for the sake of contradiction, that rr' does not appear as aforementioned in the cyclic ordering, but it is an edge of $T(s)$. Note that $s \rightsquigarrow r'$, by assumption, can not be a single edge, and that our assumption now states that the edge rr' follows sr and precedes rr'' in the aforementioned cyclic ordering. Since $s \rightsquigarrow r'$ is a shortest path, there must be at least two edges of P that intersect the line segment sr', occluding s from r', but this is not the case for sr nor rr' since the latter two were edges of $T(s)$. We now restrict our attention to the triangle $\Delta(s, r, r') \subset P$, and consider paths that start at s and end in r'. Clearly, our assumed shortest path s, r, r' is contained in (the non-strict interior of) $\Delta(s, r, r')$, but we note that since no edge of P intersects sr or rr', this implies that there is a convex chain $C \in \Delta(s, r, r')$ starting at s and ending at r' such that $|C| < |sr| + |rr'|$ contradicting that s, r, r' is shortest-path.

Next assume that $p \in \kappa_s(sr)$. Then clearly $p \rightsquigarrow s$ has the form $p \rightsquigarrow rs$ (observe that those that do not form a null-set) where s lies on uv, so $p \in K_r^P(s)$, and hence $K_r^P(s) = \kappa_s(sr)$ μ-almost everywhere. □

Corollary 13. *Let P be a simple polygon, r be a corner of P and s be a point visible from r. Then*

1. $\int_{p \in \kappa_s(sr)} d(s,p)\,\mathrm{d}\mu(p) = \int_{p \in K_r^P(s)} d(s,p)\,\mathrm{d}\mu(p)$ *and*
2. $\mu(\kappa_s(sr)) = \mu(K_r^P(s))$.

We define quantities $W_s(qr)$ and $A_s(qr)$, corresponding to the contribution stemming from all triangles associated with the subtree rooted at r in $T(s)$, and the sum of their areas, respectively. We define these functions with respect to edges and not vertices for notational convenience. In the following, we let $e_i = e_i^{s,r}$. If r is a leaf of $T(s)$, we define $W_s(qr) = 0$ and $A_s(qr) = 0$. Otherwise

$$W_s(qr) = \sum_{i=1}^{|\mathcal{N}^+(r)|-1} \left(\int_{p \in \Delta_i^{s,r}} d(p,r)\,\mathrm{d}\mu(p) \right) + \sum_{i=1}^{|\mathcal{N}^+(r)|} \left(W_s(e_i) + A_s(e_i) \cdot d(e_i) \right)$$

and accordingly

$$A_s(qr) = \sum_{i=1}^{|\mathcal{N}^+(r)|-1} \left(A_s(e_i) + \mu(\Delta_i^{s,r}) \right).$$

where $\mathcal{N}^+$ is with respect to $T(s)$. The above notion is summarized in the following lemma:

Lemma 14. *Let P be a simply polygon and r and t be points of P such that r sees t and $s \in rt$. Then for any edge qr of $T(s)$, we have $\int_{p \in \kappa_s(qr)} d(r,p)\,\mathrm{d}\mu(p) = W_s(qr)$ and $\mu(\kappa_s(qr)) = A_s(qr)$.*

Proof. We proceed by structural induction on $T(s)$. If r' is a leaf node, we have $W_s(rr') = A(rr') = \mu(\Delta^{r,r'}) = 0$. Otherwise, assume, inductively, that $\int_{p \in \kappa_s(rr')} d(r,p)\,\mathrm{d}\mu(p) = W_s(rr')$ and $\mu(\kappa_s(rr')) = A_s(rr')$ where rr' is any outgoing edge of r in $T(s)$. Again, we let $e_i = e_i^{s,r}$. Then by the induction

hypothesis, we have

$$\begin{aligned}
A_s(qr) &= \sum_i (A_s(e_i) + \mu(\Delta_i^{s,r})) \\
&= \sum_i (\mu(\kappa_s(e_i)) + \mu(\Delta_i^{s,r})) \\
&= \sum_i \mu(\kappa_s(e_i)) + \sum_i \mu(\Delta_i^{s,r}) \\
&= \sum_i \mu(\kappa_s(e_i)) + \mu(\Delta^{s,r}) \\
&= \mu(\kappa_s(qr))
\end{aligned}$$

which is what we wanted to show about A_s. The fourth equality is due to the fact that $\mu(\Delta_i^{s,r} \cap \Delta_j^{s,r}) = 0$ for $i \neq j$. The fifth equality follows since $\kappa_s(qr) = \Delta^{s,r} \cup (\cup_i \kappa_s(e_i^{s,r}))$, and since also $\mu(\Delta^{s,r} \cap (\cup_i \kappa_s(e_i^{s,r}))) = 0$.

The first term of $W_s(qr)$ accounts for each triangle associated with edges going out from r in $T^+(s)$. So for the first term, we get

$$\sum_{i=1}^{|\mathcal{N}^+(r)|-1} \left(\int_{p \in \Delta_i^{s,r}} d(p,r)\, \mathrm{d}\mu(p) \right) = \int_{p \in \Delta^{s,r}} d(p,r)\, \mathrm{d}\mu(p)$$

and furthermore, by applying the induction hypothesis to the second term, we get

$$\begin{aligned}
\sum_{i=1}^{|\mathcal{N}^+(r)|} (W_s(e_i) + A_s(e_i) \cdot d(e_i)) &= \sum_{i=1}^{|\mathcal{N}^+(r)|} \left(\int_{p \in \kappa_s(e_i)} d(\mathrm{head}(e_i), p)\, \mathrm{d}\mu(p) + \mu(\kappa_s(e_i)) \cdot d(e_i) \right) \\
&= \sum_{i=1}^{|\mathcal{N}^+(r)|} \left(\int_{p \in \kappa_s(e_i)} d(\mathrm{head}(e_i), p) + d(e_i)\, \mathrm{d}\mu(p) \right) \\
&= \sum_{i=1}^{|\mathcal{N}^+(r)|} \left(\int_{p \in \kappa_s(e_i)} d(\mathrm{head}(e_i), p) + d(\mathrm{tail}(e_i), \mathrm{head}(e_i))\, \mathrm{d}\mu(p) \right) \\
&= \sum_{i=1}^{|\mathcal{N}^+(r)|} \left(\int_{p \in \kappa_s(e_i)} d(\mathrm{tail}(e_i), p)\, \mathrm{d}\mu(p) \right) \\
&= \sum_{i=1}^{|\mathcal{N}^+(r)|} \left(\int_{p \in \kappa_s(e_i)} d(r,p)\, \mathrm{d}\mu(p) \right) \\
&= \int_{p \in \cup_i \kappa_s(e_i)} d(r,p)\, \mathrm{d}\mu(p)
\end{aligned}$$

where the fourth equality follows since $r = \mathrm{tail}(e_i)$, and $r, \mathrm{head}(e_i), p$ is a shortest path. The sixth equality is due to the fact that $\mu(\kappa_s(e_i) \cap \kappa_s(e_j)) = 0$ for $i \neq j$. Using the aforementioned reasoning about the definition of κ_s, gathering the terms yields

$$W_s(qr) = \int_{p \in \Delta^{s,r}} d(p,r)\, \mathrm{d}\mu(p) + \int_{p \in \cup_i \kappa_s(e_i)} d(r,p)\, \mathrm{d}\mu(p) = \int_{p \in \kappa_s(qr)} d(p,r)\, \mathrm{d}\mu(p)$$

which is what we wanted. □

The following somewhat trivial observation will be useful in proving the subsequent lemma:

Observation 15. *Let $p \in P$ and $A, B \subset P$ such that there is a $c \in P$ such that $c \in a \rightsquigarrow b$ for μ-almost all $(a,b) \in A \times B$. Then $\int_{A \times B} d(a,b)\, \mathrm{d}(\mu \times \mu)(a,b) = \mu(B) \cdot \int_A d(a,c)\, \mathrm{d}\mu(a) + \mu(A) \cdot \int_B d(c,b)\, \mathrm{d}\mu(b)$.*

Proof.

$$\begin{aligned}\int_{A\times B} d(a,b)\,\mathrm{d}(\mu\times\mu)(a,b) &= \int_A\int_B d(a,b)\,\mathrm{d}\mu(b)\,\mathrm{d}\mu(a)\\ &= \int_A\int_B d(a,c)+d(c,b)\,\mathrm{d}\mu(b)\,\mathrm{d}\mu(a)\\ &= \int_A\int_B d(a,c)\,\mathrm{d}\mu(b)\,\mathrm{d}\mu(a)+\int_A\int_B d(c,b)\,\mathrm{d}\mu(b)\,\mathrm{d}\mu(a)\\ &= \int_B\int_A d(a,c)\,\mathrm{d}\mu(a)\,\mathrm{d}\mu(b)+\int_A\int_B d(c,b)\,\mathrm{d}\mu(b)\,\mathrm{d}\mu(a)\\ &= \mu(B)\cdot\int_A d(a,c)\,\mathrm{d}\mu(a)+\mu(A)\cdot\int_B d(c,b)\,\mathrm{d}\mu(b).\end{aligned}$$

□

We can now characterize the integrals of Equation (11) and Equation (14) in terms of W_s and A_s:

Corollary 16. *Let P be a simply polygon and r and t be points of P such that r sees t and $s\in rt$. Then*

$$A_s(st)\cdot W_s(sr)+A_s(sr)\cdot W_s(st)=\int_{p_1\in K_r^P(t)}\int_{p_2\in K_t^P(r)} d(p_1,p_2)\,\mathrm{d}\mu(p_2)\,\mathrm{d}\mu(p_1)$$

Proof. By applying Lemma 14, Corollary 13 and Observation 15, we get

$$\begin{aligned}&A_s(st)\cdot W_s(sr)+A_s(sr)\cdot W_s(st)\\ &=\mu(\kappa_s(st))\cdot\int_{p\in\kappa_s(sr)} d(s,p)\,\mathrm{d}\mu(p)+\mu(\kappa_s(sr))\cdot\int_{p\in\kappa_s(st)} d(s,p)\,\mathrm{d}\mu(p)\\ &=\mu(K_t^P(r))\cdot\int_{p\in K_r^P(t)} d(s,p)\,\mathrm{d}\mu(p)+\mu(K_r^P(t))\cdot\int_{p\in K_t^P(r)} d(s,p)\,\mathrm{d}\mu(p)\\ &=\int_{p_1\in K_r^P(t)}\int_{p_2\in K_t^P(r)} d(p_1,p_2)\,\mathrm{d}\mu(p_2)\,\mathrm{d}\mu(p_1).\end{aligned}$$

□

Characterizing Equation (11) and Equation (14). The above corollary implies that computing the contributions corresponding to integrals of Equation (11) suffices to characterize quantities $A_s(st)$, $A_s(sr)$ and $W_s(st)$, $W_s(sr)$. We shall show how to do this, but first we address the issue of computing the contribution of Equation (14). Recall that in this case, $K^P_{s_\varphi}(r)$ is simply a line segment starting at s_φ and going to its intersection point with e_2 as defined by the associated interval of angles $I^r_{e_1,e_2}$. Denote by e_φ this endpoint. Then since $K^P_{s_\varphi}(r)=re_\varphi\setminus rs_\varphi$, we can rewrite the innermost integral of Equation (14) as

$$\int_{re_\varphi\setminus rs_\varphi} d(p_1,p_2)\,\mathrm{d}\mu(p_2)=\int_{re_\varphi} d(p_1,p_2)\,\mathrm{d}\mu(p_2)-\int_{rs_\varphi} d(p_1,p_2)\,\mathrm{d}\mu(p_2), \tag{15}$$

and substitute back into Equation (14) to obtain

$$\int_I\int_{K_r^P(s_\varphi)}\left(\int_{re_\varphi} d(p_1,p_2)\,\mathrm{d}\mu(p_2)-\int_{rs_\varphi} d(p_1,p_2)\,\mathrm{d}\mu(p_2)\right)\mathrm{d}\mu(p_1)\,\mathrm{d}\mu(\varphi) \tag{16}$$

$$=\int_I\int_{K_r^P(s_\varphi)}\int_{re_\varphi} d(p_1,p_2)\,\mathrm{d}\mu(p_2)\,\mathrm{d}\mu(p_1)\,\mathrm{d}\mu(\varphi) \tag{17}$$

$$-\int_I\int_{K_r^P(s_\varphi)}\int_{rs_\varphi} d(p_1,p_2)\,\mathrm{d}\mu(p_2)\,\mathrm{d}\mu(p_1)\,\mathrm{d}\mu(\varphi). \tag{18}$$

This shows that the integral of Equation (15) can be expressed in terms of the similar integrals that are the terms of Equations (17) and (18). We note that in particular, for any point s_φ as defined in the above, it holds that s_φ sees e_1 through r by definition. The following observation and lemma will allow us to rewrite integrals of the form found in Equations (17) and (18):

Observation 17. *Let P be a simple polygon, e a segment and r a corner of P. Let S be the set of points that see e through r. For any $s \in S$ denote by i_s the intersection of the extension of sr with e, and by e_s the edge which immediately precedes ri_s in the cyclic order of edges incident to r. Then for any $s \neq s' \in S$ where i_s and i'_s are not endpoints of e, we have $e_s = e_{s'}$ and this edge is $e_j^{s,r}$ where j is the largest s.t. $e_j^{s,r}$ is the edge which immediately precedes ri_s.*

Lemma 18. *Let P be a simple polygon, e a segment and r a corner of P. Let S be the set of points that see e through r. For any $s \in S$ denote by i_s the intersection of the extension of sr with e, and by e_s the edge which immediately precedes ri_s in the cyclic order of edges incident to r. Let c_s be any point of S which is colinear with e_s Then for any $s \in S$ s.t. i_s is not an endpoint of e, we have $\kappa_s(sr) \setminus \kappa_{c_s}(c_s r) = \Delta(e_j^{s,r}, ri_s)$ where j is the largest s.t. $e_j^{s,r}$ is the edge which immediately precedes ri_s.*

Proof. We note that $e_j^{s,t}$ is incident on one endpoint of e in $T(s) \cup P$. We apply Definition 11 to get

$$\begin{aligned}
\kappa_s(sr) \setminus \kappa_{c_s}(c_s r) &= \Delta^{s,r} \cup (\cup_i \kappa_s(e_i^{s,r})) \setminus \Delta^{c_s,r} \cup (\cup_i \kappa_{c_s}(e_i^{c_s,r})) && (19)\\
&= (\Delta^{s,r} \setminus (\Delta^{c_s,r} \cup (\cup_i \kappa_{c_s}(e_i^{c_s,r})))) \cup ((\cup_i \kappa_s(e_i^{s,r})) \setminus (\Delta^{c_s,r} \cup (\cup_i \kappa_{c_s}(e_i^{c_s,r})))) && (20)\\
&= (\Delta^{s,r} \setminus \Delta^{c_s,r}) \cup ((\cup_i \kappa_s(e_i^{s,r})) \setminus (\cup_i \kappa_{c_s}(e_i^{c_s,r}))) && (21)\\
&= \Delta^{s,r} \setminus \Delta^{c_s,r} && (22)\\
&= \cup_i \Delta_i^{s,r} \setminus \cup_i \Delta_i^{c_s,r} && (23)\\
&= \left(\cup_{i=1}^{j-1} \Delta_i^{s,r} \cup \Delta_j^{s,r}\right) \setminus \cup_{i=1}^{j-1} \Delta_i^{c_s,r} && (24)\\
&= \Delta_j^{s,r} && (25)\\
&= \Delta(e_j^{s,r}, e_{j+1}^{s,r}) && (26)\\
&= \Delta(e_j^{s,r}, ri_s) && (27)
\end{aligned}$$

where Equation (21) follows from $\Delta^{s,r} \cap (\cup_i \kappa_{c_s}(e_i^{c_s,r})) = \emptyset$ and $(\cup_i \kappa_s(e_i^{s,r})) \cap \Delta^{c_s,r} = \emptyset$, Equation (22) from the fact that $\cup_i \kappa_s(e_i^{s,r}) \subseteq \cup_i \kappa_{c_s}(e_i^{c_s,r})$, Equation (24) holds since $\Delta^{s,r}$ has one triangle $\Delta_j^{s,r}$ more than $\Delta^{c_s,r}$ and finally Equation (25) follows due to $e_i^{s,r}$ sharing endpoints (and therefore extended edges) with $e_i^{c_s,r}$ for $1 \leq i \leq j-1$. □

Again consider w.l.o.g. the integral of Equation (17). Applying Lemma 12, we get

$$\int_I \int_{K_r^P(s_\varphi)} \int_{re_\varphi} d(p_1,p_2)\,\mathrm{d}\mu(p_2)\,\mathrm{d}\mu(p_1)\,\mathrm{d}\mu(\varphi) = \int_I \int_{\kappa_{s_\varphi}(rs_\varphi)} \int_{re_\varphi} d(p_1,p_2)\,\mathrm{d}\mu(p_2)\,\mathrm{d}\mu(p_1)\,\mathrm{d}\mu(\varphi).$$

By Observation 17 all s_φ for $\varphi \in I$ have the same preceding edge ri in the cyclic ordering since they all see e_1 through r. Note that $i \in e_1$. Let c denote the intersection point of the extension of ir and e_2, which is guaranteed to exists by definition of I. Furthermore let i_φ denote the intersection of the extension of $s_\varphi r$ and e_1. By adding and subtracting the same appropriate term on the RHS of the above and applying

Lemma 18, we get

$$\int_I \int_{\kappa_{s_\varphi}(rs_\varphi)} \int_{re_\varphi} d(p_1,p_2)\,\mathrm{d}\mu(p_2)\,\mathrm{d}\mu(p_1)\,\mathrm{d}\mu(\varphi) \tag{28}$$

$$+ \int_I \int_{\kappa_c(cr)} \int_{re_\varphi} d(p_1,p_2)\,\mathrm{d}\mu(p_2)\,\mathrm{d}\mu(p_1)\,\mathrm{d}\mu(\varphi) \tag{29}$$

$$- \int_I \int_{\kappa_c(cr)} \int_{re_\varphi} d(p_1,p_2)\,\mathrm{d}\mu(p_2)\,\mathrm{d}\mu(p_1)\,\mathrm{d}\mu(\varphi) \tag{30}$$

$$= \int_I \int_{\kappa_{s_\varphi}(rs_\varphi)\setminus\kappa_c(cr)} \int_{re_\varphi} d(p_1,p_2)\,\mathrm{d}\mu(p_2)\,\mathrm{d}\mu(p_1)\,\mathrm{d}\mu(\varphi) \tag{31}$$

$$+ \int_I \int_{\kappa_c(cr)} \int_{re_\varphi} d(p_1,p_2)\,\mathrm{d}\mu(p_2)\,\mathrm{d}\mu(p_1)\,\mathrm{d}\mu(\varphi) \tag{32}$$

$$= \int_I \int_{\Delta(ri,ri_\varphi)} \int_{re_\varphi} d(p_1,p_2)\,\mathrm{d}\mu(p_2)\,\mathrm{d}\mu(p_1)\,\mathrm{d}\mu(\varphi) \tag{33}$$

$$+ \int_I \int_{\kappa_c(cr)} \int_{re_\varphi} d(p_1,p_2)\,\mathrm{d}\mu(p_2)\,\mathrm{d}\mu(p_1)\,\mathrm{d}\mu(\varphi). \tag{34}$$

We can rewrite the term of Equation (34) to get

$$\int_I \int_{\kappa_c(cr)} \int_{re_\varphi} d(p_1,p_2)\,\mathrm{d}\mu(p_2)\,\mathrm{d}\mu(p_1)\,\mathrm{d}\mu(\varphi) \tag{35}$$

$$= \int_I \int_{\kappa_c(cr)} \int_{re_\varphi} d(p_1,r) + d(r,p_2)\,\mathrm{d}\mu(p_2)\,\mathrm{d}\mu(p_1)\,\mathrm{d}\mu(\varphi) \tag{36}$$

$$= \int_I \int_{\kappa_c(cr)} \int_{re_\varphi} d(p_1,r)\,\mathrm{d}\mu(p_2)\,\mathrm{d}\mu(p_1)\,\mathrm{d}\mu(\varphi) + \int_I \int_{\kappa_c(cr)} \int_{re_\varphi} d(r,p_2)\,\mathrm{d}\mu(p_2)\,\mathrm{d}\mu(p_1)\,\mathrm{d}\mu(\varphi) \tag{37}$$

$$= \int_I \int_{re_\varphi} \int_{\kappa_c(cr)} d(p_1,r)\,\mathrm{d}\mu(p_1)\,\mathrm{d}\mu(p_2)\,\mathrm{d}\mu(\varphi) + \int_{\kappa_c(cr)} \int_I \int_{re_\varphi} d(r,p_2)\,\mathrm{d}\mu(p_2)\,\mathrm{d}\mu(\varphi)\,\mathrm{d}\mu(p_1) \tag{38}$$

$$= W_c(cr) \int_I \int_{re_\varphi} \mathrm{d}\mu(p_2)\,\mathrm{d}\mu(\varphi) + A_c(cr) \int_I \int_{re_\varphi} d(r,p_2)\,\mathrm{d}\mu(p_2).\,\mathrm{d}\mu(\varphi) \tag{39}$$

In fact, the second term of Equation (39) provides the analytic format which describes how to compute contributions corresponding to triangles of $T(s)$. We note that re_φ is an linear combination of re_{φ_0} and re_{φ_1} where φ_0 and φ_1 are the smallest, respectively, largest value assumed by $\varphi \in I$. So by the change-of-variables theorem we can apply the transformation $F(\alpha,\beta) = (\alpha\cdot(\beta\cdot re_{\varphi_0}+(1-\beta)\cdot re_{\varphi_0})+(1-\alpha)\cdot(\beta\cdot re_{\varphi_1}+(1-\beta\cdot re_{\varphi_1}))$ to get

$$W_c(cr) \int_0^1 \int_0^1 |\det J_F|\,\mathrm{d}\beta\,\mathrm{d}s + A_c(cr) \int_0^1 \int_0^1 d(r,F(\alpha,\beta))|\det J_F|\,\mathrm{d}\beta\,\mathrm{d}\alpha \tag{40}$$

and thus resolving the contribution amounts to determining values $W_c(cr)$ and $A_c(cr)$ and evaluating the integrals described above. Let us next consider the term of Equation (33). Again, since $\Delta(ri,ri_\varphi)$ is the union of all linear combinations $s\cdot ri+(1-s)\cdot ri_\varphi$, we can apply change-of-variable with the transformation

with $F'(\gamma,\delta) = (\gamma \cdot (\delta \cdot ri + (1-\delta)\cdot ri) + (1-\gamma)\cdot(\delta \cdot ri_{\varphi_1} + (1-\delta)\cdot ri_{\varphi_1}))$ and F to get

$$\int_I \int_{\Delta(ri,ri_\varphi)} \int_{re_\varphi} d(p_1,p_2)\,\mathrm{d}\mu(p_2)\,\mathrm{d}\mu(p_1)\,\mathrm{d}\mu(\varphi) \tag{41}$$

$$= \int_I \int_{re_\varphi} \int_{\Delta(ri,ri_\varphi)} d(p_1,p_2)\,\mathrm{d}\mu(p_1)\,\mathrm{d}\mu(p_2)\,\mathrm{d}\mu(\varphi) \tag{42}$$

$$= \int_I \int_{re_\varphi} \int_0^{\zeta(\varphi)} \int_0^1 d(F'(\gamma,\delta),p_2)|\det J_{F'}|\,\mathrm{d}\delta\,\mathrm{d}\gamma\,\mathrm{d}\mu(p_2)\,\mathrm{d}\mu(\varphi) \tag{43}$$

$$= \int_0^1 \int_0^1 \int_0^{\zeta^{-1}(\alpha)} \int_0^1 d(F'(\gamma,\delta),F(\alpha,\beta))|\det J_{F'}|\,\mathrm{d}\delta\,\mathrm{d}\gamma|\det J_F|\,\mathrm{d}\beta\,\mathrm{d}\alpha \tag{44}$$

where $\zeta(\varphi) = ||i_\varphi - i_{\varphi_0}||/||i_{\varphi_1} - i_{\varphi_0}||$ and the last step follows from ζ being a bijection. Since Equation (18) has the same form, it follows that the integral corresponding to each contribution of Equations (11) and (14) is the sum of a constant number of terms, where each term is either a integral over simple functions and domains, or the product of such a "simple" integral and the quantity $A_s(sr)$ or $W_s(sr)$ for appropriate choice of s and r. For any choice of s that sees r, we can apply Observation 17, Lemma 18 and Lemma 14 to get that $A_s(sr)$ and $W_s(sr)$ is the sum of $A_c(cr)$ where c corresponds to an extended edge incident on r. This is captured in the following corollary:

Corollary 19. *Let P be a simple polygon, e a segment and r a corner of P. Let S be the set of points that see e through r. For any $s \in S$ denote by i_s the intersection of the extension of sr with e, and by e_s the edge which immediately precedes ri_s in the cyclic order of edges incident to r. Let c be any vertex of S colinear with e_s. Then for any $s \in S$ where i_s is not an endpoint of e, we have $W_s(sr) = W_r(cr) + \int_{\Delta(ri_s,e_s)} d(p,r)\,\mathrm{d}p$ and $A_s(sr) = A_r(cr) + \mu(\Delta(ri_s,e_s))$.*

It follows that for any corner r of P and any s that sees r, $A_s(sr)$ can be expressed as a sum of two terms, where the first term is one of at most $O(n)$ different values $A_1^r, A_2^r, \ldots$ specific to r and the second is the contribution corresponding to the triangle formed by the extended edge and the extension of sr. The contribution of this triangle has the form of the second term of Equation (39). If we treat each such value for all r as a symbol for the value it represents, rather than an actual value, we can replace occurrences of $A_s(sr)$ with the appropriate A_i^r in Observation 15 and Equation (39). Then by Corollary 19 we increase the total number of total terms by at most a constant factor, and this implies Theorem 6:

Proof of Theorem 6. For each corner c of P we compute a list of corners visible from c. We do this so they appear in the clockwise order around c s.t. the first corner is the endpoint of the edge adjacent to c in P whose right side faces the interior of P. We can then use the ray-shooting procedure of Hershberger and Suri [15] to determine the diagonals of P that cross uv of Equation (11) as well as extended edges to determine the partition of $(\pi; 2\pi)$ of Equation (14). There can be at most $O(n^2)$ terms for diagonals and as previously argued, at most $O(n)$ terms for each partitions of $(\pi; 2\pi)$ for each of n corners of P, for a total of $O(n^2)$ terms. As argued, each of these terms are of constant size, and comprises simple integrals as well as symbols of from an alphabet of size $O(n^2)$. The above takes total time $O(n \log n)$ yielding a total running time of $O(n \log^2 n)$ across all levels. □

To compute the actual numerical value of the geodesic L_2 distance, we can assume an algorithm O which can evaluate the described types of integrals in time $O(t(d))$ s.t. the output is a d-approximation within the exact value. Then $d = 1$ and $t(d) = O(1)$ would for instance be the case, if the integrals admit a closed form (recall that each integral is of constant size with a constant number of terms); it is however unclear to the authors whether this can be done. For all practical purposes, it seems that an approximation will suffice due to the limited precision offered by the word-RAM model when evaluating simple functions like square roots and logarithms. We do, however, assume that no A_i^r values are known a priori, but remark that each of these value can be computed in time $O(n^2 \log n \cdot t(d))$ by employing a simple dynamic programming approach, which implies Corollary 7.

References

[1] Takao Asano, Tetsuo Asano, Leonidas J. Guibas, John Hershberger, and Hiroshi Imai. "Visibility of Disjoint Polygons". In: *Algorithmica* 1.1 (1986), pp. 49–63. DOI: 10.1007/BF01840436.

[2] Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008. ISBN: 9783540779735. URL: https://www.worldcat.org/oclc/227584184.

[3] Gilles Bonnet, Anna Gusakova, Christoph Thäle, and Dmitry Zaporozhets. "Sharp inequalities for the mean distance of random points in convex bodies". In: *Advances in Mathematics* 386 (2021), pp. 1–27. DOI: https://doi.org/10.1016/j.aim.2021.107813.

[4] Kevin Buchin, Bram Custers, Ivor van der Hoog, Maarten Löffler, Aleksandr Popov, Marcel Roeloffzen, and Frank Staals. *Segment Visibility Counting Queries in Polygons*. 2022. DOI: 10.48550/ARXIV.2201.03490.

[5] Kevin Buchin, Irina Kostitsyna, Maarten Löffler, and Rodrigo I. Silveira. "Region-Based Approximation of Probability Distributions (for Visibility Between Imprecise Points Among Obstacles)". In: *Algorithmica* 81.7 (2019), pp. 2682–2715. DOI: 10.1007/s00453-019-00551-2.

[6] Uwe Bäsel. *The moments of the distance between two random points in a regular polygon*. 2021. DOI: 10.48550/ARXIV.2101.03815.

[7] Sergio Cabello. "Subquadratic Algorithms for the Diameter and the Sum of Pairwise Distances in Planar Graphs". In: *ACM Trans. Algorithms* 15.2 (2019), 21:1–21:38. DOI: 10.1145/3218821.

[8] Bernard Chazelle. "Triangulating a Simple Polygon in Linear Time". In: *Discret. Comput. Geom.* 6 (1991), pp. 485–524. DOI: 10.1007/BF02574703.

[9] Bernard Chazelle, Herbert Edelsbrunner, Leonidas J. Guibas, and Micha Sharir. "Algorithms for Bichromatic Line-Segment Problems Polyhedral Terrains". In: *Algorithmica* 11.2 (1994), pp. 116–132. DOI: 10.1007/BF01182771.

[10] Emanuel Czuber. *Geometrische Wahrscheinlichkeiten und Mittelwerte*. 1884. URL: https://books.google.de/books/about/Geometrische_Wahrscheinlichkeiten_und_Mi.html?id=2jQAAAAAQAAJ.

[11] Pawel Gawrychowski, Haim Kaplan, Shay Mozes, Micha Sharir, and Oren Weimann. "Voronoi Diagrams on Planar Graphs, and Computing the Diameter in Deterministic $\tilde{O}(n^{5/3})$ Time". In: *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2018)*. 2018, pp. 495–514. DOI: 10.1137/1.9781611975031.33.

[12] Mukulika Ghosh, Nancy M. Amato, Yanyan Lu, and Jyh-Ming Lien. "Fast approximate convex decomposition using relative concavity". In: *Comput. Aided Des.* 45.2 (2013), pp. 494–504. DOI: 10.1016/j.cad.2012.10.032.

[13] Leonidas J. Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert Endre Tarjan. "Linear-Time Algorithms for Visibility and Shortest Path Problems Inside Triangulated Simple Polygons". In: *Algorithmica* 2 (1987), pp. 209–233. DOI: 10.1007/BF01840360.

[14] John Hershberger. "An Optimal Visibility Graph Algorithm for Triangulated Simple Polygons". In: *Algorithmica* 4.1 (1989), pp. 141–155. DOI: 10.1007/BF01553883.

[15] John Hershberger and Subhash Suri. "A Pedestrian Approach to Ray Shooting: Shoot a Ray, Take a Walk". In: *J. Algorithms* 18.3 (1995), pp. 403–431. DOI: 10.1006/jagm.1995.1017.

[16] Arthur C. Hsu. "The expected distance between two random points in a polygon". MA thesis. Massachusetts Institute of Technology, 1990. URL: https://dspace.mit.edu/bitstream/handle/1721.1/14232/22504138-MIT.pdf.

[17] Rolf Klein. *Algorithmische Geometrie*. 1996.

[18] Der-Tsai Lee. "Proximity and reachability in the plane". PhD thesis. University of Illinois at Urbana-Champaign, 1978.

[19] Jyh-Ming Lien and Nancy M. Amato. "Approximate convex decomposition of polygons". In: *Comput. Geom.* 35.1-2 (2006), pp. 100–123. DOI: 10.1016/j.comgeo.2005.10.005.

[20] Andrea Mantler and Jack Snoeyink. "Intersecting Red and Blue Line Segments in Optimal Time and Precision". In: *Discrete and Computational Geometry, Japanese Conference (JCDCG 2000)*. 2000, pp. 244–251. DOI: 10.1007/3-540-47738-1_23.

[21] Maple 2019.2, Maplesoft, a division of Waterloo Maple Inc., Waterloo, Ontario. Maple is a trademark of Waterloo Maple Inc.

[22] Larry F. Palazzi and Jack Snoeyink. "Counting and Reporting Red/Blue Segment Intersections". In: *CVGIP Graph. Model. Image Process.* 56.4 (1994), pp. 304–310. DOI: 10.1006/cgip.1994.1027.

[23] Günter Rote. "The degree of convexity". In: *Abstracts of the 29th European Workshop on Computational Geometry (EuroCG 2013)*. 2013, pp. 69–72. URL: http://page.mi.fu-berlin.de/rote/Papers/pdf/The+degree+of+convexity.pdf.

[24] Micha Sharir and Amir Schorr. "On Shortest Paths in Polyhedral Spaces". In: *SIAM J. Comput.* 15.1 (1986), pp. 193–215. DOI: 10.1137/0215014.

[25] Helman Stern. "Polygonal entropy: A convexity measure". In: *Pattern Recognit. Lett.* 10.4 (1989), pp. 229–235. DOI: 10.1016/0167-8655(89)90093-7.

[26] Emo Welzl. "Constructing the Visibility Graph for n-Line Segments in $O(n^2)$ Time". In: *Inf. Process. Lett.* 20.4 (1985), pp. 167–171. DOI: 10.1016/0020-0190(85)90044-4.

A Counterexample to a claim from [5]

The algorithm for computing the Beer-index of a polygon with holes from [5] works as follows. We consider a convex polygon P and a family $H_1, \ldots, H_k$ of obstacles, and we want to find the probability that for two points p, q chosen uniformly and independently at random in P, the line segment pq is disjoint from all obstacles. Note that this is just a slight reformulation of the problem of computing the Beer-index of a polygon with holes.

We partition $P \setminus \bigcup H_i$ into trapezoids $T_1, \ldots, T_t$. We now compute the Beer-index as the sum of two entities: (i) the probability that two random points p, q are in the same trapezoid, and (ii) the probability that two random points p, q are in different trapezoids and the segment pq is disjoint from all obstacles. For the second case, we compute the contribution for each pair T_i, T_j of trapezoids and sum over all these pairs.

Let us consider a fixed pair T_i, T_j of trapezoids. For each pair s_1^i, s_2^i of two segments of T_i and each pair s_1^j, s_2^j of two segments of T_j, we do as follows. Using geometric duality, we map each line intersecting all four segments $s_1^i, s_2^i, s_1^j, s_2^j$ to a point in the dual space, and these points together form a convex polygon L^*. Likewise, the dual points of the set of lines intersecting an obstacle H_l form an "hourglass-shaped" unbounded region H_l^* in the dual space. We now compute the overlay of L^* with all the regions H_l^*. This results in an arrangement $\mathcal{S}^*$ of the plane. Each cell C of $\mathcal{S}^*$ contained in L^* corresponds to a set of lines in primal space that all cross the four segments $s_1^i, s_2^i, s_1^j, s_2^j$. Furthermore, the visibility from T_i to T_j is either not blocked along any of these lines or blocked for all the lines by one or more obstacles. It is then described in the paper how to compute the contribution to the Beer-index of all pairs of points $p \in T_1$, $q \in T_2$, where the dual point $l(pq)^*$ is in C. The time it takes to compute this contribution is proportional to the complexity of C.

Running over all cells C contained in L^*, we obtain that computing the contribution for all pairs of points $p \in T_1$, $q \in T_2$, where $l(pq)$ intersects $s_1^i, s_2^i, s_1^j, s_2^j$, can be done in $O(n^2)$ time, where n is the total complexity of P and the obstacles. Since there are $O(1)$ choices of the four segments $s_1^i, s_2^i, s_1^j, s_2^j$, the contribution from all pairs $p \in T_1$, $q \in T_2$ can then also be computed in time $O(n^2)$.

It is then erroneously claimed in Lemma 3 that the total size of the arrangements $\mathcal{S}^*$, as we sum over all pairs of trapezoids T_i, T_j, is also $O(n^2)$. This is not true. For $\Omega(n^2)$ of the pairs, the complexity may be $\Omega(n^2)$, resulting in a total complexity of $\Omega(n^4)$. This can even be the case for simple polygons, as the

polygon in Figure 12 shows. The polygon is simply a rectangle with some spikes removed along the top edge. The obstacles will therefore be the removed spikes. Note that a line can intersect any consecutive interval of obstacles and no other obstacle, so there are $\Omega(n^2)$ subsets of obstacles that can be intersected. The arrangement of the hour-glasses H_l^* therefore has complexity $\Omega(n^2)$. It then follows that for $\Omega(n^2)$ pairs of trapezoids T_i, T_j, the arrangement $\mathcal{S}^*$ has complexity $\Omega(n^2)$, resulting in a running time of $\Omega(n^4)$.

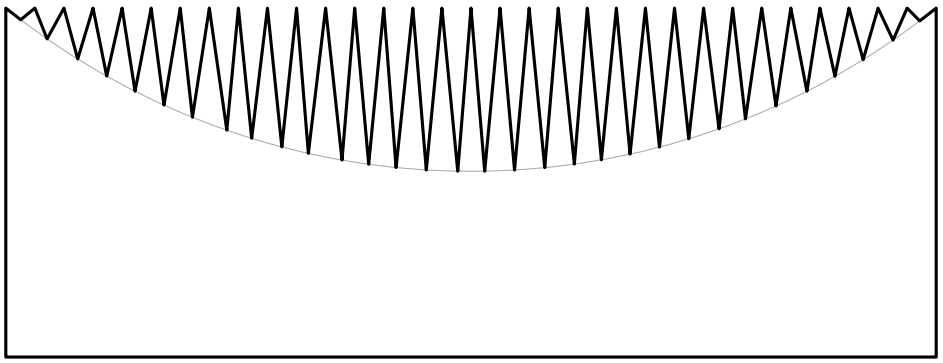


Figure 12: For polygons of this type, the algorithm from [5] has running time $\Omega(n^4)$.