Improved Bounds for Shortest Paths in Dense Distance Graphs

Paweł Gawrychowski
Institute of Computer Science, University of Wrocław, Poland
gawry@cs.uni.wroc.pl

Adam Karczmarz[1]
University of Warsaw, Poland
a.karczmarz@mimuw.edu.pl

Abstract

We study the problem of computing shortest paths in so-called *dense distance graphs*, a basic building block for designing efficient planar graph algorithms. Let G be a plane graph with a distinguished set ∂G of *boundary vertices* lying on a constant number of faces of G. A distance clique of G is a complete graph on ∂G encoding all-pairs distances between these vertices. A dense distance graph is a union of possibly many unrelated distance cliques.

Fakcharoenphol and Rao [7] proposed an efficient implementation of Dijkstra's algorithm (later called *FR-Dijkstra*) computing single-source shortest paths in a dense distance graph. Their algorithm spends $O(b\log^2 n)$ time per distance clique with b vertices, even though a clique has b^2 edges. Here, n is the total number of vertices of the dense distance graph. The invention of FR-Dijkstra was instrumental in obtaining such results for planar graphs as nearly-linear time algorithms for multiple-source-multiple-sink maximum flow and dynamic distance oracles with sublinear update and query bounds.

At the heart of FR-Dijkstra lies a data structure updating distance labels and extracting minimum labeled vertices in $O(\log^2 n)$ amortized time per vertex. We show an improved data structure with $O\left(\frac{\log^2 n}{\log^2 \log n}\right)$ amortized bounds. This is the first improvement over the data structure of Fakcharoenphol and Rao in more than 15 years. It yields improved bounds for all problems on planar graphs, for which computing shortest paths in dense distance graphs is currently a bottleneck.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases shortest paths, dense distance graph, planar graph, Monge matrix

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.61

Related Version A full version of this paper is available at [11], `https://arxiv.org/abs/1602.07013`.

Acknowledgements We thank Piotr Sankowski for helpful discussions. The first author also thanks Oren Weimann and Shay Mozes for discussions about Monge matrices and FR-Dijkstra.

[1] Supported by the grants 2014/13/B/ST6/01811 and 2017/24/T/ST6/00036 of the Polish National Science Center.

1 Introduction

Finding a truly subquadratic, strongly polynomial algorithm for many of the most basic real-weighted graph problems like the single-source shortest paths or the maximum flow on sparse digraphs seems to be very difficult. However, the situation changes significantly if we restrict ourselves to planar digraphs, which constitute an important class of sparse graphs. In this regime the ultimate goal is to obtain linear or almost linear time complexity.

In their breakthrough paper, Fakcharoenphol and Rao gave the first nearly-linear time algorithm for single-source shortest paths in real-weighted planar graphs [7]. Their algorithm had $O(n\log^3 n)$ time complexity. Although their upper bound was eventually improved to $O\left(n\frac{\log^2 n}{\log\log n}\right)$ by Mozes and Wulff-Nilsen [22], the techniques introduced in [7] proved very useful in obtaining not only nearly-linear time algorithms for other static planar graph problems, but also first sublinear dynamic algorithms for shortest paths and maximum flows.

A major contribution of Fakcharoenphol and Rao was introducing the general concept of a *dense distance graph*. Let G be a real-weighted plane digraph and let ∂G denote some subset of its vertices, called *boundary vertices*, such that there exist $\ell = O(1)$ faces $f_1,\ldots,f_\ell$ of G satisfying $\partial G \subseteq V(f_1)\cup\ldots V(f_\ell)$. Such graphs with a topologically nice boundary typically emerge after decomposing a plane graph using a cycle separator. For example, by using a cycle separator of Miller [19], one can decompose any n-vertex triangulated plane graph H into two subgraphs H_{in} and H_{out} such that (i) $H_{\text{in}}\cup H_{\text{out}} = H$, (ii) H_{in} and H_{out} are smaller than H by a constant factor, (iii) the set $\partial H_{\text{in}} = \partial H_{\text{out}} = V(H_{\text{in}})\cap V(H_{\text{out}})$ has size $O(\sqrt{n})$ and lies both on a single face of H_{in} and on a single face of H_{out}.

We define a *distance clique* of G, denoted $\text{DC}(G)$, to be a complete graph on ∂G such that the weight of an edge uv is equal to the length of the shortest path from u to v in G. A dense distance graph is a union of possibly many unrelated distance cliques.

We note that such a definition of a dense distance graph (also used in [23]) is a bit more general than that of Fakcharoenphol and Rao [7], who defined it only with respect to a recursive decomposition of G using cycle-separators. In fact, subsequently dense distance graphs have been also defined a bit differently with respect to so-called r-divisions [13], and even the two sides of a cycle-separator [15] (i.e., $\text{DC}(H_{\text{in}})\cup\text{DC}(H_{\text{out}})$ in the above example). The definition we assume in this paper captures all these cases.

Suppose we are given q distance cliques $\text{DC}(G_1),\ldots,\text{DC}(G_q)$ explicitly. Let $\text{DDG} = \bigcup_{i=1}^{q}\text{DC}(G_i)$, $V = \partial G_1\cup\ldots\cup\partial G_q$ and $n = |V|$. Clearly, DDG has $\sum_{i=1}^{q}|\partial G_i|^2$ edges in total. Fakcharoenphol and Rao showed how to compute single-source shortest paths in such graph DDG with non-negative edge weights in only $O\left(\sum_i |\partial G_i|\log^2 n\right)$ time, i.e., for each $\text{DC}(G_i)$ one only needs to spend time nearly-linear in the number of *vertices* of $\text{DC}(G_i)$, as opposed to its number of edges, i.e., $|\partial G_i|^2$. Their method is often called the *FR-Dijkstra*, as it follows the overall approach of Dijkstra's algorithm. Whereas Dijkstra's algorithm uses a priority queue to maintain its distance labels and extract a non-visited vertex with minimum label, a much more sophisticated data structure is used in FR-Dijkstra. This data structure is capable of relaxing many edges in a single step, by leveraging the fact that certain submatrices of the adjacency matrix of a distance clique are *Monge matrices.*

Applications of Dense Distance Graphs and FR-Dijkstra. Fakcharoenphol and Rao originally employed FR-Dijkstra to construct their dense distance graph recursively and answer distance queries on it. However, the applications of FR-Dijkstra proved much broader and thus it has become an important planar graph primitive used to obtain numerous breakthrough results in recent years. We briefly cover the most important of these results below.

The dense distance graphs and FR-Dijksta have been used to break the long-standing $O(n \log n)$ barrier for computing minimal s,t-cuts [12] in undirected planar graphs and global min-cuts in both undirected [17] and directed [20] planar graphs. Borradaile et al. [5] developed an oracle answering arbitrary min s,t-cut queries in an weighted undirected planar graph after only near-linear preprocessing. This result has been later generalized to bounded-genus graphs [3], thus proving the usefulness of FR-Dijkstra in more general graph classes.

The most sophisticated applications of FR-Dijkstra are probably those related to computing maximum flow in directed planar graphs. Borradaile et al. [4] gave a nearly-linear time max-flow algorithm for the case of multiple source and multiple sinks and maximum bipartite matching. Later, Łącki et al. [18] gave a nearly-linear time algorithm computing the maximum flow values between a specified source and all possible sinks.

Most recently, Asathulla et al. [2] used FR-Dijkstra to break through the $O(n^{3/2})$ barrier for minimum-cost bipartite weighted matching with integer weights. Cabello [6] showed the first truly subquadratic algorithm for computing a diameter of a weighted planar graph. Even though it mainly builds on a new concept of additively weighted Voronoi diagrams for planar graphs, dense distance graphs and FR-Dijkstra are still used extensively in his work. The diameter algorithm was later improved by Gawrychowski et al. [9] to run in $O(n^{5/3} \operatorname{polylog} n)$. Currently, [9] does not require FR-Dijkstra, but it seems that using it would be again required if one gave a more efficient Voronoi diagrams construction algorithm for planar graphs. Last but not least, FR-Dijkstra has been instrumental to obtaining virtually all *exact* dynamic algorithms for shortest paths, maximum flows and minimum cuts in planar graphs, with sublinear update/query bounds [7, 12, 13, 14, 17].

Significance. Dense distance graphs are pivotal in designing efficient planar graph algorithms, and therefore obtaining fine-grained bounds for computing and manipulating them is an important direction. Although a better algorithm (in comparison to the recursive method of [7]) running in $O((|V| + |\partial G|^2) \log n)$ time has been proposed for computing a distance clique [14], improving the FR-Dijkstra itself proved very challenging and no progress over [7] has been made so far in the most general setting that we study.

Related Work. For the important case of a *dense distance graph over an r-division*, i.e., when the individual graphs G_i are the pieces of an *r-division with few holes* of a single planar graph (see e.g., [16]), Mozes et al. [21] gave an algorithm for computing single source shortest paths in $O\left(\frac{n}{\sqrt{r}} \log^2 r\right)$ time. The original FR-Dijkstra runs in $O\left(\frac{n}{\sqrt{r}} \log n \log r\right)$ time in that case. Hence, [21] does not improve over it in the case of $r = \operatorname{poly} n$, which emerges in many important applications, e.g., [2, 3, 4, 13, 18]. However, dense distance graphs over r-divisions with $r = \operatorname{polylog}(n)$ have also found applications, most notably in $O(n \log \log n)$ algorithms for minimum cuts [12, 17, 20]. Computing shortest paths in dense distance graphs is not a bottleneck in those algorithms, though. For other applications of dense distance graphs over r-divisions with small r, consult [21].

Our Contribution. In this paper we show an algorithm for computing single-source shortest paths in a DDG in $O\left(\sum_{i=1}^{q} |\partial(G_i)| \frac{\log^2 n}{\log^2 \log n}\right)$ time, which is asymptotically faster than FR-Dijkstra in *all* cases. Specifically, for a dense distance graph defined over an r-division, the algorithm runs in $O\left(\frac{n}{\sqrt{r}} \frac{\log^2 n}{\log^2 \log n}\right)$ time.

We treat the problem of computing shortest paths in DDG from a purely data-structural perspective. At a high level, instead of developing an entirely new shortest paths algorithm, we propose a new data structure for maintaining distance labels and extracting minimum labeled vertices in amortized $O\left(\frac{\log^2 n}{\log^2 \log n}\right)$ time, as opposed to $O(\log^2 n)$ time in [7].

In [7], a distance clique is first partitioned into *square* Monge matrices, each handling a subset of its edges. For any such matrix, a separate data structure is used for relaxing the corresponding edges and extracting the labels possibly induced by these edge relaxations. Recall that in the case of Dijkstra's algorithm, the improvement from $O(m \log n)$ to $O(m + n \log n)$ is obtained by noticing that relaxing edges is cheaper than extracting minimum labeled vertices. Consequently, one can use a Fibonacci heap [8] in place of a binary heap. We show that in the case of the data structure originally used in [7] for handling Monge matrices, the situation is in a sense the opposite: label extractions can be made cheaper than edge relaxations. We make use of this fact by proposing a biased scheme of partitioning distance cliques into *rectangular* (as opposed to square) Monge matrices, different than in [7]. Whereas in [7], the partition follows from a very natural idea of splitting face boundary into halves, our partition is tailored to exploit this asymmetry between the cost of processing a row and the cost of processing a column.

Our result implies an immediate improvement by a factor of $O(\log^2 \log n)$ in the time complexity for a number of planar digraph problems such as multiple-source multiple-sink maximum flows, maximum bipartite matching [4], single-source all-sinks maximum flows [18], for which the best known time bounds were $O(n \log^3 n)$, i.e., already nearly-linear. It also yields polylog-logarithmic speed-ups to both preprocessing and query/update algorithms of dynamic algorithms for shortest paths and max-flows [12, 13, 14]. More generally, we make polylog-logarithmic improvements to all previous results (such as [2]). for which the bottleneck of the best known algorithm is computing shortest paths in a dense distance graph. A more detailed discussion on the implications of our result and on how FR-Dijkstra is used in different algorithms for planar graphs can be found in the full version [11].

It should be noted that for small values of r, such as $r = \text{polylog}(n)$, our algorithm does not improve on [21] for the case of a dense distance graph over an r-division.

Model of Computation. We assume the standard word-RAM model with word size $\Omega(\log n)$. However, we stress that our algorithm works in the very general case of *real* edge lengths, i.e., we are only allowed to perform arithmetical operations on lengths and compare them.

Outline of the Paper. In Section 2 we introduce the matrix notation that we use and state some important properties of Monge matrices. In Section 3 we give an overview of our shortest paths algorithm and also discuss the main ideas behind the improved data structure for reporting column minima of a staircase Monge matrix in an online fashion.

In Sections 4, 5 and 6 we develop the increasingly more powerful data structures for reporting column minima in online Monge matrices. Each of these data structures is used in a black-box manner in the following section.

Due to space limitations, many technical details, most proofs and discussion of the applications can be found in the full version [11].

2 Monge Matrices and Their Minima

In this paper we define a *matrix* to be a partial function $\mathcal{M} : R \times C \to \mathbb{R}$, where R (called *rows*) and C (called *columns*) are some totally ordered finite sets. Set $R = \{r_1, \ldots, r_k\}$ and $C = \{c_1, \ldots, c_l\}$, where $r_1 \leq \ldots \leq r_k$ and $c_1 \leq \ldots \leq c_l$. If for $r_i, r_j \in R$ we have $r_i \leq r_j$, we also say that r_i is (weakly) *above* r_j and r_j is (weakly) *below* r_i. Similarly, when c_i, c_j we have $c_i < c_j$, we say that c_i is *to the left* of c_j and c_j is *to the right* of c_i.

For some matrix $\mathcal{M}$ defined on rows R and columns C, for $r \in R$ and $c \in C$ we denote by $\mathcal{M}_{r,c}$ an *element* of $\mathcal{M}$. An element is the value of $\mathcal{M}$ on pair (r, c), if defined.

For $R' \subseteq R$ and $C' \subseteq C$ we define $\mathcal{M}(R', C')$ to be a *submatrix* of $\mathcal{M}$. $\mathcal{M}(R', C')$ is a partial function on $R' \times C'$ satisfying $\mathcal{M}(R', C')_{r,c} = \mathcal{M}_{r,c}$ for any $(r, c) \in R' \times C'$ such that $\mathcal{M}_{r,c}$ is defined. We sometimes abuse this notation by writing $\mathcal{M}(R', c')$ or $\mathcal{M}(r', C')$ when R' or C' are single-element, i.e., when $R' = \{r'\}$ or $C' = \{c'\}$.

The *minimum* of a matrix $\min\{\mathcal{M}\}$ is defined as the minimum value of the partial function $\mathcal{M}$. The *column minimum* of $\mathcal{M}$ in column c is defined as $\min\{\mathcal{M}(R, \{c\})\}$.

We call a matrix $\mathcal{M}$ *rectangular* if $\mathcal{M}_{r,c}$ is defined for every $r \in R$ and $c \in C$. A matrix is called *staircase* (*flipped staircase*) if $|R| = |C|$ and $\mathcal{M}_{r_i,c_j}$ is defined iff $i \leq j$ ($i \geq j$ resp.).

Finally, a *subrectangle* of $\mathcal{M}$ is a rectangular matrix $\mathcal{M}(\{r_a, \ldots, r_b\}, \{c_x, \ldots, c_y\})$ for $1 \leq a \leq b \leq k$, $1 \leq x \leq y \leq l$. We define a *subrow* to be a subrectangle with a single row.

For a matrix $\mathcal{M}$ and a function $d : R \to \mathbb{R}$, define the *offset matrix* $\text{off}(\mathcal{M}, d)$ to be a matrix $\mathcal{M}'$ such that for all r, c such that $\mathcal{M}_{r,c}$ is defined, we have $\mathcal{M}'_{r,c} = \mathcal{M}_{r,c} + d(r)$.

We say that a matrix $\mathcal{M}$ with rows R and columns C is a *Monge matrix*, if for each $r_1, r_2 \in R$, $r_1 \leq r_2$ and $c_1, c_2 \in C$, $c_1 \leq c_2$ such that all elements $\mathcal{M}_{r_1,c_1}, \mathcal{M}_{r_1,c_2}, \mathcal{M}_{r_2,c_1}, \mathcal{M}_{r_2,c_2}$ are defined, the *Monge property* holds, i.e., we have

$$\mathcal{M}_{r_2,c_1} + \mathcal{M}_{r_1,c_2} \leq \mathcal{M}_{r_1,c_1} + \mathcal{M}_{r_2,c_2}.$$

▶ **Fact 1.** *Let $\mathcal{M}$ be a Monge matrix. For any $R' \subseteq R$ and $C' \subseteq C$, $\mathcal{M}(R', C')$ is also a Monge matrix.*

▶ **Fact 2.** *Let $\mathcal{M}$ be a rectangular Monge matrix. Assume that for some $c \in C$ and $r \in R$, $\mathcal{M}_{r,c}$ is a column minimum of c. Then, for each column c^- to the left of c, there exists a row $r^- \geq r$, such that $\mathcal{M}_{r^-,c^-}$ is a column minimum of c^-. Similarly, for each column c^+ to the right of c, there exists a row $r^+ \leq r$, such that $\mathcal{M}_{r^+,c^+}$ is a column minimum of c^+.*

▶ **Fact 3.** *Let $\mathcal{M}$ be a Monge matrix and let $d : R \to \mathbb{R}$. Then off$(\mathcal{M}, d)$ is also Monge.*

▶ **Fact 4.** *Let $\mathcal{M}$ be a rectangular Monge matrix and assume R is partitioned into disjoint blocks $\mathcal{R} = R_1, \ldots, R_a$ such that each R_i is a contiguous group of subsequent rows and each R_i is above R_{i+1}. Assume also that the set C is partitioned into blocks $\mathcal{C} = C_1, \ldots, C_b$ so that C_i is to the left of C_{i+1}. Then, a matrix $\mathcal{M}'$ with rows $\mathcal{R}$ and columns $\mathcal{C}$ defined as $\mathcal{M}'_{R_i,C_j} = \min\{\mathcal{M}(R_i, C_j)\}$, is also a Monge matrix.*

▶ **Fact 5.** *Let $\mathcal{M}$ be a rectangular Monge matrix. Let $r \in R$ and $C = \{c_1, \ldots, c_l\}$. The set of columns $C_r \in C$ having one of their column minima in row r is contiguous, that is either $C_r = \emptyset$ or $C_r = \{c_a, \ldots, c_b\}$ for some $1 \leq a \leq b \leq l$.*

3 Shortest Paths in a Dense Distance Graph: an Overview

Recall that we are explicitly given q graphs $\text{DC}(G_1), \ldots, \text{DC}(G_q)$, such that each $\text{DC}(G_i)$ is a complete digraph encoding the distances between boundary vertices ∂G_i of a plane digraph G_i. Additionally, we assume that ∂G_i is distributed into some $O(1)$ faces of G_i. We also assume that the distances between the boundary vertices of G_i are non-negative.

Let $\text{DDG} = \text{DC}(G_1) \cup \ldots \cup \text{DC}(G_q)$, $V = \partial G_1 \cup \ldots \cup \partial G_q$ and $n = |V|$. Our goal is to find an efficient algorithm for computing single-source shortest paths in DDG.

As the graphs $\mathrm{DC}(G_i)$ are given explicitly, we can assume that we are allowed to preprocess each $\mathrm{DC}(G_i)$ once in time asymptotically no more than the time used to construct it, which is clearly $\Omega(|\partial G_i|^2)$. To the best of our knowledge, in all known applications this time is $\Theta((|V(G_i)| + |\partial G_i|^2)\log|V(G_i)|)$, which is the running time of Klein's algorithm [14]. After the preprocessing stage, we may need to handle multiple shortest-path queries.

In order to obtain the speedup over FR-Dijkstra we use a subtle combination of techniques. The single-source shortest paths in DDG are computed with an optimized implementation of Dijkstra's algorithm. Recall that Dijkstra's algorithm run from the source s grows a set S of *visited* vertices of the graph such that the lengths $d(v)$ of the shortest paths $s \to v$ for $v \in S$ are already known. Initially $S = \{s\}$ and we repeatedly choose a vertex $y \in V \setminus S$ such that the value (a distance estimate) $z(y) := \min_{x \in S}\{d(x) + \ell(x, y) : (x, y) \in E\}$ is the smallest. The vertex y is then added to S with $d(y) = z(y)$. The vertices $y \in V \setminus S$ are typically stored in a priority queue with keys $z(y)$, which allows to choose the best y efficiently.

Since the vertices of ∂G_i lie on $O(1)$ faces of a planar digraph G_i, we can exploit the fact that many of the shortest paths represented by $\mathrm{DC}(G_i)$ have to cross. Formally, this is captured by the following lemma. Denote by $\mathrm{DC}(G_i)[u, v]$ the weight of uv in $\mathrm{DC}(G_i)$.

▶ **Lemma 1** ([22]). *Each $DC(G_i)$ can be decomposed into $O(1)$ (possibly flipped) staircase Monge matrices D_i of at most $|\partial G_i|$ rows and columns. For each $u, v \in \partial G_i$ we have:*

- *for each $\mathcal{M} \in D_i$ such that $\mathcal{M}_{u,v}$ is defined, $\mathcal{M}_{u,v} \geq DC(G_i)[u, v]$.*
- *there exists $\mathcal{M} \in D_i$ such that $\mathcal{M}_{u,v}$ is defined and $\mathcal{M}_{u,v} = DC(G_i)[u, v]$.*

The decomposition can be computed in $O(|\partial G_i|)^2)$ time if ∂G_i is a subset of a single face of G_i and in $O((|V(G_i)| + |\partial G_i|^2)\log|V(G_i)|)$ time otherwise.

In other words, the adjacency matrix of $\mathrm{DC}(G_i)$ can be partitioned into a constant number of staircase Monge matrices. Consequently, a natural approach to maintaining the minimum distance estimate $z(y)$, for $y \notin S$, is to split the work needed to accomplish this task between the individual matrices $\mathcal{M} \in \bigcup_{i=1}^{q} D_i$ that encode the edges of DDG. Then, it is sufficient to design a data structure reporting the column minima of the offset matrix $\mathrm{off}(\mathcal{M}, d)$ in an online fashion. Specifically, the data structure has to handle *row activations* intermixed with extractions of the column minima in non-decreasing order. Once Dijkstra's algorithm establishes the distance $d(v)$ to some vertex v, the row of $\mathrm{off}(\mathcal{M}, d)$ corresponding to v is activated and becomes available to the data structure. This row contains values $d(v)+\ell(v, w)$, where $\ell(v, w)$ is, by Lemma 1, no less than the length of the edge vw in DDG. Alternatively, a minimum in some column corresponding to v (in the revealed part of $\mathrm{off}(\mathcal{M}, d)$) may be used by Dijkstra's algorithm to establish a new distance label $z(v) = d(v)$, even though not all rows of $\mathrm{off}(\mathcal{M}, d)$ have been revealed so far. In this case, we can guarantee that all the inactive rows of $\mathrm{off}(\mathcal{M}, d)$ contain entries not smaller than $d(v)$ and hence we can safely extract the column minimum of $\mathrm{off}(\mathcal{M}, d)$.

Such an approach was also used by Fakcharoenphol and Rao [7] and Mozes et al. [21], who both dealt with staircase Monge matrices by using a recursive partition into *square* Monge matrices, which are easier to handle. In particular, Fakcharoenphol and Rao showed that a sequence of row activations and column minima extractions can be performed on an $m \times m$ square Monge matrix in $O(m \log m)$ time. The recursive partition assigns each row and column to $O(\log|\partial G_i|)$ square Monge matrices. As a result, in [7], the total time for handling all the square matrices is $O(|\partial G_i| \log^2 |\partial G_i|)$. The details and the pseudocode of the above shortest path algorithm can be found in the full version [11].

The Data Structure. Developing an improved data structure reporting the column minima of an online offset staircase Monge matrix is the main contribution of this paper. This goal is achieved in three steps, presented in the following three sections in a bottom-up fashion. Below we sketch the main ideas behind these steps.

Our first component is a refined data structure for handling row activations and column minima extractions on a *rectangular* Monge matrix, described in Section 4. We show a data structure supporting any sequence of operations on a $k \times l$ matrix in $O\left(k \frac{\log m}{\log \log m} + l \log m\right)$ total time, where $m = \max(k, l)$. In comparison to [7], we do not map all the columns to active rows containing the current minima. Instead, the columns are assigned *potential row sets* of bounded size that are guaranteed to contain the "currently optimal" rows. This relaxed notion allows to remove the seemingly unavoidable binary search at the heart of [7] and instead use the SMAWK algorithm [1] to split the potential row sets once they become too large. The maintenance of a priority queue used for reporting the column minima in order is possible with the recent efficient data structure supporting subrow minimum queries in Monge matrices [10] and priority queues with $O(1)$ time Decrease-Key operation [8].

The second step is to relax the requirements posed on a data structure handling rectangular $k \times l$ Monge matrices. It is motivated by the following observation. Let $\Delta > 0$ be an integer. Imagine we have found the minima of l/Δ evenly spread, *pivot* columns $c_1, \ldots, c_{l/\Delta}$. Denote by $r_1, \ldots, r_{l/\Delta}$ some rows containing the corresponding minima. A well-known property of Monge matrices implies that for any column c' lying between c_i and c_{i+1}, we only have to look for a minimum of c' in rows $r_i, \ldots, r_{i+1}$. Thus, the minima in the remaining columns can be found in $O(k\Delta + l)$ total time. In Section 5 we show how to adapt this idea to an online setting that fits our needs. The columns are partitioned into $O(l/\Delta)$ *blocks* of size at most Δ. Each block is conceptually contracted to a single column: an entry in row r is defined as the minimum in row r over the contracted columns. For sufficiently small values of Δ, such a minimum can be computed in $O(1)$ time using the data structure of [10]. Locating a block minimum can be seen as an introduction of a new pivot column. We handle the block matrix with the data structure of Section 4 and prove that the total time needed to correctly report all the column minima is $O\left(k \frac{\log m}{\log \log m} + k\Delta + l + \frac{l}{\Delta} \log m\right)$. In particular, for $\Delta = \log^{1-\epsilon} m$, this bound becomes $O\left(k \frac{\log m}{\log \log m} + l \log^{\epsilon} m\right)$.

Finally, in Section 6 we exploit the asymmetry of per-row and per-column costs of the developed block data structure for rectangular matrices by using a different partition of a staircase Monge matrix. Our partition is biased towards columns, i.e., the matrix is split into *rectangular* (as opposed to square) Monge matrices, each with roughly polylogarithmically more columns than rows. Consequently, the total number of rows in these matrices is $O\left(|\partial G_i| \frac{\log |\partial G_i|}{\log \log |\partial G_i|}\right)$, whereas the total number of columns is only slightly larger, i.e., $O\left(|\partial G_i| \log^{1+\epsilon} |\partial G_i|\right)$. This yields a data structure handling staircase Monge matrices in $O\left(|\partial G_i| \frac{\log^2 |\partial G_i|}{\log^2 \log |\partial G_i|}\right)$ total time. By plugging this data structure into our shortest path algorithm, we obtain the following theorem.

▶ **Theorem 2.** *The single-source shortest paths computations in DDG can be performed in* $O\left(\sum_{i=1}^{q} |\partial G_i| \frac{\log^2 n}{\log^2 \log n}\right)$ *time. The required preprocessing time per each* G_i *is* $O(|\partial G_i|^2)$ *if* ∂G_i *lies on a single face of* G_i*, and* $O\left((|V(G_i)| + |\partial G_i|^2) \log |V(G_i)|\right)$ *otherwise.*

4 Online Column Minima of a Rectangular Offset Monge Matrix

Let $\mathcal{M}_0$ be a rectangular $k \times l$ Monge matrix. Let $R = \{r_1, \ldots, r_k\}$ and $C = \{c_1, \ldots, c_l\}$ be the sets of rows and columns of $\mathcal{M}_0$, respectively. Set $m = \max(k, l)$.

Let $d : R \to \mathbb{R}$ be an offset function and set $\mathcal{M} = \mathrm{off}(\mathcal{M}_0, d)$. By Fact 3, $\mathcal{M}$ is also a Monge matrix. Our goal is to design a data structure capable of reporting the column minima of $\mathcal{M}$ in increasing order of their values. However, the function d is not entirely revealed beforehand, as opposed to the matrix $\mathcal{M}_0$. There is an initially empty, growing set $\overline{R} \subseteq R$ containing the rows for which $d(r)$ is known. Alternatively, the set $\overline{R}$ can be seen as "active" rows of $\mathcal{M}$ that can be accessed by the data structure. There is also a set $\overline{C} \subseteq C$ containing the remaining columns for which we have not reported the minima yet. Initially, $\overline{C} = C$ and $\overline{C}$ shrinks over time. We also provide a mechanism to guarantee that the rows that have not been revealed do not influence the smallest of the column minima of $\mathcal{M}(R, \overline{C})$.

The exact set of operations we support is the following:

- Activate-Row(r), where $r \in R \setminus \overline{R}$ – add r to the set $\overline{R}$.
- Lower-Bound() – compute the number $\min\{\mathcal{M}(\overline{R}, \overline{C})\}$.
- Ensure-Bound-And-Get() – inform the data structure that we indeed have $\min\{\mathcal{M}(R \setminus \overline{R}, C)\} \geq \min\{\mathcal{M}(\overline{R}, \overline{C})\} =$ Lower-Bound(), that is, the smallest element of $\mathcal{M}(R, \overline{C})$ does not depend on the values of $\mathcal{M}$ located in rows $R \setminus \overline{R}$.
  Observe that such claim implies that for some column $c \in \overline{C}$ we have $\min\{\mathcal{M}(R, c)\} = \min\{\mathcal{M}(\overline{R}, \overline{C})\}$, which in turn means that we are able to find the minimum element in column c. The function returns any such c and removes it from the set $\overline{C}$.
- Current-Min-Row(c), where $c \in C$ – compute r, where $r \in \overline{R}$ is a row such that $\min\{\mathcal{M}(\overline{R}, c)\} = \mathcal{M}_{r,c}$. If $\overline{R} = \emptyset$, return **nil**. Note that c is not necessarily in $\overline{C}$.
  Additionally, we require Current-Min-Row to have the following property: once the column c is moved out of $\overline{C}$, Current-Min-Row(c) always returns the same row. Moreover, for $c_1, c_2 \in C$, $c_1 < c_2$, we have Current-Min-Row(c_1) $\geq$ Current-Min-Row(c_2).

Note that Activate-Row increases the size of $\overline{R}$ and thus cannot be called more than k times. Analogously, Ensure-Bound-And-Get decreases the size of $\overline{C}$ so it cannot be called more than l times. Actually, in order to reveal all the column minima with this data structure, the operation Ensure-Bound-And-Get has to be called *exactly* l times.

4.1 The Components

The Subrow Minimum Query Data Structure. Given $r \in \overline{R}$ and a, b, $1 \leq a \leq b \leq l$, a subrow minimum query $S(r, a, b)$ computes a column $c \in \{c_a, \ldots, c_b\}$ such that $\mathcal{M}_{r,c} = \min\{\mathcal{M}(r, \{c_a, \ldots, c_b\})\}$. We use the following theorem of Gawrychowski et al. [10].

▶ **Theorem 3** ([10]). *Given a $k \times l$ rectangular Monge matrix $\mathcal{M}$, a data structure supporting subrow minimum queries in $O(\log\log(k + l))$ time can be constructed in $O(l \log k)$ time.*

Recall that $\mathcal{M} = \mathrm{off}(\mathcal{M}_0, d)$. Adding the offset $d(r)$ to all the elements in row r of $\mathcal{M}_0$ does not change the relative order of elements in row r. Hence, the answer to a subrow minimum query $S(r, a, b)$ in $\mathcal{M}$ is the same as the answer to $S(r, a, b)$ in $\mathcal{M}_0$.

Therefor, by building a data structure of Theorem 3 for $\mathcal{M}_0$ we can answer any subrow minimum query in $\mathcal{M}$ in $O(\log\log m)$ time.

The Column Groups. The set C is internally partitioned into disjoint, contiguous *column groups* $\mathcal{C}_1, \ldots, \mathcal{C}_q$ (where $\mathcal{C}_1$ is the leftmost and $\mathcal{C}_q$ is the rightmost), so that $\bigcup_i \mathcal{C}_i = C$. For each $c \notin \overline{C}$, there is a group consisting of a single element c. Such a group is called *done*.

As the groups constitute contiguous segments of columns, we can represent the partition with a subset $F \subseteq C$ containing the first columns of individual groups. Each group is identified with its leftmost column. We use a dynamic predecessor data structure [24] for maintaining the set F. Such representation also allows to split groups and merge neighboring groups in $O(\log\log m)$ time.

The Potential Row Sets. For each $\mathcal{C}_i$ we store a set $P(\mathcal{C}_i) \subseteq \overline{R}$, called a *potential row set*. Between consecutive operations, the potential row sets satisfy the following invariants:

P.1 For any $c \in \mathcal{C}_i$ there exists a row $r \in P(\mathcal{C}_i)$ such that $\min\{\mathcal{M}(\overline{R}, c)\} = \mathcal{M}_{r,c}$.
P.2 The size of any set $P(\mathcal{C}_i)$ is less than 2α, where $\alpha = \sqrt{\log m}$.
P.3 For any $i < j$ and any $r \in P(\mathcal{C}_i)$, $r' \in P(\mathcal{C}_j)$, we have $r \geq r'$.

The sets $P(\mathcal{C}_i)$ are stored as balanced binary search trees, sorted bottom to top. Intuitively, invariant 3 can be maintained because, by Fact 1, $\mathcal{M}(\overline{R}, C)$ is a Monge matrix, so Fact 2 applies. Then, we have $|P(\mathcal{C}_i) \cap P(\mathcal{C}_{i+1})| \leq 1$, so the sum of sizes of sets $P(\mathcal{C}_i)$ is $O(k+l)$.

▶ **Lemma 4.** *An insertion or deletion of some r to $P(\mathcal{C}_i)$ (along with the update of the auxiliary structures) can be performed in $O(\log\alpha + \log\log m)$ time.*

Clearly, one can answer the Current-Min-Row(c) query by finding the relevant group $\mathcal{C}_i$, $c \in \mathcal{C}_i$, and examining the entries $\mathcal{M}_{r,c}$ for $r \in P(\mathcal{C}_i)$. This takes $O(\log\log m + \alpha)$ time.

Upon activation of a new row r, we first merge the groups $\mathcal{C}_j$ such that r contains a current minimum for each column in $\mathcal{C}_j$. The potential row set of the newly formed group is set to $\{r\}$. Next, we insert r to some (at most two) of the existing potential row sets. This might make some $P(\mathcal{C}_i)$ break invariant 2. In such case the group $\mathcal{C}_i$ along with $P(\mathcal{C}_i)$ is split, so that the resulting potential row sets are of size α. The splitting algorithm summarized by the following lemma which leverages the SMAWK algorithm [1] to decrease the per-row cost of a split.

▶ **Lemma 5.** *Let $\mathcal{M}$ be a $u \times v$ rectangular Monge matrix with rows $\{r_1, \ldots, r_u\}$ and columns $C = \{c_1, \ldots, c_v\}$. For any $i \in [1, u]$, in $O\left(u\frac{\log v}{\log u}\right)$ time we can find such $c_s \in C$ that:*

1. *Some minima of columns $c_1, \ldots, c_s$ lie in rows $r_{i+1}, \ldots, r_u$.*
2. *Some minima of columns $c_{s+1}, \ldots, c_v$ lie in rows $r_1, \ldots, r_i$.*

As the split of some fixed $P(\mathcal{C}_i)$ happens at most once per α insertions, we charge the $O\left(\alpha\frac{\log m}{\log\alpha}\right)$ cost of splitting $P(\mathcal{C}_i)$ to the α elements inserted since the last split of $P(\mathcal{C}_i)$. The total number of insertions performed on the potential row sets is $O(k+l)$.

The Priority Queue. A priority queue H contains an element c for each $c \in \overline{C}$. The queue H satisfies the following invariants between any two operations.

H.1 For each $c \in \overline{C}$, the key of c in H is greater than or equal to $\min\{\mathcal{M}(\overline{R}, c)\}$.
H.2 For each group $\mathcal{C}_j$ that is not done, there exists such column $c_j \in \mathcal{C}_j$ that the key of c_j in H is equal to $\min\{\mathcal{M}(\overline{R}, c_j)\} = \min\left\{\mathcal{M}(\overline{R}, \mathcal{C}_j)\right\}$.

Note that by invariants 1 and 2, the key at the top of H is in fact equal to $\min\{\mathcal{M}(\overline{R}, \overline{C})\}$. Hence, Lower-Bound can be implemented trivially in $O(1)$ time. The following lemma follows easily from invariant 1 and Theorem 3.

▶ **Lemma 6.** *We can ensure 2 is satisfied for a single group $\mathcal{C}_j$ in $O(\alpha\log\log m)$ time.*

The detailed description of how the individual operations are implemented can be found in the full version [11]. The performance of our data structure can be summarized as follows.

▶ **Lemma 7.** *Let $\mathcal{M}$ be a $k \times l$ offset Monge matrix. There exists a data structure built in $O(k + l \log m)$ time, supporting* Lower-Bound *in $O(1)$ time and both* Current-Min-Row *and* Ensure-Bound-And-Get *in $O(\log m)$ time. Additionally, any sequence of operations* Activate-Row *is performed in $O\left((k+l)\frac{\log m}{\log\log m}\right)$ total time, where $m = \max(k, l)$.*

5 Online Column Minima of a Block Monge Matrix

Let $\mathcal{M} = \text{off}(\mathcal{M}_0, d)$, R, C, l, k, m be defined as in Section 4. In this section we consider the problem of reporting the column minima of a rectangular offset Monge matrix, but in a slightly different setting. Again, we are given a fixed rectangular Monge matrix $\mathcal{M}_0$ and we also have an initially empty, growing set of rows $\overline{R} \subseteq R$ for which the offsets $d(\cdot)$ are known. Let $\Delta > 0$ be an integral parameter not larger than l. We partition C into a set $\mathcal{B} = \{B_1, \ldots, B_b\}$ of at most $\lceil l/\Delta \rceil$ blocks, each of size at most Δ. The columns in each B_i constitute a contiguous fragment of $c_1, \ldots, c_l$, and each block B_i is to the left of B_{i+1}. We also maintain a shrinking subset $\overline{\mathcal{B}} \subseteq \mathcal{B}$ containing the blocks B_i, such that the minima $\min\{\mathcal{M}(R, B_i)\}$ are not yet known. More formally, for each $B_i \in \mathcal{B} \setminus \overline{\mathcal{B}}$, we have $\min\{\mathcal{M}(R, B_i)\} = \min\{\mathcal{M}(\overline{R}, B_i)\}$. Initially $\overline{\mathcal{B}} = \mathcal{B}$.

For each $c \in C$ not contained in the blocks of $\overline{\mathcal{B}}$, the data structure explicitly maintains the *current minimum*, i.e., the value $\min\{\mathcal{M}(\overline{R}, c)\}$. Moreover, when a new row is activated, we provide the user with columns of $\bigcup(\mathcal{B} \setminus \overline{\mathcal{B}})$ for which the current minima have changed.

For blocks $\overline{\mathcal{B}}$, the data structure only maintains the value $\min\{\mathcal{M}(\overline{R}, \bigcup \overline{\mathcal{B}})\}$. Once the user can guarantee that $\min\{\mathcal{M}(R, \bigcup \overline{\mathcal{B}})\}$ does not depend on the "hidden offsets" of rows $R \setminus \overline{R}$, the data structure moves a block $B_i \in \overline{\mathcal{B}}$ such that $\min\{\mathcal{M}(R, \bigcup \overline{\mathcal{B}})\} = \min\{\mathcal{M}(\overline{R}, B_i)\}$ out of $\overline{\mathcal{B}}$ and makes it possible to access the current minima in the columns of B_i.

More formally, we support the following set of operations:

- Activate-Row(r), where $r \in R \setminus \overline{R}$ – add r to the set $\overline{R}$.
- Block-Lower-Bound() – return $\min\{\mathcal{M}(\overline{R}, \bigcup \overline{\mathcal{B}})\}$.
- Block-Ensure-Bound() – tell the data structure that indeed $\min\{\mathcal{M}(R \setminus \overline{R}, C)\} \geq$ Block-Lower-Bound() $= \min\{\mathcal{M}(\overline{R}, B_i)\}$, for some $B_i \in \overline{\mathcal{B}}$, i.e., the smallest element of $\mathcal{M}(R, \bigcup \overline{\mathcal{B}})$ does not depend on the entries of $\mathcal{M}$ located in rows $R \setminus \overline{R}$.
  As the minimum of $\mathcal{M}(R, B_i)$ can now be computed, B_i is removed from $\overline{\mathcal{B}}$.
- Current-Min(c), where $c \in C$ – for $c \in \bigcup(\mathcal{B} \setminus \overline{\mathcal{B}})$, return the explicitly maintained value $\min\{\mathcal{M}(\overline{R}, \{c\})\}$. For $c \in \bigcup \overline{\mathcal{B}}$, set Current-Min$(c) = \infty$.

Additionally, the data structure provides access to the queue Updates containing the columns $c \in \bigcup(\mathcal{B} \setminus \overline{\mathcal{B}})$ such that the most recent call to either Activate-Row or Block-Ensure-Bound resulted in a change (or an initialization, if $c \in B_i$ and the last update was Block-Ensure-Bound, which moved B_i out of $\overline{\mathcal{B}}$) of the value Current-Min(c).

Note that there can be at most k calls to Activate-Row and no more than $\lceil l/\Delta \rceil$ calls to Block-Ensure-Bound.

5.1 The Data Structure

An Infrastructure for Short Subrow Minimum Queries. In this section we assume that for any $r \in R$ and $1 \leq i, j \leq l$, $j - i + 1 \leq \Delta$, it is possible to compute an answer to a subrow minimum query $S(r, i, j)$ (see Section 4) on matrix $\mathcal{M}_0$ (equivalently: $\mathcal{M}$) in constant time. We call such a subrow minimum query *short*.

The Block Minima Matrix. Define a $k \times b$ matrix $\mathcal{M}'$ with rows R and columns $\mathcal{B}$, such that for all $r_i \in R$ and $B_j \in \mathcal{B}$, $\mathcal{M}'_{r_i,B_j} = \min\{\mathcal{M}(r_i, B_j)\}$. We build the data structure of Section 4 for matrix $\mathcal{M}'$.

▶ **Lemma 8.** *$\mathcal{M}'$ is a Monge matrix and its entries can be accessed in $O(1)$ time.*

The Exact Minima Array. For each column $c \in \bigcup(\mathcal{B} \setminus \overline{\mathcal{B}})$, the value $\mathtt{cmin}(c) = \min\{\mathcal{M}(\overline{R}, c)\}$ is stored explicitly. The operation Current-Min(c) returns $\mathtt{cmin}(c)$.

Rows Containing the Block Minima. For each $B_j \in (\mathcal{B} \setminus \overline{\mathcal{B}})$ we store the value $y_j = \mathcal{M}'.$Current-Min-Row(B_j). Note that the data structure of Section 4 guarantees that for $B_i, B_j \in (\mathcal{B} \setminus \overline{\mathcal{B}})$ such that $i < j$, we have $y_i \geq y_j$. The set of defined y_j's grows over time.

The Row Candidate Sets. Two subsets D_0 and D_1 of $\overline{R}$ are maintained. The set D_q for $q = 0, 1$ contains the rows of $\overline{R}$ that may still prove useful when computing the initial value of $\mathtt{cmin}(c)$ for $c \in \bigcup\{B_i : B_i \in \overline{\mathcal{B}} \wedge i \bmod 2 = q\}$. For each such c, D_q contains a row r such that $\min\{\mathcal{M}(\overline{R}, c)\} = \mathcal{M}_{r,c}$. The sets D_q are also stored in dynamic predecessor structures.

Implementation. We now sketch how the data structure's components are used. Clearly, a call to Block-Lower-Bound translates into a single call Lower-Bound executed on $\mathcal{M}'$. When Block-Ensure-Bound is called, some block B_i is moved out of $\overline{\mathcal{B}}$. Apart from calling Ensure-Bound-And-Get on $\mathcal{M}'$, we have to initialize the values $\mathtt{cmin}(c)$ for $c \in B_i$. For each such c, we examine multiple rows of $D_{i \bmod 2}$ when looking for minima, but it can be shown that most of these rows can be discarded from $D_{i \bmod 2}$ afterwards. This in turn allows us to charge the work to the insertions into row candidate sets.

When Activate-Row is called, one has to call Activate-Row on $\mathcal{M}'$ first. Moreover, the values $\mathtt{cmin}(c)$ for some $c \in \bigcup(\mathcal{B} \setminus \overline{\mathcal{B}})$ have to be updated. It turns out that all such columns reside in at most two blocks and thus only $O(\Delta)$ additional time is needed.

The detailed implementation of each operation can be found in the full version [11].

▶ **Lemma 9.** *Let $\mathcal{M} = \mathit{off}(\mathcal{M}_0, d)$ be a $k \times l$ rectangular offset Monge matrix. Let Δ be the block size. Assume we can perform subrow minima queries spanning at most Δ columns of $\mathcal{M}_0$ in $O(1)$ time. There exists a data structure initialized in $O(k + l + \frac{l}{\Delta}\log m)$ time and supporting both* Block-Lower-Bound *and* Current-Min *in $O(1)$ time. Any sequence of* Activate-Row *and* Block-Ensure-Bound *operations can be performed in $O\left(k\left(\frac{\log m}{\log\log m} + \Delta\right) + l + \frac{l}{\Delta}\log m\right)$ time, where $m = \max(k, l)$.*

6 Online Column Minima of a Staircase Offset Monge Matrix

In this section we show a data structure supporting a similar set of operations as in Section 4, but in the case when the matrices $\mathcal{M}_0$ and $\mathcal{M} = \mathrm{off}(\mathcal{M}_0, d)$ are staircase Monge matrices with m rows $R = \{r_1, \ldots, r_m\}$ and m columns $C = \{c_1, \ldots, c_m\}$. We still aim at reporting the column minima of $\mathcal{M}$, while the set $\overline{R}$ of revealed rows is extended and new bounds on $\min\{\mathcal{M}(R \setminus \overline{R}, C)\}$ are available.

In comparison to the data structure of Section 4, we loosen the conditions posed on the operations Lower-Bound and Ensure-Bound-And-Get. Now, Lower-Bound might return a value smaller than $\min\{\mathcal{M}(\overline{R}, \overline{C})\}$ and a single call to Ensure-Bound-And-Get might not report any new column minimum at all. However, Ensure-Bound-And-Get can still only be called if $\min\{\mathcal{M}(R \setminus \overline{R}, C)\} \geq$ LOWER-BOUND() and the data structure we develop in this section guarantees that a bounded number of calls to ENSURE-BOUND-AND-GET suffices to report all the column minima of $\mathcal{M}$. The exact set of supported operations is:

- ACTIVATE-ROW(r), where $r \in R \setminus \overline{R}$ – add r to the set $\overline{R}$.
- LOWER-BOUND() – return a number v such that $\min\{\mathcal{M}(\overline{R}, \overline{C})\} \geq v$.
- ENSURE-BOUND-AND-GET() – tell the data structure that the inequality $\min\{\mathcal{M}(R \setminus \overline{R}, C)\} \geq$ LOWER-BOUND() holds.
  With this knowledge, the data structure may report some column $c \in \overline{C}$ such that $\min\{\mathcal{M}(R, c)\}$ is known. However, it's also valid to not report any new column minimum (in such case **nil** is returned) and only change the known value of LOWER-BOUND().
- CURRENT-MIN(c), where $c \in C$ – if $c \in C \setminus \overline{C}$, return the known minimum in column c.

6.1 The Data Structure

The Short Subrow Minimum Queries Infrastructure. Let $\Delta = \lceil \log^{1-\epsilon/2} m \rceil$. The following lemma allows us to use the data structure of Lemma 9 with block size Δ.

▶ **Lemma 10.** *The staircase Monge matrix $\mathcal{M}_0$ can be preprocessed in $O(m\Delta \log m)$ time so that subrow minimum queries on $\mathcal{M}_0$ spanning at most Δ columns take $O(1)$ time.*

The Partition of $\mathcal{M}$ into Rectangular Matrices $\mathcal{M}_1, \ldots, \mathcal{M}_q$. We partition the staircase Monge matrix $\mathcal{M}$ into $O(m \log^{\epsilon/2} m)$ non-overlapping *rectangular* Monge matrices $\mathcal{M}_1, \ldots, \mathcal{M}_q$ using the below lemma. Each $\mathcal{M}_i$ is a subrectangle of $\mathcal{M}$, each row r (column c) appears in a set W_r (W^c, resp.) of $O\left(\frac{\log m}{\log \log m}\right)$ ($O\left(\frac{\log^{1+\epsilon/2} m}{\log \log m}\right)$, resp.) subrectangles.

▶ **Lemma 11.** *For any $\epsilon \in (0, 1)$, a staircase matrix $\mathcal{M}$ with m rows and m columns can be partitioned in $O(m)$ time into $O(m \log^{\epsilon/2} m)$ non-overlapping rectangular matrices so that each row appears in $O\left(\frac{\log m}{\log \log m}\right)$ matrices of the partition, whereas each column appears in $O\left(\frac{\log^{1+\epsilon} m}{\log \log m}\right)$ matrices of the partition.*

We build the block data structure of Section 5 for each $\mathcal{M}_i$. For each $\mathcal{M}_i$ we use the same block size Δ. As each $\mathcal{M}_i$ is a subrectangle of $\mathcal{M}$, Lemma 10 guarantees that we can perform subrow minimum queries on $\mathcal{M}_i$ spanning at most Δ columns in $O(1)$ time. Recall that the blocks of the matrix $\mathcal{M}_i$ are partitioned into two sets $\overline{\mathcal{B}}_i$ and $\mathcal{B}_i \setminus \overline{\mathcal{B}}_i$. Denote by *block*$(\mathcal{M}_i)$ the submatrix $\mathcal{M}_i(\overline{R}_i, \bigcup \overline{\mathcal{B}}_i)$ and by *exact*$(\mathcal{M}_i)$ the submatrix $\mathcal{M}_i(\overline{R}_i, \bigcup(\mathcal{B}_i \setminus \overline{\mathcal{B}}_i))$. Here, R_i and C_i denote the row and column sets of $\mathcal{M}_i$, respectively.

The Priority Queue H. The core of our data structure is a priority queue H. At any time, H contains an element c for each $c \in \overline{C}$ and at most one element $\mathcal{M}_i$ for each matrix $\mathcal{M}_i$. Thus the size of H never exceeds $O(m \log^{\epsilon/2} m)$. We maintain the following invariants after the initialization and each call ACTIVATE-ROW or ENSURE-BOUND-AND-GET resulting in $\overline{C} \neq \emptyset$:

H.1 For each $c \in \overline{C}$, the key of c in H is equal to $\min\{\mathcal{M}_i.\text{CURRENT-MIN}(c) : \mathcal{M}_i \in W^c\}$.

H.2 For each $\mathcal{M}_i$ such that *block*$(\mathcal{M}_i)$ is not empty, the key of $\mathcal{M}_i$ in H is equal to $\min\{\textit{block}(\mathcal{M}_i)\} = \mathcal{M}_i.\text{BLOCK-LOWER-BOUND}()$.

▶ **Lemma 12.** *Assume invariants 1 and 2 are satisfied. Then H.*MIN-KEY() $\leq \mathcal{M}(\overline{R}, \overline{C})$.

Implementation. The data structure always returns the top key of H when Lower-Bound is called. Each call to Ensure-Bound-And-Get removes the top element e of H. If e is a column c, the minimum of c is reported. Otherwise $e = \mathcal{M}_i$ and it can be seen that $\mathcal{M}_i$.Block-Ensure-Bound can now be called. Afterwards, $\mathcal{M}_i$ is reinserted into H. Note that the number of times some $\mathcal{M}_j$ gets reinserted into H is no more than the total number of blocks in the matrices $\mathcal{M}_1, \ldots, \mathcal{M}_q$, i.e., $O(m \log^{\epsilon} m)$. The row activations are propagated to the relevant matrices $\mathcal{M}_j$ of the partition.

Both Ensure-Bound-And-Get and Activate-Row may make invariants 1 and 2 violated. However, the keys in H may only need to be decreased. Given that each operation Decrease-Key on H takes only $O(1)$ time, the time needed to update H is dominated by the cost of operations on the individual matrices $\mathcal{M}_j$.

The detailed implementation of the individual operations and analysis can be found in the full version [11]. The following lemma summarizes the performance of our data structure.

▶ **Lemma 13.** *Let $\mathcal{M} = \mathit{off}(\mathcal{M}_0, d)$ be an $m \times m$ offset staircase Monge matrix and let $\epsilon \in (0,1)$. There exists a data structure that can be initialized in $O\left(m \log^{2-\epsilon} m\right)$ time, supporting both* Lower-Bound *and* Current-Min *in $O(1)$ time. Any sequence of* Activate-Row *and* Ensure-Bound-And-Get *operations takes $O\left(m \frac{\log^2 m}{\log^2 \log m}\right)$ total time. All the column minima are computed after $O(m \log^{\epsilon} m)$ calls to* Ensure-Bound-And-Get.

References

1 Alok Aggarwal, Maria M. Klawe, Shlomo Moran, Peter W. Shor, and Robert E. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987. doi:10.1007/BF01840359.

2 Mudabir Kabir Asathulla, Sanjeev Khanna, Nathaniel Lahn, and Sharath Raghvendra. A faster algorithm for minimum-cost bipartite perfect matching in planar graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 457–476, 2018. doi:10.1137/1.9781611975031.31.

3 Glencora Borradaile, David Eppstein, Amir Nayyeri, and Christian Wulff-Nilsen. All-pairs minimum cuts in near-linear time for surface-embedded graphs. In *32nd International Symposium on Computational Geometry, SoCG 2016, June 14-18, 2016, Boston, MA, USA*, pages 22:1–22:16, 2016. doi:10.4230/LIPIcs.SoCG.2016.22.

4 Glencora Borradaile, Philip N. Klein, Shay Mozes, Yahav Nussbaum, and Christian Wulff-Nilsen. Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. *SIAM J. Comput.*, 46(4):1280–1303, 2017. doi:10.1137/15M1042929.

5 Glencora Borradaile, Piotr Sankowski, and Christian Wulff-Nilsen. Min *st*-cut oracle for planar graphs with near-linear preprocessing time. *ACM Trans. Algorithms*, 11(3):16:1–16:29, 2015. doi:10.1145/2684068.

6 Sergio Cabello. Subquadratic algorithms for the diameter and the sum of pairwise distances in planar graphs. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2143–2152, 2017. doi:10.1137/1.9781611974782.139.

7 Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *J. Comput. Syst. Sci.*, 72(5):868–889, 2006. doi:10.1016/j.jcss.2005.05.007.

8 Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987. doi:10.1145/28869.28874.

9 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, New Orleans, LA, USA, January 7-10, 2018*, pages 495–514, 2018. doi:10.1137/1.9781611975031.33.

10 Pawel Gawrychowski, Shay Mozes, and Oren Weimann. Submatrix maximum queries in monge matrices are equivalent to predecessor search. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 580–592, 2015. doi:10.1007/978-3-662-47672-7_47.

11 Paweł Gawrychowski and Adam Karczmarz. Improved bounds for shortest paths in dense distance graphs, 2016. arXiv:1602.07013.

12 Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Improved algorithms for min cut and max flow in undirected planar graphs. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 313–322, 2011. doi:10.1145/1993636.1993679.

13 Haim Kaplan, Shay Mozes, Yahav Nussbaum, and Micha Sharir. Submatrix maximum queries in monge matrices and partial monge matrices, and their applications. *ACM Trans. Algorithms*, 13(2):26:1–26:42, 2017. doi:10.1145/3039873.

14 Philip N. Klein. Multiple-source shortest paths in planar graphs. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*, pages 146–155, 2005. URL: http://dl.acm.org/citation.cfm?id=1070432.1070454.

15 Philip N. Klein and Shay Mozes. Optimization algorithms for planar graphs, 2017. URL: http://planarity.org.

16 Philip N. Klein, Shay Mozes, and Christian Sommer. Structured recursive separator decompositions for planar graphs in linear time. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 505–514, 2013. doi:10.1145/2488608.2488672.

17 Jakub Łącki, and Piotr Sankowski and. Min-cuts and shortest cycles in planar graphs in o(n loglogn) time. In *Algorithms - ESA 2011 - 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings*, pages 155–166, 2011. doi:10.1007/978-3-642-23719-5_14.

18 Jakub Łącki, Yahav Nussbaum, Piotr Sankowski and Christian Wulff-Nilsen. Single source - all sinks max flows in planar digraphs. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 599–608, 2012. doi:10.1109/FOCS.2012.66.

19 Gary L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *J. Comput. Syst. Sci.*, 32(3):265–279, 1986. doi:10.1016/0022-0000(86)90030-9.

20 Shay Mozes, Kirill Nikolaev, Yahav Nussbaum, and Oren Weimann. Minimum cut of directed planar graphs in $O(n \log\log n)$ time. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 477–494, 2018. doi:10.1137/1.9781611975031.32.

21 Shay Mozes, Yahav Nussbaum, and Oren Weimann. Faster shortest paths in dense distance graphs, with applications. *Theor. Comput. Sci.*, 711:11–35, 2018. doi:10.1016/j.tcs.2017.10.034.

22 Shay Mozes and Christian Wulff-Nilsen. Shortest paths in planar graphs with real lengths in $O(n \log^2 n / \log\log n)$ time. In *Algorithms - ESA 2010, 18th Annual European Symposium, Liverpool, UK, September 6-8, 2010. Proceedings, Part II*, pages 206–217, 2010. doi:10.1007/978-3-642-15781-3_18.

23 Yahav Nussbaum. *Network flow problems in planar graphs*. PhD thesis, Tel Aviv University, 2014.

24 Peter van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Inf. Process. Lett.*, 6(3):80–82, 1977. doi:10.1016/0020-0190(77)90031-X.