# A Faster Algorithm for Minimum-cost Bipartite Perfect Matching in Planar Graphs



MUDABIR KABIR ASATHULLA, Virginia Tech, USA
SANJEEV KHANNA, University of Pennsylvania, USA
NATHANIEL LAHN and SHARATH RAGHVENDRA, Virginia Tech, USA

Given a weighted planar bipartite graph $G(A \cup B, E)$ where each edge has an integer edge cost, we give an $\tilde{O}(n^{4/3} \log nC)$ time algorithm to compute minimum-cost perfect matching; here $C$ is the maximum edge cost in the graph. The previous best-known planarity exploiting algorithm has a running time of $O(n^{3/2} \log n)$ and is achieved by using planar separators (Lipton and Tarjan '80).

Our algorithm is based on the bit-scaling paradigm (Gabow and Tarjan '89). For each scale, our algorithm first executes $O(n^{1/3})$ iterations of Gabow and Tarjan's algorithm in $O(n^{4/3})$ time leaving only $O(n^{2/3})$ vertices unmatched. Next, it constructs a compressed residual graph $H$ with $O(n^{2/3})$ vertices and $O(n)$ edges. This is achieved by using an $r$-division of the planar graph $G$ with $r = n^{2/3}$. For each partition of the $r$-division, there is an edge between two vertices of $H$ if and only if they are connected by a directed path inside the partition. Using existing efficient shortest-path data structures, the remaining $O(n^{2/3})$ vertices are matched by iteratively computing a minimum-cost augmenting path, each taking $\tilde{O}(n^{2/3})$ time. Augmentation changes the residual graph, so the algorithm updates the compressed representation for each partition affected by the change in $\tilde{O}(n^{2/3})$ time. We bound the total number of affected partitions over all the augmenting paths by $O(n^{2/3} \log n)$. Therefore, the total time taken by the algorithm is $\tilde{O}(n^{4/3})$.

CCS Concepts: • **Mathematics of computing → Combinatorial algorithms**; **Matchings and factors**; **Graph algorithms**; *Paths and connectivity problems*;

Additional Key Words and Phrases: Minimum-cost matching, primal-dual, scaling algorithms

**ACM Reference format:**
Mudabir Kabir Asathulla, Sanjeev Khanna, Nathaniel Lahn, and Sharath Raghvendra. 2019. A Faster Algorithm for Minimum-cost Bipartite Perfect Matching in Planar Graphs. *ACM Trans. Algorithms* 16, 1, Article 2 (November 2019), 30 pages.
https://doi.org/10.1145/3365006

## 1 INTRODUCTION

Consider a bipartite graph $G(A \cup B, E)$ where $|A| = |B| = n$, the edge set $E \subseteq A \times B$ and every edge $(a, b) \in E$ has an integer cost specified by $\mathrm{c}(a, b)$. A *matching* $M \subseteq E$ is any set of vertex-disjoint edges, and we represent its cost by $\mathrm{c}(M) = \sum_{(a,b)\in M} \mathrm{c}(a, b)$. $M$ is a *perfect matching* if $|M| = n$. A *minimum-cost perfect matching* is a perfect matching with the smallest cost. In this article, we

This work is supported in part by National Science Foundation grants CCF-1464276, CCF-1909171, and CCF-1617851.
Authors' addresses: M. K. Asathulla, N. Lahn, and S. Raghvendra, Virginia Tech, 114 McBryde Hall, 225 Stanger Street, Blacksburg, VA 24060; emails: {mudabir, lahnn, sharathr}@vt.edu; S. Khanna, Dept. of Computer & Information Science, University of Pennsylvania, 3330 Walnut Street, Philadelphia, PA 19104; email: sanjeev@cis.upenn.edu.

1549-6325/2019/11-ART2 $15.00
https://doi.org/10.1145/3365006

consider the problem of computing a minimum-cost perfect matching in a planar bipartite graph. There are two previous algorithms for this problem that run in $\tilde{O}(n^{3/2})$[1] time. The first algorithm by Lipton and Tarjan [14] is a divide-and-conquer algorithm based on the well-known planar separator theorem, whereas the second approach by Gabow and Tarjan is based on the bit-scaling paradigm [7]. In this article, we combine these two approaches to yield an improved $\tilde{O}(n^{4/3})$ time algorithm for computing a minimum-cost perfect matching in bipartite planar graphs.

*Maximum-Cardinality Matching:* For the problem of computing a maximum cardinality matching in unweighted graphs with $n$ vertices and $m$ edges, Ford and Fulkerson [6] presented an algorithm that iteratively computes $n$ augmenting paths, each of which takes $O(m)$ time, leading to an $O(mn)$ time algorithm. Hopcroft and Karp [9] presented an algorithm with an improved execution time of $O(m\sqrt{n})$. Their algorithm iteratively computes a maximal set of vertex-disjoint shortest augmenting paths in $O(m)$ time and converges to an optimal matching in $O(\sqrt{n})$ iterations. Computing a perfect matching in bipartite graphs is a special case of multiple-source multiple-sink maximum flow problem where each source produces and each sink consumes exactly one unit of commodity. For arbitrary graphs, one can easily reduce a multiple-source multiple-sink maximum flow problem to a single-source single sink case by introducing an artificial source and an artificial sink vertex and connecting them to all sources and sinks. For planar graphs, however, this reduction may not preserve planarity, and therefore the algorithms for the multiple-source multiple-sink problem in planar graphs are distinctly different from the single-source single-sink problem.

In unweighted planar graphs, there is extensive work on computing flows in planar graphs with multiple sources and sinks. For the case where it is known how much of commodity is produced and consumed at each source and sink, there is an algorithm to compute a valid flow that runs in $O(n \log^2 n/ \log \log n)$ time [15]. Computing perfect matching in bipartite graphs is a special case and, therefore, this algorithm also applies to computing perfect matching in planar bipartite graphs. After considerable effort on many special cases, an $O(n \log^3 n)$ time algorithm for the problem of computing maximum flow in planar graphs with multiple sources and multiple sinks was designed [2]. This algorithm can also be applied to compute maximum cardinality matching in bipartite planar graphs.

It is also possible to compute maximum matchings in arbitrary graphs (not necessarily bipartite) using Gaussian elimination [16]. This randomized algorithm runs in $O(n^{\omega})$ time; here $\omega$ is the best-known exponent for the matrix multiplication problem. For planar graphs, Mucha and Sankowski improved and obtained an $O(n^{\omega/2})$ time randomized algorithm for this problem [17].

*Minimum-cost Matching:* In weighted graphs with $n$ vertices and $m$ edges, the well-known Hungarian method computes a minimum-cost maximum cardinality matching in $O(mn)$ time [13]. Gabow and Tarjan designed a cost-scaling method to compute a minimum-cost perfect matching in $O(m\sqrt{n} \log nC)$ time, where $C$ is the largest cost on any edge of the graph [7]. Using a standard reduction, they showed that their algorithm can also be used to compute minimum-cost maximum cardinality matching. This reduction, however, does not preserve planarity. Ramshaw and Tarjan [18] extend the scaling approach to compute minimum-cost imperfect matching for bipartite graphs where the two sets are of different cardinalities. The running time of their algorithm is $O(m\sqrt{n} \log nC)$. Sankowski extended the Gaussian elimination-based approach to compute weighted matchings in arbitrary graphs in $O(n^{\omega}C)$ time [19]. In Reference [3], Cohen et al. give an $\tilde{O}(m^{10/7})$ algorithm for computing minimum-cost maximum flow on bipartite graphs with unit capacities. This algorithm can be used to find a minimum-cost perfect matching on a bipartite planar graph in $\tilde{O}(n^{10/7})$ time.

[1] $\tilde{O}(\cdot)$ ignores the log factors that may appear in the running time.

For weighted planar graphs, Lipton and Tarjan [14] use planar separators to design an algorithm that computes a minimum-cost maximum cardinality matching in $O(n^{3/2} \log n)$ time; see also Reference [4]. All planarity exploiting algorithms to compute maximum flow use a relation between maximum flow in a planar graph and shortest paths in its dual graph. It is not clear how to extend this relation to computation of minimum-cost flows. Therefore, despite significant progress in the design of faster planarity exploiting algorithms for the maximum cardinality bipartite matching problem, there has not been any improvement in algorithms for the weighted version of this problem. We are not aware of any faster planarity exploiting algorithms for computing minimum-cost perfect matching.

Computing a perfect matching in planar bipartite graphs is also an important intermediate step in solving the two-dimensional Euclidean bipartite matching problem. Here, as input, we are given two sets of $n$ points in a plane. An edge between two points has a cost equal to the Euclidean distance. In this problem, we wish to compute a minimum-cost perfect matching of the two point sets. Under the assumption that points have bounded integer coordinates, Sharathkumar [20] presented a reduction of the Euclidean bipartite matching problem to the problem of computing minimum-cost perfect matching in a planar bipartite graph. For an arbitrary small constant $\delta > 0$, this reduction takes $\tilde{O}(n^{3/2+\delta})$ time. The minimum-cost matching is then computed by simply using Lipton and Tarjan's $O(n^{3/2} \log n)$ time algorithm.

*Our Result and Approach:* In this article, we use the scaling paradigm to design a new planarity exploiting algorithm that computes minimum-cost perfect matching in planar bipartite graphs. Our algorithm runs in $O(n^{4/3} \log^2 n \log nC)$ time where $C$ is the largest cost edge in the graph.[2] In comparison, the previous algorithm by Lipton and Tarjan runs in $O(n^{3/2} \log n)$ time.

Both the Hungarian algorithm and the Gabow-Tarjan algorithm iteratively compute a minimum cost augmenting path and augment the matching along this path. To obtain a speed-up, Gabow and Tarjan observed that for graphs where the edges have positive integers as edge costs, and the optimal matching has a cost of $O(n)$, one can integrate several properties of the Hopcroft-Karp algorithm with the Hungarian algorithm. To do this, they introduce an additive error of 1 on every edge that is not in the matching and iteratively compute the minimum-cost augmenting path. The error of +1 on every non-matching edge results in longer paths having a larger cost. As a consequence, their algorithm would pick many short (both in cost and length) augmenting paths in each iteration. This is similar to how the Hopcroft-Karp algorithm behaves in the unweighted setting. As a result, they obtain a running time that is similar to the running time of Hopcroft and Karp's algorithm, i.e., $O(m\sqrt{n} \log nC)$. Using the scaling paradigm, they guarantee that the cost of the optimal matching on each scale is $O(n)$ while also eliminating any error in cost that is introduced due to the +1 that was added to the edge costs.

In our algorithm, we build an $r$-division of the planar graph that breaks the graph into $O(n/r)$ pieces each of $O(r)$ size. In addition, each piece has only $O(\sqrt{r})$ vertices that also participate in other pieces (such vertices are called boundary vertices). Therefore, in total, there are $O(n/\sqrt{r})$ boundary vertices.

Similar to Gabow and Tarjan's algorithm, we use a scaling approach where each scale corresponds to a bit in the edge costs. The computation proceeds over $O(\log nC)$ scales. In each scale, our algorithm iteratively computes minimum-cost augmenting paths. We speed-up the Gabow-Tarjan $O(n^{3/2})$ computation time per scale to $\tilde{O}(n^{4/3})$ time as follows:

[2]The algorithm will be presented for nonnegative edge costs. If there are negative edge costs, then a constant can be added to the weight of every edge to make them nonnegative.

- Within each scale, we first execute the Gabow-Tarjan algorithm for $O(\sqrt{r})$ iterations to match all but $O(n/\sqrt{r})$ vertices. This takes $O(n\sqrt{r})$ time.
- We introduce an additive error of $+\lceil\sqrt{r}\rceil$ on every edge that is not in the matching and is incident on a boundary vertex. We show that doing so does not asymptotically increase the total error in the cost of the matching. Using the scaling paradigm, we can eliminate this error to obtain the optimal matching (see Section 3).
- Next, we compress the residual graph into a weighted and directed multigraph $H$ with the $O(n/\sqrt{r})$ boundary and free (unmatched) vertices as the vertex set. There is an edge between two boundary vertices if and only if they are connected by a directed path in one of the pieces of the $r$-division. This compressed residual graph $H$ has $O(n/\sqrt{r})$ vertices and $O(n)$ edges (see Section 4).
- The shortest path between two free vertices in $H$ can be used to compute the minimum-cost augmenting path. Using efficient data structures and an efficient implementation of Dijkstra's algorithm by Fakcharoenphol and Rao (described in References [5], [10], and [11]), we compute the minimum-cost augmenting path in $H$ in time $O((n/\sqrt{r})\log r \log n)$. Therefore, the total time taken to compute the remaining $O(n/\sqrt{r})$ augmenting paths is $O((n^2/r)\log r \log n)$ (see Section 4.3).
- After we augment the matching, the residual graph changes and so does its compressed representation $H$. For every piece of the $r$-division that the augmenting path passes through, we recompute its edges in $H$. Each such affected piece can be updated in $O(r \log r)$ time. Although every augmenting path can potentially pass through all the $O(n/r)$ pieces, we prove that, throughout the course of the algorithm, the total number of pieces that the augmenting paths touch is $O((n/\sqrt{r})\log n)$ (see Lemma 5.11), and, therefore, the total time taken to recompute edges of these pieces is only $O(n\sqrt{r}\log r \log n)$.
- Together, the total time taken by the algorithm for computation over a single scale is $O((n^2/r)\log r \log n + n\sqrt{r}\log r \log n)$, which is $O(n^{4/3}\log^2 n)$ when $r = n^{2/3}$. By summing up over all scales, we obtain a running time of $O(n^{4/3}\log^2 n \log nC)$.

*Organization:* The remainder of the article is organized as follows: In Section 2, we present background on the matching algorithms that serve as building blocks for our approach. In Section 3, we describe our scaling algorithm. In Section 4, we present our algorithm for each scale. In Section 5, we prove the correctness and efficiency of our algorithm.

## 2 BACKGROUND

*Preliminaries on Matching:* Given a matching $M$ on this bipartite graph, an *alternating path* (or cycle) is a simple path (respectively, cycle) whose edges alternate between those in $M$ and those not in $M$. We refer to any vertex that is not matched in $M$ as a *free vertex*. An *augmenting path* $P$ is an alternating path between two free vertices. We can *augment* $M$ by one edge along $P$ if we remove the edges of $P \cap M$ from $M$ and add $P \setminus M$ to $M$. After augmenting, the new matching is given by $M \leftarrow M \oplus P$, where $\oplus$ is the symmetric difference operator. For a matching $M$, we define a directed graph called the *residual graph* $G_M(A \cup B, E_M)$. We represent a directed edge from $a$ to $b$ as $\overrightarrow{ab}$. For every edge $(a, b) \in E \cap M$, we have an edge $\overrightarrow{ab}$ in $E_M$ and for every edge $(a, b) \in E \setminus M$, there is an edge $\overrightarrow{ba}$ in $E_M$. Note that $G$ and $G_M$ have the same vertex set and edge set with the edges of $G_M$ directed depending on their membership in the matching $M$. For simplicity in presentation, we treat the vertex set and the edge set of $G$ and $G_M$ as identical. So, for example, a matching $M$ in

the graph $G$ is also a matching in the graph $G_M$. It is easy to see that a path $\overrightarrow{P}$ in $G_M$ is a directed path if and only if this path is an alternating path in $G$.

Our work uses ideas from two fundamental algorithms—namely, the Hungarian algorithm and the Gabow-Tarjan algorithm—to compute a minimum-cost matching in any bipartite graph. Both these algorithms are similar in style to our algorithm. Next, we present a high-level overview of these algorithms. In the Hungarian algorithm, for every vertex $v$ of the graph $G$, we maintain a dual weight $y(v)$. A *feasible* matching consists of a matching $M$ and a set of dual weights $y(\cdot)$ on the vertex set such that for every edge between $u \in B$ and $v \in A$, we have

$$y(u) + y(v) \leq c(u, v), \tag{1}$$
$$y(u) + y(v) = c(u, v) \quad \text{for } (u, v) \in M. \tag{2}$$

For the Hungarian algorithm, we define the *net-cost* of an augmenting path $P$ as follows:

$$\phi(P) = \sum_{(a,b)\in P\setminus M} c(a, b) - \sum_{(a,b)\in P\cap M} c(a, b).$$

We can also interpret the net-cost of a path as the increase in the cost of the matching due to augmenting it along $P$, i.e., $\phi(P) = c(M \oplus P) - c(M)$. We can extend the definition of net-cost to alternating paths and cycles in a straightforward way.

At the start of the algorithm $M = \emptyset$. In each iteration, the Hungarian algorithm computes a minimum net-cost augmenting path $P$ and updates $M$ to $M \oplus P$. The algorithm terminates when we have a perfect matching. During the course of the algorithm, it maintains the invariant that there are no alternating cycles with negative net-cost.

It can be shown that any perfect matching $M$ with a set of dual weights $y(\cdot)$ is a min-cost perfect matching if and only if there is no negative net-cost alternating cycle with respect to $M, y(\cdot)$ in $G$. Any perfect matching $M$ that satisfies the feasibility conditions (1) and (2) has this property. Thus, it is sufficient for the Hungarian algorithm to find a feasible perfect matching.

To find the minimum net-cost augmenting path, the Hungarian algorithm uses a simple Dijkstra-type search procedure called the Hungarian Search. At any stage of the algorithm, the matching $M$ and the set of dual weights $y(\cdot)$ satisfy the following invariants:

(i) $M$ and the set of dual weights $y(\cdot)$ form a feasible matching.
(ii) For every vertex $b \in B$, $y(b) \geq 0$, and if $b$ is a free vertex, then the dual weight $y(b) = \max_{v\in B} y(v)$. We refer to this dual weight of any free vertex of $B$ as $y_{max}$.
(iii) For every vertex $a \in A$, $y(a) \leq 0$, and if $a$ is a free vertex, then its dual weight $y(a) = 0$.

The Hungarian search computes the minimum net-cost path as follows: For any edge $(u, v)$, let $c(u, v) - y(u) - y(v)$ be the *slack* of $(u, v)$. By feasibility condition (1), the slack of any edge is non-negative. Consider a directed graph $G'_M$, which is the same as the residual graph $G_M$ except the cost associated with each edge is equal to its slack. It can be shown that the minimum weight directed path in $G'_M$ corresponds to the minimum net-cost augmenting path in $G$. Since the slack on every edge is non-negative, the graph $G'_M$ does not have any negative cost edges. Therefore, we can simply use (a slight variant of) Dijkstra's algorithm to compute the minimum net-cost augmenting path. After this, the dual weights are updated in such a way that the invariants are satisfied (see Reference [13] for details). The Hungarian algorithm computes $n$ augmenting paths, each of which can be computed by Hungarian search in $O(m)$ time. Therefore, the total time taken is $O(mn)$.

As in the Hungarian algorithm, the Gabow-Tarjan algorithm also maintains a dual weight for every vertex of $G$. A *1-feasible matching* consists of a matching $M$ and set of dual weights $y(\cdot)$ such

that for every edge between $u \in A$ and $v \in B$, we have

$$y(u) + y(v) \leq c(u, v) + 1, \tag{3}$$

$$y(u) + y(v) = c(u, v) \quad \text{for } (u, v) \in M. \tag{4}$$

A *1-optimal matching* is a perfect matching that is 1-feasible. The following lemma relates 1-optimal matchings to optimal matchings:

Lemma 2.1. *[7] For a bipartite graph $G(A \cup B, E)$ with an integer edge cost function c, let $M$ be a 1-optimal matching and $M_{OPT}$ be the optimal matching. Then, $c(M) \leq c(M_{OPT}) + n$.*

For every $(a, b) \in E$, suppose we redefine the edge weight to be $c^*(a, b) = (n + 1)c(a, b)$. This uniform scaling of edge costs preserves the set of optimal matchings. Lemma 2.1 implies that a 1-optimal matching of vertices in $A, B$ with edge weights $c^*(\cdot, \cdot)$ corresponds to an optimal matching with the original edge weights $c(\cdot, \cdot)$. We now describe the Gabow-Tarjan algorithm for finding a 1-optimal matching, which is based on a bit-scaling approach.

The Gabow-Tarjan algorithm consists of *scales*. For any edge $(u, v)$, let $b_1, b_2 \ldots b_\ell$ be the binary representation of $c^*(u, v)$. In the $i$th scale, the cost of an edge, $c_i(u, v)$, corresponds to the most significant $i$ bits of $c^*(u, v)$. Each scale of the Gabow-Tarjan algorithm takes as input a bipartite graph on $A, B$, with a cost function $c_i(\cdot, \cdot)$, and a set of dual weights $y(v)$ for every vertex $v \in A \cup B$ and returns a 1-optimal matching. We transfer the dual weights from scale $i$ to scale $i + 1$ by simply setting, for any vertex $v \in A \cup B$, $y(v) \leftarrow 2y(v) - 1$. Therefore, at the beginning of scale $i + 1$, the slack $s_{i+1}(u, v) = c_{i+1}(u, v) - y(u) - y(v)$ on the edges of the 1-optimal matching of scale $i$ will be positive and at most 3. Hence, the cost of an optimal matching with respect to the slack is upper bounded by $O(n)$.

For a given scale $i$, we refer to an edge $(u, v) \notin M$ as *admissible* if $y(u) + y(v) = c_i(u, v) + 1$. An *admissible graph* is the union of the set of admissible edges and edges in $M$. The algorithm will iteratively find a maximal set $\mathcal{P}$ of vertex-disjoint augmenting paths in the admissible graph by doing a depth first search. Then the matching is augmented along every path $P \in \mathcal{P}$ in the admissible graph. The dual weights are then adjusted to ensure that the new matching edges satisfy 1-feasibility condition (4). After this, it can be shown that the resulting admissible graph does not have any augmenting paths. The algorithm invokes the Hungarian search procedure, which adjusts the dual weights so more edges become admissible until it finds an augmenting path of admissible edges.

Using the fact that in each iteration Hungarian search increases the dual weight of every free vertex by at least one, Gabow and Tarjan show that their algorithm iterates only $O(\sqrt{n})$ times, and that the total length of all the augmenting paths found is $O(n \log n)$. Since each iteration runs in $O(m)$ time, the computation time for a single scale is $O(m\sqrt{n})$, and summed over all $O(\log nC)$ scales, the total time taken is $O(m\sqrt{n} \log nC)$.

*$r$-division of a Planar Graph:* We introduce the notion of an $r$-division of a planar graph that we use in our algorithm.

*Definition 2.2.* An *$r$-division* of any planar graph $G(V, E)$, denoted by $\mathcal{R}(G)$, is a set $\{\mathcal{R}_1(V_1, E_1), \ldots, \mathcal{R}_l(V_l, E_l)\}$ of $l = O(n/r)$ edge-induced subgraphs of $G$, with $E_j = \{(a, b) \mid (a, b) \in E, a, b \in V_j\}$, $\bigcup_j E_j = E$, and $\bigcup_j V_j = V$. We call each such subgraph a *partition*. Here, each edge is contained in at least one partition. A vertex that has incident edges from two or more partitions is called a *boundary* vertex. All other vertices are called *internal* vertices. For any partition $\mathcal{R}_j$, let $\mathcal{K}_j$ denote its set of boundary vertices. Specifically, the $r$-division $\mathcal{R}(G)$ has the following properties:

- For every partition $\mathcal{R}_j$, $|V_j| \leq r$ and $|\mathcal{K}_j| = O(\sqrt{r})$. There are $O(n/r)$ partitions. Let $k$ be a constant such that $\sum_{j=1}^{l} |\mathcal{K}_j| \leq kn/\sqrt{r}$.

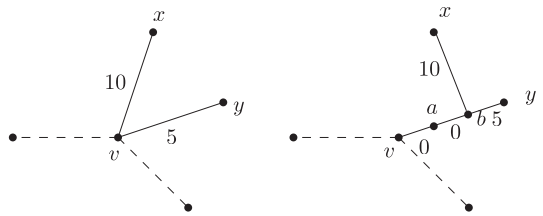


Fig. 1. Transforming any bipartite graph to a constant degree graph.

- Every partition $\mathcal{R}_j$ contains $O(1)$ faces that are not faces of a planar embedding of $G$ (also called holes).

Klein et al. [12] give an algorithm for constructing such an $r$-division in $O(n)$ time. Given a value of $r$, we use the same $r$-division during the course of the algorithm.

For construction of the $r$-division and the subsequent use of data structures for planar graphs, it is convenient to assume that the input graph has a constant maximum degree. Biedl [1] showed how any planar bipartite graph can be transformed into another planar bipartite graph such that the degree of every vertex is at most 3 and the matching is "preserved." For the sake of completion, we present this transformation in the context of weighted bipartite graphs. First, we provide a transformation of $G$ to graph $\tilde{G}$, which reduces the maximum degree in $\tilde{G}$ by 1 (provided the maximum degree is at least 4). Repeatedly applying this transformation, we obtain a graph where every vertex has a degree of at most 3. We describe this transformation next. Take any vertex $v \in G$ with a degree at least 4 and consider two vertices $x$ and $y$ such that $(v, x)$ and $(v, y)$ appear next to each other in the clockwise ordering of all edges incident on $v$. Suppose $v \in B$ (a symmetric construction also works when $v \in A$). We add two new vertices $a, b$ and add edges $(v, a)$, $(a, b)$, $(b, x)$, and $(b, y)$ with costs $\mathrm{c}(v, a) = 0, \mathrm{c}(a, b) = 0, \mathrm{c}(b, x) = \mathrm{c}(v, x)$, and $\mathrm{c}(b, y) = \mathrm{c}(v, y)$. (See Figure 1). This transformation reduces the degree of $v$ by 1 and preserves planarity. Using a straightforward proof from Reference [1], it can be shown that all optimal matchings in $G$ and $\tilde{G}$ have the same cost and an optimal matching in $G$ can be retrieved from any optimal matching in $\tilde{G}$.

*Convention for Notation:* Throughout this article, we assume $G$ is a planar bipartite graph with $A \cup B$ as the vertex set and $E$ as the set of edges. Given a matching $M$, we refer to its residual graph by $G_M$. Note that the vertex and edge sets of $G$ and $G_M$ are identical (except for the directions) and a matching, alternating path or an alternating cycle in $G$ is also a matching, directed path or a directed cycle in $G_M$. So, if there is any subset $P$ of edges in $G$, we will also use $P$ to denote the same subset of edges in $G_M$; the directions of these edges are determined by whether or not an edge is in $M$. We will define a net-cost for an alternating path (or cycle) $P$ in our algorithm and denote it by $\phi(P)$. Any directed path or cycle in $G_M$ will inherit its net-cost from $G$. Throughout the article, for any weighted and directed graph $K$, we will use the notation $K'$ to be the graph identical to $K$ where the cost of any directed edge in the graph is replaced by its slack. Recall that an $r$-division $\mathcal{R}(G)$ partitions the edges of $G$. Since $G$, $G_M$, and $G'_M$ have the same underlying set of edges, $\mathcal{R}(G)$ can be seen as an $r$-division of $G_M$ and $G'_M$ as well. Therefore, we use $\mathcal{R}(G), \mathcal{R}(G_M)$, and $\mathcal{R}(G'_M)$ to denote the same $r$-division.

In the following, we present a scaling algorithm to compute minimum-cost matching in planar graphs. Our algorithm is similar in some ways to the Gabow-Tarjan algorithm. However, by using a compressed residual graph, we achieve a faster execution time of $O(n^{4/3} \log^2 n \log nC)$ in computing the optimal matching.

## 3 OUR SCALING ALGORITHM

We next introduce a notion of feasibility that is based on an $r$-division of a planar graph. We assume that we are given an $r$-division, $\mathcal{R}(G) = \{\mathcal{R}_1(V_1, E_1), \ldots, \mathcal{R}_l(V_l, E_l)\}$ with $l = O(n/r)$. Recall that we denote the set of boundary vertices of $\mathcal{R}_j$ by $\mathcal{K}_j$. For every vertex $v \in A \cup B$, we define a 0/1 indicator variable $i_v$ to be 1 if and only if $v$ is a boundary vertex with respect to $\mathcal{R}(G)$. For any edge $(u, v) \in E$, we define a value $\delta_{uv}$ to be $\max\{1, i_u\lceil\sqrt{r}\rceil, i_v\lceil\sqrt{r}\rceil\}$. For any edge induced subgraph $G^*(V^*, E^*)$ of $G(A \cup B, E)$ and the $r$-division $\mathcal{R}(G)$, we say that a matching $M \subseteq E^*$ and a set of dual weights $y(\cdot)$ on the vertices of $V^*$ are $\mathcal{R}$*−feasible* if the following conditions are satisfied:

$$y(u) + y(v) \leq \mathrm{c}(u, v) + \delta_{uv} \quad \text{for } (u, v) \in E^*, \tag{5}$$

$$y(u) + y(v) = \mathrm{c}(u, v) \quad \text{for } (u, v) \in M. \tag{6}$$

An $\mathcal{R}$-optimal matching is a perfect matching that is $\mathcal{R}$-feasible. In our algorithm, for the graph $G(A \cup B, E)$ and the $r$-division $\mathcal{R}(G)$, we would like to compute an $\mathcal{R}$-optimal matching $M$ along with its dual weights $y(\cdot)$. Note that $\mathcal{R}$-feasible matching is defined for any edge induced subgraph of $G$. In this article, the only induced subgraphs that we consider are partitions from the $r$-division $\{\mathcal{R}_1, \ldots, \mathcal{R}_l\}$. Our algorithm will maintain an $\mathcal{R}$-feasible matching for each partition. Throughout this article, we will fix the $r$-division in the definition of $\mathcal{R}$-feasibility to be $\mathcal{R}(G)$. For any $\mathcal{R}$-feasible matching, when obvious from the context, we will not explicitly mention the induced sub-graph for which it is defined.

The following lemma bounds the cost of an $\mathcal{R}$-optimal matching on $G$:

Lemma 3.1. *For a planar bipartite graph $G(A \cup B, E)$ with a positive integer edge cost function* c*, let $M$ be an $\mathcal{R}$-optimal matching and $M_{OPT}$ be some optimal matching. Then,* $\mathrm{c}(M) \leq \mathrm{c}(M_{OPT}) + (k + 1)n$ *where $k$ is a constant in definition (2.2).*

Proof. Since $M$, $y(\cdot)$ is $\mathcal{R}$-optimal, we have that

$$\mathrm{c}(M) = \sum_{u \in A \cup B} y(u). \tag{7}$$

From condition (5), every edge $(u, v) \in M_{\mathrm{OPT}}$ will satisfy $y(u) + y(v) \leq \mathrm{c}(u, v) + \delta_{uv}$. Since $M_{\mathrm{OPT}}$ is a perfect matching, we have

$$\begin{aligned} \sum_{u \in A \cup B} y(u) &= \sum_{(u,v) \in M_{\mathrm{OPT}}} (y(u) + y(v)) \\ &\leq \mathrm{c}(M_{\mathrm{OPT}}) + \sum_{(u,v) \in M_{\mathrm{OPT}}} \delta_{uv}. \end{aligned} \tag{8}$$

There are at most $kn/\sqrt{r}$ boundary vertices, and so there are at most $kn/\sqrt{r}$ edges $(u, v)$ in $M_{\mathrm{OPT}}$ such that $\delta_{uv} = \lceil\sqrt{r}\rceil$. For the other $n - kn/\sqrt{r}$ edges $(u, v)$ of $M_{\mathrm{OPT}}$, $\delta_{uv} = 1$. Therefore,

$$\sum_{(u,v) \in M_{\mathrm{OPT}}} \delta_{uv} \leq \frac{kn}{\sqrt{r}}(\sqrt{r} + 1) + n - \frac{kn}{\sqrt{r}} = (k + 1)n. \tag{9}$$

Combining Equations (7) and (8) with (9), we get

$$\mathrm{c}(M) = \sum_{u \in A \cup B} y(u) \leq \mathrm{c}(M_{\mathrm{OPT}}) + (k + 1)n. \qquad \square$$

As in the Gabow-Tarjan algorithm, for every edge $(a, b) \in E$, we redefine its weight to be $\mathrm{c}^*(a, b) = (k + 1)(n + 1)\mathrm{c}(a, b)$. Since this uniform scaling of edge costs preserves the set of optimal matchings, Lemma 3.1 implies that an $\mathcal{R}$-optimal matching of the vertices of $A, B$ with edge weights $\mathrm{c}^*(\cdot, \cdot)$ corresponds to an optimal matching with the original edge costs $\mathrm{c}(\cdot, \cdot)$.

Our algorithm also consists of *scales*. For any edge $(u, v)$, let $b_1, b_2 \ldots b_\ell$ be the binary representation of $c^*(u, v)$. In the $i$th scale, the cost of an edge, $c_i(u, v)$, corresponds to the most significant $i$ bits of $c^*(u, v)$. Each scale of our algorithm takes as input a planar bipartite graph on $A, B$, with a cost function $c_i(\cdot, \cdot)$, and a set of dual weights $y(v)$ for every vertex $v \in A \cup B$, and returns an $\mathcal{R}$-optimal matching. We transfer the dual weights from scale $i$ to scale $i + 1$ by simply setting, for any vertex $v \in A \cup B$, $y(v) \leftarrow 2y(v) - \max\{1, i_v\lceil\sqrt{r}\rceil\}$. As in the Gabow-Tarjan algorithm, the optimal matching does not change if we replace the cost of each edge with its slack. Therefore, at the beginning of scale $i + 1$, we set the edge cost to be its slack $s_{i+1}(u, v) = c_{i+1}(u, v) - y(u) - y(v)$. For any edge $(u, v)$ of the $\mathcal{R}$-optimal matching $M_i$ from the previous scale $i$, the slack will be positive and at most 3 if both $u$ and $v$ are internal vertices, $2 + \lceil\sqrt{r}\rceil$ if either $u$ or $v$ is a boundary vertex and the other is an internal vertex, and $1 + 2\lceil\sqrt{r}\rceil$ if both $u$ and $v$ are boundary vertices. Since there are at most $kn/\sqrt{r}$ boundary nodes, the cost of the optimal matching with slack costs is at most $(k + 2)n$. Therefore, at the start of each scale, the cost of every edge is a positive integer and the cost of the optimal matching is $O(n)$. In the next section, we describe an algorithm for each scale. This algorithm takes a graph with positive integer edge costs, an optimal matching with cost $O(n)$, and computes an $\mathcal{R}$-optimal matching in $O(n^{4/3}\log^2 n)$ time. After $O(\log(nC))$ scales, the $\mathcal{R}$-optimal matching returned by our algorithm will also be an optimal matching.

## 4 ALGORITHM FOR EACH SCALE

Our algorithm takes a planar bipartite graph $G(A \cup B, E)$ with positive integer cost of $c(u, v)$ for every edge $(u, v)$ such that the optimal matching has a cost no more than $(k + 2)n$. It has two steps. The first step of the algorithm will simply execute $O(\sqrt{r})$ iterations of the Gabow-Tarjan algorithm. At the end of this step, our algorithm will have computed a 1-feasible matching $M$ with at most $O(n/\sqrt{r})$ free vertices. Additionally, from the properties of the Gabow-Tarjan algorithm, for every free vertex $a \in A$, $y(a) = 0$, and for every free vertex $b \in B$, $y(b) = \max_{b' \in B} y(b')$. Since every 1-feasible matching also satisfies conditions (5) and (6), $M$ is also an $\mathcal{R}$-feasible matching. Therefore, at the end of the first step, we have the following:

Lemma 4.1. *At the end of the first step of our algorithm, the matching $M$ and the dual weights $y(\cdot)$ form an $\mathcal{R}$-feasible matching for the graph $G$, the number of free vertices with respect to $M$ is at most $O(n/\sqrt{r})$, for every free vertex $a \in A$, $y(a) = 0$, and for every free vertex $b \in B$, $y(b) = \max_{b' \in B} y(b')$.*

For our algorithm, we will define the *net-cost* of an augmenting path $P$, denoted by $\phi(P)$, by

$$\phi(P) = \sum_{(u,v)\in P\setminus M} (c(u, v) + \delta_{uv}) - \sum_{(u,v)\in P\cap M} c(u, v).$$

This definition also extends to alternating paths and alternating cycles in a straightforward way. We also define an edge $(u, v)$ as an *admissible edge* if $(u, v) \in M$ or if $(u, v) \notin M$ and,

$$y(u) + y(v) = c(u, v) + \delta_{uv}.$$

As in the Hungarian and Gabow-Tarjan algorithms, in the second step of our algorithm, we will iteratively compute an augmenting path of admissible edges and augment the matching along this path. The augmenting path computed by our algorithm can also be seen as a minimum net-cost augmenting path.

We define the slack of an edge $(u, v) \in M$, $s(u, v) = 0$, and $(u, v) \in E \setminus M$ to be $s(u, v) = c(u, v) + \delta_{uv} - y(u) - y(v)$. From the $\mathcal{R}$-feasibility conditions (5) and (6), it follows that the slack $s(u, v) \geq 0$.

We can easily compute the minimum net-cost augmenting path by conducting a Hungarian search. The search procedure will modify the dual weights of all the vertices in $G$ and find an

augmenting path of admissible edges. Unfortunately, this search may require us to visit and update the dual weights of all the $n$ vertices of the graph requiring $O(n)$ time, which is too slow for our purpose. To speed this up, we use a compressed representation $H$ of the residual graph $G_M$. In this representation, we only maintain the boundary vertices of the $r$-division and connect two boundary vertices $u$ and $v$ with a directed edge if and only if there is a directed path from $u$ to $v$ in some partition $\mathcal{R}_j$. We assign a weight to $(u, v)$ that is equal to the net-cost of a minimum net-cost path from $u$ to $v$ in $\mathcal{R}_j$. Such a compressed representation $H$ of the residual graph has $O(n/\sqrt{r})$ vertices and $O(n)$ edges. These edges can be computed in $O(r \log r)$ time per partition with a total computation time of $O(n \log n)$. Also, any directed path $P$ in $H$ can be *projected* to an augmenting path in $G$ by simply replacing each edge of $P$ with the minimum net-cost directed path of $G_M$ that it represents.

To compute the minimum net-cost augmenting path, we show that it is sufficient to apply Hungarian search on $H$, which can be done in $O((n/\sqrt{r}) \log r \log n)$ time. However, as stated earlier, this search may affect the dual weights of all the $n$ vertices. We avoid updating all the dual weights each search by introducing the notion of a planar-feasible matching. In a planar-feasible matching, we will maintain a separate set of dual weights for each partition and another set of dual weights for all the vertices of $H$; a similar strategy was used in Reference [10]. Using the dual weights of $H$, we find the minimum net-cost augmenting path $P$ and augment the matching along its projection $\overrightarrow{P}$ on $G_M$. This augmentation changes the residual graph and, therefore, its compressed representation $H$ as well. We will then update the dual weights and the edges of every partition of the compressed residual graph that contains at least one edge of $\overrightarrow{P}$. Note that the dual weights of the vertices in other partitions may be affected as well. However, we will update them only when an augmenting path contains its edges. We show that the edges of a single partition of the compressed residual graph can be updated in $O(r \log r)$ time. The total time taken by the algorithm, therefore, is $O((n/\sqrt{r}) \log r \log n)$ to compute an augmenting path $P$ and $O(|P| r \log r)$ to update the dual weights and the edges of $H$. Therefore, the remaining $O(n/\sqrt{r})$ augmenting paths can be computed in $O((n^2/r) \log r \log n + r \log r \sum_i |P|)$ time where $\sum_i |P|$ is the sum of the lengths of all these $O(n/\sqrt{r})$ augmenting paths. Interestingly, we show that the sum of the lengths of the augmenting paths is bounded by $O((n/\sqrt{r}) \log n)$. This bounds the running time of our algorithm by $O((n^2/r + n\sqrt{r}) \log r \log n)$ or $O(n^{4/3} \log^2 n)$ when $r = n^{2/3}$.

*Handling Edges in Multiple Partitions:* Recollect that every edge of $G$ belongs to one or more partitions of the $r$-division $\mathcal{R}(G)$. However, for purposes of efficiency, we would like to assign every edge to a unique partition. To accomplish this, we use an idea from Reference [10] and convert $G$ into a multigraph. For any edge $(u, v)$, suppose $(u, v)$ participates in $t$ distinct partitions of the $r$-division $\mathcal{R}(G)$. We add $t$ copies of $(u, v)$ in $G$, one for each partition it belongs to. However, we assign only one of these $t$ copies a cost of $c(u, v)$ and all other copies are assigned a cost of $\infty$. The $r$-division $\mathcal{R}(G)$ can be seen as an edge-disjoint partition of the multigraph $G$. Note that any edge with a cost of $\infty$ will never participate in any augmenting path or any matching maintained by the algorithm and, therefore, is cosmetic in nature. However, we need these edges to ensure that the number of holes in the planar embedding remains $O(1)$. From here on, we present our algorithm by assuming that the edge $(u, v)$ belongs to a unique partition, the one that contains the copy of $(u, v)$ with its original cost.

In the following, we will define the compressed residual graph and formally introduce planar-feasible matchings. After that, we will show that we can quickly convert any $\mathcal{R}$-feasible matching to a planar-feasible matching and vice versa. Finally, we will present the second step of our algorithm that computes an $\mathcal{R}$-optimal matching.

*Compressed Residual Graph H:* We now describe the compressed residual graph $H$, which will guide the execution of each iteration of the second step of our algorithm. For a matching $M$, let $G_M$ denote the (directed) residual graph with respect to $M$. Let $\mathcal{R}(G_M) = \mathcal{R}(G)$ be the $r$-division of $G_M$ as given by Definition 2.2. Let $A_F$ and $B_F$ denote the set of free vertices (not matched by $M$) of $A$ and $B$, respectively. Our compressed graph $H$ will be a *weighted multi-graph* whose vertex set is $V_H$ and the edge set is $E_H$ is defined next.

We define the vertex set $V_j^H$ and the edge set $E_j^H$ for a single partition $\mathcal{R}_j$ of the $r$-division. The vertex set $V_H$ and the edge set $E_H$ is simply the union of all the edges and vertices over all partitions. For every partition $\mathcal{R}_j$, $V_j^H$ contains the boundary vertices $\mathcal{K}_j$. Also, if there is at least one internal vertex of $A$ (respectively, $B$) that is free (unmatched), i.e., $(V_j \setminus \mathcal{K}_j) \cap A_F \neq \emptyset$ (respectively, $(V_j \setminus \mathcal{K}_j) \cap B_F \neq \emptyset$), then we create a special vertex $a_j$ (respectively, $b_j$) to represent all vertices in this set in $V_j^H$. We call these two additional vertices for $\mathcal{R}_j$ the *free internal vertices* of $\mathcal{R}_j$. We set $V_j^H = \mathcal{K}_j \cup \{a_j, b_j\}$, $A_j^H = (\mathcal{K}_j \cap A) \cup \{a_j\}$, and $B_j^H = (\mathcal{K}_j \cap B) \cup \{b_j\}$. The free vertices of $\mathcal{R}_j$ are represented by $B_j^F = (B_F \cap \mathcal{K}_j) \cup \{b_j\}$ and $A_j^F = (A_F \cap \mathcal{K}_j) \cup \{a_j\}$. The vertex set $V_H$ of $H$ is thus given by $V_H = \bigcup_{j=1}^{l} V_j^H$, $B_H = \bigcup_{j=1}^{l} B_j^H$, and $A_H = \bigcup_{j=1}^{l} A_j^H$. The free vertices of $H$ are given by $A_H^F = \bigcup_{j=1}^{l} A_j^F$ and $B_H^F = \bigcup_{j=1}^{l} B_j^F$.

Next, we define the set of edges $E_j^H$ for each partition $\mathcal{R}_j$. For any $u, v \in V_j^H$ $(u \neq v)$ there is an edge from $u$ to $v$ if

(1) $u, v \in \mathcal{K}_j$, i.e., $u$ and $v$ are boundary vertices and there is a directed path $\overrightarrow{P}$ from $u$ to $v$ in $G_M$ that only passes through the edges of $\mathcal{R}_j$. Let $\overrightarrow{P}_{u,v,j}$ be a minimum net-cost path from $u$ to $v$ in $\mathcal{R}_j$. We denote this type of edge as a *boundary-to-boundary* edge.

(2) $u = b_j$, $v \in \mathcal{K}_j$, and there is a directed path $\overrightarrow{P}$ in $G_M$ from some free vertex in $B_F \cap (V_j \setminus \mathcal{K}_j)$ to $v$ that only passes through the edges of $\mathcal{R}_j$. Let $\overrightarrow{P}_{u,v,j}$ be a minimum net-cost path from $u$ to $v$ in $\mathcal{R}_j$.

(3) $u \in \mathcal{K}_j$, $v = a_j$, and there is a directed path $\overrightarrow{P}$ in $G_M$ from $u$ to some free vertex in $A_F \cap (V_j \setminus \mathcal{K}_j)$ that only passes through the edges of $\mathcal{R}_j$. Let $\overrightarrow{P}_{u,v,j}$ be a minimum net-cost path from $u$ to $v$ in $\mathcal{R}_j$.

(4) $u = b_j$ and $v = a_j$ are free vertices and there is a directed path $\overrightarrow{P}$ in $G_M$ from some vertex in the set $B_F \cap (V_j \setminus \mathcal{K}_j)$ to a vertex in the set $A_F \cap (V_j \setminus \mathcal{K}_j)$ that only passes through the edges in $\mathcal{R}_j$. Let $\overrightarrow{P}_{u,v,j}$ be a minimum net-cost path from $u$ to $v$ in $\mathcal{R}_j$.

We set the weight of $(u, v)$ to be $\phi(\overrightarrow{P}_{u,v,j})$. We also refer to this edge $(u, v)$ as an edge of partition $\mathcal{R}_j$ and denote the set of all edges of partition $\mathcal{R}_j$ as $E_j^H$. The set of edges of $H$ is simply $E_H = \bigcup_j E_j^H$. Note that $H$ is a multi-graph, as there can be directed path from $u$ to $v$ in multiple partitions of the $r$-division. This completes the description of the compressed residual graph $H$. See Figure 2 for an example of the construction of $H$ for a partition.

*Planar Feasibility:* Next, we will define a *planar-feasible matching*. First, we decompose the edges of the matching based on the partition of the $r$-division they belong to. We denote as $M_j$ the edges of $M$ that belong to partition $\mathcal{R}_j$. $M = \bigcup_{j=1}^{l} M_j$. For each partition $\mathcal{R}_j$, we maintain a dual weight $y_j(v)$ for every vertex in $v \in V_j$. These dual weights $y_j(\cdot)$ along with $M_j$ form an $\mathcal{R}$-feasible matching. Additionally, we also store a dual weight $\tilde{y}(v)$ for every $v \in V_H$. We say that the dual weights $\tilde{y}(\cdot)$ are *$H$-feasible* if they satisfy the following conditions: For each partition $\mathcal{R}_j$, and for every directed

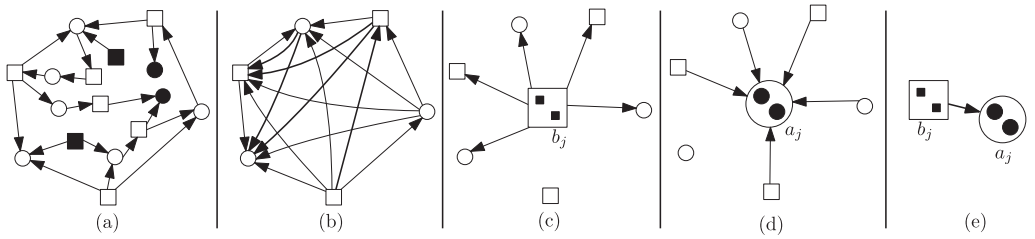


Fig. 2. (a) A partition $\mathcal{R}_j$. The squares represent vertices of $B$ and the circles represent vertices of $A$. Free vertices are filled in. (b) The boundary-to-boundary edges of $H$ for $\mathcal{R}_j$. (c) The edges from $b_j$ to $\mathcal{K}_j$. (d) The edges from $\mathcal{K}_j$ to $a_j$. (e) There is a single edge from $b_j$ to $a_j$.

edge $(u, v) \in E_j^H$,

$$\begin{aligned}
\tilde{y}(u) + \tilde{y}(v) &\leq \phi(\overrightarrow{P}_{u,v,j}) \qquad \text{if } (u,v) \in (B_j^H \times A_j^H) \\
-\tilde{y}(u) - \tilde{y}(v) &\leq \phi(\overrightarrow{P}_{u,v,j}) \qquad \text{if } (u,v) \in (A_j^H \times B_j^H) \\
\tilde{y}(u) - \tilde{y}(v) &\leq \phi(\overrightarrow{P}_{u,v,j}) \qquad \text{if } (u,v) \in (B_j^H \times B_j^H) \\
-\tilde{y}(u) + \tilde{y}(v) &\leq \phi(\overrightarrow{P}_{u,v,j}) \qquad \text{if } (u,v) \in (A_j^H \times A_j^H).
\end{aligned}$$

As in the Gabow-Tarjan algorithm,

(a) For every vertex $v \in A_H$, $\tilde{y}(v) \leq 0$ and for every free vertex $v \in A_H^F$, $\tilde{y}(v) = 0$,
(b) For every vertex $v \in B_H$, $\tilde{y}(v) \geq 0$ and for every free vertex $v \in B_H^F$, $\tilde{y}(v) = y_{max}$, where $y_{max} = \max_{v \in A_H \cup B_H} \tilde{y}(v)$.

Using (a) and (b), we can restate the $H$-feasibility conditions more compactly as

$$|\tilde{y}(u)| - |\tilde{y}(v)| \leq \phi(\overrightarrow{P}_{u,v,j}). \tag{10}$$

For a planar bipartite graph $G$, and an $r$-division $\mathcal{R}(G)$, we say that a matching $M$, a set of dual weights $y_j(\cdot)$ for the vertices of each partition $\mathcal{R}_j$, and a set of dual weights $\tilde{y}(\cdot)$ for the vertices $V_H$, form a planar-feasible matching if in addition to (a) and (b), the following three conditions are satisfied:

(c) For every partition $\mathcal{R}_j$, the matching $M_j$ and dual weights $y_j(\cdot)$ form an $\mathcal{R}$-feasible matching,
(d) The dual weights $\tilde{y}(\cdot)$ are $H$-feasible,
(e) For each partition $\mathcal{R}_j$ and any $v \in \mathcal{K}_j$ , $|\tilde{y}(v)| \geq |y_j(v)|$. For every vertex $a \in (V_j \setminus \mathcal{K}_j) \cap A_F$ (respectively, $b \in (V_j \setminus \mathcal{K}_j) \cap B_F$), $|y_j(a)| = 0$ (respectively, $y_j(b) \leq y_{max}$).

Note that a boundary vertex has many different dual weights assigned to it, one for each of the partitions it belongs to. During the course of our algorithm, the magnitudes of the dual weights of vertices in $H$ may increase. As we do not immediately update the dual weights of all vertices in $G$, for some partitions $\mathcal{R}_j$ the dual weights $y_j(\cdot)$ may not reflect the updated dual weights. This condition is captured by (e). We refer to any perfect matching that is also a planar-feasible matching as a *planar optimal matching*. For any edge $(u, v) \in E_j^H$, let the slack $s_H(u, v) = \phi(\overrightarrow{P}_{u,v,j}) - |\tilde{y}(u)| + |\tilde{y}(v)|$. Following our convention, we set $H'$ to be a graph identical to $H$ with the weight of every edge replaced by its slack.

At the end of the first step of our algorithm, we have a matching $M$ and a set of dual weights that form an $\mathcal{R}$-feasible matching. In Section 4.1, we describe how we can compute a planar-feasible

matching from this $\mathcal{R}$-feasible matching $M, y(\cdot)$. At the end of each scale, our algorithm produces a planar optimal matching. In Section 4.2, we describe how to compute an $\mathcal{R}$-optimal matching from this planar optimal matching. Using some of the procedures introduced in Sections 4.1 and 4.2, we describe the second step of our algorithm in Section 4.3.

### 4.1 Computing a Planar Feasible Matching from an $\mathcal{R}$-feasible Matching

At the end of the first step, we have an $\mathcal{R}$-feasible matching $M, y(\cdot)$ that also satisfies $y(u) = 0$ for all $u \in A_F$ and $y(v) = y_{max}$ for all $v \in B_F$. In this section, we will present an algorithm to compute a planar-feasible matching from this $\mathcal{R}$-feasible matching $M$ and its set of dual weights $y(\cdot)$. For every vertex $v \in A \cup B$, and for every partition $\mathcal{R}_j$ such that $v \in V_j$, we set $y_j(v) = y(v)$. For every boundary vertex $v \in \mathcal{K}$, we set $\tilde{y}(v) = y(v)$. We also set, for every partition $\mathcal{R}_j$, $\tilde{y}(a_j) = 0$ and $\tilde{y}(b_j) = y_{max}$. Note that, from Lemma 4.1, conditions (a)–(c) and (e) are trivially satisfied. The next lemma shows that dual weights $\tilde{y}(\cdot)$ satisfy $H$-feasibility and, therefore, (d) holds.

Lemma 4.2. *Consider a matching $M_j$ and a set of dual weights $y(\cdot)$ for a partition $\mathcal{R}_j$ such that $M_j, y(\cdot)$ is $\mathcal{R}$-feasible. Suppose the dual weights of all vertices of $A$ in $V_j$ are non-positive and the dual weights of all vertices of $B$ in $V_j$ are non-negative. For any two vertices $u, v \in V_j$, let the directed path $\overrightarrow{P}_{u,v,j}$ be a minimum net-cost alternating path from $u$ to $v$ in the residual graph of $\mathcal{R}_j$. Then,*

$$|y(u)| - |y(v)| \leq \phi(\overrightarrow{P}_{u,v,j}). \tag{11}$$

*Furthermore,*

$$\phi(\overrightarrow{P}_{u,v,j}) - |y(u)| + |y(v)| = \sum_{(a,b)\in\overrightarrow{P}_{u,v,j}} s(a,b). \tag{12}$$

Proof. Let $P = \overrightarrow{P}_{u,v,j}$. As $u$ and $v$ can belong to either $A$ or $B$, we need to consider four possible cases. We will provide a proof for the case where $u \in B$ and $v \in A$. An identical argument will also hold for the other three cases.

From the definition of net-cost, we have

$$\begin{aligned}\phi(P) &= \sum_{(a,b)\in P\setminus M} (\mathrm{c}(a,b) + \delta_{ab}) - \sum_{(a,b)\in P\cap M} \mathrm{c}(a,b)\\ &= \sum_{(a,b)\in P\setminus M} (\mathrm{c}(a,b) + \delta_{ab}) - \sum_{(a,b)\in P\cap M} (y(a) + y(b)).\end{aligned}$$

Since $u \in B$ and $v \in A$, both the first and the last edge of $P$ are not in the matching $M$. Therefore, we can write the above equation as:

$$\begin{aligned}\phi(P) &= \sum_{(a,b)\in P\setminus M} (\mathrm{c}(a,b) + \delta_{ab} - y(a) - y(b)) + y(u) + y(v)\\ &= y(u) + y(v) + \sum_{(a,b)\in P\setminus M} s(a,b)\\ &= \sum_{(a,b)\in P\setminus M} s(a,b) + |y(u)| - |y(v)|.\end{aligned}$$

The last equality holds from the fact that dual weight of $u$ is non-negative and the dual weight of $v$ is non-positive. □

Using a similar argument, we can extend Lemma 4.2 to the entire graph leading to the following:

Lemma 4.3. *Consider a matching $M$ and a set of dual weights $y(\cdot)$ on the vertices of $G(A \cup B, E)$ such that $M, y(\cdot)$ is $\mathcal{R}$-feasible. Suppose all vertices of $A$ have a non-positive dual weight and all vertices*

*of $B$ have a non-negative dual weight. For any two vertices $u, v \in A \cup B$, let the directed path $\overrightarrow{P}_{u,v}$ be a minimum net-cost path from $u$ to $v$ in $G_M$. Then,*

$$|y(u)| - |y(v)| \quad \leq \quad \phi(\overrightarrow{P}_{u,v}).$$

*Furthermore,*

$$\phi(\overrightarrow{P}_{u,v}) - |y(u)| + |y(v)| = \sum_{(a,b)\in\overrightarrow{P}_{u,v}} s(a,b).$$

Using the following lemma, we will provide an efficient procedure called Construct to compute all the edges of the compressed residual graph $H$. After that, we describe the data structures in which we store these edges.

Lemma 4.4. *Let $\mathcal{R}'_j$ be a directed graph identical to the directed graph $\mathcal{R}_j$ except that the cost of an edge $(a, b)$ is the slack $s(a, b)$. Then, for a partition $\mathcal{R}_j$ and any two vertices $u, v$ in $V_j$, the minimum net-cost directed path $\overrightarrow{P}_{u,v,j}$ from $u$ to $v$ in $\mathcal{R}_j$ is also the minimum-cost directed path between $u$ and $v$ in $\mathcal{R}'_j$. Furthermore, we can derive $\phi(\overrightarrow{P}_{u,v,j})$ from the dual weights $y(u)$, $y(v)$ and the cost of the shortest path in $\mathcal{R}'_j$.*

Proof. We highlight our argument for the case where $u \in B$ and $v \in A$; the argument for every other case is identical.

For any directed path $P$ between $u$ and $v$, since $u \in B$ and $v \in A$ and, from the proof of Lemma 4.2, we know that

$$\phi(P) \quad = \quad \sum_{(a,b)\in P\setminus M} s(a,b) + y(u) + y(v). \tag{13}$$

This is true for every path $P$ from $u$ to $v$. Furthermore, the dual weights $y(u)$ and $y(v)$ in the above equation are the same for any path $P$. Therefore, we conclude that the minimum net-cost path will also be the minimum-cost path between $u$ and $v$ in $\mathcal{R}'_j$. Let $P^*$ be this minimum net-cost path. Since $P^*$ also satisfies Equation (13), the sum of the slacks of the edges on the path $P^*$ along with $y(u)$ and $y(v)$ will give us the minimum net-cost between $u$ and $v$. □

For the rest of the algorithm, we define the slack on any directed edge $(u, v) \in E^H_j$ to be $\phi(\overrightarrow{P}_{u,v,j}) - |\tilde{y}(u)| + |\tilde{y}(v)|$. From Lemma 4.4, it follows that slack of the edge $(u, v)$ is non-negative and exactly equal to $\sum_{(a,b)\in\overrightarrow{P}_{u,v,j}} s(a, b)$, provided that the first vertex $\hat{u}$ of $\overrightarrow{P}_{u,v,j}$ has $y_j(\hat{u}) = \tilde{y}(u)$ and the last vertex $\hat{v}$ of $\overrightarrow{P}_{u,v,j}$ has $y_j(\hat{v}) = \tilde{y}(v)$. Following our convention, we use $H'$ to denote the compressed residual graph with the same edge set as $H$ but with the edge weights being replaced with their slacks.

Our initial choice of $\tilde{y}(\cdot)$ is $H$-feasible. To assist in the execution of the second step of our algorithm, we explicitly compute the edges of $H$ and store them and their slacks in a data structure. This data structure will assist us in the fast execution of Dijkstra's algorithm on $H'$.

Using Lemma 4.4 the Construct procedure will, for any partition $\mathcal{R}_j$ of the $r$-division of $G_M$, compute the edges of $E^H_j$ in $O(r \log r)$ time.

*The* Construct *Procedure:* This procedure takes a partition $\mathcal{R}_j$ of $G_M$ as input and constructs the edges of $E^H_j$. We assume the matching $M_j$ and the dual weights $y_j(\cdot)$ are $\mathcal{R}$-feasible. Let $\mathcal{R}'_j$ be a graph identical to $\mathcal{R}_j$, where each edge has a cost equal to its slack as defined by the current dual assignment $y_j(\cdot)$. We note that all edges in $\mathcal{R}'_j$ are non-negative. Since the dual assignment is feasible with respect to $M_j$, we know by Lemma 4.4 that the path of minimum net-cost between

two vertices is also the path of minimum total slack in $\mathcal{R}_j^{'}$. Therefore, it is sufficient to compute the shortest path lengths in $\mathcal{R}_j^{'}$ and use Equation (12) to compute the net-cost of the minimum net-cost path in constant time. We will describe how to compute the shortest paths in $\mathcal{R}_j^{'}$.

Recollect that there are four types of edges in $E_j^H$. We first explain how to compute the boundary-to-boundary edges $(u, v) \in E_j^H$; that is, $(u, v) \in \mathcal{K}_j \times \mathcal{K}_j$. We construct a multiple-source shortest paths data structure described in Reference [11] on $\mathcal{R}_j'$ in $O(|V_j| \log |V_j|) = O(r \log r)$ time. This structure can answer the following query in $O(\log |V_j|) = O(\log r)$ time: Given any vertex $s$ in $V_j$, and a vertex $t$ on a distinguished face of $\mathcal{R}_j'$, what is the shortest $s$ to $t$ distance in $\mathcal{R}_j'$? Using this, we can obtain the boundary-to-boundary edges of $\mathcal{R}_j$ by simply querying this data structure and using Equation (12). We note that the boundary vertices may be located at $O(1)$ different holes; for each such hole $\mathcal{H}$, we build a multiple source shortest paths data structure with $\mathcal{H}$ as the distinguished face. By querying this data structure, we can obtain all the boundary-to-boundary edges from vertices of $\mathcal{H}$ to the boundary vertices not in $\mathcal{H}$. Since there are a constant number of holes that contain all the boundary vertices of each partition, this will not increase the asymptotic complexity. To compute the other edges of $E_j^H$, we execute Dijkstra's algorithm three times. Let $\mathcal{R}_j''$ be a graph where every edge of $\mathcal{R}_j'$ is reversed. The first execution of Dijkstra's algorithm is on $\mathcal{R}_j'$ starting from the vertices of $B_F \cap (V_j \setminus \mathcal{K}_j)$ to all vertices of $\mathcal{K}_j$. For any vertex $v \in \mathcal{K}_j$, since the dual weight of every vertex $B_F \cap (V_j \setminus \mathcal{K}_j)$ is the same, the cost $\phi(\overrightarrow{P}_{b_j, v, j})$ can be simply computed using Lemma 4.4. Similarly, the second execution of Dijkstra's algorithm is on $\mathcal{R}_j''$ where we find distance to every vertex $v \in \mathcal{K}_j$ from some $A_F \cap (V_j \setminus \mathcal{K}_j)$ and compute $\phi(\overrightarrow{P}_{v, a_j, j})$. The final edge from $b_j$ to $a_j$ can be computed by simply executing Dijkstra's algorithm starting from the vertices of $B_F \cap (V_j \setminus \mathcal{K}_j)$ and compute the shortest path cost to any point in $A_F \cap (V_j \setminus \mathcal{K}_j)$. Since the dual weights of every free vertex of $B_F \cap (V_j \setminus \mathcal{K}_j)$ is identical, and the dual weights of every free vertices of $A_F \cap (V_j \setminus \mathcal{K}_j)$ is 0, using Lemma 4.4, we can obtain the weight $\phi(\overrightarrow{P}_{b_j, a_j, j})$. Together, these three executions of Dijkstra's algorithm will compute the remaining edges of $E_j^H$. The total time taken to compute all the edges of $E_j^H$ is $O(r \log r)$.

*Monge Property of the Boundary-to-boundary Edge Costs:* We say that any matrix $M$ satisfies the Monge property if for any $i < i'$ and $j < j'$, $M_{i,j} + M_{i',j'} \leq M_{i,j'} + M_{i',j}$. Multiple previous results described how to decompose the boundary-to-boundary edge costs of a partition $\mathcal{R}_j$ into matrices with the Monge property [5, 10, 15]. We will briefly provide the intuition for any partition $\mathcal{R}_j$ where there are no holes; see Reference [10] for a detailed description on how cases involving holes are handled. Suppose all the boundary vertices $\mathcal{K}_j$ are on the infinite face. We can partition the boundary-to-boundary edges into ordered bipartite groups. The first two groups are simply formed by dividing any clockwise ordering of all the boundary vertices into two equal parts (we refer to the first half as the left side and the second half as the right side). Now, we create a group containing all edges going from a vertex in the left side to a vertex in the right side. We also create another group to represent all the edges going from a vertex in the right side to the left side. To capture the edges that go between two vertices of the left (respectively, right) side, we simply recursively divide the left (respectively, right) partition into two sets of equal sizes and again capture all the edges that go between them. Note that any boundary vertex participates in at most $O(\log r)$ groups within a partition. Also, if we represent the edge-costs of any bipartite group as a matrix, then this matrix satisfies the Monge property.

*Preprocessing the boundary-to-boundary edges:* For the sake of our algorithm, it is useful to build a data structure to store the boundary-to-boundary edges and their slacks in the ordered bipartite

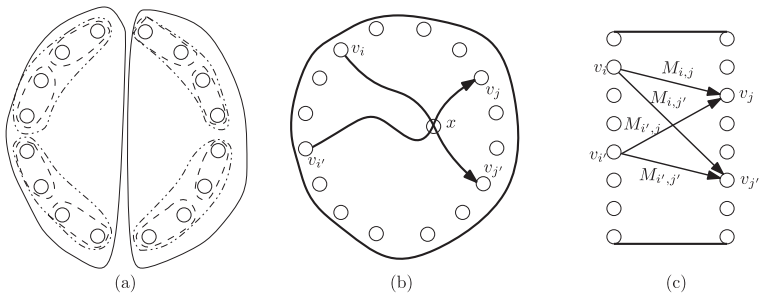


Fig. 3. (a) A division of the boundary vertices of a partition into $O(\log n)$ bipartite groups. (b) Let $d(u, v)$ be the shortest path distance within the partition from $u$ to $v$. The shortest path from $v_i$ to $v'_j$ and the shortest path from $v'_i$ to $v_j$ must cross at a vertex $x$. (c) The Monge property within a bipartite group. $M_{i,j} + M_{i',j'} \leq M_{i',j} + M_{i,j'}$.

groups described above. For each ordered bipartite group, we have a cost matrix $\mathcal{M}$. For any edge $(u, v) \in E_j^H$ that participates in the bipartite group, we store its cost, i.e., $\phi(\overrightarrow{P}_{u,v,j})$, in the cost matrix. Note that the slack, $\phi(\overrightarrow{P}_{u,v,j}) - |y(u)| + |y(v)|$ of any edge can be computed in $O(1)$ time given its cost and the dual weights of its end vertices. Thus, we can implicitly represent a matrix $\mathcal{M}'$ based on $\mathcal{M}$ with slacks as costs. As shown in Reference [5], shortest path distances in planar graphs between vertices of the ordered bipartite groups satisfy the Monge-property (see Figure 3). From Lemma 4.2, the slack on any edge $(u, v) \in E_j^H$ is the length of the shortest path from $u$ to $v$ in $\mathcal{R}'_j$, meaning $\mathcal{M}'$ also satisfies the Monge property. In the Construct procedure, we build the Monge matrix range-minimum data structure (described in Reference [10]) on the slack matrix $\mathcal{M}'$ in time $O(\sqrt{x} \log x)$ where $x$ is the number of vertices in the bipartite group. For any vertex $v$ on the left, and any interval of nodes on the right, this data structure can answer the query, what is the minimum edge from $v$ to a vertex in the interval within the group? The Monge matrix range-minimum data structure in conjunction with the bipartite groups can be used to execute Dijkstra's algorithm on $H'$ quickly, which helps us compute a minimum net-cost augmenting path in $H$ efficiently.

Lemma 4.5. *Given an $\mathcal{R}$ feasible matching $M_j$, $y_j(\cdot)$, the Construct procedure builds the edges of $E_j^H$ in $O(r \log r)$ time. In addition, it stores the boundary-to-boundary edges in ordered bipartite groups in $O(r \log r)$ time where the edge cost matrix for each group satisfies the Monge property. It also builds a Monge-matrix range minimum data structure on the slacks of the boundary-to-boundary edges of $E_j^H$ for every bipartite group of $\mathcal{R}_j$ in $O(\sqrt{r} \log^2 r)$ time.*

Corollary 4.6. *Let $M, y(\cdot)$ be the $\mathcal{R}$-feasible matching computed at the end of the first step of our algorithm. Given $M, y(\cdot)$, we can use the Construct procedure to compute the graph $H$ in $O(n \log r)$ time.*

Using these data structures, one can compute shortest path between a vertex $b \in B_H^F$ to a vertex $a \in A_H^F$ in $H'$ using the following algorithm nearly identical to the one presented in Reference [10] in $O((n/\sqrt{r}) \log r \log n)$ time. The algorithm also computes the distance $c_v$ to a vertex $v \in V_H$ from an arbitrary vertex of $B_H^F$.

- In the first step, the algorithm finds the distance from $b_j$ to every boundary vertex of $\mathcal{K}_j$ within $\mathcal{R}'_j$. The minimum net-cost paths have corresponding edges explicitly pre-computed in $H$, and their slacks can each be computed in $O(1)$ time. For each boundary vertex $v$, we

set $c_v = \min_{\mathcal{R}_j \in \mathcal{R}, (b_j, v) \in E_j^H} s_H(b_j, v)$ as the initial shortest-path *distance estimate* from a free vertex. This takes $O(\sqrt{r})$ per partition and a total of $O(n/\sqrt{r})$ time for the entire graph $H'$.

- In the second step, the algorithm executes an implementation of Dijkstra's algorithm given by Fakcharoenphol and Rao [5], which uses the bipartite groups and range-minimum query data structures. Dijkstra's algorithm uses the initial distance estimates $c_v$ for each vertex $v \in \mathcal{K}$. The algorithm updates the value $c_v$ to reflect the minimum distance from any free vertex of $B$ to $v$. Fakcharoenphol and Rao have shown how to execute such a procedure in $O((n/\sqrt{r}) \log r \log n)$ time[3] by efficiently computing the next edge to add to the shortest path tree. For a description of their algorithm, see the appendix of Reference [10].
- In the third step, we set the shortest path distance to every free internal vertex $a_j$ as $c_{a_j} = \min_{\mathcal{R}_j \in \mathcal{R}, (u, a_j) \in E_j^H} c_u + s_H(u, a_j)$. As in step one, the net-costs of such paths are precomputed in $H$, and the slacks can be computed in $O(1)$ time.
- Let $\alpha$ be the vertex $a \in A_H^F$ with minimum $c_a$. We return the path to $\alpha$ in $H'$. We can compute this path from the shortest path tree in $O(n/\sqrt{r})$ time.

Therefore, we obtain the following lemma:

Lemma 4.7. *Given the edges of $H$ stored in ordered bipartite groups along with a Monge-matrix range minimum data structure built on the slack matrix for each group, one can compute the shortest path (in terms of slack) between any free vertex of $B_H^F$ to any free vertex of $A_H^F$ in $O((n/\sqrt{r}) \log r \log n)$ time.*

### 4.2 Computing an $\mathcal{R}$-feasible Matching from a Planar Feasible Matching

Recollect that an $\mathcal{R}$-feasible matching was obtained at the end of the first step of our algorithm. In the previous section, we described how we can compute a planar-feasible matching from this $\mathcal{R}$-feasible matching. The second step of our algorithm will compute a planar optimal matching from this planar-feasible matching. At the end of each scale, however, we desire an $\mathcal{R}$-optimal matching. In this section, we describe how to convert any planar optimal matching to an $\mathcal{R}$-optimal matching in $O(n \log r)$ time. In fact, we will show that any planar-feasible matching can be converted into an $\mathcal{R}$-feasible matching.

To assist with the presentation, for certain vertices of $G$, we define their *representative* in $H$ as follows: For any vertex $v \in \mathcal{K}_j \cup (V_j \cap (A_F \cup B_F))$, if $v \in \mathcal{K}_j$, then the representative $v'$ in $H$ will be the same boundary vertex in $V_j^H$, and if $v$ is a free internal vertex of $A$ (respectively, $B$), then its representative $v'$ in $H$ will be $a_j$ (respectively, $b_j$).

In a planar-feasible matching, any boundary vertex or free internal vertex can have multiple dual weights, one corresponding to each of the partitions it belongs to. In addition, its representative in $H$ also has a dual weight. We introduce a synchronization procedure (called Sync) that will take a planar-feasible matching along with a partition $\mathcal{R}_j$ and update the dual weights $y_j(\cdot)$ so the new dual weights $y_j(\cdot)$ and the matching $M_j = M \cap E_j$ continue to be $\mathcal{R}$-feasible and for every boundary vertex and free internal vertex $v \in \mathcal{K}_j$, $y_j(v) = \tilde{y}(v')$ where $v'$ is the representative of $v$ in $H$. We can convert a planar-feasible matching into an $\mathcal{R}$-feasible matching by repeatedly invoking this procedure for every partition.

The Sync procedure is implemented as follows:

- Recollect that the graph $\mathcal{R}'_j$ is a graph identical to $\mathcal{R}_j$ with slacks as the edge costs. First, temporarily add a new source vertex $s$ to $\mathcal{R}'_j$. Let $B_{in}$ denote all vertices of $B \cap V_j$ that are matched inside $\mathcal{R}_j$ and let $A_{in}$ be the matches of vertices in $B_{in}$. For any vertex

[3] A recent result by Reference [8] improves this time by a factor of $\log \log n$.

$v \in \mathcal{K}_j \cup (V_j \cap (A_F \cup B_F))$), if $v \notin B_{in}$, we add a directed edge from $s$ to $v$. Otherwise, if $v \in B_{in}$, then let $u \in A_{in}$ be its match in $M_j$. We add a directed edge from $s$ to $u$. Consider any $v \in \mathcal{K}_j \cup (V_j \cap (A_F \cup B_F))$ and its representative $v' \in V_j^H$. For any such boundary or free internal vertex $v$, let $\kappa_v = |\tilde{y}(v')| - |y_j(v)|$, and let $\kappa = \max_{v \in \mathcal{K}_j \cup (V_j \cap (A_F \cup B_F))} \kappa_v$. If $v \notin B_{in}$, then set the weight of the newly added edge from $s$ to $v$ to $\kappa - \kappa_v$. Otherwise, if $v \in B_{in}$, then set the weight of the edge from $s$ to $u$ to $\kappa - \kappa_v$. This new graph has only non-negative edge costs.

- Execute Dijkstra's algorithm on this graph beginning from the source vertex $s$. Let $\ell_v$ be the length of the shortest path from $s$ to $v$ as computed by this execution of Dijkstra's algorithm. For each vertex $v \in V_j$, if $\ell_v \geq \kappa$, then do not change its dual weight. Otherwise, if $v \in B \cap V_j$, then set the new dual weight to be $y_j(v) \leftarrow y_j(v) + \kappa - \ell_v$ and if $v \in A \cap V_j$, then set its new dual weight to be $y_j(v) \leftarrow y_j(v) - \kappa + \ell_v$.

This completes the description of the Sync procedure. The Sync procedure executes Dijkstra's algorithm on $\mathcal{R}'_j$ with an additional vertex $s$ and updates the dual weights of all the $O(r)$ vertices. The total time taken for this is $O(r \log r)$. To prove the correctness of this procedure, we have to show the following:

(1) For any vertex $v \in \mathcal{K}_j \cup (V_j \cap (A_F \cup B_F))$ and its representative $v' \in V_j^H$, the dual weight after the execution of Sync procedure, $\tilde{y}(v')$ is equal to $y_j(v)$.
(2) The new dual weights $y_j(\cdot)$ along with the matching $M_j$ form an $\mathcal{R}$-feasible matching.

Lemma 4.8. *At the end of the Sync procedure, both (1) and (2) hold.*

Proof. Let us denote the dual weights before and after applying the Sync procedure as $y'_j(\cdot)$ and $y_j(\cdot)$, respectively. Consider any $v \in \mathcal{K}_j \cup (V_j \cap (A_F \cup B_F))$. Recall the definition of the representative of $v$ in $H$. Either (i) $v \notin B_{in}$ or (ii) $v \in B_{in}$.

In case (i), let $v'$ be the representative of $v$ in $H$. We will show that the shortest path from $s$ to $v$ in $\mathcal{R}'_j$ is the edge $(s, v)$. Given this, if $v \in A$, $\ell_v = \kappa - y'_j(v) + \tilde{y}(v')$ and $y_j(v) = y'_j(v) - \kappa + \ell_v = \tilde{y}(v')$. Similarly, if $v \in B$, $\ell_v = \kappa + y'_j(v) - \tilde{y}(v')$ and $y_j(v) = y'_j(v) + \kappa - \ell_v = \tilde{y}(v)$, as desired. Therefore, for case (i), it suffices to show that the shortest path from $s$ to $v$ is the edge $(s, v)$; we show this next. For the sake of contradiction, let the shortest path from $s$ to $v$ be strictly less than the cost of the edge $(s, v)$. Let $P$ be this shortest path from $s$ to $v$ and let $\hat{v}$ be the first vertex that appears on $P$ after $s$. By construction, $\hat{v} \in \mathcal{K}_j \cup (V_j \cap (A_F \cup B_F))$ and let $\hat{v}'$ be the representative of $\hat{v}$ in $H$. Let $P'$ be the sub-path of $P$ with the vertex $s$ removed, i.e., $P'$ is a path from $\hat{v}$ to $v$. By the optimal substructure property of shortest paths, $P'$ has to be the smallest slack path from $\hat{v}$ to $v$ in $\mathcal{R}'_j$. We know that the cost of $P$ in $\mathcal{R}'_j$ is $(\kappa - |\tilde{y}(\hat{v}')| + |y'_j(\hat{v})|) + \sum_{(a,b) \in P'} s(a, b)$. Since the cost of $P$ is smaller than the cost of the edge from $s$ to $v$, we have

$$\kappa - |\tilde{y}(v')| + |y'_j(v)| > \kappa - |\tilde{y}(\hat{v}')| + |y'_j(\hat{v})| + \sum_{(a,b) \in P'} s(a, b),$$

$$|\tilde{y}(\hat{v}')| - |\tilde{y}(\hat{v})| > |y'_j(v')| - |y'_j(v)| + \sum_{(a,b) \in P'} s(a, b) = \phi(P').$$

The last statement follows from the fact that $P'$ is the minimum slack path from $v'$ to $v$ and since the matching $M_j$ and dual weights $y'_j(\cdot)$ form an $\mathcal{R}$-feasible matching. The above inequality, $|\tilde{y}(\hat{v}')| - |\tilde{y}(\hat{v})| > \phi(P')$, contradicts the $H$-feasibility condition.

In case (ii), let $u \in A_{in}$ be the match of $v$ and let $v'$ be the representative of $v$ in $H$. Using a proof similar to case (i), one can show that the shortest path from $s$ to $v$ is the path $\langle s, u, v \rangle$. Since $(u, v)$ is an edge in the matching $M_j$, $(u, v)$ has a cost of 0 in $\mathcal{R}'_j$. So, the shortest path cost from

$s$ to $v$ is equal to the cost of the edge $(s, u)$, i.e., $\ell_u = \kappa - y'_j(v) + \tilde{y}(v')$ and the new dual weight $y_j(v) = y'_j(v) - \kappa + \ell_v = \tilde{y}(v')$. Thus, we have shown that at the end of Sync procedure, (1) holds.

To prove (2), we need to show that the matching $M_j$ along with the new dual weights $y_j(\cdot)$ are $\mathcal{R}$-feasible. We show this for the two cases (i) $(a, b) \in M_j$ and (ii) $(a, b) \in E_j \setminus M_j$ separately. For case (i), from $\mathcal{R}$-feasibility of $M_j$ and $y'_j(\cdot)$, $y'_j(a) + y'_j(b) = \mathrm{c}(a, b)$. We note that by construction of $\mathcal{R}'_j$ (that includes $s$) $b$ has exactly one incoming edge from $a$ to $b$. From $\mathcal{R}$-feasibility, the slack on the edge from $a$ to $b$ edge is 0. Therefore, $\ell_a = \ell_b$. So, the dual weights as updated by the Sync procedure satisfy

$$y_j(a) + y_j(b) = y'_j(a) - \kappa + \ell_a + y'_j(b) + \kappa - \ell_b = \mathrm{c}(a, b),$$

implying that $(a, b)$ satisfies condition (6).

For case (ii), the edge $(a, b)$ is not in the matching $M_j$, and, therefore, the edge is directed from $b$ to $a$. Consider if $a \in \mathcal{K}_j \cup ((V_j \setminus \mathcal{K}_j) \cap (A_F \cup B_F))$. The shortest path cost from $s$ to $a$ satisfies $\ell_a \leq \ell_b + s(a, b)$; otherwise, there would be a shorter path to $a$ through $b$. Since $M_j$ and $y'_j(\cdot)$ are $\mathcal{R}$-feasible, we have $y'_j(a) + y'_j(b) + s(a, b) = \mathrm{c}(a, b) + \delta_{ab}$. Therefore, the dual weights as updated by the Sync procedure satisfy

$$\begin{aligned} y_j(a) + y_j(b) &= y'_j(a) - \kappa + \ell_a + y'_j(b) + \kappa - \ell_b \\ &= y'_j(a) + y'_j(b) + \ell_a - \ell_b \\ &\leq y'_j(a) + y'_j(b) + s(a, b) \\ &= \mathrm{c}(a, b) + \delta_{ab}, \end{aligned}$$

implying that every non-matching edge $(a, b)$ satisfies the $\mathcal{R}$-feasibility condition (5).

Therefore, the matching $M_j$ and the new dual weights $y_j(\cdot)$ are $\mathcal{R}$-feasible completing the proof of (2). □

Given the correctness of the Sync procedure, we can convert a planar-feasible matching into an $\mathcal{R}$-feasible matching by simply applying the Sync procedure to all the partitions. This will guarantee that the dual weight of any vertex $v \in A \cup B$ is the same across all partitions it participates in and also equal to the dual weight of its representative in $H$ if one exists. Let $y(v)$ be this dual weight. Every edge $(u, v)$ of the graph $G$ belongs to some partition in the $r$-division $\mathcal{R}(G)$ and so it must be $\mathcal{R}$-feasible. Therefore, the matching $M$ along with the dual weights $y(\cdot)$ form an $\mathcal{R}$-feasible matching. Also note that every free vertex $v \in A_F$ will have a dual weight $y(v) = 0$ and every free vertex $v \in B_F$ will have a dual weight $y(v) = y_{max}$. This follows from (1), and the fact that the input to Sync is a planar-feasible matching.

Lemma 4.9. *Given a planar-feasible matching $M$, we can convert it into an $\mathcal{R}$-feasible matching $M$ with a set of dual weights $y(\cdot)$ in $O(n \log r)$ time. Furthermore, for every $a \in A_F$, $y(a) = 0$ and for every $b \in B_F$, $y(b) = y_{max}$.*

### 4.3 Second Step of the Algorithm

In this section, given a planar-feasible matching with $O(n/\sqrt{r})$ free vertices, we show how to compute a planar optimal matching in $O((n^2/r + n\sqrt{r}) \log r \log n)$ time. Our algorithm will then convert the planar optimal matching into an $\mathcal{R}$-optimal matching in time $O(n \log r)$, by Lemma 4.9. The second step of our algorithm will consist of multiple phases. In each phase, the size of the matching will increase by one while maintaining planar feasibility. Recall that $H'$ is the graph identical to $H$ with slacks as the edge cost. We will execute the following during each phase:

(i) Compute the shortest path distance $\mathrm{c}_v$, to every $v \in V_H$ from some free vertex of $B_H^F$ using Lemma 4.7. Let $\alpha = \arg\min_{v \in A_H^F} \mathrm{c}_v$ and let $P$ be this shortest path to $\alpha$.

(ii) Let $\mathcal{T}$ be the set of vertices $v \in V_H$ such that $c_v \leq c_\alpha$. For every vertex $v \in \mathcal{T} \cap B_H$, we increase the dual weight $\tilde{y}(v) \leftarrow \tilde{y}(v) + c_\alpha - c_v$, and for every vertex $v \in \mathcal{T} \cap A_H$, we decrease the dual weight $\tilde{y}(v) \leftarrow \tilde{y}(v) - c_\alpha + c_v$. Note that in either case, this results in an increase in the magnitude of $\tilde{y}(v)$.

(iii) Execute the AUGMENT procedure (described below). This procedure will augment the matching while maintaining planar feasibility.

(iv) (i)–(iii) will change $\tilde{y}(\cdot)$ and, therefore, also the slack on the edges of $H'$. Thus, the algorithm rebuilds the Monge matrix range minimum data structure for every partition in $\mathcal{R}(G)$. By using Lemma 4.5, this can be done in $O(\sqrt{r}\log^2 r)$ time per partition and $O((n/\sqrt{r})\log^2 r)$ time in total.

Step 2(i) and Step 2(ii) modify the dual weight of vertices in $H$. Lemma 5.1 shows that this modification does not violate $H$-feasibility. In Lemma 5.2, we show that, after execution of Step 2(ii) the edges of $P$ will have zero slack with respect to $\tilde{y}(\cdot)$. Next, we will describe the details of AUGMENT procedure.

*Augment Procedure:* After executing Step 2(ii), we have a path $P$ in $H$ and a set of $H$-feasible dual weights $\tilde{y}(\cdot)$ such that all edges in $P$ have zero slack. Now, we need to use $P$ to augment the matching. Let $b$ be the first vertex of $P$ and let $a$ be the last vertex of $P$. For any edge $(u, v)$ on the path $P$, if $(u, v)$ is an edge of $E_j^H$, then we refer to $\mathcal{R}_j$ as an *affected* partition. Let $\mathcal{A}$ be the set of all such affected partitions. We execute the following operations during the AUGMENT procedure:

(a) Call the SYNC procedure on every affected partition $\mathcal{R}_j \in \mathcal{A}$. This procedure will update the dual weights $y_j(\cdot)$.

(b) For any edge $(u, v) \in P$, suppose $(u, v)$ is an edge in $E_j^H$. Project $(u, v)$ to obtain the path $\overrightarrow{P}_{u,v,j}$. This can be done by executing a BFS on the admissible graph of the partition $\mathcal{R}_j$. Combine all the projections and obtain an augmenting path $\overrightarrow{P}$ of admissible edges in the residual graph $G_M$. Next, augment the matching $M$ along $\overrightarrow{P}$. Augmenting the matching will change the residual graph of every affected partition $\mathcal{R}_j \in \mathcal{A}$. Lemma 5.3 shows that $\overrightarrow{P}_{u,v,j}$ consists of admissible edges with respect to $M_j$, $y_j(\cdot)$ after the call to SYNC. Lemma 5.4 shows that $\overrightarrow{P}$ is a simple path, i.e., it has no self-intersections.

(c) For any vertex $v \in A \cap \overrightarrow{P}$, let $u$ be the vertex to which $v$ is matched after the augmentation. Set $y_j(v) \leftarrow y_j(v) - \delta_{uv}$ for all partitions $\mathcal{R}_j \in \mathcal{A}$ that contain $v$. If $v \in A_H \cap P$ and $v \neq a_{j'}$, then set $\tilde{y}(v) \leftarrow \tilde{y}(v) - \delta_{uv}$. Lemma 5.5 shows that after this operation, for every affected partition $\mathcal{R}_j$, the matching $M_j$ and $y_j(\cdot)$ is $\mathcal{R}$-feasible. Lemma 5.6 shows that after this operation, $\tilde{y}(\cdot)$ continues being $H$-feasible.

(d) Call the CONSTRUCT procedure on every affected partition $\mathcal{R}_j$ to rebuild the edges of $H$ for each affected partition. Due to the augmentation, there may be several new edges added to $H$ and several other edges whose costs have changed. We refer to the set $\mathcal{E}$ of these edges as the *affected edges* of $H$.

## 5 ANALYSIS OF THE ALGORITHM

### 5.1 Correctness

In this section, we show that the matching the algorithm maintains is planar-feasible at the end of each phase of the second step. We also show that the augmenting path computed in each phase is a simple path. Since there are only $O(n/\sqrt{r})$ unmatched vertices after the first step of the algorithm, it will conduct $O(n/\sqrt{r})$ phases, where each phase increases the size of the matching by one.

Therefore, after $O(n/\sqrt{r})$ phases, we will have a planar optimal matching. By Lemma 4.9, a planar optimal matching can generate an $\mathcal{R}$-optimal matching by use of the Sync procedure.

*Overview of the Correctness Proof:* To prove that the matching generated at the end of each phase is planar-feasible, we must show conditions (a)–(e) of planar feasibility are satisfied. Lemma 5.8 shows that conditions (a) and (b) of planar feasibility hold at the end of every phase. Condition (c) requires $\mathcal{R}$-feasibility of every partition at the end of each phase. We show, in Lemma 5.5, that condition (c) holds. To show that condition (d) holds, we have to establish that the dual weights $\tilde{y}(\cdot)$ are $H$-feasible at the end of each phase. In each phase, Step 2(i), Step 2(ii), and the Augment procedure modify the dual weights $\tilde{y}(\cdot)$ of $H$. Lemma 5.1 will show that the modifications by Step 2(i) and Step 2(ii) do not violate $H$-feasibility. In Lemma 5.6, we show that the Augment procedure does not violate the $H$-feasibility and condition (d) holds. Finally, we establish that condition (e) holds at the end of each phase in Lemma 5.7. Together, these lemmas prove that we maintain a planar-feasible matching at the end of each phase.

Also, Lemma 5.4 and Lemma 5.2 show that the augmenting path computed by the algorithm in each phase does not have any self-intersections.

Lemma 5.1. *Let us assume the dual weights $\tilde{y}(\cdot)$ are $H$-feasible prior to an execution of Step 2(ii). After executing Step 2(ii), $\tilde{y}(\cdot)$ remains $H$-feasible.*

Proof. We show that after the execution of Step 2(ii), for every directed edge $(u, v)$ in $E_j^H$,

$$|\tilde{y}(u)| - |\tilde{y}(v)| \leq \phi(\overrightarrow{P}_{u,v,j}). \tag{14}$$

Let $\tilde{y}_0(\cdot)$ be the dual weights prior to executing Step 2(ii). Initially, we have that,

$$|\tilde{y}_0(u)| - |\tilde{y}_0(v)| + s_H(u, v) = \phi(\overrightarrow{P}_{u,v,j}).$$

Additionally, Step 2(i) produces a distance assignment c such that for the directed edge $(u, v)$,

$$c_u + s_H(u, v) \geq c_v. \tag{15}$$

Recall that for any vertex $v \in V_H$, $c_v$ is the minimum distance in $H'$ to $v$ from some vertex of $B_H^F$. Let $\alpha$ be the vertex $v \in A_H^F$ with minimum $c_v$. Let $\mathcal{T}$ be the set of vertices $v$ for which $c_v \leq c_\alpha$. We consider four cases:

(1) $u \in \mathcal{T}, v \in \mathcal{T}$. We increase the magnitude of $\tilde{y}(u)$ by $c_\alpha - c_u$ and increase the magnitude of $\tilde{y}(v)$ by $c_\alpha - c_v$. Thus, we have,

$$\begin{aligned} |\tilde{y}(u)| - |\tilde{y}(v)| &\leq (|\tilde{y}_0(u)| + c_\alpha - c_u) - ((|\tilde{y}_0(v)| + c_\alpha - c_v) \\ &\leq (|\tilde{y}_0(u)| - |\tilde{y}_0(v)|) + (c_v - c_u). \end{aligned} \tag{16}$$

Combining inequality (15) with inequality (16) yields,

$$\begin{aligned} |\tilde{y}(u)| - |\tilde{y}(v)| &\leq |\tilde{y}_0(u)| - |\tilde{y}_0(v)| + s_H(u, v) \\ &= \phi(\overrightarrow{P}_{u,v,j}), \end{aligned}$$

and inequality (14) holds.

(2) $u \in \mathcal{T}$, $v \notin \mathcal{T}$. $|\tilde{y}(u)| = |\tilde{y}_0(u)| + c_\alpha - c_u$ and $|\tilde{y}(v)| = |\tilde{y}_0(v)|$. Since $v \notin \mathcal{T}$, $c_\alpha < c_v$, and, therefore, $c_\alpha - c_u < c_v - c_u$. From inequality (15), we have that $c_v - c_u \leq s_H(u, v)$. Therefore,

$$\begin{aligned} |\tilde{y}(u)| - |\tilde{y}(v)| &< |\tilde{y}_0(u)| - |\tilde{y}_0(v)| + s_H(u, v) \\ &= \phi(\overrightarrow{P}_{u,v,j}), \end{aligned}$$

and inequality (14) holds.

(3) $u \notin \mathcal{T}, v \in \mathcal{T}$. In this case, $|\tilde{y}(u)| = |\tilde{y}_0(u)|$ and $|\tilde{y}(v)| \geq |\tilde{y}_0(v)|$. Therefore,

$$\begin{aligned} |\tilde{y}(u)| - |\tilde{y}(v)| &\leq |\tilde{y}_0(u)| - |\tilde{y}_0(v)| + s_H(u,v) \\ &= \phi(\overrightarrow{P}_{u,v,j}), \end{aligned}$$

and inequality (14) holds.

(4) $u \notin \mathcal{T}$, $v \notin \mathcal{T}$. In this case, $\tilde{y}(u) = \tilde{y}_0(u)$ and $\tilde{y}(v) = \tilde{y}_0(v)$. Therefore, inequality (14) holds. □

Lemma 5.2. *Let $P$ be the path found in $H$ during Step 2(i). Then after Step 2(ii), all edges in $P$ have zero slack.*

Proof. We show that for any edge $(u, v) \in P$,

$$|\tilde{y}(u)| - |\tilde{y}(v)| = \phi(\overrightarrow{P}_{u,v,j}).$$

Since $(u, v)$ is part of the shortest path, it must be true that $c_u + s_H(u, v) = c_v$. We increase the magnitude of $\tilde{y}(u)$ by $c_\alpha - c_u$ and we increase the magnitude of $\tilde{y}(v)$ by $c_\alpha - c_v$. Thus, $|\tilde{y}(u)| - |\tilde{y}(v)|$ increased by $s_H(u, v)$. Since the matching $M_j$ is $\mathcal{R}$-feasible, by Lemma 4.2, prior to Step 2(ii), we have $|\tilde{y}(u)| - |\tilde{y}(v)| + s_H(u, v) = \phi(\overrightarrow{P}_{u,v,j})$. Therefore, after Step 2(ii), we have $|\tilde{y}(u)| - |\tilde{y}(v)| = \phi(\overrightarrow{P}_{u,v,j})$. □

The following lemma shows that the edges of an augmenting path are admissible prior to augmentation:

Lemma 5.3. *After the execution of the Sync procedure on all partitions in $\mathcal{A}$, for every partition $\mathcal{R}_j \in \mathcal{A}$, any edge $(u, v) \in \overrightarrow{P} \cap E_j$ is admissible with respect to $M_j, y_j(\cdot)$. Here $\overrightarrow{P}$ is the augmenting path computed in part (b) of the Augment procedure.*

Proof. Let $P$ be the path found in $H$ in Step 2(i). By Lemma 5.2, all edges in $P$ have zero slack with respect to the dual weights in $H$. In part (a), we call the Sync procedure on every $\mathcal{R}_j \in \mathcal{A}$, which, by Lemma 4.8, creates a matching $M_j$ and dual weights $y_j(\cdot)$ that are both feasible and synchronized with the weights $\tilde{y}(\cdot)$ in $H$. Since the dual weights of $H$ did not change during the Sync procedure, for any edge $(u, v) \in E_j^H$, $(u, v)$ still has zero slack, i.e., $|\tilde{y}(u)| - |\tilde{y}(v)| = |y_j(u)| - |y_j(v)| = \phi(\overrightarrow{P}_{u,v,j})$. So, by Lemma 4.2, the sum of slacks on all edges of $\overrightarrow{P}_{u,v,j}$, for every $(u, v) \in P$, must be zero. Since slacks are non-negative, every edge in $\overrightarrow{P}_{u,v,j}$ must have zero slack and, therefore, be admissible. □

By our construction of $H$, any path through $H$ maps to a path in $G_M$. The augmenting path $P$ found in $H$ is a simple path, but it is conceivable that the corresponding projection $\overrightarrow{P}$ is not. However, the following lemma shows that $\overrightarrow{P}$ is in fact a simple path:

Lemma 5.4. *Let $P$ be a path found in $H$ during Step 2(i) of the algorithm and let $\overrightarrow{P}$ be the projection of $P$ onto $G_M$ (computed in part (b) of the Augment procedure). Then, $\overrightarrow{P}$ is a simple path, i.e., $\overrightarrow{P}$ has no self-intersections.*

Proof. We show that our algorithm never introduces a cycle of admissible edges in $G_M$. Since the path $\overrightarrow{P}$ chosen by our algorithm consists only of admissible edges, $\overrightarrow{P}$ is a simple path.

The first step of our algorithm executes $O(\sqrt{r})$ iterations of the Gabow-Tarjan algorithm. The Gabow-Tarjan algorithm never creates any cycle of admissible edges [7, Lemma 2.3]. Therefore, after the first step of our algorithm, the algorithm has a 1-feasible matching $M$ and dual weights $y(\cdot)$ with no cycle of admissible edges. The admissibility conditions for a 1-feasible matching are different from the admissibility conditions for an $\mathcal{R}$-feasible matching. However, given any 1-feasible matching, an edge that is not admissible is also not admissible as defined for $\mathcal{R}$-feasibility. Therefore, there is no cycle of admissible edges with respect to the $\mathcal{R}$-feasible matching $M$ and $y(\cdot)$. The algorithm converts this $\mathcal{R}$-feasible matching into a planar-feasible matching and constructs the compressed residual graph $H$. At the start, $H$ will not have any cycle consisting only of zero slack edges, simply because the projection of any such cycle would form a cycle of admissible edges in the residual graph $G_M$.

Note that during any phase of the algorithm, if the path $\overrightarrow{P}$ computed contains a cycle $\overrightarrow{C}$ of admissible edges (in $G_M$), then either (1) $\overrightarrow{C}$ contains edges of only one partition or (2) $\overrightarrow{C}$ contains edges from more than one partition. We present our proof for case (2). The proof for case (1) uses similar arguments but is significantly simpler. In case (2), the boundary vertices on $\overrightarrow{C}$ partition the cycle into paths where each path is between two successive boundary nodes. We can construct a cycle $C$ in $H$ by replacing this path between two successive boundary nodes in the cycle $\overrightarrow{C}$ with a directed edge of $H$ between them. Before augmentation along $\overrightarrow{P}$, due to execution of the Sync procedure, the dual weight $y(v)$ of any boundary vertex and $\tilde{y}(v')$ of its representative $v'$ in $H$ are the same. Since all the edges of $\overrightarrow{C}$ are admissible, from Lemma 4.2, the slack on the edges of $C$ is zero. Therefore, to show that case (2) does not occur, it suffices to show that during the course of the algorithm, $H$ does not have any cycle consisting of only zero slack edges.

As already claimed, after the first step of the algorithm, $H$ does not contain any cycle of zero slack edges. The dual weights of $H$ change in either step 2(ii) or in the Augment procedure. We will argue that neither of these steps will create a cycle consisting only of zero slack edges.

Using a proof by contradiction, we show that step 2(ii) cannot create a zero slack cycle in $H$. For the sake of contradiction, consider some phase of the algorithm such that a cycle $C'$ has a non-zero slack at the start of the phase but has only zero slack edges at the end of the phase. In step 2(i), we compute shortest distance from vertices of $B_F^H$ to every vertex in $H$, including all vertices on $C'$. If all the vertices of $C'$ have the same shortest path distance, then the change in the magnitude of the dual weights of all these vertices will be the same and, therefore, the slack on every edge remains the same, implying that not all the edges of $C'$ are zero slack at the end of Step 2(ii). Therefore, we can assume that at least one vertex on the cycle $C'$ has a distance that is different from other vertices on $C'$. Note that when the shortest path distances to vertices of $C'$ are different, then there will be at least one edge of the cycle $C'$ from a vertex $u$ to a vertex $v$ such that the shortest distance to $u$ as computed by step 2(i) is greater than the shortest distance to $v$. In this case, the magnitude of the dual weight of $v$ changes by a larger value than the magnitude of the dual weight of $u$ and, therefore, the slack on the edge $(u, v)$ strictly increases. This implies that $C'$ will have at least one edge with a non-zero slack at the end of Step 2(ii). So, step 2(ii) will not introduce a cycle that consists only of zero slack edges in $H$.

The Augment procedure may introduce new edges and change the costs of some edges in $H$. We divide this set of affected edges $\mathcal{E}$ into two groups. Edges of the first group are affected edges that are directed from some vertex $u \in A_H \cap P$ to $v$ ($P$ is the path found by Step 2(i)). The remaining edges in the affected set are of the second type. As we show in Lemma 5.6, all edges of the first type have a slack of at least zero, and edges of the second type will have a slack of at least 1. So, these affected edges of the second group do not create a cycle of zero slack edges in $H$. For any

affected edge $(u, v)$ in the first group, we show that every incoming edge to $u$ has a slack of at least 1. This is because, after augmenting the matching, we increase the magnitude of the dual weight of $u$ (equivalently, since $u \in A_H$, reducing the dual weight of $u$), we guarantee that all edges of the residual graph that enter $u$ have a slack of at least 1. So, although an affected edge of type 2 $(u, v)$ can have zero slack in $H$, since all incoming edges to $u$ in $H$ have a slack of at least 1, it cannot participate in a cycle containing only zero slack edges. Therefore, at the end of this phase, there is no cycle in $H$ containing only zero slack edges, leading to a contradiction. Hence, case (2) will never occur. Since both (1) and (2) cannot occur, the path $\overrightarrow{P}$ will not contain any cycle of admissible edges, i.e., $\overrightarrow{P}$ is a simple path. □

We note that the matchings $M_j$ and the dual weights $y_j(\cdot)$ only change during Step 2(iii) of the algorithm; that is, during Augment. The following lemma argues that the Augment procedure does not violate $\mathcal{R}$-feasibility in any $\mathcal{R}_j$ for which $M_j$ or $y_j(\cdot)$ was modified. Therefore, planar feasibility condition (c) is satisfied.

Lemma 5.5. *After the Augment procedure, for any affected partition $\mathcal{R}_j$, the matching $M_j$ and the dual weights $y_j(\cdot)$ form an $\mathcal{R}$-feasible matching.*

Proof. Let $\overrightarrow{P}$ be the augmenting path that the algorithm augments along. By Lemma 5.3, the Sync procedure produces an $\mathcal{R}$-feasible $M_j$, $y_j(\cdot)$ for every affected partition where the edges of $\overrightarrow{P}$ are admissible. We will show that the edges of $\overrightarrow{P}$ are feasible after augmentation.

After augmentation, for any $v \in A \cap \overrightarrow{P}$, let $u$ be the vertex to which $v$ is matched. Then, we reduce $y_j(v)$ by $\delta_{uv}$ for each $\mathcal{R}_j \in \mathcal{A}$. First consider any edge $(u, v) \notin M_j$. Prior to augmentation, $(u, v)$ was an admissible edge in the matching, so $y_j(u) + y_j(v) = c(u, v)$. The reduction of the dual weight of $v$ only decreases the sum $y_j(u) + y_j(v)$, meaning the feasibility condition in (5) is satisfied. However, we must still show that condition (6) is satisfied for all edges $(u, v) \in M_j$. Consider an edge $(u, v) \in M_j$. We know that $(u, v)$ was admissible prior to augmentation from Lemma 5.3. Let $y'_j(\cdot)$ be the dual weights prior to augmentation. Then $y'_j(u) + y'_j(v) = c(u, v) + \delta_{uv}$, since $(u, v)$ was not in the matching prior to augmentation. $y_j(u) = y'_j(u)$ and $y_j(v) = y'_j(v) - \delta_{uv}$. Therefore, $y_j(u) + y_j(v) = c(u, v)$ and $M_j, y_j(\cdot)$ form an $\mathcal{R}$-feasible matching. □

From Lemma 5.1, we have that $\tilde{y}(\cdot)$ is $H$-feasible after Step 2(ii). The Augment procedure, however, modifies the dual weights $\tilde{y}(\cdot)$, may add new edges to $H$, and may edit the cost of some of the old edges. Let $\mathcal{E}$ be the set of *affected* edges that are either newly added or whose cost has changed. The following lemma establishes that, despite the changes to $H$, the dual weights $\tilde{y}(\cdot)$ remain $H$-feasible after Augment, meaning requirement (d) of planar-feasibility is satisfied. Furthermore, every affected edge of $\mathcal{E}$ whose tail is not in $A_H$ has a slack of at least 1.

Lemma 5.6. *After the Augment procedure, the dual assignment $\tilde{y}(\cdot)$ is $H$-feasible. Let $\mathcal{E}$ be the edges that were either newly added to $H$ or whose cost changed after Augment. We claim that every edge $(u, v) \in \mathcal{E}$ directed from $u$ to $v$ in $H$ has $s_H(u, v) \geq 0$ and if $u \notin A_H$, then $s_H(u, v) \geq 1$.*

Proof. We address $H$-feasibility separately for $H$-edges of each piece $\mathcal{R}_j$. There are two cases: (i) $\mathcal{R}_j \notin \mathcal{A}$ and (ii) $\mathcal{R}_j \in \mathcal{A}$. First consider case (i). Then no edge $(u, v) \in E_j^H$ has experienced a change in net-cost, since the augmenting path did not pass through $\mathcal{R}_j$. However, it is possible that the dual weight $\tilde{y}(u)$ or $\tilde{y}(v)$ changed. Specifically, step (c) of Augment increases the magnitude of $\tilde{y}(x)$ for any vertex of $x \in A_H$ along the augmenting path. Consider if $u \in A_H$. Then the first edge of $\overrightarrow{P}_{u,v,j}$ must be a matching edge that was along the augmenting path, and $\mathcal{R}_j$ would be an affected piece, contradicting the assumption of case (i). Therefore, we can assume that $\tilde{y}(u)$

remains constant. It is possible that the magnitude of $\tilde{y}(v)$ increases. However, this will only increase $s_H(u,v)$. Therefore, we can conclude that $H$-feasibility is maintained for case (i).

Next, we address case (ii). Since $\mathcal{R}_j \in \mathcal{A}$, the Sync procedure was invoked on $\mathcal{R}_j$. By Lemma 4.8, this means for any vertex $x \in V_j$ and its representative vertex $x'$, $y_j(x) = \tilde{y}(x')$. Step (c) of Augment modifies the dual weights $y_j(x)$ and $\tilde{y}(x')$ by the same amounts, except if $x' = a_j$. However, if $x' = a_j$, then $x$ was the last vertex of the augmenting path; $x$ is now matched, meaning $x$ is no longer represented by $a_j$. Therefore, for all vertices $x \in V_j$ with representative vertex $x'$, $y_j(x) = \tilde{y}(x')$ after Augment.

Consider any edge $(u,v) \in E_j^H$. By Lemma 5.5, the matching after Augment is $\mathcal{R}$-feasible, and we can invoke Equation (12) of Lemma 4.2. Specifically, since $\tilde{y}(u) = y_j(u)$ and $\tilde{y}(v) = y_j(v)$, we get that

$$s_H(u,v) = \phi(\overrightarrow{P}_{u,v,j}) - |\tilde{y}(u)| + |\tilde{y}(v)| = \sum_{(a,b)\in\overrightarrow{P}_{u,v,j}} s(a,b).$$

Since $y_j(\cdot), M_j$ is $\mathcal{R}$-feasible, each edge of $E_j$ has non-negative slack. Therefore, $s_H(u,v)$ is also non-negative, and $H$-feasibility holds for all edges of $E_j^H$. This completes case (ii).

We next argue the properties on the edges of $\mathcal{E}$ hold. Trivially, since $H$-feasibility holds, $s_H(u,v) \geq 0$ for any $(u,v) \in \mathcal{E}$. However, it remains to show that $s_H(u,v) \geq 1$ for any $(u,v) \in \mathcal{E}$ such that $u \notin A_H$. Let $\overrightarrow{P}_{u,v,j}$ (respectively, $\overrightarrow{P}'_{u,v,j}$) be the shortest path from $u$ to $v$ in $\mathcal{R}_j$ prior to (respectively, after) augmentation. There are two cases: (1) $\overrightarrow{P}'_{u,v,j}$ does not share an edge with the augmenting path and (2) $\overrightarrow{P}'_{u,v,j}$ does share an edge with the augmenting path.

Consider case (1). Since $\overrightarrow{P}'_{u,v,j}$ does not share an edge with the augmenting path, it also existed prior to augmentation. Therefore, $\phi(\overrightarrow{P}_{u,v,j}) \leq \phi(\overrightarrow{P}'_{u,v,j})$. Since $(u,v) \in \mathcal{E}$, by definition, $\phi(\overrightarrow{P}_{u,v,j}) \neq \phi(\overrightarrow{P}'_{u,v,j})$. Therefore, $\phi(\overrightarrow{P}_{u,v,j}) < \phi(\overrightarrow{P}'_{u,v,j})$. The dual weights $\tilde{y}(u)$ and $\tilde{y}(v)$ did not change, because step (c) of Augment only changes dual weights along the augmenting path, and, by our assumption for case (1), $\overrightarrow{P}'_{u,v,j}$ has no vertices on the augmenting path. Therefore, $\phi(\overrightarrow{P}_{u,v,j}) - |\tilde{y}(u)| + |\tilde{y}(v)| < \phi(\overrightarrow{P}'_{u,v,j}) - |\tilde{y}(u)| + |\tilde{y}(v)|$ and $s_H(u,v)$ only increased as a result of the augmentation. Since $H$-feasibility held prior to augmentation, $s_H(u,v)$ must be at least 1 after augmentation. This completes case (1).

Next, consider case (2). $\overrightarrow{P}'_{u,v,j}$ must intersect the augmenting path at some edge, implying there is some vertex $x \in A$ on $\overrightarrow{P}'_{u,v,j}$ that was also on the augmenting path. Since we assume $u \notin A$, $u \neq x$, and there must be at least one edge $(x',x) \in \overrightarrow{P}'_{u,v,j}$ from some $x' \in B$ to $x$. $s(x',x) \geq 0$ prior to step (c) of Augment. Step (c) of Augment decreases $y(x)$ by at least 1, because $x$ was a vertex of $A$ on the augmenting path. Since $x' \in B$, $y(x')$ does not change from step (c) of Augment. Thus, the slack on $(x',x)$ (a non-matching edge) strictly increases, and $s(x',x) \geq 1$. Since all other edges of $\overrightarrow{P}'_{u,v,j}$ have non-negative slack, we conclude that $s_H(u,v) \geq 1$, completing case (2). □

The following lemma shows that requirement (e) for planar feasibility is satisfied:

Lemma 5.7. *Assuming we begin the phase with a planar-feasible matching, at the end of that phase, for each vertex $v \in V_H$ and for every partition $\mathcal{R}_j$ such that $v \in \mathcal{K}_j$, $|\tilde{y}(v)| \geq |y_j(v)|$ for all $j$. Furthermore, if $v = a_j$ (respectively, $v = b_j$), then for every vertex $v' \in (V_j \setminus \mathcal{K}_j) \cap A_F$ (respectively, $v' \in (V_j \setminus \mathcal{K}_j) \cap B_F$)), $y_j(v') = 0$ (respectively, $y_j(v') \leq |\tilde{y}(b_j)|$).*

Proof. Step 2(ii) only increases the magnitude of dual weights or leaves them unchanged. This follows from the fact that for a vertex $v \in B_H$, $\tilde{y}(v)$ increases by $c_\alpha - c_v$ only if $c_\alpha - c_v \geq 0$.

Similarly, for a vertex $v \in A_H$, $\tilde{y}(v)$ decreases by $c_\alpha - c_v$ only if $c_\alpha - c_v \geq 0$. Since we assume that the matching at the beginning of the phase is planar-feasible, any $\tilde{y}(v)$ for $v \in B_H$ is non-negative and any $\tilde{y}(v)$ for $v \in A_H$ is non-positive. Therefore, the magnitudes of $\tilde{y}(\cdot)$ only increase during Step 2(ii), and the claims hold after Step 2(ii).

Step (a) of the AUGMENT procedure calls SYNC on the affected partitions. The properties of SYNC ensure that the claims are satisfied after step (a). Finally, in step (c) of AUGMENT, for some vertices $v \in A$, we lower the value of $y_j(v)$, increasing its magnitude. However, if $v \in \mathcal{K}_j$, then the magnitude of $\tilde{y}(v)$ increases by the same amount as $y_j(v)$. There are no free vertices $v \in A_F$ along the path after augmentation. Therefore, the claims hold true at the end of a phase. □

The following lemma guarantees that planar feasibility conditions (a) and (b) are satisfied:

LEMMA 5.8. *Assuming we are given a planar-feasible matching at the beginning of a phase, then at the end of that phase, every vertex $v \in A_H$ has a non-positive dual weight $\tilde{y}(v)$ and for each free (internal or boundary) vertex $v$ of $A_H$, $\tilde{y}(v) = 0$. Every vertex $v \in B_H$ has a non-negative dual weight $\tilde{y}(v)$ and for each free (internal or boundary) vertex $v$ of $B_H$, $\tilde{y}(v) = y_{max}$, where $y_{max} = \max_{v \in A_H \cup B_H} \tilde{y}(v)$.*

PROOF. As discussed in the proof of Lemma 5.7, the operations we apply do not decrease the magnitude of $\tilde{y}(\cdot)$. That is, for every $v \in B_H$, $\tilde{y}(v)$ only increases, and for every $v \in A_H$, $\tilde{y}(v)$ only decreases. It remains to prove the properties of $\tilde{y}(\cdot)$ for free vertices. For any vertex $v \in A_H^F$, Step 2(ii) does not change $\tilde{y}(v)$. This follows from the fact that if $v \in A_H^F$ and we have $c_v > c_\alpha$, then $\tilde{y}(v)$ is unchanged. Otherwise the magnitude of $\tilde{y}(v)$ increases by $c_\alpha - c_v$, which is 0, since $\alpha$ is a vertex in $A_H^F$ with minimum distance. For any vertex $v \in B_H^F$, $c_v = 0$. All distance values c are non-negative, so $c_\alpha - c_v$ is maximum over all vertices of $V_H$. The dual weights of all free vertices of $B_H$ increase by the same amount and continue being the largest among all $\tilde{y}(\cdot)$. Therefore, the claim holds true after Step 2(ii). The AUGMENT procedure reduces the dual weights of vertices in $P \cap A_H$ after augmenting along $P$. However, $P$ has no free vertices after augmentation. Therefore, at the end of a phase, for every $v \in A_F$, $\tilde{y}(v)$ remains 0 and for every $v \in B_F$, $\tilde{y}(v)$ remains maximum over all $\tilde{y}(\cdot)$. □

Combining lemmas (5.5), (5.6), (5.7), and (5.8) yields the following lemma:

LEMMA 5.9. *The matching at the end of each phase of the algorithm is a planar-feasible matching.*

Lemma 5.9 guarantees that at the end of each scale, we have a planar-feasible matching. We can convert a planar-feasible matching to an $\mathcal{R}$-feasible matching in time $O(n \log r)$ by calling the SYNC procedure on every partition.

### 5.2 Efficiency

Step 1 of our algorithm for each scale requires time $O(n\sqrt{r})$, since it executes $O(\sqrt{r})$ iterations of the Gabow-Tarjan algorithm, with each iteration taking $O(n)$. This results in $O(n/\sqrt{r})$ free vertices remaining.

In Step 2, our algorithm increases the size of the matching by 1 each iteration and, therefore, executes $O(n/\sqrt{r})$ iterations. Step 2(i) takes $O((n/\sqrt{r}) \log r \log n)$ time. Step 2(ii) takes $O(n/\sqrt{r})$ time, since $\tilde{y}(v)$ could be changed for each $v \in V_H$. In Step 2(iv), we recompute all of the interval-min search data structures, taking $O((n/\sqrt{r}) \log r \log n)$ time. Therefore, over $O(n/\sqrt{r})$ iterations, the total time for Step 2(i), Step 2(ii), Step 2(iv) is $O((n^2/r) \log r \log n)$. It remains to bound the time taken by Step 2(iii), i.e., the AUGMENT procedure.

Let $P_i$ be the path in $H$ computed in Step 2(i) during the $i$th phase (i.e., the phase of the second step of the algorithm that produces the $i$th augmenting path). During AUGMENT, SYNC is called

on all affected partitions. This takes $O(r \log r)$ time per partition. Next, $P$ is projected to yield an augmenting path $\overrightarrow{P}$ in $G_M$. This requires a BFS search for each projected edge of $P$, each search taking $O(r)$ time. Then, the AUGMENT procedure augments the edges of $\overrightarrow{P}$ and adjusts dual weights along $\overrightarrow{P}$. This takes time proportional to $|\overrightarrow{P}|$, which is at most $O(r)$ for each affected partition. Finally, we call CONSTRUCT on all the affected partitions, taking time $O(r \log r)$ per partition. Thus, the total time taken by the AUGMENT procedure is $O(\Delta r \log r)$ where $\Delta = \sum_{i=1}^{O(n/\sqrt{r})} |P_i|$ is the total length of all augmenting paths found in $H$ by Step 2(i) during the entire second step. Lemma 5.11 bounds the total length of all paths in $H$, i.e., $\Delta = O((n/\sqrt{r}) \log n)$. Therefore, over all the $O(n/\sqrt{r})$ phases, the total time taken by the AUGMENT procedure is $O(n\sqrt{r} \log r \log n)$.

Setting $r = n^{2/3}$ yields a total complexity of $O(n^{4/3} \log^2 n)$ for each scale. Our algorithm executes $O(\log nC)$ scales, where $C$ is the largest edge cost of the original graph. Therefore, the total complexity is $O(n^{4/3} \log^2 n \log nC)$.

The following lemma shows that we compute the minimum net-cost augmenting path in $G$, which is useful for bounding $\Delta$:

LEMMA 5.10. *The augmenting path computed in each phase is a minimum net-cost augmenting path.*

PROOF. Let $\overrightarrow{P}$ be the augmenting path found in $G_M$. By Lemma 5.3, any edge $(u, v) \in \overrightarrow{P} \cap E_j$ is admissible with respect to the matching $M_j$ and the dual weights $y_j(\cdot)$. Since the SYNC procedure has already been executed for partitions in $\mathcal{A}$, we can invoke the SYNC procedure on the remaining partitions that are not in $\mathcal{A}$ to obtain an $\mathcal{R}$-feasible matching $M$, $y(\cdot)$ on $G$ by Lemma 4.9. The dual weights $y_j(\cdot)$ for all $\mathcal{R}_j \in \mathcal{A}$ are unchanged and, therefore, the edges of $\overrightarrow{P}$ would remain admissible with respect to the matching $M$, $y(\cdot)$. This means the total slack along $\overrightarrow{P}$ is 0 with respect to $M$. Since the matching $M$ is feasible, all edges must have non-negative slack, and $\overrightarrow{P}$ is a minimum slack directed path in $G'_M$. By Lemma 4.3, we have for any path from $u$ to $v$ in $G_M$,

$$\phi(\overrightarrow{P}_{u,v}) - |y(u)| + |y(v)| = \sum_{(a,b) \in \overrightarrow{P}_{u,v}} s(a, b).$$

We maintain that for a vertex $v \in B_H^F$, $\tilde{y}(v) = y_{max}$ and for a vertex $v \in A_H^F$, $\tilde{y}(v) = 0$. Therefore, by the properties of the SYNC procedure, for any vertex $v \in B_F$, $y(v) = y_{max}$, and for any vertex $v \in A_F$, $y(v) = 0$. Since all free vertices of $B$ (respectively, $A$) have the same dual weight, a zero slack augmenting path with respect to $M$ and $y(\cdot)$ is also an augmenting path of minimum net-cost. □

The following lemma bounds the total length of all the augmenting paths computed by our algorithm. Gabow and Tarjan assign an error of 1 on every non-matching edge and show that the total error on all the augmenting paths generated by their algorithm is $O(n \log n)$. We use a similar proof strategy and show that the total error of all augmenting paths generated by our algorithm is also $O(n \log n)$. However, in our case, we assigned a cost of $\lceil \sqrt{r} \rceil$ for every edge incident on a boundary vertex. So, the number of such edges and, therefore, the number of boundary vertices on the augmenting paths, can be no more than $O(\frac{n}{\sqrt{r}} \log n)$, leading to a bound on their lengths in $H$.

LEMMA 5.11. *Let $P_i$ be the $i$th path in $H$ computed by the second step of our algorithm, and let $M_i$ be the matching just before computing $P_i$. Let $t$ be the number of phases of the second step of our algorithm. Recall $t = O(n/\sqrt{r})$. Then, $\Delta = \sum_{i=1}^{t} |P_i| = O((n/\sqrt{r}) \log n)$.*

Proof. We first show that

$$\sum_{i=1}^{t} \phi(P_i) = O(n \log n). \tag{17}$$

Consider $M_i \oplus M_{\text{OPT}}$ where $M_{\text{OPT}}$ is some minimum cost perfect matching. $M_i \oplus M_{\text{OPT}}$ forms alternating paths and cycles. Let the set $S_i$ consist of all the augmenting paths in $M_i \oplus M_{\text{OPT}}$. Then $|S_i| = t - i + 1$. We will now show that

$$\sum_{P \in S_i} \phi(P) \leq (2k + 3)n. \tag{18}$$

By definition of net cost, given that all edge costs are non-negative, we have:

$$\sum_{P \in S_i} \phi(P) \leq \sum_{P \in S_i} \sum_{(u,v) \in M_{\text{OPT}} \cap P} (\mathrm{c}(u, v) + \delta_{uv}).$$

Since all edge costs are non-negative, the edges in $M_{\text{OPT}}$ that are also in some $P \in S_i$ must have a total cost at most $\mathrm{c}(M_{\text{OPT}})$. Thus,

$$\sum_{P \in S_i} \phi(P) \leq \left( \sum_{P \in S_i} \sum_{(u,v) \in M_{\text{OPT}} \cap P} \delta_{uv} \right) + (k + 2)n. \tag{19}$$

By Lemma 3.1, we have

$$\sum_{(u,v) \in M_{\text{OPT}}} \delta_{uv} \leq (k + 1)n,$$

$$\sum_{P \in S_i} \sum_{(u,v) \in M_{\text{OPT}} \cap P} \delta_{uv} \leq (k + 1)n. \tag{20}$$

Combining inequalities (19) and (20) yields

$$\sum_{P \in S_i} \phi(P) \leq (2k + 3)n.$$

Since $|S_i| = t - i + 1$, we can say that the average net-cost of the paths in $S_i$ is at most $(2k + 3)n/(t - i + 1)$. Our algorithm finds the minimum net-cost path available, by Lemma 5.10, which is at most the average net cost path in $S$. Therefore, we have

$$\sum_{i=1}^{t} \phi(P_i) \leq \sum_{i=1}^{t} ((2k + 3)n/(t - i + 1)).$$

The denominator of the right side forms a harmonic series. Therefore,

$$\sum_{i=1}^{t} \phi(P_i) = O(n \log n).$$

Let $M_{\text{START}}$ be the matching computed after the first step of the algorithm and let $M$ be the final perfect matching computed by the second step of our algorithm. By definition of the net-cost, from Lemma 3.1 and the fact that $\mathrm{c}(M) \geq 0$, we have that

$$\begin{aligned} \sum_{i=1}^{t} \phi(P_i) &= \mathrm{c}(M) - \mathrm{c}(M_{\text{START}}) + \sum_{P_i} \sum_{(u,v) \in P_i \setminus M_i} \delta_{uv} \\ &\geq -\mathrm{c}(M_{\text{START}}) + \sum_{i=1}^{t} \sum_{(u,v) \in P_i \setminus M_i} \delta_{uv}. \end{aligned}$$

$M_{\text{START}}$ is guaranteed to have a cost less than $c(M_{\text{OPT}}) + n = O(n)$ by the properties of the Gabow-Tarjan algorithm. Therefore,

$$\sum_{i=1}^{t} \sum_{(u,v) \in P_i \setminus M_i} \delta_{uv} = O(n \log n).$$

For any $(u, v) \notin M_i$ where $u \in \mathcal{K}_j$ or $v \in \mathcal{K}_j$, we have that $\delta_{uv} = \lceil \sqrt{r} \rceil$. This means that among all augmenting paths computed in $G$, there can only be $O((n/\sqrt{r}) \log n)$ boundary vertices used. The length of any path in $H$ is at most the number of boundary vertices along the path plus 1, and there are $t = O(n/\sqrt{r})$ augmenting paths computed during the second step of the algorithm. Therefore,

$$\sum_{i=1}^{t} |P_i| = O((n/\sqrt{r}) \log n). \qquad \square$$

## REFERENCES


[1] Therese Biedl. 2001. Linear reductions of maximum matching. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA'01)*. SIAM, 825–826.

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

[3] Michael B. Cohen, Aleksander Madry, Piotr Sankowski, and Adrian Vladu. 2017. Negative-weight shortest paths and unit capacity minimum cost flow in $\tilde{O}(m^{10/7} \log W)$ time. In *Proceedings of the 28th ACM-SIAM Symposium on Discrete Algorithms (SODA'17)*. SIAM, 752–771. DOI: https://doi.org/10.1137/1.9781611974782.48

[4] Sabine Cornelsen, Andreas Karrenbauer, and Shujun Li. 2012. Leveling the grid. In *Proceedings of the Workshop on Algorithm Engineering and Experimentation (ALENEX'12)*. SIAM, 45–54. DOI: https://doi.org/10.1137/1.9781611972924.4

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

[6] L. R. Ford and D. R. Fulkerson. 1957. A simple algorithm for finding maximal network flows and an application to the Hitchcock problem. *Canadian J. Math.* 9 (1957), 210–218. DOI: https://doi.org/10.4153/CJM-1957-024-0

[7] H. N. Gabow and R. E. Tarjan. 1989. Faster scaling algorithms for network problems. *SIAM J. Comput.* 18 (Oct. 1989), 1013–1036. Issue 5. DOI: https://doi.org/10.1137/0218069

[8] Pawel Gawrychowski and Adam Karczmarz. 2018. Improved bounds for shortest paths in dense distance graphs. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP'18)*, Vol. 107. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 61:1–61:15. DOI: https://doi.org/10.4230/LIPIcs.ICALP.2018.61

[9] J. Hopcroft and R. Karp. 1973. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.* 2, 4 (1973), 225–231. DOI: https://doi.org/10.1137/0202019

[10] Haim Kaplan, Shay Mozes, Yahav Nussbaum, and Micha Sharir. 2017. Submatrix maximum queries in Monge matrices and partial Monge matrices, and their applications. *ACM Trans. Alg.* 13, 2 (2017), 26. DOI: https://doi.org/10.1145/3039873

[11] Philip Klein. 2005. Multiple-source shortest paths in planar graphs. In *Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms (SODA'05)*. SIAM, 146–155.

[12] Philip N. Klein, Shay Mozes, and Christian Sommer. 2013. Structured recursive separator decompositions for planar graphs in linear time. In *Proceedings of the 45th ACM Symposium on Theory of Computing (STOC'13)*. ACM, 505–514. DOI: https://doi.org/10.1145/2488608.2488672

[13] Harold Kuhn. 1956. Variants of the Hungarian method for assignment problems. *Nav. Res. Log.* 3, 4 (1956), 253–258.

[14] Richard J. Lipton and Robert Endre Tarjan. 1980. Applications of a planar separator theorem. *SIAM J. Comput.* 9, 3 (1980), 615–627. DOI: https://doi.org/10.1137/0209046

[15] Shay Mozes and Christian Wulff-Nilsen. 2010. Shortest paths in planar graphs with real lengths in $O(n \log^2 n/(\log \log n))$. In *Proceedings of the European Symposium on Algorithms (ESA'10)*. Springer, Berlin, 206–217. DOI: https://doi.org/10.1007/978-3-642-15781-3_18

[16] Marcin Mucha and Piotr Sankowski. 2004. Maximum matchings via Gaussian elimination. In *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science (FOCS'04)*. IEEE, 248–255. DOI: https://doi.org/10.1109/FOCS.2004.40

[17] Marcin Mucha and Piotr Sankowski. 2006. Maximum matchings in planar graphs via Gaussian elimination. *Algorithmica* 45, 1 (2006), 3–20. DOI: https://doi.org/10.1007/s00453-005-1187-5

[18] L. Ramshaw and R. E. Tarjan. 2012. A weight-scaling algorithm for min-cost imperfect matchings in bipartite graphs. In *Proceedings of the 53rd IEEE Symposium on Foundations of Computer Science (FOCS'12)*. IEEE, 581–590. DOI: https://doi.org/10.1109/FOCS.2012.9.

[19] Piotr Sankowski. 2009. Maximum weight bipartite matching in matrix multiplication time. *Theor. Comput. Sci.* 410 (2009), 4480–4488. Issue 44. DOI: https://doi.org/10.1016/j.tcs.2009.07.028

[20] R. Sharathkumar. 2013. A sub-quadratic algorithm for bipartite matching of planar points with bounded integer coordinates. In *Proceedings of the 29th Symposium on Computational Geometry (SOCG'13)*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 9–16. DOI: https://doi.org/10.1145/2462356.2480283

Received March 2018; revised August 2019; accepted September 2019