

# Geometric Partial Matching and Unbalanced Transportation

**Pankaj K. Agarwal**

Duke University, USA

[pankaj@cs.duke.edu](mailto:pankaj@cs.duke.edu)

**Hsien-Chih Chang**

Duke University, USA

[hsienchih.chang@duke.edu](mailto:hsienchih.chang@duke.edu)

**Allen Xiao**

Duke University, USA

[axiao@cs.duke.edu](mailto:axiao@cs.duke.edu)

## Abstract

Let  $A$  and  $B$  be two point sets in the plane with uneven sizes  $r$  and  $n$  respectively (assuming  $r$  is at most  $n$ ), and let  $k$  be a parameter. The geometric partial matching problem asks to find the minimum-cost size- $k$  matching between  $A$  and  $B$  under powers of  $L_p$  distances. Applying combinatorial algorithms for partial matching in general graphs to our setting naively requires quadratic time due to existence of many edges between point sets  $A$  and  $B$ . Most previous work for geometric matching has focused on the setting when  $k$ ,  $r$ , and  $n$  are equal. The best algorithm in this setting, due to Sharathkumar and Agarwal [STOC 2012], runs in time  $O(n \text{ polylog } n \text{ poly } \varepsilon^{-1})$ , but is limited to matching objectives that are sum-of-distances.

We present the first set of geometric algorithms which work for any powers of  $L_p$ -norm matching objective: An exact algorithm which runs in  $O((n + k^2) \text{ polylog } n)$  time, and a  $(1 + \varepsilon)$ -approximation which runs in  $O((n + k\sqrt{k}) \text{ polylog } n \log \varepsilon^{-1})$  time. Both algorithms are based on primal-dual flow augmentation scheme; the main improvements are obtained by using dynamic data structures to achieve efficient flow augmentations. Using similar techniques, we give an exact algorithm for the transportation problem in the plane, which runs in  $O(rn(r + \sqrt{n}) \text{ polylog } n)$  time. This is the first sub-quadratic time exact algorithm when  $r = o(\sqrt{n})$ , which improves over the state-of-art quadratic time algorithm by Agarwal *et al.* [SOCG 2016].

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms

**Keywords and phrases** partial matchings, transportation, minimum-cost flow,

**Lines** 1025

## 1 Introduction

### «REWRITE AFTER TECHNICAL SECTIONS»

Consider the problem of finding a minimum-cost bichromatic matching between a set of red points  $A$  and a set of blue points  $B$  lying in the plane, where the cost of a matching edge  $(a, b)$  is the Euclidean distance  $\|a - b\|$ ; in other words, the minimum-cost bipartite matching problem on the Euclidean complete graph  $G = (A \cup B, A \times B)$ . Let  $r := |A|$  and  $n := |B|$ . Without loss of generality, assume that  $r \leq n$ . We consider the problem of *partial matching*, where the task is to find a minimum-cost matching of size  $k \leq r$ . When  $k = r = n$ , we say the matching instance is **balanced**. When  $k = r < n$  ( $A$  and  $B$  have different sizes, but the matching is maximal), we say the matching instance is **unbalanced**. We call the geometric problem of finding a size  $k$  matching



of point sets  $A$  and  $B$  the *geometric partial matching problem*. **«talk about the near-linear time alg»**

## 1.1 Contributions

In this paper, we present two algorithms for geometric partial matching that are based on fitting nearest-neighbor (NN) and geometric closest pair (BCP) oracles into primal-dual algorithms for non-geometric bipartite matching and minimum-cost flow. This pattern is not new, see for example (...) **«TODO cite»**. Unlike these previous works, we focus on obtaining running time dependencies on  $k$  or  $r$  instead of  $n$ , that is, faster for inputs with small  $r$  or  $k$ . We begin in Section 2 by introducing notation for matching and minimum-cost flow.

First in Section 3, we show that the Hungarian algorithm [7] combined with a BCP oracle solves geometric partial matching exactly in time  $O((n + k^2) \text{polylog } n)$ . Mainly, we show that we can separate the  $O(n \text{polylog } n)$  preprocessing time for building the BCP data structure from the augmenting paths' search time, and update duals in a lazy fashion such that the number of dual updates per augmenting path is  $O(k)$ .

► **Theorem 1.1.** *Let  $A$  and  $B$  be two point sets in the plane with  $|A| = r$  and  $|B| = n$  satisfying  $r \leq n$ , and let  $k$  be a parameter. A minimum-cost geometric partial matching of size  $k$  can be computed between  $A$  and  $B$  in  $O((n + k^2) \text{polylog } n)$  time.*

**«State the settings separately so no need to repeat in the theorem statement.»**

Next in Section 4, we apply a similar technique to the unit-capacity min-cost circulation algorithm of Goldberg, Hed, Kaplan, and Tarjan [4]. The resulting algorithm finds a  $(1 + \epsilon)$ -approximation to the optimal geometric partial matching in  $O((n + k\sqrt{k}) \text{polylog } n \log(n/\epsilon))$  time.

► **Theorem 1.2.** *Let  $A$  and  $B$  be two point sets in the plane with  $|A| = r$  and  $|B| = n$  satisfying  $r \leq n$ , and let  $k$  be a parameter. A  $(1 + \epsilon)$  geometric partial matching of size  $k$  can be computed between  $A$  and  $B$  in  $O((n + k\sqrt{k}) \text{polylog } n \log(1/\epsilon))$  time.*

Our third algorithm solves the transportation problem in the unbalanced setting. The transportation problem is a weighted generalization of the matching problem. Each point of  $A$  is weighted with an integer *supply* and each point of  $B$  is weighted with integer *demand* such that the sum of supply and demand are equal. The goal of the transportation problem is to find a minimum-cost mapping of all supplies to demands, where the cost of moving a unit of supply at  $a \in A$  to satisfy a unit of demand at  $b \in B$  is  $\|a - b\|$ . For this, we use the strongly polynomial uncapacitated min-cost flow algorithm by Orlin [8]. The result is an  $O(n^{3/2}r \text{polylog } n)$  time algorithm for unbalanced transportation. This improves over the  $O(n^2 \text{polylog } n)$  time algorithm of Agarwal *et al.* [1] when  $r = o(\sqrt{n})$ .

► **Theorem 1.3.** *Let  $A$  and  $B$  be two point sets in the plane with  $|A| = r$  and  $|B| = n$  satisfying  $r \leq n$ , with supplies and demands given by the function  $\lambda : (A \cup B) \rightarrow \mathbb{Z}$  such that  $\sum_{a \in A} \lambda(a) = \sum_{b \in B} \lambda(b)$ . An optimal transportation map can be computed in  $O(rn(r/\sqrt{n} + \sqrt{n}) \text{polylog } n)$  time.*

By nature of the BCP/NN oracles we use, these results generalize to when  $\|a - b\|$  is any  $L_p$  distance, and if we use  $p'$ -th power costs  $c(a, b) = \|a - b\|^{p'}$  for any  $1 \leq p' < \infty$ .

## 2 Preliminaries

### 2.1 Matching

Let  $G$  be a bipartite graph between vertex sets  $A$  and  $B$  and edge set  $E$ , with costs  $c(v, w)$  for each edge  $e$  in  $E$ . **«Should we define the problem on point sets instead, and construct the graph afterwards?»** We use  $C := \max_{e \in E} c(e)$ , and assume that the problem is scaled such that  $\min_{e \in E} c(e) = 1$ . **«where do we use this assumption?»** A *matching*  $M \subseteq E$  is a set of edges where no two edges share an endpoint. We use  $V(M)$  to denote the vertices matched by  $M$ . The *size* of a matching is the number of edges in the set, and the *cost* of a matching is the sum of costs of its edges. The *minimum-cost partial matching problem (MPM)* asks to find a size- $k$  matching  $M^*$  of minimum cost.

**«Define LP-duality and admissibility for matchings»**

## 3 Computing Min-cost Partial Matching using Hungarian algorithm

The Hungarian algorithm maintains a 0-optimal (initially empty) matching  $M$ , and repeatedly augments by alternating augmenting paths of admissible edges until  $|M| = k$ . To this end, the algorithm maintains a set of feasible potentials  $\pi$  and updates them to find augmenting paths of admissible edges. **«admissible augmenting path?»** It maintains the invariant that matching edges are admissible. Since there are  $k$  augmentations and each alternating path has length at most  $2k - 1$ , the total time spent on bookkeeping the matching is  $O(k^2)$ . This leaves the analysis of the subroutine that updates the potentials and finds an admissible augmenting path; we call this subroutine the *Hungarian search*.

► **Theorem 3.1 Time for Hungarian algorithm.** *Let  $G = (A \cup B, A \times B)$  be an instance of geometric partial matching with  $r := |A|$ ,  $n := |B|$ ,  $r \leq n$ , and parameter  $k \leq r$ . Suppose the Hungarian search finds each augmenting path in  $T(n, k)$  time after a one-time  $P(n, k)$  preprocessing time. Then, the Hungarian algorithm finds the optimal size  $k$  matching in time  $O(P(n, k) + kT(n, k) + k^2)$ .*

### 3.1 Hungarian search

Let  $S$  be the set of vertices that can be reached from an unmatched  $a \in A$  by admissible residual edges, initially the unmatched vertices of  $A$ . The Hungarian search updates potentials in a Dijkstra's algorithm-like manner, expanding  $S$  until it includes an unmatched  $b \in B$  (and thus an admissible alternating augmenting path). The “search frontier” of the Hungarian search is  $(S \cap A) \times (B \setminus S)$ . We *relax* the minimum-reduced cost edge in the frontier, changing the potentials of vertices  $S$  such that the edge becomes admissible, and adding the head of the edge into  $S$ . **«Well, either you add both endpoints into  $S$ , or an admissible augmenting path is found.»**

The potential update uniformly decreases the reduced costs of the frontier edges. Since  $(a', b')$  is the minimum reduced cost frontier edge, the potential update in line ?? does not make any reduced cost negative, and thus preserves the dual feasibility constraint for all edges. The algorithm is shown below as Algorithm ??.

By tracking the forest of relaxed edges (e.g. back pointers), it is straightforward to recover the alternating augmenting path  $\Pi$  once we reach an unmatched  $b' \in B$ . We make the following observation about the Hungarian search:

108 ► **Lemma 3.2.** *There are at most  $k$  edge relaxations before the Hungarian search finds an alternating*  
 109 *augmenting path.*

110 **Proof.** Each edge relaxation either leads to a matched vertex in  $B$  (there are at most  $k - 1$  such  
 111 vertices), or finds an unmatched vertex and ends the search. ◀

112 In general graphs, the minimum edge is typically found by pushing all encountered  $(S \cap A) \times$   
 113  $(B \setminus S)$  edges into a priority queue. However, in the bipartite complete graph, this may take  
 114  $\Theta(rn \text{ polylog } n)$  time for each Hungarian search — edges are being pushed into the queue even  
 115 when they are not relaxed. We avoid this problem by finding an edge with minimum cost using  
 116 **bichromatic closest pair** (BCP) queries on an additively weighted Euclidean distances, for which  
 117 there exist fast dynamic data structures. Given two point sets  $P$  and  $Q$  in the plane, the BCP is  
 118 the pair of points  $p \in P$  and  $q \in Q$  minimizing the (adjusted) distance  $\|p - q\| - \omega(p) + \omega(q)$ , for  
 119 some real-valued vertex weights  $\omega(p)$ . In our setting, the vertex weights will mostly be set as the  
 120 potentials; the adjusted distance is equal to the reduced cost.

121 **«Short history on BCP?»** The state of the art dynamic BCP data structure from Kaplan,  
 122 Mulzer, Roditty, Seiferth, and Sharir [6] supports point insertions and deletions in  $O(\text{polylog } n)$   
 123 time, and answers queries in  $O(\log^2 n)$  time. The following lemma, combined with Theorem 3.1,  
 124 completes the proof of Theorem 1.1.

125 ► **Lemma 3.3.** *Using the dynamic BCP data structure from Kaplan et al., we can implement*  
 126 *Hungarian search with  $T(n, k) = O(k \text{ polylog } n)$  and  $P(n, k) = O(n \text{ polylog } n)$ .*

127 **Proof.** Recall that we maintain a BCP data structure between  $P = (S \cap A)$  and  $Q = (B \setminus S)$ .  
 128 Changes to the  $P$  and  $Q$  are entirely driven by changing  $S$ ; that is, updates to  $S$  incur BCP  
 129 insertions/deletions. We first analyze the bookkeeping besides the potential updates, and then  
 130 show how potential updates can be implemented efficiently.

131 **«Untangles the algorithm with the proof; move the algorithm out of the proof of**  
 132 **the lemma.»**

- 133 1. Let  $S_0^t$  **«Is there a reason why you want the subscript? Do you ever define  $S^t$ ?»**  
 134 denote the initial set  $S$  at the beginning of the  $t$ -th Hungarian search, that is, the set of  
 135 unmatched points in  $A$  after  $t$  augmentations. At the very beginning of the Hungarian  
 136 algorithm, we initialize  $S_0^0 \leftarrow A$  (meaning that  $P = A$  and  $Q = B$ ), which is a one-time  
 137 insertion of  $O(n)$  points into BCP, attributed to  $P(n, k)$ . On each successive Hungarian search,  
 138  $S_0^t$  shrinks as more and more points in  $A$  are matched. Assume for now that, at the beginning  
 139 of the  $(t + 1)$ -th Hungarian search, we are able to construct  $S_0^t$  from the previous iteration. To  
 140 construct  $S_0^{t+1}$ , we simply remove the point in  $A$  that was matched by the  $t$ -th augmenting  
 141 path. Thus, with that assumption, we are able to initialize  $S$  using one BCP deletion operation  
 142 per augmentation.
- 143 2. During each Hungarian search, points are added to  $P$  (that is, some points in  $A$  are added to  
 144  $S$ ) and removed from  $Q$  (points in  $B$  added to  $S$ ), which will happen at most once per edge  
 145 relaxation. By Lemma 3.2 the number of relaxed edges is at most  $k$ , so the number of such  
 146 BCP operations is also at most  $k$ .
- 147 3. To obtain  $S_0^t$ , we keep track **«give a name to such points»** of the points added since  $S_0^t$  in  
 148 the last Hungarian search (i.e. those of (2)). **«Unclear»** After the augmentation, we use this  
 149 log **«use the name»** to delete the added vertices from  $S$  and recover  $S_0^t$ . By the argument  
 150 in (2) there are  $O(k)$  of such points to delete, so reconstructing  $S_0^t$  takes  $O(k)$  BCP operations.  
 151 **«TODO change to full persistence: loglogm overhead with  $m = r$  modifications»**

We spend  $P(n, k) = O(n \text{ polylog } n)$  time to build the initial BCP. The number of BCP operations associated with each Hungarian search is  $O(k)$ , so the time spent on BCP operations in each Hungarian search is  $O(k \text{ polylog } n)$ .

As for the potential updates, we modify a trick from Vaidya [10] to batch potential updates. Potentials have a **stored value**, i.e. the current value of  $\pi(v)$ , and a **true value**, which may have changed from  $\pi(v)$ . The algorithm uses the true value when dealing with reduced costs and updates the stored value rarely; we explain the mechanism shortly.

Throughout the course of the algorithm, we maintain a nonnegative value  $\delta$  (initially 0) which aggregates potential changes. Vertices that are added to  $S$  are immediately added to a BCP data structure with weight  $\omega(p) \leftarrow \pi(p) - \delta$ , for whatever value  $\delta$  is at the time of insertion. When the points of  $S$  have potentials increased by  $\gamma$  in (2), we instead raise  $\delta \leftarrow \delta + \gamma$ . Thus, true value for any potential of a point in  $S$  is  $\omega(p) + \delta$ . For points of  $(A \cup B) \setminus S$ , the true potential is equal to the stored potential.

Since potentials for  $S$  points are uniformly offsetted by  $\delta$ , the minimum edge returned by the BCP oracle does not change. Once a point is removed from  $S$ , we update its stored potential to be  $\pi(p) \leftarrow \omega(p) + \delta$ , for the current value of  $\delta$ . Importantly,  $\delta$  is not reset at the end of a Hungarian search, and persists throughout the entire algorithm. This way, the unmatched points in each  $S_0^t$  have their true potentials accurately represented by  $\delta$  and  $\omega(p)$ .

The number of updates to  $\delta$  is equal to the number of edge relaxations, which is  $O(k)$  per Hungarian search. We update stored potentials when removing a point from  $S$  (by the rewind mechanism, or due to an augmentation) which occurs  $O(k)$  times per Hungarian search. The time spent on potential updates per Hungarian search is therefore  $O(k)$ . Overall, the time spent per Hungarian search is  $T(n, k) = O(k \text{ polylog } n)$ .

«The proof gets more handwavy as the paragraph progresses. Consider a revision after this round.»

## 4 Approximating Min-Cost Partial Matching through Cost-Scaling

The goal of section is to prove Theorem 1.2; that is, to compute a size- $k$  geometric partial matching between two point sets  $A$  and  $B$  in the plane, with cost at most  $(1 + \varepsilon)$  times the optimal matching, in time  $O((n + k\sqrt{k}) \text{ polylog } n \log(1/\varepsilon))$ .

After introducing the necessary terminologies in Section 4.1, we reduce the partial matching problem to computing an approximate minimum-cost flow on a unit-capacity reduction network in Section 4.2. In Section 4.3 we outline the high-level overview of the cost-scaling algorithm. We postpone the fast implementation using dynamic data structures to Section 5.

### 4.1 Preliminaries on Network Flows

**Network.** Let  $G = (V, E)$  be a directed graph, augmented by edge costs  $c$  and capacities  $u$ , and a supply-demand function  $\phi$  defined on the vertices. One can turn the graph  $G$  into a **network**  $N = (V, \vec{E})$ : For each directed edge  $(v, w)$  in  $E$ , insert two **arcs**  $v \rightarrow w$  and  $w \rightarrow v$  into the arc set  $\vec{E}$ ; the **forward arc**  $v \rightarrow w$  inherits the capacity and cost from the directed graph  $G$ , while the **backward arc**  $w \rightarrow v$  satisfies  $u(w \rightarrow v) = 0$  and  $c(w \rightarrow v) = -c(v \rightarrow w)$ . This we ensure that the graph  $(V, \vec{E})$  is **symmetric** and the cost function  $c$  is **antisymmetric** on  $N$ . The positive values of  $\phi(v)$  are referred to as **supply**, and the negative values of  $\phi(v)$  as **demand**. We assume that all capacities are nonnegative, all supplies and demands are integers, and the sum of supplies and demands is equal to zero. A **unit-capacity** network has all its edge capacities equal to 1. In this section we assume all networks are of unit-capacity.

**Pseudoflows.** Given a network  $N := (V, \vec{E}, c, u, \phi)$ , a **pseudoflow** (or **flow** to be short)  $f : \vec{E} \rightarrow \mathbb{Z}^1$  on  $N$  is an antisymmetric function on the arcs of  $N$  satisfying  $f(v \rightarrow w) \leq u(v \rightarrow w)$  for every arc  $v \rightarrow w$ . We sometimes abuse the terminology by allowing pseudoflow to be defined on a directed graph, in which case we are actually referring to the pseudoflow on the corresponding network by extending the flow values antisymmetrically to the arcs. We say that  $f$  **saturates** an arc  $v \rightarrow w$  if  $f(v \rightarrow w) = u(v \rightarrow w)$ ; an arc  $v \rightarrow w$  is **residual** if  $f(v \rightarrow w) < u(v \rightarrow w)$ . The **support** of  $f$  in  $N$ , denoted as  $\text{supp}(f)$ , is the set of arcs with positive flows:

$$\text{supp}(f) := \{v \rightarrow w \in \vec{E} \mid f(v \rightarrow w) > 0\}.$$

Given a pseudoflow  $f$ , we define the **imbalance** of a vertex (with respect to  $f$ ) to be

$$\phi_f(v) := \phi(v) + \sum_{w \rightarrow v \in \vec{E}} f(w \rightarrow v) - \sum_{v \rightarrow w \in \vec{E}} f(v \rightarrow w).$$

We call positive imbalance **excess** and negative imbalance **deficit**; and vertices with positive and negative imbalance **excess vertices** and **deficit vertices**, respectively. A vertex is **balanced** if it has zero imbalance. If all vertices are balanced, the pseudoflow is a **circulation**. The **cost** of a pseudoflow is defined to be

$$\text{cost}(f) := \sum_{v \rightarrow w \in \text{supp}(f)} c(v \rightarrow w) \cdot f(v \rightarrow w).$$

The **minimum-cost flow problem (MCF)** asks to find a circulation of minimum cost inside a given directed graph.

**Residual graph.** Given a pseudoflow  $f$ , one can define the **residual network** as follows. Recall that the set of **residual arcs**  $\vec{E}_f$  under  $f$  are those arcs  $v \rightarrow w$  satisfying  $f(v \rightarrow w) < u(v \rightarrow w)$ . In other words, an arc that is not saturated by  $f$  is a residual arc; similarly, given an arc  $v \rightarrow w$  with positive flow value, the backward arc  $w \rightarrow v$  is a residual arc.

Let  $N = (V, \vec{E}, c, u, \phi)$  be a network constructed from graph  $G$ , with a pseudoflow  $f$  on  $N$ . The **residual graph**  $G_f$  of  $f$  has  $V$  as its vertex set and  $\vec{E}_f$  as its arc set. The **residual capacity**  $u_f$  with respect to pseudoflow  $f$  is defined to be  $u_f(v \rightarrow w) := u(v \rightarrow w) - f(v \rightarrow w)$ . Observe that the residual capacity is always nonnegative. We can define residual arcs differently using residual capacities:

$$\vec{E}_f = \{v \rightarrow w \mid u_f(v \rightarrow w) > 0\}.$$

In other words, the set of residual arcs are precisely those arcs in the residual graph, each of which has nonzero residual capacity.

**LP-duality and admissibility.** To solve the minimum-cost flow problem, we focus on the primal-dual algorithms using linear programming. Let  $G = (V, E)$  be a given directed graph with the corresponding network  $N = (V, \vec{E}, c, u, \phi)$ . Formally, the **potentials**  $\pi(v)$  are the variables of the linear program dual to the standard linear program for the minimum-cost flow problem with variables  $f(v, w)$  for each directed edge in  $E$ . Assignments to the primal variables satisfying the capacity constraints extend naturally into a pseudoflow on the network  $N$ . Let  $G_f = (V, \vec{E}_f)$  be the residual graph under pseudoflow  $f$ . The **reduced cost** of an arc  $v \rightarrow w$  in  $\vec{E}_f$  with respect to  $\pi$  is defined as

$$c_\pi(v \rightarrow w) := c(v \rightarrow w) - \pi(v) + \pi(w).$$

<sup>1</sup> In general the pseudoflows are allowed to take real-values. Here under the unit-capacity assumption any optimal flows are integer-valued. **⟨⟨cite integrality theorem?⟩⟩**



236 Notice that the cost function  $c_\pi$  is also antisymmetric.

237 The **dual feasibility constraint** says that  $c_\pi(v \rightarrow w) \geq 0$  holds for every directed edge  $(v, w)$   
 238 in  $E$ ; potentials  $\pi$  which satisfy this constraint are said to be **feasible**. Suppose we relax the dual  
 239 feasibility constraint to allow some small violation in the value of  $c_\pi(v \rightarrow w)$ . We say that a pair  
 240 of pseudoflow  $f$  and potential  $\pi$  is  **$\varepsilon$ -optimal** [?, ?] if  $c_\pi(v \rightarrow w) \geq -\varepsilon$  for every residual arc  $v \rightarrow w$   
 241 in  $\vec{E}_f$ . Pseudoflow  $f$  is  **$\varepsilon$ -optimal** if it is  $\varepsilon$ -optimal with respect to some potentials  $\pi$ ; potential  
 242  $\pi$  is  **$\varepsilon$ -optimal** if it is  $\varepsilon$ -optimal with respect to some pseudoflow  $f$ . Given a pseudoflow  $f$  and  
 243 potentials  $\pi$ , a residual arc  $v \rightarrow w$  in  $\vec{E}_f$  is **admissible** if  $c_\pi(v \rightarrow w) \leq 0$ . We say that a pseudoflow  
 244  $g$  in  $G_f$  is **admissible** if all support arcs of  $g$  on  $G_f$  are admissible; in other words,  $g(v \rightarrow w) > 0$   
 245 holds only on admissible arcs  $v \rightarrow w$ .

246 ► **Lemma 4.1.** *Let  $f$  be an  $\varepsilon$ -optimal pseudoflow in  $G$  and let  $f'$  be an admissible flow in  $G_f$ . Then*  
 247  *$f + f'$  is also  $\varepsilon$ -optimal. ((Lemma 5.3 in [5]? Also, where is this used?))*

## 248 4.2 Reduction to Unit-Capacity Min-Cost Flow Problem

249 The goal of the subsection is to reduce the minimum-cost partial matching problem to the unit-  
 250 capacity minimum-cost flow problem with a polynomial bound on diameter. To this end we  
 251 first provide an upper bound on the size of support of an integral pseudoflow on the standard  
 252 reduction network between the two problems. This upper bound in turn provides an additive  
 253 approximation on the cost of an  $\varepsilon$ -optimal circulation. Next we employ a technique by Sharathku-  
 254 mar and Agarwal [9] to transform an additive  $\varepsilon$ -approximate solution into a multiplicative  
 255  $(1 + \varepsilon)$ -approximation for the geometric partial matching problem. The reduction does not work  
 256 out of the box, as Sharathkumar and Agarwal were tackling a similar but different problem on  
 257 geometric transportations.

258 ► **Lemma 4.2.** *Computing a  $(1 + \varepsilon)$ -approximate geometric partial matching can be reduced to*  
 259 *the following problem in  $O(n \text{ polylog } n)$  time: Given a reduction network  $N$  over a point set with*  
 260 *diameter at most  $K \cdot kn^3$  for some constant  $K$ , compute a  $(K \cdot \varepsilon / 6k)$ -optimal circulation on  $N$ .*

261 **Additive approximation.** Given a bipartite graph  $G = (A, B, E_0)$  for the geometric partial  
 262 matching problem with cost function  $c$ , we construct the **reduction network**  $N_H$  as follows:  
 263 Direct the edges in  $E_0$  from  $A$  to  $B$ , and assign each directed edge with capacity 1. Now add a  
 264 dummy vertex  $s$  with directed edges to all vertices in  $A$ , and add a dummy vertex  $t$  with directed  
 265 edges from all vertices in  $B$ ; each edge added this way has cost 0 and capacity 1. Denote the new  
 266 graph with vertex set  $V = A \cup B \cup \{s, t\}$  and edge set  $E$  as the **reduction graph**  $H$ . Assign vertex  $s$   
 267 with supply  $k$  and vertex  $t$  with demand  $k$ ; the rest of the vertices in  $H$  have zero supply-demand  
 268 We call the network naturally corresponds to  $H$  as the **reduction network**, denoted by  $N_H$ .

269 It is straightforward to show that any integer circulation  $f$  on  $N_H$  uses exactly  $k$  of the  $A$ -to- $B$   
 270 arcs, which correspond to the edges of a size- $k$  matching  $M_f$ . Notice that the cost of the circulation  
 271  $f$  is equal to the cost of the corresponding matching  $M_f$ . In other words, a  $(1 + \varepsilon)$ -approximation  
 272 to the MCF problem on the reduction network  $N_H$  translates to a  $(1 + \varepsilon)$ -approximation to the  
 273 geometric matching problem on the input graph  $G$ .

274 First we show that the number of arcs used by any integer pseudoflow in  $N_H$  is asymptotically  
 275 bounded by the excess of the pseudoflow.

276 ► **Lemma 4.3.** *Let  $f$  be an integer circulation in the reduction network  $N_H$ . Then, the size of the*  
 277 *support of  $f$  is at most  $3k$ . As a corollary, the number of residual backward arcs is at most  $3k$ .*

278 Using the bound on the support size, we show that an  $\varepsilon$ -optimal integral circulation gives an  
 279 additive  $O(k\varepsilon)$ -approximation to the MCF problem.

280 ► **Lemma 4.4.** *Let  $f$  be an  $\varepsilon$ -optimal integer circulation in  $N_H$ , and  $f^*$  be an optimal integer*  
 281 *circulation for  $N_H$ . Then,  $\text{cost}(f) \leq \text{cost}(f^*) + 6k\varepsilon$ .*

282 **Multiplicative approximation.** Now we employ a technique from Sharathkumar and Agar-  
 283 wal [9] to convert the additive approximation into a multiplicative one. Here we sketch a proof  
 284 to Lemma 4.2; a complete proof can be found in the Appendix.

285 **⟨Sketch Sharathkumar and Agarwal [9] with our modification.⟩**

### 286 4.3 High-Level Description of Cost-Scaling Algorithm

290 Our main algorithm for the unit-capacity minimum-cost flow problem is based on the **cost-scaling**  
 291 technique, originally due to Goldberg and Tarjan [5]; Goldberg, Hed, Kaplan, and Tarjan [4]  
 292 applied the technique on unit-capacity networks. The algorithm finds  $\varepsilon$ -optimal circulations  
 293 for geometrically shrinking values of  $\varepsilon$ . Each fixed value of  $\varepsilon$  is called a **cost scale**. Once  $\varepsilon$  is  
 294 sufficiently small, the  $\varepsilon$ -optimal flow is a suitable approximation according to Lemma 4.2.<sup>2</sup>

295 The cost-scaling algorithm initializes the flow  $f$  and the potential  $\pi$  to be zero. Note that the  
 296 zero flow is trivially a  $kC$ -optimal flow. At the beginning of each scale starting at  $\varepsilon = kC$ ,

- 297 ■ SCALE-INIT takes the previous circulation (now  $2\varepsilon$ -optimal) and transforms it into an  $\varepsilon$ -optimal  
 298 pseudoflow with  $O(k)$  excess.
- 299 ■ REFINE then reduces the excess in the newly constructed pseudoflow to zero, making it an  
 300  $\varepsilon$ -optimal circulation.

301 Thus, the algorithm produces an  $\varepsilon^*$ -optimal circulation after  $O(\log(kC/\varepsilon^*))$  scales. Using the  
 302 reduction in Lemma 4.2, we have the diameter of the point set, thus maximum cost  $C$ , bounded  
 303 by  $O(K \cdot kn^3)$  for some value  $K$ . By setting  $\varepsilon^*$  to be  $K \cdot \varepsilon/6k$ , the number of cost scales is bounded  
 304 above by  $O(\log(n/\varepsilon))$ .

305 **Scale initialization.** Recall that  $H$  is the *reduction graph* and  $N_H$  is the *reduction network*, both  
 306 constructed in Section 4.2. The vertex set of  $H$  consists of two point sets  $A$  and  $B$ , as well as two  
 307 dummy vertices  $s$  and  $t$ . The directed edges in  $H$  are pointed from  $s$  to  $A$ , from  $A$  to  $B$ , and from  
 308  $B$  to  $t$ . We call those arcs in  $N_H$  whose direction is consistent with their corresponding directed  
 309 edges as **forward arcs**, and those arcs that points in the opposite direction as **backward arcs**.

310 The procedure SCALE-INIT transforms a  $2\varepsilon$ -optimal circulation from the previous cost scale  
 311 into an  $\varepsilon$ -optimal flow with  $O(k)$  excess, by raising the potentials  $\pi$  of all vertices in  $A$  by  $\varepsilon$ , those  
 312 in  $B$  by  $2\varepsilon$ , and the potential of  $t$  by  $3\varepsilon$ . The potential of  $s$  remains unchanged. Now the reduced  
 313 cost of every forward arc is dropped by  $\varepsilon$ , and thus all the forward arcs have reduced cost at  
 314 least  $-\varepsilon$ .

315 As for backward arcs, the procedure SCALE-INIT continues by setting the flow on  $v \rightarrow w$  to  
 316 zero for each backward arc  $w \rightarrow v$  violating the  $\varepsilon$ -optimality constraint. In other words, we set  
 317  $f(v \rightarrow w) = 0$  whenever  $c_\pi(w \rightarrow v) < -\varepsilon$ . This ensures that all such backward arcs are no longer  
 318 residual, and therefore the flow (now with excess) is  $\varepsilon$ -optimal.

319 Because the arcs are of unit-capacity in  $N_H$ , each arc desaturation creates one unit of excess.  
 320 By Lemma 4.3 the number of backward arcs is at most  $3k$ . Thus the total amount of excess  
 321 created is also  $O(k)$ .

287 <sup>2</sup> When the costs are integers, an  $\varepsilon$ -optimal circulation for a sufficiently small  $\varepsilon$  (say less than  $1/n$ ) is itself an  
 288 optimal solution [4, 5]. We present this algorithm without the integral-cost assumption because in the geometric  
 289 partial matching setting (with respect to  $L_p$  norms) the costs are generally not integers.



322 In total, potential updates and backward arc desaturations, thus the whole procedure SCALE-  
323 INIT, take  $O(n)$  time.

324 **Refinement.** The procedure REFINE is implemented using a primal-dual augmentation al-  
325 gorithm, which sends flows on admissible arcs to reduce the total excess, like the Hungarian  
326 algorithm. Unlike the Hungarian algorithm, it uses *blocking flows* instead of augmenting paths.  
327 An *augmenting path* is a path in the residual network from an excess vertex to a deficit vertex.  
328 We call a pseudoflow  $f$  on residual network  $N_g$  a **blocking flow** if  $f$  saturates at least one residual  
329 arc in every augmenting path in  $N_g$ . In other words, there is no admissible augmenting path in  
330  $N_{f+g}$  from an excess vertex to a deficit vertex.

331 Each iteration of REFINE finds an admissible blocking flow that is then added to the current  
332 pseudoflow in two stages:

- 333 1. A *Hungarian search*, which increases the dual variables  $\pi$  of vertices that are reachable from  
334 an excess vertex by at least  $\varepsilon$ , in a Dijkstra-like manner, until there is an excess-deficit path of  
335 admissible edges.
- 336 2. A *depth-first search* through the set of admissible edges to construct an admissible blocking  
337 flow. It suffices to repeatedly extract admissible augmenting paths until no more admissible  
338 excess-deficit paths remain. By definition, the union of such paths is a blocking flow. **«Move  
339 to where the blocking flow is introduced?»**

340 The algorithm continues until the total excess becomes zero and the  $\varepsilon$ -optimal flow is now a  
341 circulation.

342 First we analyze the number of iterations executed by REFINE. The proof follows the strategy  
343 in Goldberg *et al.* [4, Section 3.2]. **«and maybe §5 of Goldberg-Tarjan?»** To this end we need  
344 a bound on the size of the support of  $f$  right before and throughout the execution of REFINE.

345 ► **Lemma 4.5.** *Let  $f$  be an integer pseudoflow in  $N_H$  with  $O(k)$  excess. Then, the size of the support  
346 of  $f$  is at most  $O(k)$ .*

347 ► **Corollary 4.6.** *The size of  $\text{supp}(f)$  is at most  $O(k)$  for pseudoflow  $f$  right before or during the  
348 execution of REFINE.*

349 ► **Lemma 4.7.** *Let  $f$  be a pseudoflow in  $N_H$  with  $O(k)$  excess. The procedure REFINE runs for  
350  $O(\sqrt{k})$  iterations before the excess of  $f$  becomes zero.*

351 The goal of the next section is to show that after  $O(n \text{ polylog } n)$  time preprocessing, each  
352 Hungarian search and depth-first search can be implemented in  $O(k \text{ polylog } n)$  time. Combined  
353 with the  $O(\sqrt{k})$  bound on the number of iterations we just proved, the procedure REFINE can be  
354 implemented in  $O((n + k\sqrt{k}) \text{ polylog } n)$  time. Together with our analysis on scale initialization and  
355 the bound on number of cost scales, this concludes the proof to Theorem 1.2.

## 356 5 Fast Implementation

357 Both Hungarian search and depth-first search are implemented in a Dijkstra-like fashion, traversing  
358 through the residual graph using admissible arcs starting from the excess vertices. Each step of  
359 the search procedures *relaxes* a minimum-reduced-cost arc from the set of visited vertices to an  
360 unvisited vertex, until a deficit vertex is reached. At a high level, our analysis strategy is to charge  
361 the relaxation events to the support arcs of  $f$ , which has size at most  $O(k)$  by Corollary 4.6.

## 5.1 Null vertices and shortcut graph

«A figure might be helpful for this section.»

As it turns out, there are some vertices visited by a relaxation event which we cannot charge to  $\text{supp}(f)$ . Unfortunately the number of such vertices can be as large as  $\Omega(n)$  (consider the residual graph under the zero flow). To overcome this issue, we replace the residual graph with an equivalent graph that excludes all the null vertices, and run the Hungarian search and depth-first search on the resulting graph instead.

**Null vertices.** We say a vertex  $v$  in the residual graph  $N_f$  is a **null vertex** if  $\phi_f(v) = 0$  and no arcs of  $\text{supp}(f)$  is incident to  $v$ . We use  $A_\emptyset$  and  $B_\emptyset$  to denote the null vertices  $A$  and  $B$  respectively. Vertices that are not null are called **normal vertices**. A **null 2-path** is a length-2 subpath in  $N_f$  from a normal vertex to another normal vertex, passing through a null vertex. As every vertex in  $A$  has in-degree 1 and every vertex in  $B$  has out-degree 1 in the residual graph, the null 2-paths must be of the form either  $(s, v, b)$  for some vertex  $b$  in  $B \setminus B_\emptyset$  or  $(a, v, t)$  for some vertex  $a$  in  $A \setminus A_\emptyset$ . In either case, we say that the null 2-path **passes through** null vertex  $v$ . Similarly, we define the length-3 path from  $s$  to  $t$  that passes through two null vertices to be a **null 3-path**. Because reduced costs telescope for residual paths, the reduced cost of any null 2-path or null 3-path does not depend on the null vertices it passes through.

**Shortcut graph.** We construct the **shortcut graph**  $\tilde{H}_f$  from the reduction network  $H$  by removing all null vertices and their incident edges, followed by inserting an arc from the head of each each null path  $\Pi$  to its tail, with cost equals to the sum of costs on the arcs. We call this arc the **shortcut** of null path  $\Pi$ , denoted as **short**( $\Pi$ ). The resulting multigraph  $\tilde{H}_f$  contains only normal vertices of  $H_f$ , and the reduced cost of any path between normal vertices are preserved. We argue now that  $\tilde{H}_f$  is fine as a surrogate for  $H_f$ . Let  $\tilde{\pi}$  be an  $\varepsilon$ -optimal potential on  $\tilde{H}_f$ . Construct potentials  $\pi$  on  $H_f$  which extends  $\tilde{\pi}$  to null vertices, by setting  $\pi(a) := \tilde{\pi}(s)$  for  $a \in A_\emptyset$  and  $\pi(b) := \tilde{\pi}(t)$  for  $b \in B_\emptyset$ .

► **Lemma 5.1.** Consider  $\tilde{\pi}$  an  $\varepsilon$ -optimal potential on  $\tilde{H}_f$  and  $\pi$  the corresponding potential constructed on  $H_f$ . Then,

1. potential  $\pi$  is  $\varepsilon$ -optimal on  $H_f$ , and
2. if arc  $\text{short}(\Pi)$  is admissible under  $\tilde{\pi}$ , then every arc in  $\Pi$  is admissible under  $\pi$ .

## 5.2 Dynamic data structures for search procedures

**Hungarian search.** «Shortly describe the Hungarian search and depth-first search implementations.»

Conceptually, we are executing the Hungarian search on the shortcut graph  $\tilde{H}_f$ . We describe how we can query the minimum-reduced cost arc leaving  $\tilde{S}$  in  $O(\text{polylog } n)$  time for the shortcut graph, without constructing  $\tilde{H}_f$  explicitly. For this purpose, let  $\mathcal{S}$  be a set of “reached” vertices maintained, identical to  $\tilde{S}$  except whenever a shortcut is relaxed, we add the null vertices passed by the corresponding null path to  $\mathcal{S}$  in addition to its (normal) endpoints. Observe that the arcs of  $\tilde{H}_f$  leaving  $\tilde{S}$  fall into  $O(1)$  categories.

By construction, the distance returned by each of the BCP data structure is equal to the reduced cost of the shortcut, which is equal to the reduced cost of the corresponding null path. Each of the above data structures requires one query per relaxation, and an update operation whenever a new vertex moves into  $\tilde{S}$ . The data structures above can perform both queries and updates in  $O(\text{polylog } n)$  time each, so the running time of the Hungarian search other than the potential updates can be charged to the number of relaxation steps.

**Depth-first search.** The depth-first search is similar to Hungarian search in that it uses the relaxation of minimum-reduced-cost arcs/null paths, this time to identify admissible arcs/null paths in a depth-first manner. This requires some adjustments to the data structures for finding the minimum-reduced-cost arc leaving  $v' \in \tilde{S}$ .

Each data structure performs a constant number of queries and updates per relaxation, each of which can be implemented in  $O(\text{polylog } n)$  time []; so the running time is again bounded by  $O(\text{polylog } n)$  times the number of relaxations.

### 5.3 Time analysis

First we bound the number of relaxations performed by both the Hungarian search and the depth-first search.

► **Lemma 5.2.** *Both Hungarian search and depth-first search performs  $O(k)$  relaxations before a deficit vertex is reached.*

Now we complete the time analysis by showing that each Hungarian search and depth-first search can be implemented in  $O(k \text{ polylog } n)$  time after a one-time  $O(n \text{ polylog } n)$ -time preprocessing.

► **Lemma 5.3.** *After  $O(n \text{ polylog } n)$ -time preprocessing, each Hungarian search and depth-first search can be implemented in  $O(k \text{ polylog } n)$  time.*

### 5.4 Number of potential updates on null vertices

In our implementation of REFINE, we do not explicitly construct  $\tilde{H}_f$ ; instead we query its edges using BCP/NN oracles and min/max heaps on elements of  $H_f$ . Potentials on the null vertices are only required right before an augmentation sends a flow through a null path, making the null vertices it passes normal. We use the construction from Lemma 5.1 to obtain potential  $\pi$  on  $H_f$  such that the flow  $f$  is both  $\varepsilon$ -optimal and admissible with respect to  $\pi$ .

**Size of blocking flows.** Now we bound the total number arcs whose flow is updated by a blocking flow during the course of REFINE. This bounds both the time spent updating the flow on these arcs and also the time spent on null vertex potential updates (Lemma 5.5).

► **Lemma 5.4.** *The support of each blocking flow found in REFINE is of size  $O(k)$ .*

► **Lemma 5.5.** *The number of end-of-REFINE null vertex potential updates is  $O(n)$ . The number of augmentation-induced null vertex potential updates in each invocation of REFINE is  $O(k \log k)$ .*

Now combining Lemma B.3, Lemma B.4, and Lemma 5.5 completes the proof of Theorem 1.2.

## 6 Unbalanced transportation

In this section, we give an exact algorithm which solves the transportation problem in  $O(rn(r + \sqrt{n}) \text{ polylog } n)$  time, proving Theorem 1.3. This algorithm is a geometric implementation of the uncapacitated min-cost flow algorithm due to Orlin [8], combined with some of the tools developed in Sections 3 and 4. Mainly, we batch potential updates and use the rewinding mechanism to initialize each Hungarian search in time proportional to the previous Hungarian search.

Let  $A$  and  $B$  be points in the plane with  $r := |A|$  and  $n := |B|$ . Let  $\lambda : A \cup B \rightarrow \mathbb{Z}$  be a **supply-demand function** with positive value on points of  $A$ , negative value on points of  $B$ , and  $\sum_{a \in A} \lambda(a) = -\sum_{b \in B} \lambda(b)$ . We use  $U := \max_{p \in A \cup B} |\lambda(p)|$ . A **transportation map** is a function  $\tau : A \times B \rightarrow \mathbb{R}_{\geq 0}$ . A transportation map  $\tau$  is **feasible** if  $\sum_{b \in B} \tau(a, b) = \lambda(a)$  for all  $a \in A$ , and  $\sum_{a \in A} \tau(a, b) = -\lambda(b)$  for all  $b \in B$ . In other words, the value  $\tau(a, b)$  describes how much supply at  $a$  should be sent to meet demands at  $b$ , and we require that all supplies are sent and all demands are met. We define the cost of  $\tau$  to be

$$\text{cost}(\tau) := \sum_{(a,b) \in A \times B} \|a - b\|_p^q \cdot \tau(a, b).$$

Given  $A$ ,  $B$ , and  $\lambda$ , the **transportation problem** asks to find a feasible transportation map of minimum cost. We focus on analyzing the **unbalanced** setting where  $r \leq n$ .

There is a simple reduction from the transportation problem to uncapacitated min-cost flow. Consider the complete bipartite graph  $G$  between  $A$  and  $B$  (with all edges directed  $A$ -to- $B$ ), set the costs  $c(a, b) = \|a - b\|_p^q$ , all capacities to infinity, and use  $\phi = \lambda$ . Any circulation  $f$  in the network  $N = (G, c, u, \phi)$  can be converted into a feasible transportation map  $\tau_f$  by taking  $\tau_f(a, b) := f(a \rightarrow b)$ . Furthermore,  $\text{cost}(f) = \text{cost}(\tau_f)$ .

## 6.1 Uncapacitated MCF by excess scaling

We give an outline of the strongly polynomial algorithm for uncapacitated MCF from Orlin [8]. Orlin's algorithm follows an **excess-scaling** paradigm originally due to Edmonds and Karp [2]. Consider the basic primal-dual skeleton used in the previous sections: The algorithm begins with  $f = 0$ ,  $\pi = 0$ , then repeatedly runs a *Hungarian search* that raises potentials (while maintaining dual feasibility) to create an admissible augmenting excess-deficit path, on which it then augments flow. If supplies/demands are integral and each augmentation is at least one unit of flow, then such an algorithm terminates. In terms of cost,  $f$  is maintained to be 0-optimal with respect to  $\pi$  and augmentations over admissible edges preserve this by Lemma 4.1. Thus, the final circulation must be optimal. The excess-scaling paradigm tunes this skeleton by specifying (i) between which excess and deficit we augment, and (ii) how much flow is sent by the augmentation.

The excess-scaling algorithm maintains a **scale parameter**  $\Delta$ , initially  $\Delta = U$ . Let vertices with  $|\phi_f(v)| \geq \Delta$  be **active**. Each augmenting path is chosen between an active excess vertex and an active deficit vertex. Once there are either no more active excess or no more active deficit vertices,  $\Delta$  is halved. Each sequence of augmentations where  $\Delta$  holds a constant value is called an **excess scale**. There are  $O(\log U)$  excess scales before  $\Delta < 1$  and, by integrality of supplies/demands,  $f$  is a circulation.

With some modifications to the excess-scaling algorithm, Orlin [8] obtains an algorithm with a strongly polynomial bound on the number of augmentations and excess scales. First, an **active** vertex is redefined to be one where  $|\phi_f(v)| \geq \alpha\Delta$ , for a parameter  $\alpha \in (1/2, 1)$ . Second, arcs which have  $f(v \rightarrow w) \geq 3n\Delta$  at the beginning of a scale are **contracted**, creating a new vertex  $\hat{v}$  which inherits all the arcs of  $v$  and  $w$ , and has  $\phi(\hat{v}) \leftarrow \phi(v) + \phi(w)$ . We use  $\hat{G} = (\hat{V}, \hat{E})$  to denote the resulting **contracted graph**, where each  $\hat{v} \in \hat{V}$  is a contracted component of vertices from  $V$ . Intuitively, the flow is so high on contracted arcs that no set of future augmentations can remove the arc from  $\text{supp}(f)$ . Third,  $\Delta$  is lowered to  $\max_{v \in V} \phi_f(v)$  if there are no active excess vertices, and  $f(v, w) = 0$  on all non-contracted arcs  $(v, w) \in \hat{E}$ . Finally, flow values are not tracked within contracted components, but once an optimal circulation is found on  $\hat{G}$ , optimal potentials  $\pi^*$  can be **recovered** for  $G$  in linear time by sequentially undoing contractions. The algorithm performs a post-processing step which finds the optimal circulation  $f^*$  on  $G$  by solving a max-flow problem on the set of admissible arcs under  $\pi^*$ .

► **Theorem 6.1 Orlin [8], Theorems 2 and 3.** *Orlin's algorithm finds optimal potentials after  $O(n \log n)$  scaling phases, and  $O(n \log n)$  total augmentations.*

The remainder of the section focuses on showing that each augmentation can be implemented in  $O(r(r/\sqrt{n} + \sqrt{n} \text{polylog } n))$  time (after preprocessing). Additionally, we show that  $f^*$  can be recovered from  $\pi^*$  very quickly for our specific graph.

**Implementing contractions.** Following Agarwal *et al.* [1], our geometric data structures must deal with real points  $(A, B)$ , rather than the contracted components  $(\hat{V})$ . We will track the contracted components described in  $\hat{G}$  (e.g. with a disjoint-set data structure) and mark the arcs of  $\text{supp}(f)$  that get contracted. We maintain potentials on the points  $(A \cup B)$  directly, instead of the contracted components  $(\hat{V})$ .

When conducting the Hungarian search, we initialize  $S$  with all vertices from **active excess contracted components** who (in sum) meet the imbalance criteria. Upon relaxing any  $v \in \hat{v}$ , we immediately relax all the contracted  $\text{supp}(f)$  arcs which span  $\hat{v}$ . Since the input network is uncapacitated, each contracted component is strongly connected in the residual network by the admissible forward/reverse arcs of each contracted arc. For relaxing arcs of  $\hat{E}$ , we relax support arcs before attempting to relax any non-support arcs. Relaxations of support arcs can be performed without further potential changes, since they are admissible by invariant.

During augmentations, contracted residual arcs are considered to have infinite capacity, and we do not update the value of flow on these arcs. We allow augmenting paths to begin from any  $a \in \hat{v} \cap A$  of an active excess component  $\hat{v}$ , and end in any  $b \in \hat{w} \cap B$  of an active deficit component  $\hat{w}$ .

**Recovering optimal flow.** **«Summarize the results.»** Naively recovering an optimal flow requires  $O(rn^2)$  time. Use a strategy from Agarwal *et al.* [1], we can recover the optimal flow in time  $O(n \text{polylog } n)$ . If furthermore the cost function is the  $p$ -norm (without the  $q$ -power), an even stronger result stands: In this case, the set of admissible arcs under an optimal potential forms a planar graph, and thus we can apply the planar maximum-flow algorithm [?, 3] which runs in  $O(n \log n)$  time.

## 6.2 Dead vertices and support stars

Given Theorem 6.1, our goal is to implement each augmentation in  $O(r(r/\sqrt{n} + n^{1/2}) \text{polylog } n)$  time. To find an augmenting path, we again use a Hungarian search with geometric data structures to perform relaxations quickly. Like in Section 4, there are vertices which cannot be charged to the flow support. Even worse, the flow support may have size  $\Omega(n)$  (consider an instance with  $r = 1$ , and demand uniformly distributed among the vertices of  $B$ ). Our strategy is summarized as follows:

- Discard vertices which lead to dead ends in the search (are not on a path to a deficit vertex).
- Cluster parts of the flow support, such that the number of support arcs outside clusters is  $O(r)$ . The number of relaxations we perform is proportional to the number of support arcs outside of clusters.
- Querying/updating clusters degrades our amortized time per relaxation from  $O(\text{polylog } n)$  to  $O((r/\sqrt{n} + \sqrt{n}) \text{polylog } n)$ .

Let  $E(\text{supp}(f)) := \{(v, w) \mid v \rightarrow w \in \text{supp}(f)\}$  be the set of undirected edges corresponding to the arcs in  $\text{supp}(f)$ . Clearly,  $|\text{supp}(f)| = |E(\text{supp}(f))|$ . Let the **support degree** of a vertex be its degree in  $E(\text{supp}(f))$ .

**Dead vertices.** We call a vertex  $b \in B$  *dead* if it has support degree 0 and is not an active excess or deficit vertex; call it *live* otherwise. Dead vertices are essentially equivalent to the *null vertices* of Section 4. Since the reduction in this section does not use a super-source/super-sink, we can simply remove these from consideration during a Hungarian search — they will not terminate the search, and have no outgoing residual arcs. Like null vertices, we ignore dead vertex potentials and infer feasible potentials when they become live again. We use  $A_\ell$  and  $B_\ell$  to denote the living vertices of points in  $A$  and  $B$ , respectively. Note that being dead/alive is a notion strictly for vertices, and not contracted components.

We say a dead vertex is *revived* when it stops meeting either condition of the definition. Dead vertices are only revived after  $\Delta$  decreases (say, in a subsequent excess scale) as no augmenting path will cross a dead vertex and they cannot meet the criteria for contractions. When a dead vertex is revived, we must add it back into each of our data structures and give it a feasible potential. For revived  $b \in B$ , a feasible choice of potential is  $\pi(b) \leftarrow \max_{a \in A} \pi(a) - c(a, b)$  which we can query by maintaining a weighted nearest neighbor data structure on the points of  $A$ . The total number of revivals is bounded above by the number of augmentations: since the final flow is a circulation on  $\hat{G}$  and a newly revived vertex  $v$  has no adjacent  $\text{supp}(f)$  arcs and cannot be contracted, there is at least one subsequent augmentation which uses  $v$  as its beginning or end. Thus, the total number of revivals is  $O(n \log n)$ .

**Support stars.** The vertices of  $B$  with support degree 1 are partitioned into subsets  $\Sigma_a \subset B$  by the  $a \in A$  lying on the other end of their single support arc. We call  $\Sigma_a$  the *support star* centered around  $a \in A$ .

Roughly speaking, we would like to handle each support star as a single unit. When the Hungarian search reaches  $a$  or any  $b \in \Sigma_a$ , then the entirety of  $\Sigma_a$  (as well as  $a$ ) is also admissible-reachable and can be included into  $S$  without further potential updates. Additionally, the only outgoing residual arcs of every  $b \in \Sigma_a$  lead to  $a$ , the only way to leave  $\Sigma_a \cup \{a\}$  is through an arc leaving  $a$ . Once a relaxation step reaches some  $b \in \Sigma_a$  or  $a$  itself, we would like to quickly update the state such that the rest of  $b \in \Sigma_a$  is also reached without performing relaxation steps to each individual  $b \in \Sigma_a$ .

### 6.3 Implementation details

Before describing our workaround for support stars, we analyze the number of relaxation steps for arcs outside of support stars.

► **Lemma 6.2.** *Suppose we have stripped the graph of dead vertices. The number of relaxation steps in a Hungarian search outside of support stars is  $O(r)$ .*

The running time of a Hungarian search will be  $O(r)$  times the time it takes us to implement each relaxation.

**Relaxations outside support stars.** For relaxations that don't involve support star vertices, we can once again maintain a BCP to query the minimum  $A_\ell$ -to- $B_\ell$  arc. To elaborate, this is the BCP between  $P = A_\ell \cap S$  and  $Q = (B_\ell \setminus (\bigcup_{a \in A_\ell} \Sigma_a)) \setminus S$ , weighted by potentials. This can be queried in  $O(\log n)$  time and updated in  $O(\text{polylog } n)$  time per point. Since it doesn't deal with support stars, there is at most one insertion/deletion per relaxation step.

For  $B_\ell$ -to- $A_\ell$ , backward (support) arcs are kept admissible by invariant, so we relax them immediately when they arrive on the frontier.



**Relaxing a support star.** We classify support stars into two categories: *big stars* are those with  $|\Sigma_a| > \sqrt{n}$ , and *small stars* are those with  $|\Sigma_a| \leq \sqrt{n}$ . Let  $A_{\text{big}} \subseteq A$  denote the centers of big stars and  $A_{\text{small}} \subseteq A$  denote the centers of small stars. We keep the following data structures to manage support stars.

1. For each big star  $\Sigma_a$ , we use a data structure  $\mathcal{D}_{\text{big}}(a)$  to maintain BCP between  $P = A_\ell \cap S$  and  $Q = \Sigma_a$ , weighted by potentials. We query this until  $a \in S$  or any vertex of  $\Sigma_a$  is added to  $S$ .
2. All small stars are added to a single BCP data structure  $\mathcal{D}_{\text{small}}$  between  $P = A_\ell \cap S$  and  $Q = (\bigcup_{a \in A_{\text{small}}} \Sigma_a) \setminus S$ , weighted by potentials. When an  $a \in A_{\text{small}}$  or any vertex of its support star is added to  $S$ , we remove the points of  $\Sigma_a$  from  $\mathcal{D}_{\text{small}}$  using  $|\Sigma_a|$  deletion operations.

We will update these data structures as each support star center is added into  $S$ . If a relaxation step adds some  $b \in B_\ell$  and  $b$  is in a support star  $\Sigma_a$ , then we immediately relax  $b \rightarrow a$ , as all support arcs are admissible. Relaxations of non-support star  $b \in B_\ell$  will not affect the support star data structures.

Suppose a relaxation step adds some  $a \in A_\ell$  to  $S$ . For the support star data structures, we must (i) remove  $a$  from every  $\mathcal{D}_{\text{big}}$ , (ii) remove  $a$  from  $\mathcal{D}_{\text{small}}$ . If  $a \in A_{\text{big}}$ , we also (iii) deactivate  $\mathcal{D}_{\text{big}}(a)$ . If  $a \in A_{\text{small}}$ , we also (iv) remove the points of  $\Sigma_a$  from  $\mathcal{D}_{\text{small}}$ . The operations (i–iii) can be performed in  $O(\text{polylog } n)$  time each, but (iv) may take up to  $O(\sqrt{n} \text{polylog } n)$  time.

On the other hand, there are now  $O(\sqrt{n})$  data structures to query during each relaxation step, as there are  $O(n/\sqrt{n})$  data structures  $\mathcal{D}_{\text{big}}(\cdot)$ . Thus, the query time within each relaxation step is  $O(\sqrt{n} \log n)$ . We can now bound the time spent within the Hungarian search.

► **Lemma 6.3.** *Hungarian search takes  $O(r\sqrt{n} \text{polylog } n)$  time.*

**Updating support stars.** As the flow support changes, the membership of support stars may shift and a big star may eventually become small (or vice versa). To efficiently support this, introduce some fuzziness to when a star should be big or small. Standard charging argument shows that the amortized update time is  $O(r\sqrt{n}(r + \sqrt{n}) \text{polylog } n)$ .

Membership of support stars can only be changed by augmentations, so the number of star membership changes by a single augmenting path is bounded above by twice its length (each vertex may be removed from one star, and/or added to another star). Thus, individual membership changes can be performed in  $O(\text{polylog } n)$  time each, and there are  $O(rn \log n)$  total.

**Preprocessing time.** To build the very first set of data structures, we take  $O(rn \text{polylog } n)$  time. There are  $r|\Sigma_a|$  points in each  $\mathcal{D}_{\text{big}}(a)$ , but the  $\Sigma_a$  are disjoint, so the total points to insert is  $O(rn)$ .  $\mathcal{D}_{\text{small}}$  also has at most  $O(rn)$  points. Each BCP data structure can be constructed in  $O(\text{polylog } n)$  times its size, so the total preprocessing time is  $O(rn \text{polylog } n)$ .

**Between searches.** After an augmentation, we reset the above data structures to their initial state plus the change from the augmentation using the rewinding mechanism. By reversing the sequence of insertions/deletions to each data structure over the course of the Hungarian search, we can recover the versions data structures as they were when the Hungarian search began. This takes time proportional to the time of the Hungarian search,  $O(r\sqrt{n} \text{polylog } n)$  by Lemma 6.3. The most recent augmentation may have deactivated at most one active excess and at most one active deficit, which we can update in the data structures in  $O(\sqrt{n} \text{polylog } n)$  time. Additionally, the augmentation may have changed the membership of some support stars, but we analyzed the time for membership changes earlier. Finally, we note that an augmenting path cannot reduce the support degree of a vertex to zero, and therefore no new dead vertices are created by augmentation.

**Between excess scales.** When the excess scale changes, vertices that were previously inactive may become active, and vertices that were dead may be revived (however, no active vertices deactivate, and no live vertices die as the result of  $\Delta$  decreasing). If we have the data structures built on the active excesses at the end of the previous scale, then we can add in each newly active  $a \in A$  and charge this insertion to the (future) augmenting path or contraction which eventually makes the vertex inactive, or absorbs it into another component. By Theorem 6.1, there are  $O(n \log n)$  such newly active vertices. The time to perform data structure updates for each of them is  $O(\sqrt{n} \text{polylog } n)$ , so the total time spent bookkeeping newly active vertices is  $O(n^{3/2} \text{polylog } n)$ .

**Putting it together.** After  $O(rn \text{polylog } n)$  preprocessing, we spend  $O(r\sqrt{n} \text{polylog } n)$  time each Hungarian search by Lemma 6.3. After each augmentation, we spend the same amount of time (plus  $O(\text{polylog } n)$  extra) to initialize data structures for the next Hungarian search. We spend up to  $O((rn + r^2\sqrt{n}) \text{polylog } n)$  total time making big-small star switching updates. We spend  $O(n^{3/2} \text{polylog } n)$  time activating and reviving vertices. Thus, the algorithm takes  $O(rn(r/\sqrt{n} + \sqrt{n}) \text{polylog } n)$  time to produce optimal potentials  $\pi^*$ , from which we can recover  $f^*$  in  $O(r\sqrt{n} \text{polylog } n)$  additional time. This completes the proof of Theorem 1.3.

**Acknowledgment.**

## References

- 1 P. K. Agarwal, K. Fox, D. Panigrahi, K. R. Varadarajan, and A. Xiao. Faster algorithms for the geometric transportation problem. In *33rd International Symposium on Computational Geometry, SoCG 2017, July 4-7, 2017, Brisbane, Australia*, pages 7:1–7:16, 2017.
- 2 J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264, 1972.
- 3 J. Erickson. Maximum flows and parametric shortest paths in planar graphs. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 794–804, 2010.
- 4 A. V. Goldberg, S. Hed, H. Kaplan, and R. E. Tarjan. Minimum-cost flows in unit-capacity networks. *Theory Comput. Syst.*, 61(4):987–1010, 2017.
- 5 A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by successive approximation. *Math. Oper. Res.*, 15(3):430–466, 1990.
- 6 H. Kaplan, W. Mulzer, L. Roditty, P. Seiferth, and M. Sharir. Dynamic planar voronoi diagrams for general distance functions and their algorithmic applications. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2495–2504, 2017.
- 7 H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics (NRL)*, 2(1-2):83–97, 1955.
- 8 J. B. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Operations Research*, 41(2):338–350, 1993.
- 9 R. Sharathkumar and P. K. Agarwal. Algorithms for the transportation problem in geometric settings. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 306–317, 2012.
- 10 P. M. Vaidya. Geometry helps in matching. *SIAM J. Comput.*, 18(6):1201–1225, 1989.

## A

 Proofs from Section 4

► **Lemma 4.1.** *Let  $f$  be an  $\varepsilon$ -optimal pseudoflow in  $G$  and let  $f'$  be an admissible flow in  $G_f$ . Then  $f + f'$  is also  $\varepsilon$ -optimal. («Lemma 5.3 in [GT90]? Also, where is this used?»)*

**Proof.** Augmentation by  $f'$  will not change the potentials, so any previously  $\varepsilon$ -optimal arcs remain  $\varepsilon$ -optimal. However, it may introduce new arcs  $v \rightarrow w$  with  $u_{f+f'}(v \rightarrow w) > 0$ , that previously had  $u_f(v \rightarrow w) = 0$ . We will verify that these arcs satisfy the  $\varepsilon$ -optimality condition.

If an arc  $v \rightarrow w$  is newly introduced this way, then by definition of residual capacities  $f(v \rightarrow w) = u(v \rightarrow w)$ . At the same time,  $u_{f+f'}(v \rightarrow w) > 0$  implies that  $(f + f')(v \rightarrow w) < u(v \rightarrow w)$ . This means that  $f'$  augmented flow in the reverse direction of  $v \rightarrow w$  ( $f'(w \rightarrow v) > 0$ ). By assumption, the arcs of  $\text{supp}(f')$  are admissible, so  $w \rightarrow v$  was an admissible arc ( $c_\pi(w \rightarrow v) \leq 0$ ). By antisymmetry of reduced costs, this implies  $c_\pi(v \rightarrow w) \geq 0 \geq -\varepsilon$ . Therefore, all arcs with  $u_{f+f'}(v, w) > 0$  respect the  $\varepsilon$ -optimality condition, and thus  $f + f'$  is  $\varepsilon$ -optimal. ◀

► **Lemma 4.3.** *Let  $f$  be an integer circulation in the reduction network  $N_H$ . Then, the size of the support of  $f$  is at most  $3k$ . As a corollary, the number of residual backward arcs is at most  $3k$ .*

**Proof.** Because  $f$  is a circulation,  $\text{supp}(f)$  can be decomposed into  $k$  paths from  $s$  to  $t$ . Each  $s$ -to- $t$  path in  $N_H$  is of length three, so the size of  $\text{supp}(f)$  is at most  $3k$ . As every backward arc in the residual network must be induced by positive flow in the opposite direction, the total number of residual backward arcs is at most  $3k$ . ◀

► **Lemma 4.4.** *Let  $f$  be an  $\varepsilon$ -optimal integer circulation in  $N_H$ , and  $f^*$  be an optimal integer circulation for  $N_H$ . Then,  $\text{cost}(f) \leq \text{cost}(f^*) + 6k\varepsilon$ .*

**Proof.** By Lemma 4.3, the total number of backward arcs in the residual network  $N_f$  is at most  $3k$ . Consider the residual flow in  $N_f$  defined by the difference between  $f^*$  and  $f$ . Since both  $f$  and  $f^*$  are both circulations and  $N_H$  has unit-capacity, the flow  $f - f^*$  is comprised of unit flows on a collection of edge-disjoint residual cycles  $\Gamma_1, \dots, \Gamma_\ell$ . Observe that each residual cycle  $\Gamma_i$  must have exactly half of its arcs being backward arcs, and thus we have  $\sum_i |\Gamma_i| \leq 6k$ .

Let  $\pi$  be some potential certifying that  $f$  is  $\varepsilon$ -optimal. Because  $\Gamma_i$  is a residual cycle, we have  $c_\pi(\Gamma_i) = c(\Gamma_i)$  since the potential terms telescope. We then see that

$$\text{cost}(f) - \text{cost}(f^*) = \sum_i c(\Gamma_i) = \sum_i c_\pi(\Gamma_i) \geq \sum_i (-\varepsilon) \cdot |\Gamma_i| \geq -6k\varepsilon,$$

where the second-to-last inequality follows from the  $\varepsilon$ -optimality of  $f$  with respect to  $\pi$ . Rearranging the terms we have that  $\text{cost}(f) \leq \text{cost}(f^*) + 6k\varepsilon$ . ◀

Let  $T$  be the minimum spanning tree on input graph  $G$  and order its edges by increasing length as  $e_1, \dots, e_{r+n-1}$ . Let  $T_\ell$  denote the subgraph of  $T$  obtained by removing the heaviest  $\ell$  edges in  $T$ . Let  $i$  be the largest index so that the optimal solution to the MPM problem has edges between components of  $T_i$ . Choose  $j$  to be the smallest index larger than  $i$  satisfying  $c(e_j) \geq kn \cdot c(e_i)$ . For each component  $K$  of  $T_j$ , let  $G_K$  be the subgraph of  $G$  induced on vertices of  $K$ ; let  $A_K := K \cap A$  and  $B_K := K \cap B$ , respectively. We partition  $A$  and  $B$  into the collection of sets  $A_K$  and  $B_K$  according to the components  $K$  of  $T_j$ . Since  $j < i$ , the optimal partial matching in  $G$  can be partitioned into edges between  $A_K$  and  $B_K$  within  $G_K$ ; no optimal matching edges lie between components.

► **Lemma A.1 (Sharathkumar and Agarwal [SA12, §3.5]).** *Let  $G = (A, B, E_0)$  be the input to MPM problem, and consider the partitions  $A_K$  and  $B_K$  defined as above. Let  $M^*$  be the optimal partial matching in  $G$ . Then,*

- 701 (i)  $c(e_i) \leq \text{cost}(M^*) \leq kn \cdot c(e_i)$ , and  
 702 (ii) the diameter of  $G_K$  is at most  $kn^2 \cdot c(e_i)$  for every  $K \in T_j$ ,

703 To prove Lemma 4.2, we need to further modify the point set so that the cost of the optimal  
 704 solution does not change, while the diameter of the *whole* point set is bounded. Move the points  
 705 within each component in *translation* so that the minimum distances between points across  
 706 components are at least  $kn \cdot c(e_i)$  but at most  $O(n \cdot kn^2 \cdot c(e_i))$ . This will guarantee that the  
 707 optimal solution still uses edges within the components by Lemma A.1. The simplest way of  
 708 achieving this is by aligning the components one by one into a “straight line”, so that the distance  
 709 between the two farthest components is at most  $O(n)$  times the maximum diameter of the cluster.

710 Now one can prove Lemma 4.2 by computing an  $(\varepsilon c(e_i)/6k)$ -optimal circulation  $f$  on the  
 711 point set after translations using additive approximation from Lemma 4.4, together with the  
 712 bound  $c(e_i) \leq \text{cost}(M^*)$  from Lemma A.1.

713 One small problem remains: We need to show that such reduction can be performed in  
 714  $O(n \text{ polylog } n)$  time. Sharathkumar and Agarwal [SA12] have shown that the partition of  $A$  and  
 715  $B$  into  $A_K$ s and  $B_K$ s can be computed in  $O(n \text{ polylog } n)$  time, assuming that the indices  $i$  and  $j$  can  
 716 be determined in such time as well. However in our application the choice of index  $i$  depends on  
 717 the optimal solution of MPM problem which we do not know.

718 To solve this issue we perform a binary search on the edges  $e_1, \dots, e_{r+n-1}$ . **«Hmm, we have  
 719 no way to check Lemma 4.5(i); but in fact a polynomial bound is good enough.»**  
 720 **«UNRESOLVED ISSUE»**

721 ► **Lemma 4.5.** *Let  $f$  be an integer pseudoflow in  $N_H$  with  $O(k)$  excess. Then, the size of the support  
 722 of  $f$  is at most  $O(k)$ .*

723 **Proof.** Observe that the reduction graph  $H$  is a directed acyclic graph, and thus the support of  $f$   
 724 does not contain a cycle. Now  $\text{supp}(f)$  can be decomposed into a set of inclusion-maximal paths,  
 725 each of which contributes a single unit of excess to the flow if the path does not terminate at  $t$   
 726 or if more than  $k$  paths terminate at  $t$ . By assumption, there are  $O(k)$  units of excess to which  
 727 we can associate to the paths, and at most  $k$  paths (those that terminate at  $t$ ) that we cannot  
 728 associate with a unit of excess. The length of any such paths is at most three by construction of  
 729 the reduction graph  $H$ . Therefore we can conclude that the number of arcs in the support of  $f$  is  
 730  $O(k)$ . ◀

731 ► **Lemma 4.7.** *Let  $f$  be a pseudoflow in  $N_H$  with  $O(k)$  excess. The procedure REFINE runs for  
 732  $O(\sqrt{k})$  iterations before the excess of  $f$  becomes zero.*

733 **Proof.** Let  $f_0$  and  $\pi_0$  be the flow and potential at the start of the procedure REFINE. Let  $f$  and  $\pi$   
 734 be the current flow and the potential. Let  $d(v)$  defined to be the amount of potential increase at  
 735  $v$ , measured in units of  $\varepsilon$ ; in other words,  $d(v) := (\pi(v) - \pi_0(v))/\varepsilon$ .

736 Now divide the iterations executed by the procedure REFINE into two phases: The transition  
 737 from the first phase to the second happens when every excess vertex  $v$  has  $d(v) \geq \sqrt{k}$ . At most  
 738  $\sqrt{k}$  iterations belong to the first phase as each Hungarian search increases the potential  $\pi$  by at  
 739 least  $\varepsilon$  for each excess vertex (and thus increases  $d(v)$  by at least one).

740 The number of iterations belonging to the second phase is upper bounded by the amount  
 741 of total excess at the end of the first phase, because each subsequent push of a blocking flow  
 742 reduces the total excess by at least one. We now show that the amount of such excess is at most  
 743  $O(\sqrt{k})$ . Consider the set of arcs  $E^+ := \{v \rightarrow w \mid f(v \rightarrow w) < f_0(v \rightarrow w)\}$ . The total amount of excess is  
 744 upper bounded by the number of arcs in  $E^+$  that crosses an arbitrarily given cut  $X$  that separates  
 745 the excess vertices from the deficit vertices, when the network has unit-capacity [GHKT17,  
 746 Lemma 3.6]. Consider the set of cuts  $X_i := \{v \mid d(v) > i\}$  for  $0 \leq i < \sqrt{k}$ ; every such cut separates

the excess vertices from the deficit vertices at the end of first phase. Each arc in  $E^+$  crosses at most 3 cuts of type  $X_i$  [GHKT17, Lemma 3.1]. So there is one  $X_i$  crossed by at most  $3|E^+|/\sqrt{k}$  arcs in  $E^+$ . The size of  $E^+$  is bounded by the sum of support sizes of  $f$  and  $f_0$ ; by Corollary 4.6 the size of  $E^+$  is  $O(k)$ . This implies an  $O(\sqrt{k})$  bound on the total excess after the first phase, which in turn bounds the number of iterations in the second phase.  $\blacktriangleleft$

## B Proofs from Section 5

**Lemma 5.1.** Consider  $\tilde{\pi}$  an  $\varepsilon$ -optimal potential on  $\tilde{H}_f$  and  $\pi$  the corresponding potential constructed on  $H_f$ . Then,

1. potential  $\pi$  is  $\varepsilon$ -optimal on  $H_f$ , and
2. if arc  $\text{short}(\Pi)$  is admissible under  $\tilde{\pi}$ , then every arc in  $\Pi$  is admissible under  $\pi$ .

**Proof.** Reduced costs for any arc from a normal vertex another is unchanged under either  $\tilde{\pi}$  or  $\pi$ . Recall that a null path is comprised of one  $A$ -to- $B$  arc, and one or two zero-cost arcs (connecting the null vertex/vertices to  $s$  and/or  $t$ ). With our choice of null vertex potentials, we observe that the zero-cost arcs still have zero reduced cost. It remains to prove that an arbitrary **«residual?»** arc  $(a, b)$  **«arc or directed edge?»** satisfies the  $\varepsilon$ -optimality condition and admissibility when either  $a$  or  $b$  is a null vertex.

By construction of the shortcut graph, there is always a null path  $\Pi$  that contains  $(a, b)$ . Observe that  $c_\pi(a, b) = c_\pi(\Pi)$ , independent to the type of null path. Again by construction,  $c_\pi(\Pi) = c_{\tilde{\pi}}(\text{short}(\Pi))$ , so we have  $c_\pi(a, b) = c_{\tilde{\pi}}(\text{short}(\Pi)) \geq -\varepsilon$ . Additionally, if  $\text{short}(\Pi)$  is admissible under  $\tilde{\pi}$ , then so is  $(a, b)$  under  $\pi$ . This proves the lemma.  $\blacktriangleleft$

1. Non-shortcut backward arcs  $(v, w)$  with  $(w, v) \in \text{supp}(f)$ . For these, we can maintain a min-heap on  $\text{supp}(f)$  arcs as each  $v$  arrives in  $\tilde{S}$ .
2. Non-shortcut  $A$ -to- $B$  forward arcs. For these, we can use a BCP data structure between  $(A \setminus A_\emptyset) \cap \tilde{S}$  and  $(B \setminus B_\emptyset) \setminus \tilde{S}$ , weighted by potential.
3. Non-shortcut forward arcs from  $s$ -to- $A$  and from  $B$ -to- $t$ . For  $s$ , we can maintain a min-heap on the potentials of  $B \setminus \tilde{S}$ , queried while  $s \in \tilde{S}$ . For  $t$ , we can maintain a max-heap on the potentials of  $A \cap \tilde{S}$ , queried while  $t \notin \tilde{S}$ .
4. Shortcut arcs  $(s, b)$  corresponding to null 2-paths from  $s$  to  $b \in (B \setminus B_\emptyset) \setminus S$ . For these, we maintain a BCP data structure with  $P = A_\emptyset$ ,  $Q = (B \setminus B_\emptyset) \setminus S$  with weights  $\omega(p) = \pi(s)$  for all  $p \in P$ , and  $\omega(q) = \pi(q)$  for all  $q \in Q$ . A response  $(a, b)$  corresponds to th null 2-path  $(s, a, b)$ . This is only queried while  $s \in S$ .
5. Shortcut arcs  $(a, t)$  corresponding to null 2-paths from  $a \in (A \setminus A_\emptyset) \cap S$  to  $t$ . For these, we maintain a BCP data structure with  $P = (A \setminus A_\emptyset) \cap S$ ,  $Q = B_\emptyset \setminus S$  with weights  $\omega(p) = \pi(p)$  for all  $p \in P$ , and  $\omega(q) = \pi(t)$  for all  $q \in Q$ . A response  $(a, b)$  corresponds to th null 2-path  $(a, b, t)$ . This is only queried while  $t \notin S$ .
6. Shortcut arcs  $(s, t)$  corresponding to null 3-paths. For these, we maintain in a BCP data structure with  $P = A_\emptyset \setminus S$ ,  $Q = B_\emptyset \setminus S$  with weights  $\omega(p) = \pi(s)$  for all  $p \in P$ , and  $\omega(q) = \pi(t)$  for all  $q \in Q$ . A response  $(a, b)$  corresponds to th null 3-path  $(s, a, b, t)$ . This is only queried while  $s \in S$  and  $t \notin S$ .

Given  $v' \in \tilde{S}$ , we would like to query:

1. Non-shortcut backward arcs  $(v', w')$  with  $(w', v') \in \text{supp}(f)$ . For these, we can maintain a min-heap on  $(w', v') \in \text{supp}(f)$  arcs for each normal  $v' \in V$ .
2. Non-shortcut  $A$ -to- $B$  forward arcs. For these, we maintain a NN data structure over  $P = (B \setminus B_\emptyset) \setminus \tilde{S}$ , with weights  $\omega(p) = \pi(p)$  for each  $p \in P$ . We subtract  $\pi(v')$  from the NN distance to recover the reduced cost of the arc from  $v'$ .

- 792 3. Non-shortcut forward arcs from  $s$ -to- $A$  and from  $B$ -to- $t$ . For  $s$ , we can maintain a min-heap  
793 on the potentials of  $B \setminus \tilde{S}$ , queried only if  $v' = s$ . For  $B$ -to- $t$  arcs, there is only one arc to check  
794 if  $v' \in B$ , which we can examine manually.
- 795 4. Shortcut arcs  $(s, b)$  corresponding to null 2-paths from  $s$  to  $b \in (B \setminus B_\emptyset) \setminus S$ . For these, we  
796 maintain a BCP data structure with  $P = A_\emptyset$ ,  $Q = (B \setminus B_\emptyset) \setminus S$  with weights  $\omega(p) = \pi(s)$  for all  
797  $p \in P$ , and  $\omega(q) = \pi(q)$  for all  $q \in Q$ . A response  $(a, b)$  corresponds to th null 2-path  $(s, a, b)$ .  
798 This is only queried if  $v' = s$ .
- 799 5. Shortcut arcs  $(a, t)$  corresponding to null 2-paths from  $a \in (A \setminus A_\emptyset) \cap S$  to  $t$ . For these, we  
800 maintain a NN data structure over  $P = B_\emptyset \setminus S$  with weights  $\omega(p) = \pi(t)$  for each  $p \in P$ .  
801 A response  $(v', b)$  corresponds to th null 2-path  $(v', b, t)$ . We subtract  $\pi(v')$  from the NN  
802 distance to recover the reduced cost of the arc from  $v'$ . This is not queried if  $t \in \tilde{S}$ .
- 803 6. Shortcut arcs  $(s, t)$  corresponding to null 3-paths. For these, we maintain in a BCP data  
804 structure with  $P = A_\emptyset \setminus S$ ,  $Q = B_\emptyset \setminus S$  with weights  $\omega(p) = \pi(s)$  for all  $p \in P$ , and  $\omega(q) = \pi(t)$   
805 for all  $q \in Q$ . A response  $(a, b)$  corresponds to th null 3-path  $(s, a, b, t)$ . This is only queried  
806 while  $v' = s$  and  $t \notin S$ .

807 ► **Lemma B.1.** *Hungarian search performs  $O(k)$  relaxations before a deficit vertex is reached.*

808 **Proof.** **«TO BE REWRITTEN.»** First we prove that there are  $O(k)$  non-shortcut relaxations.  
809 Each edge relaxation adds a new vertex to  $S$ , and non-shortcut relaxations only add normal  
810 vertices. The vertices of  $V \setminus S$  fall into several categories: (i)  $s$  or  $t$ , (ii) vertices of  $A$  or  $B$  with  
811 0 imbalance, and (iii) deficit vertices of  $A$  or  $B$  ( $S$  contains all excess vertices). The number of  
812 vertices in (i) and (iii) is  $O(k)$ , leaving us to bound the number of (ii) vertices.

813 An  $A$  or  $B$  vertex with 0 imbalance must have an even number of  $\text{supp}(f)$  edges. There is  
814 either only one positive-capacity incoming arc (for  $A$ ) or outgoing arc (for  $B$ ), so this quantity  
815 is either 0 or 2. Since the vertex is normal, this must be 2. We charge 0.5 to each of the two  
816  $\text{supp}(f)$  arcs; the arcs of  $\text{supp}(f)$  have no more than 1 charge each. Thus, the number of type  
817 (ii) vertex relaxations is  $O(|\text{supp}(f)|)$ . By Corollary 4.6,  $O(|\text{supp}(f)|) = O(k)$ .

818 Next we prove that there are  $O(k)$  shortcut relaxations. Recall the categories of shortcuts from  
819 the list of datastructures above. We have shortcuts corresponding to (i) null 2-paths surrounding  
820  $a \in A_\emptyset$ , (ii) null 2-paths surrounding  $b \in B_\emptyset$ , and (iii) null 3-paths, which go from  $s$  to  $t$ . There is  
821 only one relaxation of type (iii), since  $t$  can only be added to  $S$  once. The same argument holds  
822 for type (ii).

823 Each type (i) relaxation adds some normal  $b \in B \setminus B_\emptyset$  into  $S$ . Since  $b$  is normal, it must either  
824 have deficit or an adjacent arc of  $\text{supp}(f)$ . We charge this relaxation to  $b$  if it is deficit, or the  
825 adjacent arc of  $\text{supp}(f)$  otherwise. No vertex is charged more than once, and no  $\text{supp}(f)$  edge is  
826 charged more than twice, therefore the total number of type (i) relaxations is  $O(|\text{supp}(f)|)$ . By  
827 Corollary 4.6,  $O(|\text{supp}(f)|) = O(k)$ . ◀

828 Similarly we can prove that there are  $O(k)$  relaxations during the DFS.

829 ► **Corollary B.2.** *Depth-first search performs  $O(k)$  relaxations before a deficit vertex is reached.*

830 ► **Lemma B.3.** *After  $O(n \text{ polylog } n)$ -time preprocessing, each Hungarian search can be implemented  
831 in  $O(k \text{ polylog } n)$  time.*

832 **Proof.** Each of the constant number of data structures used by the Hungarian search can be  
833 constructed in  $O(n \text{ polylog } n)$  time. For each data structure queried during a relaxation, the new  
834 vertex moved into  $S$  causes a constant number of updates, each of which can be implemented  
835 in  $O(\text{polylog } n)$  time. We first prove that the number of BCP operations during the Hungarian  
836 search over is bounded by  $O(k)$ .



- 837 1. Let  $S^t$  denote the initial set  $S$  at the beginning of the  $t$ -th Hungarian search, Assume for now  
 838 that, at the beginning of the  $(t + 1)$ -th Hungarian search, we have the set  $S^t$  from the previous  
 839 iteration. To construct  $S^{t+1}$ , we remove the vertices that had excess decreased to zero by the  
 840  $t$ -th blocking flow. Thus, we are able to initialize  $S$  at the cost of one BCP deletion per excess  
 841 vertex, which sums to  $O(k)$  over the entire course of REFINE. **«Too strong as a bound? Is  
 842 it enough to look at one Hungarian search?»**
- 843 2. During each Hungarian search, a vertex entering  $S$  may incur one BCP insertion/deletion.  
 844 We can charge the updates to the number of relaxations over the course of Hungarian search.  
 845 The number of relaxations in a Hungarian search is  $O(k)$  by Lemma B.1.
- 846 3. To obtain  $S^t$ , we keep track of the points added to  $S^t$  since the last Hungarian search. After  
 847 the augmentation, we remove those points added to  $S^t$ . By (2) there are  $O(k)$  such points to  
 848 be deleted, so reconstructing  $S^t$  takes  $O(k)$  BCP operations.

849 For potential updates, we use the same trick by Vaidya [Vai89] to lazily update potentials after  
 850 vertices leave  $S$  (similar to Lemma 3.3), but this time only for normal vertices. Normal vertices  
 851 are stored in each data structure with weight  $\omega(v) = \pi(v) - \delta$ , and  $\delta$  is increased in lieu of  
 852 increasing the potential of vertices in  $S$ . When a vertex leave  $S$  (through the rewind mechanism  
 853 above), we restore its potential as  $\pi(v) \leftarrow \omega(v) + \delta$ . With lazy updates, the number of potential  
 854 updates on normal vertices is bounded by the number of relaxations in the Hungarian search,  
 855 which is  $O(k)$  by Lemma B.1. Note that null vertex potentials are not handled in the Hungarian  
 856 search. **«then where? Lemma 5.5»** ◀

857 There are no potentials to update within DFS, so the running time of DFS boils down to the  
 858 time spent to querying and updating the data structures.

859 ► **Lemma B.4.** *After  $O(n \text{ polylog } n)$ -time preprocessing, each depth-first search can be implemented  
 860 in  $O(k \text{ polylog } n)$  time.*

861 **Proof.** At the beginning of REFINE, we can initialize the  $O(1)$  data structures used in DFS in  
 862  $O(n \text{ polylog } n)$  time. We use the same rewinding mechanism as in Hungarian search (Lemma B.3)  
 863 to avoid reconstructing the data structures across iterations of REFINE, so the total time spent  
 864 is bounded by the  $O(\text{polylog } n)$  times the number of relaxations. By Corollary B.2, the running  
 865 time for depth-first search is  $O(k \text{ polylog } n)$ . ◀

866 ► **Lemma 5.4.** *The support of each blocking flow found in REFINE is of size  $O(k)$ .*

867 **Proof.** Let  $i$  be fixed and consider the invocation of DFS which produces the  $i$ -th blocking flow  
 868  $f_i$ . DFS constructs  $f_i$  as a sequence of admissible excess-deficit paths, which appear as path  $P$   
 869 in Algorithm ???. Every arc in  $P$  is an arc relaxed by DFS, so  $N_i$  is bounded by the number of  
 870 relaxations performed in DFS. Using Corollary B.2, we have  $N_i = O(k)$ . ◀

871 ► **Lemma 5.5.** *The number of end-of-REFINE null vertex potential updates is  $O(n)$ . The number of  
 872 augmentation-induced null vertex potential updates in each invocation of REFINE is  $O(k \log k)$ .*

873 **Proof.** The number of end-of-REFINE potential updates is  $O(n)$ . Each update due to flow aug-  
 874 mentation involves a blocking flow sending positive flow through an null path, causing a potential  
 875 update on the passed null vertex. We charge this potential update to the edges of that null path,  
 876 which are in turn arcs with positive flow in the blocking flow. For each blocking flow, no positive  
 877 arc is charged more than twice. It follows that the number of augmentation-induced updates  
 878 is at most the size of support of the blocking flow, which is  $O(k)$  by Lemma 5.4. According to  
 879 Lemma 4.7 there are  $O(\sqrt{k})$  iterations of REFINE before it terminates. Summing up we have an  
 880  $O(k\sqrt{k})$  bound over the course of REFINE. ◀

## C

 Proofs from Section 6

**Recovering the optimal flow.** «use this one if we want to use exponent  $> 1$ . »  
«Move everything to appendix and left a pointer to the socg 2016 paper.»

We use a strategy from Agarwal *et al.* [AFP<sup>+</sup>17]. Instead of finding a max flow in the entire admissible network under  $\pi^*$ , we claim that is sufficient to find a max flow in a *spanning tree* of admissible arcs, e.g. a shortest path tree on reduced costs. There are some details to explain — like where the tree should be rooted, how to ensure the underlying network is strongly connected by admissible arcs — but we give the intuition first: Such a spanning tree is a maximal set of linearly independent dual LP constraints for the optimal dual ( $\pi^*$ ), so there exists an optimal primal solution ( $f^*$ ) with support only on these arcs. To see this, we can use a perturbation argument: raising the cost of each non-tree edge by  $\varepsilon > 0$  does not change  $\text{cost}(\pi^*)$  or the feasibility of  $\pi^*$ , but does raise the cost of any circulation  $f$  using non-tree edges. Strong duality suggests that  $\text{cost}(f^*) = \text{cost}(\pi^*)$  is unchanged, therefore  $f^*$  must have support only on the tree edges.

«TODO: the SPT construction requires describing Dijkstra and promising strong connectivity»

**Recovering the optimal flow.** «use this one if we only use exponent 1. »

Instead of running a generic max-flow algorithm after finding the optimal potentials, we use the following observation.

Up until now, we have not placed restrictions on coincidence between  $A$  and  $B$ , but for the next proof it is useful to do so. We can assume that all points within  $A \cup B$  are distinct, otherwise we can replace all points coincident at  $x \in \mathbb{R}^2$  with a single point whose supply/demand is  $\sum_{v \in A \cup B: v=x} \lambda(v)$ . This is roughly equivalent to transporting as much as we can between coincident supply and demand, and is optimal by triangle inequality. So without loss of generality, we assume all points of  $A \cup B$  are distinct.

Without loss of generality, assume  $\pi^*$  is nonnegative (raising  $\pi^*$  uniformly on all points does not change the objective or feasibility). Recall that feasibility of  $\pi^*$  states that, for all  $a \in A$  and  $b \in B$

$$c_{\pi^*}(a, b) = \|a - b\|_p - \pi^*(a) + \pi^*(b) \geq 0.$$

An arc  $a \rightarrow b$  is admissible when

$$c_{\pi^*}(a, b) = \|a - b\|_p - \pi^*(a) + \pi^*(b) = 0.$$

We note that these definitions have a nice visual: Place disks  $D_q$  of radius  $\pi(q)$  at each  $q \in A \cup B$ . Feasibility states that for all  $a \in A$  and  $b \in B$ ,  $D_a$  cannot contain  $D_b$  with a gap between their boundaries. The arc  $a \rightarrow b$  is admissible when  $D_a$  contains  $D_b$  and their boundaries are tangent.

► **Lemma C.1.** *Let  $\pi^*$  be a set of optimal potentials for the point sets  $A$  and  $B$ , under costs  $c(a, b) = \|a - b\|_p$ . Then, the set of admissible arcs under  $\pi^*$  form a planar graph.*

**Proof.** We assume the points of  $A \cup B$  are in general position (e.g. by symbolic perturbation) such that no three points are collinear. Let  $a_1 \rightarrow b_1$  and  $a_2 \rightarrow b_2$  be any pair of admissible arcs under  $\pi^*$ . We will isolate them from the rest of the points, considering  $\pi^*$  restricted to the four points  $\{a_1, a_2, b_1, b_2\}$ . Clearly, this does not change whether the two arcs cross. Observe that we can raise  $\pi^*(a_2)$  and  $\pi^*(b_2)$  uniformly, until  $c_{\pi^*}(a_2, b_1) = 0$ , without breaking feasibility or changing admissibility of  $a_1 \rightarrow b_1$  and  $a_2 \rightarrow b_2$ . Henceforth, we assume that we have modified  $\pi^*$  in this way to make  $a_2 \rightarrow b_1$  admissible. Given positions of  $a_1, a_2$ , and  $b_1$ , we now try to place  $b_2$  such that

924  $a_1 \rightarrow b_1$  and  $a_2 \rightarrow b_2$  cross. Specifically,  $b_2$  must be placed within a region  $\mathcal{F}$  that lies between the  
 925 rays  $\overrightarrow{a_2 a_1}$  and  $\overrightarrow{a_2 b_1}$ , and within the halfplane bounded by  $\overleftrightarrow{a_1 b_1}$  that does not contain  $a_2$ .

926 Let  $g_a(q) := \|a - q\| - \pi^*(a)$  for  $a \in A$  and  $q \in \mathbb{R}^2$ . Let the **bisector** between  $a_1$  and  $a_2$  be  
 927  $\beta := \{q \in \mathbb{R}^2 \mid g_{a_1}(q) = g_{a_2}(q)\}$ .  $\beta$  is a curve subdividing the plane into two open faces, one where  
 928  $g_{a_1}$  is minimized and the other where  $g_{a_2}$  is. From these definitions, admissibility of  $a_1 \rightarrow b_1$  and  
 929  $a_2 \rightarrow b_1$  imply that  $b_1$  is a point of the bisector.

930 We show that  $\mathcal{F}$  lies entirely on the  $g_{a_1}$  side of the bisector. First, we prove that the closed  
 931 segment  $\overline{a_1 b_1}$  lies entirely on the  $g_{a_1}$  side, except  $b_1$  which lies on  $\beta$ . Any  $q \in \overline{a_1 b_1}$  can be written  
 932 parametrically as  $q(t) = (1 - t)b_1 + ta_1$  for  $t \in [0, 1]$ . Consider the single-variable functions  
 933  $g_{a_1}(q(t))$  and  $g_{a_2}(q(t))$ .

$$\begin{aligned} g_{a_1}(q(t)) &= (1 - t)\|a_1 - b_1\| - \pi(a_1) \\ g_{a_2}(q(t)) &= \|(a_2 - b_1) - t(a_1 - b_1)\| - \pi(a_2) \end{aligned}$$

934 At  $t = 0$ , these expressions are equal. Observe that the derivative with respect to  $t$  of  $g_{a_1}(q(t))$   
 935 is less than  $g_{a_2}(q(t))$ . Indeed, the value of  $\frac{d}{dt}\|(a_2 - b_1) - t(a_1 - b_1)\|$  is at least  $-\|a_1 - b_1\| =$   
 936  $\frac{d}{dt}g_{a_1}(q(t))$ , which is realized iff  $\frac{(a_2 - b_1)}{\|a_2 - b_1\|} = \frac{(a_1 - b_1)}{\|a_1 - b_1\|}$ . This corresponds to  $\overrightarrow{a_2 b_1}$  and  $\overrightarrow{a_1 b_1}$  being  
 937 parallel, but this is disallowed since  $a_1, a_2, b_1$  are in general position. Thus,  $g_{a_1}(q(t)) \leq g_{a_2}(q(t))$   
 938 with equality only at  $b_1$ .

939 Now, we parameterize each point of  $\mathcal{F}$  in terms of points on  $\overline{a_1 b_1}$ . Every  $q \in \mathcal{F}$  can be written  
 940 as  $q(t') = q' + t'(q' - a_2)$  for some  $q' \in \overline{a_1 b_1}$  and  $t \geq 0$ , i.e.  $q' = \overline{a_1 b_1} \cap \overrightarrow{a_2 q}$ . We call  $q'$  the  
 941 **projection** of  $q$  onto  $\overline{a_1 b_1}$ . We can write  $g_{a_1}$  and  $g_{a_2}$  in terms of  $t'$  and observe that  $\frac{d}{dt'}g_{a_1}(q(t')) \leq$   
 942  $\frac{d}{dt'}g_{a_2}(q(t'))$ , as the derivative of  $g_a(q(t'))$  is maximized if  $(q(t') - a)$  is parallel to  $(q(t') - a_2)$   
 943 and lower otherwise. Notably,  $q(t')$  with projection  $b_1$  have  $\frac{d}{dt'}g_{a_1}(q(t')) < \frac{d}{dt'}g_{a_2}(q(t'))$ , since  
 944  $a_1, a_2, b_1$  are in general position. Any  $q(t')$  with a different projection do not have strict inequality,  
 945 but the projection itself has  $g_{a_1}(q') < g_{a_2}(q')$  for  $q' \neq b_1$  since it lies on  $\overline{a_1 b_1}$ . Therefore, for all  
 946  $q \in \mathcal{F} \setminus \{b_1\}$ ,  $g_{a_1}(q') < g_{a_2}(q')$ , and  $\mathcal{F}$  lies on the  $g_{a_1}$  side of the bisector except for  $b_1$  which lies  
 947 on  $\beta$ . We can eliminate  $b_1$  as a candidate position for  $b_2$ , since points of  $B$  cannot coincide.

948 Observe that  $g_{a_1}(b) < g_{a_2}(b)$  for  $b \in B$  implies that  $c_\pi(a_1, b) < c_\pi(a_2, b)$ , and  $c_\pi(a_1, b) =$   
 949  $c_\pi(a_2, b)$  if and only if  $b$  lies on  $\beta$ . This holds for all  $b \in \mathcal{F}$  including our prospective  $b_2$ , but then  
 950  $c_\pi(a_1, b_2) < c_\pi(a_2, b_2) = 0$  since  $a_2 \rightarrow b_2$  is admissible. This violates feasibility of  $a_1 \rightarrow b_2$ , so there  
 951 is no feasible placement of  $b_2$  which also crosses  $a_1 \rightarrow b_1$  with  $a_2 \rightarrow b_2$ . ◀

952 We can construct the entire set of admissible arcs by repeatedly querying the minimum-  
 953 reduced cost outgoing arc for each  $a \in A$  until the result is not admissible. By Lemma C.1 the  
 954 resulting arc set forms a planar graph, so by Euler's formula the number of arcs to query is  
 955  $O(n)$ . We can then find the maximum flow in time  $O(n \log n)$  time, using for example the planar  
 956 maximum-flow algorithm by Erickson [Eri10]. [?]

957 By prioritizing the relaxation of support arcs, we also have the following lemma.

958 ▶ **Lemma C.2 (Agarwal et al. [AFP<sup>+</sup>17]).** *If arcs of  $\text{supp}(f)$  are relaxed first as they arrive on  
 959 the frontier, then  $E(\text{supp}(f))$  is acyclic.*

960 **Proof.** Let  $f_i$  be the pseudoflow after the  $i$ -th augmentation, and let  $T_i$  be the forest of relaxed  
 961 arcs generated by the Hungarian search for the  $i$ -th augmentation. Namely, the  $i$ -th augmenting  
 962 path is an excess-deficit path in  $T_i$ , and all arcs of  $T_i$  are admissible by the time the augmentation  
 963 is performed. Let  $E(T_i)$  be the undirected edges corresponding to arcs of  $T_i$ . Notice that,  
 964  $E(\text{supp}(f_{i+1})) \subseteq E(\text{supp}(f_i)) \cup E(T_i)$ . We prove that  $E(\text{supp}(f_i)) \cup E(T_i)$  is acyclic by induction  
 965 on  $i$ ; as  $E(\text{supp}(f_{i+1}))$  is a subset of these edges, it must also be acyclic. At the beginning with  
 966  $f_0 = 0$ ,  $E(\text{supp}(f_0))$  is vacuously acyclic.

Let  $E(\text{supp}(f_i))$  be acyclic by induction hypothesis. Since  $T_i$  is a forest (thus, acyclic), any hypothetical cycle  $\Gamma$  that forms in  $E(\text{supp}(f_i)) \cup E(T_i)$  must contain edges from both  $E(\text{supp}(f_i))$  and  $E(T_i)$ . To give a visual analogy, we will color  $e \in \Gamma$  **purple** if  $e \in E(\text{supp}(f_i)) \cap E(T_i)$ , **red** if  $e \in E(\text{supp}(f_i))$  but  $e \notin E(T_i)$ , and **blue** if  $e \in E(T_i)$  but  $e \notin E(\text{supp}(f_i))$ . Then,  $\Gamma$  is neither entirely red nor entirely blue. We say that red and purple edges are **red-tinted**, and similarly blue and purple edges are **blue-tinted**. Roughly speaking, our implementation of the Hungarian search prioritizes relaxing red-tinted admissible arcs over pure blue arcs.

We can sort the blue-tinted edges of  $\Gamma$  by the order they were relaxed into  $S$  during the Hungarian search forming  $T_i$ . Let  $(v, w) \in \Gamma$  be the last pure blue edge relaxed, of all the blue-tinted edges in  $\Gamma$  — after  $(v, w)$  is relaxed, the remaining unrelaxed, blue-tinted edges of  $\Gamma$  are purple.

Let us pause the Hungarian search the moment before  $(v, w)$  is relaxed. At this point,  $v \in S$  and  $w \notin S$ , and the Hungarian search must have finished relaxing all frontier support arcs. By our choice of  $(v, w)$ ,  $\Gamma \setminus (v, w)$  is a path of relaxed blue edges and red-tinted edges which connect  $v$  and  $w$ . Walking around  $\Gamma \setminus (v, w)$  from  $v$  to  $w$ , we see that every vertex of the cycle must be in  $S$  already:  $v \in S$ , relaxed blue edges have both endpoints in  $S$ , and any unrelaxed red-tinted edge must have both endpoints in  $S$ , since the Hungarian search would have prioritized relaxing the red-tinted edges to grow  $S$  before relaxing  $(v, w)$  (a blue edge). It follows that  $w \in S$  already, a contradiction.

No such cycle  $\Gamma$  can exist, thus  $E(\text{supp}(f_i)) \cup E(T_i)$  is acyclic and  $E(\text{supp}(f_{i+1})) \subseteq E(\text{supp}(f_i)) \cup E(T_i)$  is acyclic. By induction,  $E(\text{supp}(f_i))$  is acyclic for all  $i$ . ◀

Let  $E(\Sigma_a)$  (**only used once**) be the underlying edges of the support star centered at  $a$  and  $F := E(\text{supp}(f)) \setminus \bigcup_{a \in A} E(\Sigma_a)$ . Using Lemma C.2, we can show that the number of support arcs outside support stars ( $|F|$ ) is small.

► **Lemma C.3.**  $|B_\ell \setminus \bigcup_{a \in A} \Sigma_a| \leq r$ .

**Proof.**  $F$  is constructed from  $E(\text{supp}(f))$  by eliminating edges in support stars, therefore all edges in  $F$  must adjoin vertices in  $B$  of support degree at least 2. By Lemma C.2,  $E(\text{supp}(f))$  is acyclic and therefore forms a spanning forest over  $A \cup B_\ell$ , so  $F$  is also a bipartite forest. All leaves of  $F$  are therefore vertices of  $A$ .

Pick an arbitrary root for each connected component of  $F$  to establish parent-child relationships for each edge. As no vertex in  $B$  is a leaf, each vertex in  $B$  has at least one child. Charge each vertex in  $B$  to one of its children in  $F$ , which must belong to  $A$ . Each vertex in  $A$  is charged at most once. Thus, the number of  $B_\ell$  vertices outside of support stars is no more than  $r$ . ◀

► **Lemma 6.2.** Suppose we have stripped the graph of dead vertices. The number of relaxation steps in a Hungarian search outside of support stars is  $O(r)$ .

**Proof.** If there are no dead vertices, then each non-support star relaxation step adds either (i) an active deficit vertex, (ii) a non-deficit vertex  $a \in A_\ell$ , or (iii) a non-deficit vertex  $b \in B_\ell$  of support degree at least 2. There is a single relaxation of type (i), as it terminates the search. The number of vertices of type (ii) is  $r$ , and the number of vertices of type (iii) is at most  $r$  by Lemma C.3. The lemma follows. ◀

► **Lemma 6.3.** Hungarian search takes  $O(r\sqrt{n} \text{polylog } n)$  time.

**Proof.** The number of relaxation steps outside of support stars is  $O(r)$  by Lemma 6.2. The time per relaxation outside of support stars is  $O(\sqrt{n} \text{polylog } n)$ . The time spent processing relaxations within a support star is  $O(\sqrt{n} \text{polylog } n)$ , and at most  $r$  are relaxed during the search. The total time is therefore  $O(r\sqrt{n} \text{polylog } n)$ . ◀

Initially, we label stars big or small according to the  $\sqrt{n}$  threshold. A star that is currently big is turned into a small star once  $|\Sigma_a| \leq \sqrt{n}/2$ . A star that is currently small is turned into a big star once  $|\Sigma_a| \geq 2\sqrt{n}$ . This way, the time spent rebuilding/updating the respective data structures can be amortized to the insertions/deletions that preceeded the switch, plus some  $O(r)$  extra work if the the update is small-to-big.

A star  $\Sigma_a$  that is switching from big-to-small has size  $|\Sigma_a| \leq \sqrt{n}/2$ . When switching, we delete  $\mathcal{D}_{\text{big}}(a)$  and insert  $\Sigma_a$  into  $\mathcal{D}_{\text{small}}$ . Thus, the time spent for big-to-small update is  $O(\sqrt{n} \text{polylog } n)$ , and there were at least  $\sqrt{n}/2$  points removed from  $\Sigma_a$  since it was last big.

A star  $\Sigma_a$  that is switching from small-to-big has size  $|\Sigma_a| = \sqrt{n} + x \geq 2\sqrt{n}$ , for some integer  $x \geq \sqrt{n}$ . Rearranging, we have  $|\Sigma_a| \leq 2x$ . When switching, we delete all  $|\Sigma_a|$  points from  $\mathcal{D}_{\text{small}}$  and construct a new  $\mathcal{D}_{\text{big}}(a)$ . Constructing  $\mathcal{D}_{\text{big}}(a)$  requires inserting  $O(r)$  points of  $A$  (into  $P$ ) and the  $|\Sigma_a|$  points of the star (into  $Q$ ). Thus, the time spent for a small-to-big update is  $O((r+x) \text{polylog } n)$ , and there were at least  $x \geq \sqrt{n}$  points added to  $\Sigma_a$  since it was last small.

## References for the Appendix

- AFP<sup>+</sup>17** Pankaj K. Agarwal, Kyle Fox, Debmalya Panigrahi, Kasturi R. Varadarajan, and Allen Xiao. Faster algorithms for the geometric transportation problem. In *33rd International Symposium on Computational Geometry, SoCG 2017, July 4-7, 2017, Brisbane, Australia*, pages 7:1–7:16, 2017.
- Eri10** Jeff Erickson. Maximum flows and parametric shortest paths in planar graphs. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 794–804, 2010.
- GHKT17** Andrew V. Goldberg, Sagi Hed, Haim Kaplan, and Robert E. Tarjan. Minimum-cost flows in unit-capacity networks. *Theory Comput. Syst.*, 61(4):987–1010, 2017.
- GT90** Andrew V. Goldberg and Robert E. Tarjan. Finding minimum-cost circulations by successive approximation. *Math. Oper. Res.*, 15(3):430–466, 1990.
- SA12** R. Sharathkumar and Pankaj K. Agarwal. Algorithms for the transportation problem in geometric settings. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 306–317, 2012.
- Vai89** Pravin M. Vaidya. Geometry helps in matching. *SIAM J. Comput.*, 18(6):1201–1225, 1989.