Geometric Partial Matching and Unbalanced Transportation

Pankaj K. Agarwal Allen Xiao

November 22, 2018

1 Introduction

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 **unblanced**. We call the geometric problem of finding a size k matching of point sets A and B the **geometric partial matching problem**.

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 [6] combined with a BCP oracle solves geometric partial matching exactly in time $O((n + k^2) \operatorname{polylog} n)$. Mainly, we show that we can separate the $O(n \operatorname{polylog} n)$ 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 $O((n + k^2) \operatorname{polylog} n)$ time.*

⟨⟨**State the settings separately so no need to repeat in the theorem statement.**⟩⟩

Next in Section 5, we apply a similar technique to the unit-capacity min-cost circulation algorithm of Goldberg, Hed, Kaplan, and Tarjan [3]. The resulting algorithm finds a $(1 + \varepsilon)$-approximation to the optimal geometric partial matching in $O((n + k\sqrt{k}) \operatorname{polylog} n \log(n/\varepsilon))$ 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 $(1 + \varepsilon)$ geometric partial matching of size k can be computed between A and B in $O((n + k\sqrt{k}) \operatorname{polylog} n \log(n/\varepsilon))$ 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 satsify 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 [7]. The result is an $O(n^{3/2} r \operatorname{polylog} n)$ time algorithm for unbalanced transportation. This improves over the $O(n^2 \operatorname{polylog} n)$ 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 $\lambda : (A \cup B) \to \mathbb{Z}$ such that $\sum_{a \in A} \lambda(a) = \sum_{b \in B} \lambda(b)$. An optimal transportation map can be computed in $O(rn^{3/2} \operatorname{polylog} n)$ time.*

By nature of the BCP/NN oracles we use, these results generalize to when $\|a - b\|$ is any L_p distance, and not just the Euclidean distance between a and b.

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. We use $C := \max_{e \in E} c(e)$, and assume that the problem is scaled such that $\min_{e \in E} c(e) = 1$. 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.

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 π 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 1 Hungarian algorithm

```
1: function MATCH(G = (A ∪ B, E), k)
2:     M ← ∅
3:     π(v) ← 0 for all v ∈ A ∪ B
4:     while |M| < k do
5:         Π ← HUNGARIAN-SEARCH(G, M, π)
6:         M ← M ⊕ Π
7:     return M
```

Algorithm 2 Hungarian Search (matching)

Requirement: $c_\pi(a,b) = 0$ for all $(a,b) \in M$

```
1: function HUNGARIAN-SEARCH(G = (A ∪ B, E), M, π)
2:     S ← a ∈ (A \ V(M))  ⟨⟨arbitrary a?⟩⟩
3:     repeat
4:         (a', b') ← arg min{c_π(a,b) | (a,b) ∈ (S ∩ A) × (B \ S)}
5:         γ ← c_π(a', b')
6:         π(v) ← π(v) + γ, ∀v ∈ S                ▷ make (a', b') admissible
7:         S ← S ∪ {b'}
8:         if b' ∉ V(M) then                       ▷ b' unmatched
9:             Π ← alternating augmenting path from a to b'
10:            return Π
11:        else                                    ▷ b' is matched to some a'' ∈ A ∩ V(M)
12:            S ← S ∪ {a''}
13:    until S = A ∪ B
14:    return failure
```

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 6 does not make any reduced cost negative, and thus preserves the dual feasibility constraint for all edges. The algorithm is shown below as Algorithm 2.

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

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

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

In general graphs, the minimum edge is typically found by pushing all encountered $(S \cap A) \times$

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

⟨⟨**Short history on BCP?**⟩⟩ The state of the art dynamic BCP data structure from Kaplan, Mulzer, Roditty, Seiferth, and Sharir [5] supports point insertions and deletions in $O(\operatorname{polylog} n)$ time, and answers queries in $O(\log^2 n)$ time. The following lemma, combined with Theorem 3.1, completes the proof of Theorem 1.1.

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

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

⟨⟨**Untangles the algorithm with the proof; move the algorithm out of the proof of the lemma.**⟩⟩

1. Let S_0^t ⟨⟨**Is there a reason why you want the subscript? Do you ever define S^t?**⟩⟩ denote the initial set S at the beginning of the t-th Hungarian search, that is, the set of unmatched points in A after t augmentations. At the very beginning of the Hungarian algorithm, we initialize $S_0^0 \leftarrow A$ (meaning that $P = A$ and $Q = B$), which is a one-time insertion of $O(n)$ points into BCP, attributed to $P(n,k)$. On each successive Hungarian search, S_0^t shrinks as more and more points in A are matched. Assume for now that, at the beginning of the $(t+1)$-th Hungarian search, we are able to construct S_0^t from the previous iteration. To construct S_0^{t+1}, we simply remove the point in A that was matched by the t-th augmenting path. Thus, with that assumption, we are able to initialize S using one BCP deletion operation per augmentation.

2. During each Hungarian search, points are added to P (that is, some points in A are added to S) and removed from Q (points in B added to S), which will happen at most once per edge relaxation. By Lemma 3.2 the number of relaxed edges is at most k, so the number of such BCP operations is also at most k.

3. To obtain S_0^t, we keep track ⟨⟨**give a name to such points**⟩⟩ of the points added since S_0^t in the last Hungarian search (i.e. those of (2)). ⟨⟨**Unclear**⟩⟩ After the augmentation, we use this log ⟨⟨**use the name**⟩⟩ to delete the added vertices from S and recover S_0^t. By the argument in (2) there are $O(k)$ of such points to delete, so reconstructing S_0^t takes $O(k)$ BCP operations. ⟨⟨**TODO instead of reversing a log, is persistence an easier solution to this?**⟩⟩

We spend $P(n,k) = O(n \operatorname{polylog} n)$ 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 $O(k \operatorname{polylog} n)$.

As for the potential updates, we modify a trick from Vaidya [9] 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 δ (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 δ is at the time of insertion. When the points of S have potentials increased by γ 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 δ, 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 δ. Importantly, δ 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 δ and $\omega(p)$.

The number of updates to δ 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 $T(n,k) = O(k \operatorname{polylog} n)$.

⟨⟨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 geometric partial matching of size k between two point sets A and B in the plane, with cost at most $(1+\varepsilon)$ times the optimal matching, in time $O((n + k\sqrt{k}) \operatorname{polylog} n \log(n/\varepsilon))$.

⟨⟨Insert outline of the section.⟩⟩

4.1 Preliminaries on Network Flows

Network. Let $G = (V, E)$ be a directed graph. One can augmented G into a **network** (V, A, c, u, ϕ), consisting of vertex set V, arc set A, arc costs c, arc capacities u, and a supply-demand function ϕ defined on the vertices. For each directed edge (v, w) in E, insert two **arcs** $v{\to}w$ and $w{\to}v$ into the arc set A, satisfying $u(w{\to}v) = 0$ and $c(w{\to}v) = -c(v{\to}w)$. This we ensure that the graph (V, A) is *symmetric* and the cost function c is *antisymmetric*. 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; in other words,

$$\sum_{v \in V(G)} \phi(v) = 0.$$

A **unit-capacity** network has all its edge capacities equal to 1. **⟨⟨Do we every have non-unit-capacity network in the section? Residual ones?⟩⟩**

Pseudoflows. Given a network $N := (V, A, c, u, \phi)$, a **pseudoflow** $f \colon A \to \mathbb{Z}$ on N is a function on the arcs of N satisfying $f(v{\to}w) \le u(v{\to}w)$ for every arc $v{\to}w$.[1] We say that f **saturates** an arc

[1] In general the pseudoflows are allowed to take real-values. Here under the unit-capacity assumption all the flows are integer-valued.

$v \to w$ if $f(v \to w) = u(v \to w)$; an arc $v \to w$ is **residual** if $f(v \to w) < u(v \to w)$. The **support** of f in N, denoted as $\mathbf{supp}(f)$, is the set of arcs with positive flows:

$$\operatorname{supp}(f) := \{v \to w \in A \mid f(v \to w) > 0\}.$$

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

$$e_f(v) := \phi(v) + \sum_{(w,v) \in E(G)} f(w,v) - \sum_{(v,w) \in E(G)} f(v,w).$$

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

$$\mathbf{cost}(f) := \sum_{(v,w) \in E(G)} c(v,w) \cdot f(v,w).$$

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

Residual network. For each edge (v,w) in G, we refer to the two **arcs** of (v,w) as $v \to w$ and $w \to v$. The **residual capacity** u_f with respect to pseudoflow f is defined to be $u_f(v \to w) := u(v,w) - f(v,w)$ if (v,w) is an edge, and $u_f(v \to w) := f(v,w)$ if (w,v) is an edge. Observe that the residual capacity is always nonnegative. The set of **residual arcs** is defined as

$$E_f := \{v \to w \mid u_f(v \to w) > 0\}.$$

The **residual graph** G_f consists of vertex set $V(G)$ and arc set E_f. We emphasize that edges with reduced capacity zero is not in the residual graph; in other words, if f saturates an edge (v,w) then $v \to w$ is not in G_f. (However, arc $w \to v$ might still be in G_f.) ⟨⟨**Do we ever need the residual network?**⟩⟩ Define N_f to be the **residual network** with respect to pseudoflow f, consisting of the **residual graph** G_f on vertex set $V(G)$ and arc set E_f, together with the antisymmetric cost function c ⟨⟨**Hmm, we need reduced costs**⟩⟩, residual capacities u_f, and the supply-demand function e_f. ⟨⟨**Is this how the function should change? We should rename e_f to ϕ_f?**⟩⟩

LP-duality and admissibility. Formally, the **potentials** $\pi(v)$ are the variables of the linear program dual to the standard LP for the minimum-cost flow problem. ⟨⟨**which primal problem? State the correspodning linear problems explicitly**⟩⟩. The **reduced cost** of an arc $v \to w$ in E_f with respect to π is defined as

$$c_\pi(v \to w) := c(v,w) - \pi(v) + \pi(w)$$

when (v,w) is an edge. The reduced cost of the reverse arc $w \to v$ (when exists) is then set to $-c_\pi(v \to w)$ so that the cost function c_π is made antisymmetric. The **dual feasibility constraint** says that $c_\pi(v \to w) \geq 0$ holds for every edge (v,w); potentials π which satisfy this constraint are said to be **feasible**. The linear programming **optimality condition** states that, for an optimal circulation f^*, there are feasible potentials π^* satisfying $c_{\pi^*}(v \to w) = 0$ for every edge (v,w) in the support of f^*.

Suppose we relax the dual feasibility constraint to allow some small violation in the value of $c_\pi(v \to w)$. We say that a pseudoflow f is **ε-optimal** with respect to π if $c_\pi(v \to w) \geq -\varepsilon$ for every

nonsaturated support edge (v, w) of f. A residual arc $v \to w$ is **admissible** if $c_\pi(v \to w) \leq 0$. We say that a flow f' in G_f is **admissible** if all arcs in the support of f' on G_f are admissible; in other words, $f'(v, w) > 0$ holds only on admissible arcs (v, w). A 0-optimal admissible pseudoflow f must have zero reduced cost on all its support edges.

Lemma 4.1. *Let f be an ε-optimal pseudoflow in G and let f' be an admissible flow in G_f. Then $f + f'$ is also ε-optimal.*

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

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

4.2 Reduction to unit-capacity Min-Cost Flow Problem

Corollary 4.2. *If an algorithm can compute $(\varepsilon \cdot K/6kn)$-optimal circulation for the reduction network N_H of a point set with diameter $C \leq kn^2 \cdot K$ for some constant K, then we can find a $(1 + \varepsilon)$-approximate partial matching betwen A and B, after $O(n \operatorname{polylog} n)$ extra preprocessing time.*

4.3 High-Level Description of Cost-Scaling Algorithm

4.4 Fast Implementation

5 Approximating Min-cost Partial Matchings — OLD

⟨⟨**THIS IS THE OLD SECTION**⟩⟩

In this section, we describe a $(1 + \varepsilon)$-approximation algorithm for the geometric partial matching problem and prove Theorem 1.2. We build atop a **cost-scaling** algorithm for unit-capacity min-cost flow from Goldberg, Hed, Kaplan, and Tarjan [3]. First, we give a cost-preserving near-linear time reduction from geometric partial matching to unit-capacity min-cost flow, which allows us to apply the cost-scaling algorithm toffind partial matchings. ⟨⟨**reduction-algorithm-implementation**⟩⟩

5.1 MPM to unit-capacity MCF reduction

For a partial matching problem on a bipartite graph $G = (A \cup B, E_0)$ with parameter k, we direct each bipartite edge in E_0 from A to B, with cost equal to the original cost $c(a, b)$ and capacity 1. Next, we add a dummy vertex s with arcs (s, a) to every vertex a in A, and a dummy vertex t with arcs (b, t) for every vertex b in B, all with cost 0 and capacity 1. For each of the above arcs (v, w), we also add a reverse arc (w, v) with cost $c(w, v) = -c(v, w)$ and capacity 0. ⟨⟨**is this consistent with the other residual graph description?**⟩⟩ Let the complete set of arcs be E, and $V = A \cup B \cup \{s, t\}$. Set $\phi(s) = k$, $\phi(t) = -k$, and $\phi(v) = 0$ for all other vertices. ⟨⟨**What is ϕ? You are extending the supply-demand fuction from the *network* on G.**⟩⟩ Let the resulting graph be $H = (V, E)$. Define the network $N_H = ((V, E), c, u, \phi)$. We call H the *reduction*

graph ⟨⟨**ever used?**⟩⟩ of the partial matching instance on A and B with parameter k, and N_H the **reduction network**.

⟨⟨**Describe everything using networks.**⟩⟩

First we show that the number of arcs used by any integer pseudoflow in H is asymptotically bounded by the excess of the pseudoflow.

Lemma 5.1. *Let f be an integer pseudoflow in H with $O(k)$ excess. Then, $|\operatorname{supp}(f)| = O(k)$.*

Proof. Observe that the arcs of H with positive capacity form a directed acyclic graph, and thus the positive-flow edges of f do not contain a cycle. Thus, $\operatorname{supp}(f)$ can be decomposed into a set of inclusion-maximal paths, each of which creates a single unit of excess if it does not terminate at t. A path may also create excess at t if there are at least k other paths terminating at t. By assumption, there are $O(k)$ units of excess to which we can associate paths, and at most k paths that we cannot associate with a unit of excess. The maximum length of any path with positive-flow arcs in H is 3 by construction. We conclude that the number of positive flow arcs in f is $O(k)$. □

It is straightforward to show that any integer circulation on H uses exactly k of the A-to-B arcs, which correspond to the edges of a size-k matching. For a circulation f in H, we use $\boldsymbol{M_f}$ to denote the corresponding matching. Notice that the cost of the circulation f is equal to the cost of the corresponding matching M_f. In other words, an α-approximation to the MCF problem on H is an α-approximation to the matching problem on G.

In the next lemma, we show how ε-optimality ⟨⟨**adj.**⟩⟩ implies an approximation ⟨⟨**noun**⟩⟩ on H. ⟨⟨**The sentece is unparsable.**⟩⟩

⟨⟨**Move the definitions of ε-optimality and admissibility for flows here?**⟩⟩

Lemma 5.2. *Let f an ε-optimal integer circulation in H, and f^* an optimal integer circulation for H. Then, $\operatorname{cost}(f) \leq \operatorname{cost}(f^*) + 6k\varepsilon$.*

Proof. We label the arcs of the residual network H_f as follows:

- **forward arcs**: arcs directed from s-to-A or A-to-B or B-to-t, and
- **reverse arcs**: arcs point in the opposite directions.

⟨⟨**Consider moving the definitions outside the proof.**⟩⟩ Reverse arcs must be induced by positive flow in the opposite direction (forward arc between the same points), since H only has arcs in the forward direction. Since f is a circulation, $\operatorname{supp}(f)$ can be decomposed into k paths from s to t. Each s-to-t path in H is length 3, so the total number of reverse arcs is $3k$.

There exists ⟨⟨**"there is", for simplicity**⟩⟩ a residual flow g in H_f such that $f + g = f^*$. Since both f and f^* are both circulations and H is unit-capacity ⟨⟨**unit-capacity is not an adj.; "has"**⟩⟩, g is comprised of unit flows on a collection edge-disjoint residual cycles $\Gamma_1, \ldots, \Gamma_\ell$. Observe that each residual cycle Γ_i must have exactly half of its arcs being reverse arcs, and therefore we have $\sum_i |\Gamma_i| \leq 6k$.

Let π be a set of potentials which certify that f is ε-optimal. For residual cycles, we have that $c_\pi(\Gamma_i) = c(\Gamma_i)$, since the potential terms telescope. We then see that

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

where the second-to-last inequality follows from the ε-optimality of f with respect to π. Rearranging, we have that $\operatorname{cost}(f) \leq \operatorname{cost}(f^*) + 6k\varepsilon$. □

We use a technique from Sharathkumar and Agarwal [8] to transform the additive ε-approximation into a relative $(1+\varepsilon)$-approximation for geometric matching. Let $\mathcal{T}$ ⟨⟨**change font faces only the the objects are of different types; here $\mathcal{T}$, T_i are both subset of edges**⟩⟩ be the minimum spanning tree of $A \cup B$ and order its edges by decreasing length as $e_1, \ldots, e_{r+n-1}$. Let T_i be the subgraph of $\mathcal{T}$ induced by $e_{i+1}, \ldots, e_{r+n-1}$; that is, the graph obtained from removing $e_1, \ldots, e_i$ from $\mathcal{T}$. For each component T of T_i, let $A_T := T \cap A$ and $B_T := T \cap B$ respectively.

Let j_1 be the minimum index such that there exists a component T of T_{j_1} satisfying $|A_T| \neq |B_T|$. Choose j_2 to be the maximum index less than j_1 satisfying $c(e_{j_2}) \geq n^2 \cdot c(e_{j_1})$. We partition A and B into the collection os sets A_T and B_T according to the components T of T_{j_2}. ⟨⟨**double-subscripts, bad. replace j_1 and j_2 with j and k or something?**⟩⟩ Since $j_2 < j_1$, $|A_T| = |B_T|$ for every component T of T_{j_2}.

⟨⟨**Move the proof to the appendix and summarize the assumptions in a paragraph.**⟩⟩

Lemma 5.3 (Sharathkumar and Agarwal [8, §3.5])**.** *Consider the partition of A and B into collection of sets A_T and B_T using components of T_{j_2}. Let M^* be the optimal matching between A and B, and M^*_T be the optimal matching on $T \in T_{j_2}$* ⟨⟨***ambiguous. DO you mean between A_T and B_T for all T?***⟩⟩*, and let the diameter of a component T be $C_T := \max_{p,q \in A_T \cup B_T} \|p - q\|$. Then,*

(i) $M^* = \bigcup_{T \in T_{j_2}} M^*_T$,

(ii) $C_T \leq kn^2 \cdot c(e_{j_1})$ *for all* $T \in T_{j_2}$, *and*

(iii) $c(e_{j_1}) \leq \operatorname{cost}(M^*)$. ⟨⟨***swap order between (ii) and (iii)?***⟩⟩

Furthermore, this partition can be constructed in $O(n \operatorname{polylog} n)$ time.

The proof of these properties can be found in the original paper [8, Section 3.5], but we reproduce the rest of proofs below (tailored for ε-optimality). Given this lemma, we can construct the MCF reduction network N_H for each component $T = A_T \cup B_T$, find an $(\varepsilon c(e_{j_1})/6kn)$-optimal circulation f_T for each, and then $\bigcup_{T \in T_{j_2}} M_{f_T}$ will be a $(1+\varepsilon)$-approximate partial matching between A and B. Let f^*_T be the optimal flow on H for the component T. Combining Lemma 5.2 and Lemma 5.3, one has

$$\begin{aligned}
\operatorname{cost}\Big(\bigcup_{T \in T_{j_2}} M_{f_T}\Big) &= \sum_{T \in T_{j_2}} \operatorname{cost}(M_{f_T}) \\
&= \sum_{T \in T_{j_2}} \operatorname{cost}(f_T) \\
&\leq \sum_{T \in T_{j_2}} \left(\operatorname{cost}(f^*_T) + \varepsilon c(e_{j_1})/n\right) && \text{[Lemma 5.2]} \\
&= \sum_{T \in T_{j_2}} \left(\operatorname{cost}(M^*_T) + \varepsilon c(e_{j_1})/n\right) \\
&\leq \operatorname{cost}(M^*) + \varepsilon c(e_{j_1}) && \text{[Lemma 5.3(i)]} \\
&\leq (1+\varepsilon)\operatorname{cost}(M^*). && \text{[Lemma 5.3(iii)]}
\end{aligned}$$

Corollary 5.4. *If an algorithm can compute $(\varepsilon c(e_{j_1})/6kn)$-optimal circulation for the reduction network N_H of a point set with diameter $C \leq kn^2 \cdot c(e_{j_1})$, then we can find a $(1+\varepsilon)$-approximate partial matching betwen A and B, after $O(n \operatorname{polylog} n)$ extra preprocessing time.* ⟨⟨***Consider to remove dependency on $c(e_{j_i})$.***⟩⟩

5.2 Algorithm description

Pseudocode for the cost-scaling algorithm is given in Algorithm 3. The Goldberg *et al.* [3] algorithm is based on **cost-scaling** or **successive approximation**, originally due to Goldberg and Tarjan [4]. The algorithm finds ε-optimal circulations for geometrically shrinking values of ε. Each iteration of the outer loop (where ε holds single value) is called a **cost scale**. Once ε is sufficiently small, the ε-optimal flow is a suitable approximation (or even an optimal flow itself when costs are integers [4, 3]). We present this algorithm without the integral-cost assumption because in the geometric partial matching setting (with respect to Euclidean distances) the costs are generally not integers.

Algorithm 3 Cost-Scaling MCF

```
1: function MCF(H, ε*)
2:     ε ← kC, f ← 0, π ← 0
3:     while ε > ε*/6 do
4:         (f, π) ← SCALE-INIT(H, f, π)
5:         (f, π) ← REFINE(H, f, π)
6:         ε ← ε/2
7:     return f
```

Note that the zero flow is trivially kC-optimal for H. At the beginning of each scale, SCALE-INIT takes the previous circulation (now 2ε-optimal) and transforms it into an ε-optimal pseudoflow with $O(k)$ excess. For the rest of the scale, the procedure REFINE, reduces the excess in the newly constructed pseudoflow to zero, making it an ε-optimal circulation. Thus, the algorithm produces an ε^*-optimal circulation after $O(\log(kC/\varepsilon^*))$ scales.

Lemma 5.5. *For each subproblem in Corollary 5.4, the cost-scaling algorithm requires $O(\log(n/\varepsilon^*))$ scales.*

Proof. Recall that the subproblems in Corollary 5.4 have diameter $C \leq kn^2 \cdot c(e_{j_1})$ and ask for an $(\varepsilon^* c(e_{j_1})/6kn)$-optimal circulation. The number of cost scales is bounded above by

$$O(\log(kC/\varepsilon^*)) = O\left(\log\left(\frac{kn^2 \cdot c(e_{j_1})}{\varepsilon^* c(e_{j_1})/6kn}\right)\right) = O(\log(n/\varepsilon^*)).$$

□

5.3 SCALE-INIT

⟨⟨**merge with previous section**⟩⟩

The procedure is described in Algorithm 4.

Let the H_f arcs directed from $s \to A$ or $A \to B$ or $B \to t$ be **forward arcs**, and let those in the opposite directions be **reverse arcs**. ⟨⟨**Already did so in Lemma 4.3; might want to move this definition to somewhere before lemma 4.3.**⟩⟩ The first four lines ⟨⟨**try not use use specific numbers, in case you change the pseudocode later**⟩⟩ of SCALE-INIT raise the reduced cost of each forward arc by ε, therefore making all forward arcs ε-optimal. ⟨⟨**Instead of an example, mention that at the start of the iteration, every forward arc is 2ε-optimal.**⟩⟩ In the lines after ⟨⟨**the for-loop**⟩⟩, we deal with the reduced cost of reverse arcs by simply de-saturating them if they violate ε-optimality. Note that forward arcs will not be de-saturated in this step, since they are now ε-optimal.

Algorithm 4 Scale Initialization

```
1: function SCALE-INIT(H, f, π)
2:     ∀a ∈ A, π(a) ← π(a) + ε
3:     ∀b ∈ B, π(b) ← π(b) + 2ε
4:     π(t) ← π(t) + 3ε
5:     for all (v, w) ∈ supp(f) do
6:         if c_π(w, v) < −ε then
7:             f(v, w) ← 0
8:     return (f, π)
```

Lemma 5.6. SCALE-INIT *turns a* 2ε*-optimal circulation into an* ε*-optimal pseudoflow with* $O(k)$ *excess in* $O(n)$ *time.*

Proof. The potential updates affect every vertex except s, so this takes $O(n)$ time. As for the arc de-saturation, every reverse arc is induced by positive flow on a forward arc, and the number of positive flow edges in f is $O(k)$ by Lemma 5.1. The total number of edges examined by the loop is $O(k)$. In total, this takes $O(n)$ time.

Notice that new excess vertex is only created due to the de-saturation of reverse arcs. Because the arcs in the graph has unit capacity, each de-saturation creates one unit of excess. By Lemma 5.1, there are $|\operatorname{supp}(f)| = O(k)$ reverse arcs, so the total excess created must be $O(k)$. □

5.4 REFINE

REFINE is implemented using a primal-dual augmentation algorithm, which sends improving flows on admissible edges like the Hungarian algorithm. Unlike the Hungarian algorithm, it uses blocking flows instead of augmenting paths.

Algorithm 5 Refinement

```
1: function REFINE(H = (V, E), f, π)
2:     while Σ_{v∈V} |e_f(v)| > 0 do
3:         π ← HUNGARIAN-SEARCH2(H, f, π)
4:         f′ ← DFS(H, f, π)              ▷ f′ is an admissible blocking flow
5:         f ← f + f′
6:     return (f, π)
```

Using the properties of blocking flows and the unit-capacity input graph, Goldberg *et al.* [3] prove that there are $O(k)$ blocking flows before excess becomes 0, but on a slightly different reduction graph and under a slightly different model of minimum-cost flow. We provide a sketch of their proof technique adapted for the reduction network N_H.

⟨⟨HC: I am not familiar enough with their algorithm; do you think it's a straightforward application of the technique, or are there something subtle that requires a complete proof?⟩⟩

Lemma 5.7 (Goldberg *et al.* [3, Lemma 3.11 and §6])**.** *Let* f *be a pseudoflow in* H *with* $O(k)$ *excess. There are* $O(\sqrt{k})$ *blocking flows before excess is 0.*

Proof. ⟨⟨**TODO: proof sketch**⟩⟩ □

An **iteration** of REFINE is a complete execution of the main loop in Algorithm 5. Each iteration of REFINE finds an admissible blocking (improving) flow in two stages, and then augments the current pseudoflow by the blocking flow.

1. A **Hungarian search**, which updates dual variables in a Dijkstra-like manner until there is an excess-deficit path of admissible edges. This is different from the procedure HUNGARIAN-SEARCH used for matching. We call the procedure HUNGARIAN-SEARCH2 to distinguish.

2. A **depth-first search** (DFS) through the set of admissible edges to construct an admissible blocking flow. It suffices to repeatedly extract admissible augmenting paths until no more admissible excess-deficit paths remain. By definition, the union of such paths is a blocking flow.

⟨⟨**This paragraph is hard to follow**⟩⟩ Both procedures traverse the residual graph using admissible arcs from the set of excess vertices. Each step of these procedures **relaxes** a minimum-reduced cost arc from a visited vertex to an unvisited vertex, until a deficit vertex is visited. We associate each relaxation step with its newly-visited vertex.
⟨⟨**Describe in a paragraph instead of stated as a lemma.**⟩⟩

Lemma 5.8. *Suppose* HUNGARIAN-SEARCH2 *can be implemented in* $T_1(n,k)$ *time after a once-per-*REFINE $P_1(n,k)$ *time preprocessing, and* DFS *can be implemented in* $T_2(n,k)$ *time after* $P_2(n,k)$ *preprocessing. Then,* REFINE *can be implemented in time*

$$O(P_1(n,k) + P_2(n,k) + \sqrt{k}T_1(n,k) + \sqrt{k}T_2(n,k) + k\sqrt{k}).$$

As we will show shortly (Lemmas 5.15 and 5.19), the total running time for REFINE is $O((n + k\sqrt{k})\operatorname{polylog} n)$. Combining with Lemmas 5.5 and 5.6 completes the proof of Theorem 1.2. At a high level, our analysis strategy is to charge relaxation events in the search to arcs of $\operatorname{supp}(f)$. We first extend Lemma 5.1 to bound the size of $\operatorname{supp}(f)$ throughout REFINE, by observing that the amount of excess decreases in each iteration of REFINE.

Corollary 5.9. *Let* f *be the pseudoflow before or after any iteration of* REFINE. *Then,* $|\operatorname{supp}(f)| = O(k)$.

We discuss some challenges of our analysis and resolve them, before giving the details of HUNGARIAN-SEARCH2 and DFS.

5.4.1 Empty vertices and the shortcut graph

⟨⟨**A figure might be helpful for this section.**⟩⟩
As it turns out, there are some vertices whose relaxation events we cannot charge to the support size. However, we can replace H_f with an equivalent graph that excludes them, and run HUNGARIAN-SEARCH2 and DFS on the resulting graph.

We say $v \in A \cup B$ is an **empty vertex** if $e_f(v) = 0$ and no edges of $\operatorname{supp}(f)$ adjoin v. ⟨⟨**Hmm. How do you feel about calling them "irrelavant vertices" or "null vertices"?**⟩⟩ We are unable to charge relaxation steps involving empty vertices to $|\operatorname{supp}(f)|$, so the algorithm must deal with them separately. Namely, there is no edge with $f(e) > 0$ adjacent to an empty vertex, reaching an empty vertex does not terminate the search, and there may be $\Omega(n)$ empty vertices at once (consider $H_{f=0}$ ⟨⟨**notation overload**⟩⟩, the residual graph of the empty flow). We use $A_\emptyset$

and $B_\emptyset$ to denote the empty vertices of A and B respectively. Vertices that are not empty are called **non-empty vertices**.

For an empty vertex v, either residual in-degree ($v \in A_\emptyset$) or residual out-degree ($v \in B_\emptyset$) is 1. Call a length 2 paths through v to/from non-empty vertices an **empty 2-path**. For example, if $v \in A_\emptyset$ (resp. $v \in B_\emptyset$), then its empty 2-paths have the form (s, v, b) (resp. (a, v, t)) for each $b \in B \setminus B_\emptyset$ (resp. $a \in A \setminus A_\emptyset$). We say that (s, v, b) is an empty 2-path **surrounding** empty vertex v. Separately, we define the length 3 s-t paths that pass through two empty vertices to be **empty 3-paths**. As with 2-paths, we say an empty 3-path (s, v_1, v_2, t) surrounds $v_1 \in A_\emptyset$ and $v_2 \in B_\emptyset$.

As for the costs of empty paths, consider an empty 2-path (s, v, b) that surrounds $v \in A_\emptyset$. Because reduced costs telescope for residual paths, the reduced cost of (s, v, b) does not depend on the potential of v.

$$c_\pi((s, v, b)) = c_\pi(s, v) + c_\pi(v, b) = c(v, b) - \pi(s) + \pi(b)$$

Something similar holds for empty 2-paths surrounding $B_\emptyset$ vertices, and empty 3-paths.

We construct the **shortcut graph** $\tilde{H}_f$ from H_f by removing all empty vertices and their adjacent edges, and then inserting a direct arc between the end points of each empty path Π of equal cost. We call this direct edge the **shortcut** $\text{short}(\Pi)$ of empty path Π. For example, the empty 2-path (s, v, b) for $v \in A_\emptyset$ is replaced with a shortcut (s, b) of cost $c(\text{short}(s, v, b)) := c(v, b)$. Similarly, the empty 3-path (s, v_1, v_2, t) would be replaced with a shortcut (s, t) of cost $c(\text{short}((s, v_1, v_2 t))) := c(v_1, v_2)$.

The resulting multigraph $\tilde{H}_f$ contains only the non-empty vertices of V, and has the same connectivity between non-empty vertices as H_f. Consider a path Π from non-empty v to non-empty w in H_f. Any empty vertex in Π is surrounded by an empty 2- or 3-path contained in Π, since the only nontrivial residual paths through an empty vertex are its surrounding empty paths. Thus, there is a corresponding v-to-w path $\tilde{\Pi}$ in $\tilde{H}_f$ by replacing each empty path contained in Π with its shortcut. Furthermore, we have $c(\Pi) = c(\tilde{\Pi})$. We argue now that $\tilde{H}_f$ is fine as a surrogate for H_f, by showing that we can recover ε-optimal potentials for the non-empty vertices.

Lemma 5.10. *Let $\tilde{\pi}$ be a ε-optimal set of potentials for non-empty vertices of H_f. Construct potentials π, extending $\tilde{\pi}$ to empty vertices, by setting $\pi(a) \leftarrow \tilde{\pi}(s)$ for $a \in A_\emptyset$ and $\pi(b) \leftarrow \tilde{\pi}(t)$ for $b \in B_\emptyset$. Then,*

1. *π is a set of ε-optimal potentials for H_f, and*
2. *if a shortcut $\text{short}(\Pi)$ is admissible under $\tilde{\pi}$, then every arc of Π is admissible under π.*

Proof. Reduced costs for non-empty to non-empty arcs are unchanged between $\tilde{\pi}$ and π, so ε-optimality are preserved for these. Recall that an empty path is comprised of one A-to-B arc, and 1 or 2 zero-cost arcs (connecting the empty vertex/vertices to s and t). With our choice of empty vertex potentials, we observe that the zero-cost arcs have reduced cost 0: for an empty $a \in A_\emptyset$, $c_\pi(s, a) = 0$, for an empty $b \in B_\emptyset$, $c_\pi(b, t) = 0$. These arcs are both ε-optimal ($\geq -\varepsilon$) and admissible (≤ 0), so it remains to prove ε-optimality and admissibility for arcs (a, b) where either a or b is an empty vertex.

Let $(a, b) \in A \times B$ such that at least one of a or b is empty. There exists an empty path Π that contains (a, b). Observe that $c_\pi(a, b) = c_\pi(\Pi)$, which we can prove for all varieties of empty paths.

- If $\Pi = (s, a, b)$ for $a \in A_\emptyset$:

$$c_\pi(a, b) = c(a, b) - \pi(a) + \pi(b) = c(a, b) - \pi(s) + \pi(b) = c_\pi(\Pi)$$

- If $\Pi = (a, b, t)$ for $b \in B_\emptyset$:

$$c_\pi(a, b) = c(a, b) - \pi(a) + \pi(b) = c(a, b) - \pi(a) + \pi(t) = c_\pi(\Pi)$$

Algorithm 6 Hungarian Search (cost-scaling)

function HUNGARIAN-SEARCH2($H = (V, E)$, f, π)
  $\tilde{H}_f \leftarrow$ the shortcut graph of H_f
  $S \leftarrow \{v \in V \mid e_f(v) > 0\}$
  repeat
    $(v', w') \leftarrow \arg\min\{c_\pi(v', w') \mid v' \in S, w' \notin S, (v', w') \in \tilde{H}_f)\}$
    $\gamma \leftarrow c_\pi(v', w')$
    if $\gamma > 0$ **then** ▷ make (v', w') admissible if it isn't
      $\pi(v) \leftarrow \pi(v) + \gamma\lceil\frac{\gamma}{\varepsilon}\rceil, \forall v \in S$
    $S \leftarrow S \cup \{w'\}$
    if $e_f(w') < 0$ **then** ▷ reached a deficit
      return π
  until $S = (A \setminus A_\emptyset) \cup (B \setminus B_\emptyset)$
  return failure

- If $\Pi = (s, a, b, t)$ for $a \in A_\emptyset$ and $b \in B_\emptyset$:

$$c_\pi(a, b) = c(a, b) - \pi(a) + \pi(b) = c(a, b) - \pi(s) + \pi(t) = c_\pi(\Pi)$$

By construction, $c_\pi(\Pi) = c_{\tilde{\pi}}(\text{short}(\Pi))$, so we have $c_\pi(a, b) = c_{\tilde{\pi}}(\text{short}(\Pi)) \geq -\varepsilon$ and (a, b) is ε-optimal. Additionally, if short(Π) is admissible under $\tilde{\pi}$, then so is (a, b) under π. Empty paths cover all arcs adjoining empty vertices, so we have proved both parts of the lemma for all arcs in H_f. □

In REFINE, we do not explicitly construct $\tilde{H}_f$ for running HUNGARIAN-SEARCH2 or DFS, but query its edges using BCP/NN oracles and min/max heaps on elements of H_f. Potentials for empty vertices are only required at the end of REFINE (for the next scale), and right before an augmentation sends flow through an empty path, making its surrounded vertices non-empty. During these occasions, we use the procedure in Lemma 5.10 to find feasible, ε-optimal potentials for empty vertices which also preserve the structure of admissibility.

Lemma 5.11. *The number of end-of-*REFINE *empty vertex potential updates is $O(n)$. The number of augmentation-induced empty vertex potential updates in each invocation of* REFINE *is $O(\sum_i N_i)$ where N_i is the number of positive flow arcs in the i-th blocking flow.*

Proof. The number of end-of-REFINE potential updates is $O(n)$. Each update due to flow augmentation involves a blocking flow sending positive flow through an empty path, causing a potential update on the surrounded empty vertex. We charge this potential update to the edges of that empty path, which are in turn arcs with positive flow in the blocking flow. For each blocking flow, no positive arc is charged more than twice. It follows that the number of augmentation-induced updates is $O(N_i)$ for the i-th blocking flow, and $O(\sum_i N_i)$ over the course of REFINE. □

Ultimately, we prove that $\sum_i N_i = O(k\sqrt{k})$, but this requires that we explain the process creating each blocking flow. We revisit this lemma after analyzing DFS.

5.4.2 Hungarian search

Logically, we are executing the Hungarian search ("raise prices") from [3, Section 3.2] on the shortcut graph $\tilde{H}_f$. We describe how we can query the minimum-reduced cost arc leaving S in $O(\text{polylog}\, n)$ time, for the shortcut graph, without constructing $\tilde{H}_f$ explicitly. For this purpose, let S' be a set of "reached" vertices maintained alongside S, identical except whenever a shortcut is relaxed, we add its surrounded empty vertices to S' in addition to its (non-empty) endpoints. Observe that the arcs of $\tilde{H}_f$ leaving S fall into $O(1)$ categories.

1. Non-shortcut reverse 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 S.
2. Non-shortcut A-to-B forward arcs. For these, we can use a BCP data structure between $(A \setminus A_\emptyset) \cap S$ and $(B \setminus B_\emptyset) \setminus S$, 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 $B \setminus S$, queried while $s \in S$. For t, we can maintain a max-heap on the potentials of $A \cap S$, queried while $t \notin S$.
4. Shortcut arcs (s, b) corresponding to empty 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 empty 2-path (s, a, b). This is only queried while $s \in S'$.
5. Shortcut arcs (a, t) corresponding to empty 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 empty 2-path (a, b, t). This is only queried while $t \notin S'$.
6. Shortcut arcs (s, t) corresponding to empty 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 empty 3-path (s, a, b, t). This is only queried while $s \in S'$ and $t \notin S'$.

By construction, the BCP distance of each datastructure in (4-6) is equal to the reduced cost of the shortcut, which is equal to the reduced cost of the corresponding empty path. Each of the above data structures requires one query per relaxation, and an insertion/deletion operation whenever a new vertex moves into S. The data structures above can perform both in $O(\text{polylog}\, n)$ time each, so the running time of HUNGARIAN-SEARCH2 outside of potential updates can be bounded in the number of relaxation steps.

Lemma 5.12. *There are $O(k)$ non-shortcut relaxations in* HUNGARIAN-SEARCH2 *before a deficit vertex is reached.*

Proof. Each edge relaxation adds a new vertex to S, and non-shortcut relaxations only add non-empty vertices. The vertices of $V \setminus S$ fall into several categories: (i) s or t, (ii) vertices of A or B with 0 imbalance, and (iii) deficit vertices of A or B (S contains all excess vertices). The number of vertices in (i) and (iii) is $O(k)$, leaving us to bound the number of (ii) vertices.

An A or B vertex with 0 imbalance must have an even number of $\text{supp}(f)$ edges. There is either only one positive-capacity incoming arc (for A) or outgoing arc (for B), so this quantity is either 0 or 2. Since the vertex is non-empty, this must be 2. We charge 0.5 to each of the two $\text{supp}(f)$ arcs; the arcs of $\text{supp}(f)$ have no more than 1 charge each. Thus, the number of type (ii) vertex relaxations is $O(|\text{supp}(f)|)$. By Corollary 5.9, $O(|\text{supp}(f)|) = O(k)$. □

Lemma 5.13. *There are $O(k)$ shortcut relaxations in* HUNGARIAN-SEARCH2 *before a deficit vertex is reached.*

Proof. Recall the categories of shortcuts from the list of datastructures above. We have shortcuts corresponding to (i) empty 2-paths surrounding $a \in A_\emptyset$, (ii) empty 2-paths surrounding $b \in B_\emptyset$, and (iii) empty 3-paths, which go from s to t.

There is only one relaxation of type (iii), since t can only be added to S once. The same argument holds for type (ii).

Each type (i) relaxation adds some non-empty $b \in B \setminus B_\emptyset$ into S. Since b is non-empty, it must either have deficit or an adjacent arc of $\text{supp}(f)$. We charge this relaxation to b if it is deficit, or the adjacent arc of $\text{supp}(f)$ otherwise. No vertex is charged more than once, and no $\text{supp}(f)$ edge is charged more than twice, therefore the total number of type (i) relaxations is $O(|\text{supp}(f)|)$. By Corollary 5.9, $O(|\text{supp}(f)|) = O(k)$. □

Corollary 5.14. *There are $O(k)$ relaxations in* HUNGARIAN-SEARCH2 *before a deficit vertex is reached.*

In the following lemma, we complete the time analysis of HUNGARIAN-SEARCH2 by proving that potentials can be maintained in $O(k)$ time over the course of the search.

Lemma 5.15. *Using a dynamic BCP, we can implement* HUNGARIAN-SEARCH2 *with $T_1(n,k) = O(k \operatorname{polylog} n)$ and $P_1(n,k) = O(n \operatorname{polylog} n)$.*

Proof. The initial sets for each data structure can be constructed in $O(n \operatorname{polylog} n)$ time. For each of the $O(1)$ data structures that are queried during a relaxation, the new vertex moved into S as a result of the relaxation causes $O(1)$ insertion/deletion operations. For each of the data structures mentioned above, insertions and deletions can be performed in $O(\operatorname{polylog} n)$ time. Using Lemma 3.3 as a basis, we first analyze the number of BCP operations over the course of HUNGARIAN-SEARCH2.

1. Let S_0^t denote the initial set S at the beginning of the t-th Hungarian search, i.e. the set of $v \in V$ with $e_f(v) > 0$ after t blocking flows. Assume for now that, at the beginning of the $(t+1)$-th Hungarian search, we have on hand the S_0^t from the previous iteration. To construct S_0^{t+1}, we remove the vertices that had excess decreased to 0 by the t-th blocking flow. Thus, with that assumption, we are able to initialize S at the cost of one BCP deletion per excess vertex, which sums to $O(k)$ over the entire course of REFINE.

2. During each Hungarian search, a vertex entering S may cause P or Q to update and incur one BCP insertion/deletion. Like before, we can charge these to the number of edge relaxations over the course of HUNGARIAN-SEARCH2. The number of these is $O(k)$ by Corollary 5.14.

3. Like before, we can meet the assumption in (1) by rewinding a log of point additions to S, and recover S_0^t.

For potential updates, we use the same trick as in Lemma 3.3 to lazily update potentials after vertices leave S, but only for non-empty vertices. Non-empty vertices are stored in each data structure with weight $\omega(v) = \pi(v) - \delta$, and δ is increased in lieu of increasing the potential of all S vertices. When vertices leave S (through the rewind mechansim above), we restore their potentials as $\pi(v) \leftarrow \omega(v) + \delta$. With lazy updates, the number of potential updates on non-empty vertices is bounded by the number of relaxations in the Hungarian search, which is $O(k)$ by Corollary 5.14. Note that empty vertex potentials are not handled in HUNGARIAN-SEARCH2. □

Algorithm 7 Depth-first search

function DFS($H = (V, E)$, f, π)
  $\tilde{H}_f \leftarrow$ the shortcut graph of H_f
  $f' \leftarrow 0$.
  $S \leftarrow \{v \in V \mid e_f(v) > 0\}$
  $S_0 \leftarrow \{v \in V \mid e_f(v) > 0\}$ ▷ stack of excess vertices
  $P \leftarrow \textsc{Pop}(S_0)$ ▷ current path; stack
  repeat
    $v' \leftarrow \textsc{Peek}(P)$
    if $e_f(v') < 0$ **then** ▷ if we reached a deficit, save the path to f'
      add to f' a unit flow on the path P
      $P \leftarrow \textsc{Pop}(S_0)$
    else
      $w' \leftarrow \arg\min\{c_\pi(v', w') \mid w' \notin S, (v', w') \in \tilde{H}_f\}$
      $\gamma \leftarrow c_\pi(v', w')$
      if $\gamma \leq 0$ **then** ▷ if (v', w') is admissible, extend the current path
        $S \leftarrow S \cup \{w'\}$
        $P \leftarrow \textsc{Push}(P, w')$
      else ▷ No admissible arcs leaving v', remove from P
        $\textsc{Pop}(P)$
  until $S_0 = \emptyset$
  return f'

5.4.3 Depth-first search

The depth-first search is similar to HUNGARIAN-SEARCH2 in that it uses the relaxation of minimum-reduced cost arcs/empty paths, this time to identify admissible arcs/empty paths in a depth-first manner. This requires some adjustments to the data structures for finding the minimum-reduced cost arc leaving $v' \in S$. Given $v' \in S$, we would like to query:

1. Non-shortcut reverse 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 non-empty $v' \in V$.

2. Non-shortcut A-to-B forward arcs. For these, we maintain a NN data structure over $P = (B \setminus B_\emptyset) \setminus S$, 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'.

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 $B \setminus S$, queried only if $v' = s$. For B-to-t arcs, there is only one arc to check if $v' \in B$, which we can examine manually.

4. Shortcut arcs (s, b) corresponding to empty 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 empty 2-path (s, a, b). This is only queried if $v' = s$.

5. Shortcut arcs (a, t) corresponding to empty 2-paths from $a \in (A \setminus A_\emptyset) \cap S'$ to t. For these, we maintain a NN data structure over $P = B_\emptyset \setminus S'$ with weights $\omega(p) = \pi(t)$ for each $p \in P$.

A response (v', b) corresponds to th empty 2-path (v', b, t). We subtract $\pi(v')$ from the NN distance to recover the reduced cost of the arc from v'. This is not queried if $t \in S$.

6. Shortcut arcs (s, t) corresponding to empty 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 empty 3-path (s, a, b, t). This is only queried while $v' = s$ and $t \notin S'$.

Each data structure above performs $O(\text{polylog}\, n)$ time worth of query and insertion/deletion per relaxation, so the running time is again bounded by $O(\text{polylog}\, n)$ times the number of relaxations.

Lemma 5.16. *There are $O(k)$ non-shortcut relaxations in* DFS.

Lemma 5.17. *There are $O(k)$ shortcut relaxations in* DFS.

Corollary 5.18. *There are $O(k)$ relaxations in* DFS *before a deficit vertex is reached.*

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

Lemma 5.19. *Using a dynamic NN, we can implement* DFS *with $T_2(n, k) = O(k\,\text{polylog}\, n)$ and $P_2(n, k) = O(n\,\text{polylog}\, n)$.*

Proof. At the beginning of REFINE, we can initialize the $O(1)$ data structures used in DFS in $P_2(n, k) = O(n\,\text{polylog}\, n)$ time. We use the same rewinding mechanism as HUNGARIAN-SEARCH2 (Lemma 5.15) to avoid reconstructing the data structures across iterations of REFINE, so the total time spent is bounded by the $O(\text{polylog}\, n)$ times the number of relaxations. By Corollary 5.18, we obtain $T_2(n, k) = O(k\,\text{polylog}\, n)$. □

5.4.4 Size of the blocking flow and completing time analysis

With Lemmas 5.15 and 5.19, we can complete the proof of Lemma 5.8 (time per REFINE) by bounding the total number of arcs whose flow is updated by a blocking flow during REFINE. This bounds both the time spent updating the flow value of these arcs, and also the time spent on empty vertex potential updates (Lemma 5.11).

Lemma 5.20. *Let N_i be the number of positive flow arcs in the i-ith blocking flow of* REFINE. *Then, $\sum_i N_i = O(k\sqrt{k})$.*

Proof. Let i be fixed and consider the invocation of DFS which produces the i-th blocking flow f_i. DFS constructs f_i as a sequence of admissible excess-deficit paths, which appear as path P in Algorithm 7. Every arc in P is an arc relaxed by DFS, so N_i is bounded by the number of relaxations performed in DFS. Using Corollary 5.18, we have $N_i = O(k)$.

By Lemma 5.7, there are $O(\sqrt{k})$ iterations of REFINE before it terminates. Summing, we see that $\sum_i N_i = O(k\sqrt{k})$. □

We now complete the proof of Lemma 5.8. There $O(\sqrt{k})$ iterations of REFINE, each of which executes HUNGARIAN-SEARCH2 and DFS. By Lemmas 5.15 and 5.19, these calls take $O(T_1(n, k) + T_2(n, k)) = O(k\,\text{polylog}\, n)$ time per iteration. HUNGARIAN-SEARCH2 and DFS require some once-per-REFINE preprocessing to initialize data structures in $P_1(n, k) + P_2(n, k) = O(n\,\text{polylog}\, n)$ time.

Outside of these, we need to account for the time spent on flow value updates and augmentation-induced empty vertex potential updates. By Lemma 5.20, the former is $O(k\sqrt{k})$ over the course of REFINE. Combining Lemmas 5.20 and 5.11, the time for the latter is also $O(k\sqrt{k})$.

Filling in the values of $P_1(n,k)$, $P_2(n,k)$, $T_1(n,k)$, and $T_2(n,k)$, the total time for REFINE is $O((n + k\sqrt{k})\operatorname{polylog} n)$. Together with Lemmas 5.5 and 5.6, this completes the proof of Theorem 1.2.

6 Unbalanced transportation

In this section, we give an exact algorithm which solves the transportation problem in $O(rn^{3/2}\operatorname{polylog} n)$ time, proving Theorem 1.3. This algorithm is a geometric implementation of the uncapacitated min-cost flow algorithm due to Orlin [7], combined with some of the tools developed in Sections **??** and 5 (e.g. rewinding relaxation updates).

Let A and B be points in the plane with $r := |A|$ and $n := |B|$. Let $\lambda : A \cup B \to \mathbb{Z}$ 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 $\tau : A \times B \to \mathbb{R}_{\geq 0}$. A transportation map τ 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 τ to be

$$\operatorname{cost}(\tau) := \sum_{a,b\in A\times B} \|a-b\|\tau(a,b).$$

Given A, B, and λ, the **transportation problem** asks to find a feasible transportation map of minimum cost. Using terminology from matchings, 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 $c(a,b) = \|a-b\|$, 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 τ_f by taking $\tau_f(a,b) \leftarrow f(a,b)$. Furthermore, $\operatorname{cost}(f) = \operatorname{cost}(\tau_f)$.

6.1 Uncapacitated MCF by excess scaling

We give an outline of the strongly polynomial algorithm for uncapacited MCF from Orlin [7]. Orlin's algorithm follows an **excess-scaling** paradigm originally due to Edmonds and Karp [2]. Recall that we can compute an optimal flow by beginning with $f = 0$, $\pi = 0$ and repeatedly augmenting f with improving flows on 0-admissible arcs. The excess-scaling algorithm maintain a parameter Δ, and uses for its improving flow an augmenting path between an excess vertex with $e_f(v) \geq \Delta$, and a deficit vertex with $e_f(v) \leq -\Delta$. Vertices with $|e_f(v)| \geq \Delta$ are called **active**. Once there are either no more active excess or no more active deficit vertices, Δ is halved. Each sequence of augmentations where Δ holds a constant value is called an **excess scale**. The algorithm initializes $\Delta = U$, so 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 [7] 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 $|e_f(v)| \geq \alpha\Delta$, for a parameter $\alpha \in (1/2, 1)$. Second, arcs which have $f(v,w) \geq 3n\Delta$ are **contracted**, creating a new vertex $\hat{v}$ which inherits all the arcs of v and

w, and has $\phi(\hat{v}) \leftarrow \phi(\hat{v}) + \phi(\hat{w})$. 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. Third, Δ is aggressively lowered to $\max_{v \in V} e_f(v)$ 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 π^* can be **recovered** for G 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 π^*.

Theorem 6.1 (Orlin [7], Theorems 2 and 3)**.** *Orlin's algorithm finds optimal potentials after $O(n \log n)$ scaling phases, and $O(n \log n)$ total augmentations.*

6.2 Dead vertices and support stars

Given Theorem 6.1, our goal is to implement each augmentation in $O(rn^{1/2} \operatorname{polylog} n)$ time. To find an augmenting path, we again use some form of Hungarian search with geometric data structures to perform relaxations quickly.

Like in Section 5, there are vertices which cannot charge to the flow support. Even worse, the size of the flow support have size $\Omega(n)$ for the transportation map (consider an instance with $r = 1$, and the demand uniformly distributed among vertices of B). Still, by classifying vertices carefully, we can do better than an $O(n)$ bound on the number of relaxations.

Dead vertices. We call a vertex $b \in B$ **dead** if it has no adjoining $\operatorname{supp}(f)$ arcs and is not an active excess or deficit. These are essentially equivalent to the *empty vertices* of Section 5. 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. Dead vertices may only stop being dead after Δ decreases, i.e. in a subsequent excess scale, as no augmenting path will cross a dead vertex. When this happens, we must add it back into a number of data structures.

Support stars. Let the **support degree** of a vertex be the number of adjoining arcs in $\operatorname{supp}(f)$. 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 Σ_a the **support star** centered around $a \in A$. The number of support stars is $O(r)$.

6.3 Implementation details

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] 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.

[4] A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by successive approximation. *Math. Oper. Res.*, 15(3):430–466, 1990.

[5] 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.

[6] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics (NRL)*, 2(1-2):83–97, 1955.

[7] J. B. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Operations Research*, 41(2):338–350, 1993.

[8] 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.

[9] P. M. Vaidya. Geometry helps in matching. *SIAM J. Comput.*, 18(6):1201–1225, 1989.