Geometric Partial Matching and Unbalanced Transportation

Pankaj K. Agarwal Allen Xiao

November 20, 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 be the number of vertices in A, and n be the number of vertices in B. Without loss of generality assume that $r \leq n$. We consider the problem of *partial matching*, where the task is to find the minimum-cost matching of size k (which by definition is at most r). When $k = r = n$, we say the matching instance is *balanced* and call the problem *perfect matching* or the *assignment problem*. When $k = r < n$ (A and B have different sizes, but the matching is maximal), we say the matching instance is *unbalanced*. ⟨⟨**Don't mix up the definition of *balanced* and *perfect*; the former is between r and n, where the latter is between k and r.**⟩⟩ Partial matching generalizes both perfect matching and unbalanced matching. We will refer to the geometric problem as *geometric partial matching*. Maybe bad nameing; there is nothing geometric about this name.

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**⟩⟩. 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 [4] 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.*

Next in Section 4, we apply a similar technique to the unit-capacity min-cost circulation algorithm of Goldberg, Hed, Kaplan, and Tarjan [1]. 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. This time, we use the strongly polynomial uncapacitated min-cost flow algorithm by Orlin [**?**] ⟨⟨**Cite**⟩⟩. 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 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 $\lambda(\cdot)$ ⟨⟨**from $A \cup B$ 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 $O(n^{3/2} r \operatorname{polylog} n)$ time.*

By nature of the BCP/NN oracles we use, our results generalize to any L_p distances. ⟨⟨**Mention Euclidean distance earlier.**⟩⟩

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.

2.2 Minimum-cost flow

⟨⟨**Collect definitions into paragraphs.**⟩⟩

Network. For minimum-cost flow, let $G_0 = (V, E_0)$ be a directed graph with nonnegative arc capacities $u(v, w)$ and costs $c(v, w)$ for each arc $(v, w) \in E_0$. We say G_0 is *unit-capacity* if $u(v, w) = 1$ holds for all arc (v, w). Let $\phi : V \to \mathbb{N}_{\geq 0}$ be a supply-demand function on V, satisfying $\sum_{v \in V} \phi(v) = 0$. The positive values of $\phi(v)$ are referred to as *supply*, and the negative values of $\phi(v)$ as *demand*.

We augment G_0 to make it *symmetric* and the costs *antisymmetric* by creating an arc (w, v) for each $(v, w) \in E_0$ and define $u(w, v) = 0$ and $c(w, v) = -c(v, w)$. Denote this set of new *reverse arcs* by E^R. ⟨⟨**How do you feel about using darts and arcs to describe the distinction?**⟩⟩ From here forward, we work with the symmetric multigraph $G = (V, E = E_0 \cup E^R)$. ⟨⟨**Oh man. This is a mouthful, use English.**⟩⟩ A *network* (G, c, u, ϕ) is a graph G augmented with arc costs, capacities, and a supply-demand function on vertices of G.

Pseudoflows. A *pseudoflow* f is an antisymmetric function on arcs ⟨⟨**define codomain; integers?**⟩⟩ satisfying $f(v, w) \leq u(v, w)$ for all arcs (v, w). We say that f *saturates* an arc e if $f(v, w) = u(v, w)$. The *support* of f is $E_{>0}(f) := \{(v, w) \mid f(v, w) > 0\}$. All our algorithms will handle integer-valued pseudoflows, so in the unit-capacity setting an arc is either saturated or has zero flow. Given a pseudoflow f, we define the *imbalance* of a vertex to be

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

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

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

The *minimum-cost flow problem (MCF)* asks to find the circulation f^* of minimum cost.

Residual network. For each arc (v,w), the *residual capacity* with respect to pseudoflow f is defined to be $u_f(v,w) := u(v,w) - f(v,w)$. The set of *residual edges* ⟨⟨**arcs?**⟩⟩ is defined as

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

We call $G_f = (V, E_f)$ the *residual graph* with respect to pseudoflow f. A pseudoflow f' in G_f can be "added" or "augmented" to f to produce a new pseudoflow (that is, the arc-wise addition $f + f'$ is a valid pseudoflow in G). A pseudoflow f' in G_f is an *improving flow* if

(1) $0 \le e_{f'}(v) \le e_f(v)$ for all excess vertices v,

(2) $0 \le -e_{f'}(v) \le -e_f(v)$ for all deficit vertices v, and

(3) $\sum_{v\in V} |e_{f'}(v)| < \sum_{v\in V} |e_f(v)|$ holds.

If improving flow f' is on a simple path (from an excess vertex to a deficit vertex), we call it an *augmenting-path flow* ⟨⟨**path flow for short? only used twice throughout the paper**⟩⟩ and its underlying support path ⟨⟨**support undefined**⟩⟩ an *augmenting path*. If f' saturates at least one residual arc in every augmenting path in G_f, we call f' a *blocking flow*. In other words, for blocking flow f', there is no augmenting-path flow f'' in G_f for which $f + (f' + f'')$ is a feasible pseudoflow in G. ⟨⟨**Move to Section 4.4 where blocking flow is first used.**⟩⟩

2.3 Primal-dual augmentation algorithms

The Hungarian algorithm begins with an empty matching and gradually increases its size to k using *alternating augmenting paths*. Given a non-maximal matching M, an alternating augmenting path P is a path between an unmatched vertex $a \in A$ and unmatched $b \in B$. ⟨⟨**What is A and B? Remind the readers about the bipartite graph again.**⟩⟩ Then, $M' = M \oplus P$ is a matching of size 1 greater. ⟨⟨$\oplus$ **undefined. Personally I believe it's easier to define in words; using notation is fine too.**⟩⟩ By restricting alternating augmenting paths to edges ⟨⟨**when do you use edges and when do you use arcs?**⟩⟩ which satisfy a certain cost condition (admissibility, defined momentarily) ⟨⟨**define it or don't say it**⟩⟩, one can prove that each intermediate matching of size $j \le k$ is of minimum-cost among matchings of size j.

There is a similar augmentation procedure for flows, which sends improving flows (e.g. augmenting-path flows) ⟨⟨**no reason to give out details that does not help with sketch of ideas**⟩⟩ to gradually reduce the imbalance in a pseudoflow to 0, making it a circulation. ⟨⟨**imbalance of a pseudoflow is undefined; you only define it on the vertices.**⟩⟩ By restricting augmentations to residual arcs satisfying a certain cost condition (admissibility), one can prove that the resulting circulation is minimum cost.

⟨⟨**The above paragraph might be clearer if you put it after the definition of admissibility; because you can actually provide a formal proof. Sketch of ideas are not that useful because for experts they don't need to read it, for beginners they won't understand without former definitions.**⟩⟩

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

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

⟨⟨**Mention that reduced costs are still antisymmetric.**⟩⟩ The *dual feasibility constraint* is that $c_\pi(v, w) \geq 0$ holds for all residual arcs ⟨⟨**naturally, not the reverse arcs; thus the distinction between arcs and darts**⟩⟩; potentials which satisfy this constraint are said to be *feasible*. The linear programming *optimality conditions* state that, for an optimal circulation f^*, there are feasible potentials π^* which satisfy $c_\pi(v, w) = 0$ ⟨⟨π^***?**⟩⟩ on all arcs with $f^*(v, w) > 0$. We can similarly define potentials and reduced costs for matchings, using $c_\pi(a, b) = c(a, b) - \pi(a) + \pi(b)$ for $(a, b) \in A \times B$. ⟨⟨**How about the reverse arcs?**⟩⟩

Suppose we relax the dual feasibility constraint to allow for a violation of $\varepsilon > 0$. We say that a pseudoflow f is *ε-optimal* ⟨⟨**with respect to** π⟩⟩ if $c_\pi(v, w) \geq -\varepsilon$ for all arcs (v, w) in E_f with $u_f(v, w) > 0$. Note that 0-optimality coincides with the optimality conditions. ⟨⟨**Careful here. Technically it's dual feasibility.**⟩⟩ We say that a residual arc (v, w) satisfying $c_\pi(v, w) \leq 0$ is *admissible*. We say that an improving flow f' is *admissible* if $f'(v, w) > 0$ only on admissible arcs (v, w).

For matchings, we say that matching M is ε-optimal if $c_\pi(a, b) \leq \varepsilon$ for $(a, b) \in M$ and $c_\pi(a, b) \geq -\varepsilon$ $(a, b) \in E \setminus M$. Matching edges (resp. nonmatching edges) are *admissible* if $c_\pi(a, b) \geq 0$ (resp. $c_\pi(a, b) \leq 0$); and an alternating augmenting path is *admissible* if all its edges are. For both matching and flows, 0-optimal f implies the admissibility condition is with equality, instead ($c_\pi(v, w) = 0$).

⟨⟨**I got the sense that it might be helpful to define admissibility at the start of the flow section and the matching section separately.**⟩⟩

Lemma 2.1. *Let f be an ε-optimal pseudoflow in G and let f' be an admissible improving flow in G_f. Then $f + f'$ is also ε-optimal.*

Proof. Augmentation by f' will not change the potentials, but may introduce new arcs with $u_{f+f'}(v, w) > 0$. We will verify that these arcs satisfy the ε-optimality condition. Such an arc (v, w) must have $u(v, w) = f(v, w) > (f + f')(v, w)$, implying $f'(w, v) > 0$. ⟨⟨**I don't understand. Never write a sentense that requires multiple steps to decode. I think what you meant is $u_f = 0$ thus $u = f$, and $u_{f+f'} > 0$ thus $u - (f + f') > 0$; finally $f'(v, w) > 0$ thus $f'(w, v) < 0$.**⟩⟩ By assumption that f' is admissible, (w, v) was an admissible arc, thus $c_\pi(w, v) \leq 0$, implying $c_\pi(v, w) \geq 0$. Thus, all such arcs have $c_\pi(v, w) \geq 0 \geq -\varepsilon$, and $f + f'$ is ε-optimal. □

With a similar argument, we could prove the same for matchings. ⟨⟨**Proof it or cite it, at least in full version.**⟩⟩ Finally, we show that ε-optimality is sufficient to certify that a circulation is an approximate MCF solution, when the underlying graph is of unit-capacity.

Lemma 2.2. *Let G be a unit-capacity graph with n vertices and m arcs, let f be an ε-optimal circulation in G, and let f^* be an optimal circulation for G. Then,* $\text{cost}(f) \leq \text{cost}(f^*) + m\varepsilon$.

⟨⟨**AX: this may be the wrong cost analysis for approximation. would like a stronger statement that addresses 0 cost edges (for instance)**⟩⟩

Proof. By the flow decomposition theorem ⟨⟨**cite?**⟩⟩, there is a residual pseudoflow f' such that $f + f' = f^*$, and f' can be decomposed into a set of unit flows on edge-disjoint cycles in G_f. The

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
```

number of edges used by these cycles is at most m. The cost of a residual cycle is equal to its reduced cost, since the potentials telescope, so the cost of each f' cycle Γ is at least $-|\Gamma|\varepsilon$ by ε-optimality of f. Thus, $\text{cost}(f') \geq -m\varepsilon$, and therefore $\text{cost}(f) \leq \text{cost}(f^*) + m\varepsilon$. □

This bound can be improved if we have a better upper bound on the number of edges used in the cycles of f'.

3 Matching with the 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 subrouotine 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 $|A| = r \geq |B| = n$ ⟨⟨**$r \leq n$?**⟩⟩, 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 edge in the frontier with minimum reduced cost, 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.

Algorithm 2 Hungarian Search (matching)

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

function HUNGARIAN-SEARCH($G = (A \cup B, E)$, M, π)
  $S \leftarrow a \in (A \setminus V(M))$ ⟨⟨**arbitrary *a*?**⟩⟩
  repeat
    $(a', b') \leftarrow \arg\min\{c_\pi(a,b) \mid (a,b) \in (S \cap A) \times (B \setminus S)\}$
    $\gamma \leftarrow c_\pi(a', b')$
    $\pi(v) \leftarrow \pi(v) + \gamma, \forall v \in S$ ▷ make (a', b') admissible
    $S \leftarrow S \cup \{b'\}$
    if $b' \notin V(M)$ **then** ▷ b' unmatched
      $\Pi \leftarrow$ alternating augmenting path from S to b' ⟨⟨**not uniquely defined; do you mean from *a*?**⟩⟩
      return Π
    else ▷ b' is matched to some $a'' \in A \cap V(M)$
      $S \leftarrow S \cup \{a''\}$
  until $S = A \cup B$
  return failure

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 task for BCP problem is to find two 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 be set as the potentials; the corresponding adjusted distance then will be the reduced costs.

⟨⟨**Short history on BCP?**⟩⟩ The state of the art dynamic BCP data structure from Kaplan, Mulzer, Roditty, Seiferth, and Sharir [3] 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.

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 A point ⟨⟨**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 [6] 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 Matching with the Goldberg *et al.* algorithm

⟨⟨**Remind the readers what you want to achieve in this section.**⟩⟩

The basis of the algorithm in this section is a *cost-scaling* algorithm ⟨⟨**try not to repeat words**⟩⟩ for unit-capacity min-cost flow from [1] ⟨⟨**don't use citation as a noun**⟩⟩. Before describing the algorithm, we first give a linear-time reduction from min-cost partial matching to unit-capacity min-cost flow, which allows us to apply the Goldberg *et al.* algorithm ⟨⟨**never defined**⟩⟩ to partial matching.

4.1 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? Undefined in this section.**⟩⟩ Let the resulting graph be $H = (V, E)$. ⟨⟨**Do you mean network, as you have defined cost, capacity, and supply-demand function?**⟩⟩

Observation 4.1. *The arcs of H with positive capacity form a directed acyclic graph.*

In other words, there will be no cycles of positive flow in circulations on H. With this, we can show that the number of arcs used by any integer pseudoflow in H is asymptotically bounded by the excess of the pseudoflow.

Lemma 4.2. *Let f be an integer pseudoflow in H with $O(k)$ excess. Then, $|E_{>0}(f)| = O(k)$.*

Proof. By Observation 4.1, the positive-flow edges of f do not contain a cycle. Thus, $E_{>0}(f)$ 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 ⟨⟨**size-k**⟩⟩ matching. For a circulation f in H, we use M_f to denote the corresponding matching. Observe that $\text{cost}(f) = \text{cost}(M_f)$, so an α-approximation to the MCF problem on H is an α-approximation to the matching problem on G.

We can improve the additive error bound in Lemma 2.2 on H. Note that for integer-valued capacities (in particular, unit capacity), there is always an integer-valued optimal circulation.

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

Lemma 4.3. *Let f an ε-optimal integer circulation in H, and f^* an optimal integer circulation for H. Then, $\text{cost}(f) \leq \text{cost}(f^*) + 6k\varepsilon$.*

Proof. We label the arcs of H_f ⟨⟨**what is H_f? residual network?**⟩⟩ 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.

Observe that a residual cycle Γ ⟨⟨**directed, by definition**⟩⟩ must have exactly half of its edges being reverse arcs. The reverse arcs may either be

(i) on one of the M_f edges ⟨⟨**what does this mean? the underlying edge is in M_f?**⟩⟩,

(ii) between s and A, or

(iii) between B and t.

If it is of type (ii) or (iii), there is an adjacent type (i) reverse arc. Thus, we can charge the reverse arcs of Γ to $M_f \cap \Gamma$ edges ⟨⟨**now this is notation abuse; Γ cannot both be an edge set and an arc set**⟩⟩ with at most 3 charges per edge in $M_f \cap \Gamma$. We can then charge all arcs of $f' = (f^* - f) = \sum \Gamma_i$ ⟨⟨**cannot parse; just arcs in the difference of f^* and f?**⟩⟩ to M_f with at most 6 charge per edge in M_f. As $|M_f| = k$, the number of arcs in f' is at most $6k$. The rest of the argument proceeds as in Lemma 2.2. ⟨⟨**State at the start of the lemma what property do you need to prove in order to apply the black box. Even better, rewrite Lemma 2 the take care of both situation assuming some bounds, and derive two corollaries from it.**⟩⟩ □

Suppose we scaled arc costs (via uniform scaling of the input points) such that the minimum cost (closest pair distance) is 1. ⟨⟨**You already made this assumption in Section 2.1; recall preliminaries.**⟩⟩ Then, $\text{cost}(f^*) \geq k$, and we can turn Lemma 4.3 into a relative approximation.

⟨⟨**I am not sure if you want to state the assumption again in the statements of the corollaries, since this assumption is something you can make when constructing H.**⟩⟩

Corollary 4.4. *Let f an ε-optimal integer circulation in H, and f^* an optimal integer circulation for H. Suppose costs are scaled to be at least 1 on all arcs. Then,* $\text{cost}(f) \leq (1 + 6\varepsilon)\,\text{cost}(f^*)$.

Corollary 4.5. *Let f an $(\varepsilon'/6)$-optimal integer circulation in H, and M^* an optimal size k matching of G. Suppose costs are scaled to be at least 1 on all arcs. Then,* $\text{cost}(M_f) \leq (1 + \varepsilon')\,\text{cost}(M^*)$.

⟨⟨**Where do you use these two corollaries?**⟩⟩

In other words, a $(\varepsilon'/6)$-optimal circulation is sufficient for a $(1 + \varepsilon')$-approximate matching.

⟨⟨**Put the spread reduction here?**⟩⟩

4.2 Algorithm description

The Goldberg *et al.* [1] algorithm is based on *cost-scaling* or *successive approximation*, originally due to Goldberg and Tarjan [2]. The algorithm finds ε-optimal circulations for geometrically shrinking values of ε. Each period ⟨⟨**what's a period?**⟩⟩ where ε holds constant is called a *scale*. Once ε is sufficiently small, the ε-optimal flow is a suitable approximation (or even an optimal flow itself when costs are integers) [2, 1]. We present this algorithm as an approximation is because costs in the geometric partial matching settings (with respect to Euclidean distances) are generally not integers.

Note that the zero flow is trivially kC-optimal for H. At the beginning of each scale, ⟨⟨**it feels weird to use "scale" this way; "scale" sounds like the manitute of ε. How about "iteration"?**⟩⟩ 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, in REFINE ⟨⟨**undefined**⟩⟩, reduces the excess in the newly constructed pseudoflow to zero, making it an ε-optimal circulation.

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
```

The bulk of this section will be the descriptions and analysis of SCALE-INIT and REFINE. ⟨⟨**Mention Algorithm 3 somehow, to justify the placement of the pseudocode.**⟩⟩

First we analyze the number of scales (iterations of the outer loop). Initially $\varepsilon = kC$, and the algorithm stops once ε is at most $\varepsilon^*/6$. Thus, the number of scales is $O(\log(kC/\varepsilon^*))$. For the geometric matching problem, there is a simple way to preprocess the point set such that $C = O(n^2)$, effectively, by Sharathkumar and Agarwal [5]. ⟨⟨**With that assumption that the minimum distance is at least 1?**⟩⟩ This gives us $O(\log(n/\varepsilon^*))$ scales as a result. We briefly describe this preprocessing step at the end of this section. ⟨⟨**where? do it here if needed.**⟩⟩

Lemma 4.6. *The cost-scaling algorithm finds a $(1+\varepsilon^*)$-approximate matching after $O(\log(n/\varepsilon^*))$ scales.*

4.3 SCALE-INIT

The procedure is described in Algorithm 4.

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) ∈ E_{>0}(f) do
6:         if c_π(w, v) < −ε then
7:             f(v, w) ← 0
8:     return (f, π)
```

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.

Lemma 4.7. 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 4.2. 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 4.2, there are $|E_{>0}(f)| = O(k)$ reverse arcs, so the total excess created must be $O(k)$. □

4.4 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 ∑_{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.* [**?**] ⟨⟨**cite**⟩⟩ prove that there are $O(\sqrt{k})$ blocking flows before excess becomes 0. ⟨⟨**This is not honest, I think. There is a derivation almost identical to theirs, but they are using a slightly different reduction graph.**⟩⟩ ⟨⟨**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 4.8 (Goldberg *et al.* [1, 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.*

We can combine this with Lemma 4.2 to argue that the amount of time spent updating the flow within REFINE is $O(k\sqrt{k})$.

Each step of REFINE finds an admissible blocking flow in two stages.

1. A *Hungarian search*, which updates duals in a Dijkstra-like manner until there is an excess-deficit path of admissible edges. There are slight differences from the Hungarian algorithm's "Hungarian search," but the final running time is identical. ⟨⟨**In your paper there are only two. So just mention that this one is different from the one for matchings.**⟩⟩ 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.

For HUNGARIAN-SEARCH2, we again use a dynamic BCP data structure to accelerate the Hungarian search after a once-per-REFINE preprocessing. To perform DFS quickly, we can use a dynamic *nearest-neighbor* (NN) data structure, to discover admissible edges without handling the set of admissible edges explicitly. This is applied in a similar way as the BCP is for Hungarian search.

Lemma 4.9. *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 4.16 and 4.20), the total running time for REFINE is $O((n + k\sqrt{k}) \operatorname{polylog} n)$. Combining with Lemmas 4.6 and 4.7 completes the proof of Theorem 1.2. At a high level, our analysis strategy is to charge relaxation events in the search to the arcs in $E_{>0}(f)$. We first extend Lemma 4.2 to bound the size of $E_{>0}(f)$ throughout REFINE, by observing that the amount of excess decreases in each iteration of REFINE.

Corollary 4.10. *Let* f *be the pseudoflow before or after any iteration of* REFINE. *Then,* $|E_{>0}(f)| = O(k)$.

⟨⟨**Move to the start of next subsubsection?**⟩⟩ 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 without them ⟨⟨**who? the empty vertices of course, but here the readers have yet to see them, so the proof sketch is not super helpful**⟩⟩, and run HUNGARIAN-SEARCH2 and DFS on the resulting graph. We describe these vertices and their properties, before describing HUNGARIAN-SEARCH2 and DFS.

4.4.1 Empty vertices and the shortcut graph

We say $v \in A \cup B$ is an ***empty vertex*** if $e_f(v) = 0$ and no edges of $E_{>0}(f)$ adjoin v. ⟨⟨**Hmm. How do you feel about calling them "irrelavant vertices"?**⟩⟩ We are unable to charge relaxation steps involving empty vertices to $|E_{>0}(f)|$, 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 4.11. *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 $b \in B_\emptyset$:

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

- 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 $\text{short}(\Pi)$ 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 4.11 to find feasible, ε-optimal potentials for empty vertices which also preserve the structure of admissibility.

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 (f, π)
  until $S = (A \setminus A_\emptyset) \cup (B \setminus B_\emptyset)$
  return failure

Lemma 4.12. *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.

4.4.2 Hungarian search

Logically, we are executing the Hungarian search ("raise prices") from [1, 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 $(w, v) \in E_{>0}(f)$. For these, we can maintain a min-heap on $E_{>0}(f)$ 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 4.13. *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) A or B vertex with 0 imbalance, and (iii) A or B vertex with deficit (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 $E_{>0}(f)$ edges. There is either only one positive-capacity incoming edge (for A) or outgoing edge (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 $E_{>0}(f)$ edges; the edges of $E_{>0}(f)$ have no more than 1 charge each. Thus, the number of (ii) vertex relaxations is $O(|E_{>0}(f)|)$. By Corollary 4.10, $O(|E_{>0}(f)|) = O(k)$. □

Lemma 4.14. *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 edge of $E_{>0}(f)$. We charge this relaxation to b if it is deficit, or the adjacent $E_{>0}(f)$ edge otherwise. No vertex is charged more than once, and no $E_{>0}(f)$ edge is charged more than twice, therefore the total number of type (i) relaxations is $O(|E_{>0}(f)|)$. By Corollary 4.10, $O(|E_{>0}(f)|) = O(k)$. □

Corollary 4.15. *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 4.16. *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 4.15.

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 4.15. Note that empty vertex potentials are not handled in HUNGARIAN-SEARCH2. □

4.4.3 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 $(w', v') \in E_{>0}(f)$. For these, we can maintain a min-heap on $(w', v') \in E_{>0}(f)$ 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.

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 $S_0 = \emptyset$

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. Since the pseudoflow is not changed within DFS (since the end of HUNGARIAN-SEARCH2), the proofs from Lemmas 4.13 and 4.14 can be recycled for DFS.

Lemma 4.17. *There are $O(k)$ non-shortcut relaxations in* DFS *before a deficit vertex is reached.*

Lemma 4.18. *There are $O(k)$ shortcut relaxations in* DFS *before a deficit vertex is reached.*

Corollary 4.19. *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 4.20. *Using a dynamic NN, we can implement* DFS *with* $T_2(n,k) = O(k \operatorname{polylog} n)$ *and* $P_2(n,k) = O(n \operatorname{polylog} n)$.

Proof. At the beginning of REFINE, we can initialize the $O(1)$ data structures used in DFS in $P_2(n,k) = O(n \operatorname{polylog} n)$ time. We use the same rewinding mechanism as HUNGARIAN-SEARCH2 (Lemma 4.16) to avoid reconstructing the data structures across iterations of REFINE, so the total time spent is bounded by the $O(\operatorname{polylog} n)$ times the number of relaxations. By Corollary 4.19, we obtain $T_2(n,k) = O(k \operatorname{polylog} n)$. □

4.4.4 Size of the blocking flow and completing time analysis

With Lemmas 4.16 and 4.20, we can complete the proof of Lemma 4.9 (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 4.12).

Lemma 4.21. *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 4.19, we have $N_i = O(k)$.

By Lemma 4.8, 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 4.9. There $O(\sqrt{k})$ iterations of REFINE, each of which executes HUNGARIAN-SEARCH2 and DFS. By Lemmas 4.16 and 4.20, these calls take $O(T_1(n,k) + T_2(n,k)) = O(k \operatorname{polylog} n)$ time per iteration. HUNGARIAN-SEARCH2 and DFS require some once-per-REFINE preprocessing to initialize data structures in $P_1(n,k) + P_2(n,k) = O(n \operatorname{polylog} n)$ 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 4.21, the former is $O(k\sqrt{k})$ over the course of REFINE. Combining Lemmas 4.21 and 4.12, 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 4.6 and 4.7, this completes the proof of Theorem 1.2.

5 Unbalanced transportation

References

[1] 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.

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

[3] 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.

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

[5] 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.

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