Geometric Partial Matching and Unbalanced Transportation

Pankaj K. Agarwal Allen Xiao

November 18, 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 *unblanced*. **⟨⟨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 argmented 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 $f(v, w) = u(v, w)$. 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. **⟨⟨this follows from (1) and (2) too?⟩⟩**

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.

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 **$\langle\langle$which primal problem? State the correspodning linear problems explicitly$\rangle\rangle$**. The *reduced cost* of an arc (v, w) in E_f with respect to π is

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

$\langle\langle$Mention that reduced costs are still antisymmetric.$\rangle\rangle$ The *dual feasibility constraint* is that $c_\pi(v, w) \geq 0$ holds for all residual arcs **$\langle\langle$naturally, not the reverse arcs; thus the distinction between arcs and darts$\rangle\rangle$**; 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 $c_\pi(v, w) = 0$ **$\langle\langle\pi^*$?$\rangle\rangle$** 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 $(a, b) \in A \times B$. **$\langle\langle$How about the reverse arcs?$\rangle\rangle$**

Suppose we relax the dual feasibility constraint to allow for a violation of $\varepsilon > 0$. We say that a pseudoflow f is *ε-optimal* **$\langle\langle$with respect to $\pi\rangle\rangle$** 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. **$\langle\langle$Careful here. Technically it's dual feasibility.$\rangle\rangle$** 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 ($c_\pi(v, w) = 0$).

$\langle\langle$I got the sense that it might be helpful to define admissibility at the start of the flow section and the matching section separately.$\rangle\rangle$

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 $f'(w, v) > 0$. **$\langle\langle$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 $f'(w, v) < 0$.$\rangle\rangle$** 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. **$\langle\langle$Proof it or cite it, at least in full version.$\rangle\rangle$** 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,* $\text{cost}(f) \leq \text{cost}(f^*) + m\varepsilon$.

$\langle\langle$AX: this may be the wrong cost analysis for approximation. would like a stronger statement that addresses 0 cost edges (for instance)$\rangle\rangle$

Proof. By the flow decomposition theorem **$\langle\langle$cite?$\rangle\rangle$**, 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($G = (A \cup B, E), k$)
2:     $M \leftarrow \emptyset$
3:     $\pi(v) \leftarrow 0$ for all $v \in A \cup B$
4:     while $|M| < k$ do
5:         $\Pi \leftarrow$ HUNGARIAN-SEARCH($G$, $M$, $\pi$)
6:         $M \leftarrow M \oplus \Pi$
       ⟨⟨indentation is enough for scope⟩⟩
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'. Indeed, the algorithm in Section 4 gives a bound of $6k$, and converts the statement from an additive approximation to a relative approximation. **⟨⟨Maybe not here; move to Sec. 4.⟩⟩** Since our matching algorithm uses a 0-optimal (exact) solution, we do not include a proof for approximation quality of ε-optimal matchings. **⟨⟨Which means you don't have to explain.⟩⟩**

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 $|A| = r \geq |B| = n$, 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 $(S \cap A) \times (B \setminus S)$. From the frontier, we *relax* the edge with minimum reduced cost, changing the duals such that the edge becomes admissible, and adding the opposite B vertex into S.

The dual 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 5 does not make any reduced cost

Algorithm 2 Hungarian Search (matching)

```
Require: $c_\pi(a,b) = 0$ for all $(a,b) \in M$
1:  function HUNGARIAN-SEARCH($G = (A \cup B, E)$, $M$, $\pi$)
2:      $S \leftarrow a \in (A \setminus V(M))$
3:      repeat
4:          $(a', b') \leftarrow \arg\min\{c_\pi(a,b) \mid (a,b) \in (S \cap A) \times (B \setminus S)\}$
5:          $\pi(v) \leftarrow \pi(v) + c_\pi(a', b'), \forall v \in S$          ▷ make $(a', b')$ admissible
6:          $S \leftarrow S \cup \{b'\}$
7:          if $b' \notin V(M)$ then                                          ▷ $b'$ unmatched
8:              $\Pi \leftarrow$ alternating augmenting path from $S$ to $b'$
9:              return $\Pi$
10:         else                                  ▷ $b'$ is matched to some $a'' \in A \cap V(M)$
11:             $S \leftarrow S \cup \{a''\}$
12:     until $S = A \cup B$
13:     return failure
```

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 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. We will maintain a BCP ⟨⟨**BCP is a *problem*, not a *data structure***⟩⟩ between $P = (S \cap A)$ and $Q = (B \setminus S)$. Changes to the BCP sets ⟨⟨**undefined**⟩⟩ are entirely driven by changing

S; that is, updates to S incur BCP insertions/deletions. We first analyze the bookkeeping besides the dual updates, and then show how dual updates can be implemented efficiently.

1. Let S_0^t **$\langle\langle$Is there a reason why you want the subscript? Do you ever define S^t?$\rangle\rangle$** 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. 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 **$\langle\langle$point in $A\rangle\rangle$** 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 **$\langle\langle$give a name to such points$\rangle\rangle$** of the points added since S_0^t in the last Hungarian search (i.e. those of (2)). **$\langle\langle$Unclear$\rangle\rangle$** After the augmentation, we use this log **$\langle\langle$use the name$\rangle\rangle$** 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. **$\langle\langle$TODO instead of reversing a log, is persistence an easier solution to this?$\rangle\rangle$**

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 such that the dual updates we do perform can be charged to a BCP insertion or deletion. **$\langle\langle$Mouthful$\rangle\rangle$** Throughout the course of the Hungarian algorithm, we maintain a value δ (initially 0) which aggregates the dual changes. Vertices that are added to S are saved into P **$\langle\langle$I don't understand; do you mean only when vertices in A are added to S?$\rangle\rangle$** with weight $\omega(p) \leftarrow \pi(p) - \delta$. When the points of S have their duals **$\langle\langle$Undefined. You mean their potentials?$\rangle\rangle$** increased in (2), we instead raise $\delta \leftarrow \delta + c_\pi(\hat{a}, \hat{b})$. **$\langle\langle$What is $\hat{a}$ and $\hat{b}$?$\rangle\rangle$** Thus, the "true" potential for any point in S is $\omega(p) + \delta$. For points outside S (i.e. $B \setminus S$) **$\langle\langle$only point in B?$\rangle\rangle$**, we simply use the true potential as the BCP weight. Since all $S \cap A$ BCP weights **$\langle\langle$you mean BCP weights of points in P?$\rangle\rangle$** are uniformly offsetted by δ, the solution returned by the BCP oracle does not change. Once a point is removed from S, we update its true potential to be $\pi(p) \leftarrow \omega(p) + \delta$.

Obviously **$\langle\langle$Never use this word$\rangle\rangle$**, the number of updates to δ is equal to the number of edge relaxations, which is $O(k)$ per Hungarian search. The number of times we have to compute the true potential is bounded by the number times we remove a point from S, which is at most $O(k)$ per Hungarian search as well. The total time spent on potential updates per Hungarian search is therefore $O(k)$. Overall, the time spent per Hungarian search is $T(n,k) = O(k \operatorname{polylog}(n))$.

$\langle\langle$The proof gets more handwavy as the paragraph progresses. Consider a revision after this round.$\rangle\rangle$ □

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 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. 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 (with $O(k)$ excess) is $O(k)$.

Lemma 4.2. *Let f be an integer pseudoflow in H with $O(k)$ excess, and let $E_{>0}(f) := \{(v,w) \mid f(v,w) > 0\}$.* ***⟨⟨Define support earlier outside the lemma statement.⟩⟩*** *Then, $|E_{>0}(f)| = O(k)$.*

Proof. By Observation 4.1, the positive-flow edges of f do not contain a cycle. Thus, the flow decomposition of f is a series of paths, each of which can create a single unit of excess if it does not terminate at t. By assumption, there are $O(k)$ such paths. 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 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 (in particular, unit) capacities, there is always an integer-valued optimal circulation.

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* directed from $s \to A$ or $A \to B$ or $B \to t$, and *reverse arcs* in the opposite directions. Observe that a residual cycle Γ 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?⟩⟩** or else (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 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⟩⟩** to M_f with at most 6 charge per M_f edge. 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. Then, $\text{cost}(f^*) \geq k$, and we can turn Lemma 4.3 into a relative approximation.

Corollary 4.4. *Let f an ε-optimal integer circulation in H, and f^* an optimal integer circulation for H. Suppose costs are scaled such that* $\min \|a - b\| = 1$. *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 such that* $\min \|a - b\| = 1$. *Then,* $\text{cost}(M_f) \leq (1 + \varepsilon')\,\text{cost}(M^*)$.

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

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 where ε holds constant is called a *scale*. Once ε is sufficiently small, the ε-optimal flow is a suitable approximation or even optimal when costs are integer [2, 1]. We present this algorithm as an approximation is because costs in geometric partial matching (i.e. Euclidean distances) are generally not integer.

Algorithm 3 Cost-Scaling MCF

```
1:  function MCF($H, \varepsilon'$)
2:      $\varepsilon \leftarrow kC$
3:      $f \leftarrow 0$
4:      $\pi \leftarrow 0$
5:      repeat
6:          $(f, \pi) \leftarrow$ SCALE-INIT($H, f, \pi$)
7:          $(f, \pi) \leftarrow$ REFINE($H, f, \pi$)
8:          $\varepsilon \leftarrow \varepsilon/2$
9:      until $\varepsilon \leq \varepsilon'/6$
10:     return $f$
```

Note that the 0 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. The rest of the scale, in REFINE, reduces the excess in this pseudoflow to 0, making an ε-optimal circulation. The bulk of this section will describe and analyze SCALE-INIT and REFINE.

For now, we analyze the number of scales (iterations of the outer loop). Initially $\varepsilon = kC$, and the algorithm is stopped once $\varepsilon \leq \varepsilon'/6$. Thus, the number of scales is $O(\log(kC/\varepsilon'))$. For bichromatic matching, there is a simple way to preprocess the point set such that $C = O(n^2)$, effectively, by Sharathkumar and Agarwal [5]. Using this preprocessing gives us $O(\log(n/\varepsilon'))$ scales, instead. We briefly describe this preprocessing step at the end of this section.

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

4.3 SCALE-INIT

Algorithm 4 Scale Initialization

```
1: function SCALE-INIT($H, f, \pi$)
2:     $\forall a \in A, \pi(a) \leftarrow \pi(a) + \varepsilon$
3:     $\forall b \in B, \pi(b) \leftarrow \pi(b) + 2\varepsilon$
4:     $\pi(t) \leftarrow \pi(t) + 3\varepsilon$

5:     for all $(v, w) \in E_f$ do
6:         if $c_\pi(w, v) < -\varepsilon$ then
7:             $f(v, w) \leftarrow 0$
8:     return $(f, \pi)$
```

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*. The first 4 lines of SCALE-INIT raise the reduced cost of each forward arc by ε, therefore making all forward arcs ε-optimal. For example, a forward arc of $A \to B$ now has reduced cost

$$c(a, b) - (\pi(a) + \varepsilon) + (\pi(b) + 2\varepsilon) = c_\pi(a, b) + \varepsilon \geq -2\varepsilon + \varepsilon = -\varepsilon.$$

In the lines after, 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. *In $O(n)$ time,* SCALE-INIT *turns a 2ε-optimal circulation into an ε-optimal pseudoflow with $O(k)$ excess.*

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

For the amount of excess, notice that new excess is only created due to the de-saturations of reverse arcs. Because the graph is unit-capacity, each de-saturation creates one unit of excess. There are $|E_{>0}(f)| = O(k)$ reverse arcs possible, so the total created excess 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.

Using the properties of blocking flows and the unit-capacity input graph, Goldberg *et al.* prove that there are $O(\sqrt{k})$ blocking flows before excess becomes 0.

Lemma 4.8 (Goldberg *et al.* [1] Lemma 3.11 and Section 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 use this, alongside 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.

Algorithm 5 Refinement

```
1: function REFINE($H = (V, E), f, \pi$)
2:     while $\sum_{v \in V} |e_f(v)| > 0$ do
3:         $\pi \leftarrow$ HUNGARIAN-SEARCH2($H, f, \pi$)
4:         $f' \leftarrow$ DFS($H, f, \pi$)                    ▷ $f'$ is an admissible blocking flow
5:         $f \leftarrow f + f'$
6:     return $(f, \pi)$
```

1. A *Hungarian search*, which updates duals in a Dijkstra-like manner until there exists 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. 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 respectively* DFS *in $T_2(n, k)$ time after $P_2(n, k)$ preprocessing. Then,* REFINE *can be implemented in $O(P_1(n, k) + P_2(n, k) + \sqrt{k}[T_1(n, k) + T_2(n, k)] + k\sqrt{k})$ time.*

As we will show shortly (Lemmas 4.12, 4.13), the total running time for REFINE is ultimately $O((n + k\sqrt{k})\,\mathrm{polylog}(n))$. Combining with Lemmas 4.6 and 4.7 completes the proof of Theorem 1.2.

4.4.1 Hungarian search

Compared to HUNGARIAN-SEARCH, this algorithm does not maintain admissibility as an invariant for edges with $f(v, w) > 0$, so these edges are not relaxed immediately in a special case. Instead, we store the $O(|E_{>0}(f)|)$ of them leaving S in a min heap and compare their reduced costs with the BCP result, relaxing the edge with lower reduced cost. Uniformly raising the potentials of S preserves reduced costs for arcs within S and decreases them for arcs leaving S. Raising potentials in intervals of ε will also not decrease a reduced cost to less than $-\varepsilon$, so arcs leaving S still satisfy ε-optimality.

Vertices of $A \cup B$ with $e_f(v) = 0$ and 0 incoming and outgoing flow cannot be charged to k using the previous analysis, so the algorithm treats these *empty vertices* 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 $H_{f=0}$, the residual graph of the empty flow). We use $A_\emptyset$ and $B_\emptyset$ to denote the empty vertices of A and B respectively.

For an empty vertex v, either residual in-degree ($v \in A_\emptyset$) or residual out-degree ($v \in B_\emptyset$) is 1. Instead of querying an empty vertex during the search, we shortcut it using the length 2 paths to/from non-empty vertices, called *empty 2-paths*. 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

Algorithm 6 Hungarian Search (cost-scaling)

```
1:  function HUNGARIAN-SEARCH2($H = (V, E), f, \pi$)
2:      $S \leftarrow \{v \in V \mid e_f(v) > 0\}$
3:      repeat
4:          $\Pi \leftarrow \arg\min\{c_\pi(v, w) \mid \Pi$ is an edge between non-empty vertices of $S \times V \setminus S$, or empty 2-/3-path ou
5:          $\gamma \leftarrow c_\pi(\Pi)$                    ▷ relax the minimum edge or empty path
6:          if $\gamma > 0$ then                        ▷ make $\Pi$ admissible if it isn't
7:              $\pi(v) \leftarrow \pi(v) + \gamma\lceil \frac{\gamma}{\varepsilon} \rceil, \forall v \in S$
8:          $S \leftarrow S \cup \Pi$

9:          Let $\Pi = v_1, \ldots, v_\ell$
10:         if $e_f(v_\ell) < 0$ then                            ▷ $\Pi$ reached a deficit
11:             return $(f, \pi)$
12:     until $S = (A \cup B)$
13:     return failure
```

that (s, v, b) is an empty 2-path *surrounding* empty vertex v. Separately, we consider the length 3 *s*-*t* paths that pass through two empty vertices, called *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 $v_2 \in B_\emptyset$.

Each relaxation (Line Relaxation steps involving an empty 2- or 3-path will relax the entire path at once, moving all vertices of the path into S. Relaxation steps that do not involve an empty 2- or 3-path are called *non-empty*.

Consider an empty 2-path (s, v, b) that surrounds v. Since v has zero excess/deficit, the search cannot "make progress" through v until it reaches b. Since reduced costs telescope for residual paths, the reduced cost of (s, v, b) does not depend on the potential of v.

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

Thus, we can safely ignore the potential of v until it ceases to be empty, e.g. after an augmentation across one of its empty paths. At that point, we can infer $\pi(v)$ from the potentials of the empty 2-path augmented through (this path must be admissible). Before we go into the details of updating the sets of empty vertices and their potentials, we describe the mechanism for querying the minimum-reduced cost empty 2- and 3-paths.

When the search has $s \in S$, we can query the minimum-reduced cost empty 2-path surrounding $A_\emptyset$ vertices using a data structure $D_\emptyset(A)$ maintaining $BCP(P = A_\emptyset, Q = (B \setminus B_\emptyset) \setminus S)$ with weights $\omega(p) = \pi(s)$ for all $p \in P$, and $\omega(q) = \pi(q)$ for all $q \in Q$. It is simple to verify that if (p, q) is the BCP of P, Q above, then its BCP distance is precisely the reduced cost of the 2-path (s, p, q). We can build similar query data structures for other empty 2-paths and empty 3-paths, also reporting a pair whose weighted distance is equal to the reduced cost of the corresponding empty 2- or 3-path. Empty 2-paths surrounding $B_\emptyset$ vertices can be queried using $D_\emptyset(B)$ which maintains $BCP(P = (A \setminus A_\emptyset) \cap S, Q = B_\emptyset)$ with weights $\omega(p) = \pi(p)$ for all $p \in P$, and $\omega(q) = \pi(t)$ for all $q \in Q$. We query $D_\emptyset(B)$ so long as $t \notin S$. Lastly, the minimum-reduced cost empty 3-path can be queried using a data structure $D_\emptyset(A, B)$ maintaining $BCP(P = A_\emptyset \setminus S, Q = B_\emptyset)$ with weights $\omega(p) = \pi(s)$ for all $p \in P$, and $\omega(q) = \pi(t)$ for all $q \in Q$. We query $D_\emptyset(A, B)$ only while $s \in S$ and $t \notin S$.

To summarize, each relaxation step of HUNGARIAN-SEARCH2 takes the minimum between:

1. Reverse arcs of $E_{>0}(f)$ edges, maintained in a min-heap as their endpoints arrive in S.

2. Forward arcs of $(A \setminus A_\emptyset) \cap S \times (B \setminus B_\emptyset) \setminus S$.

3. Forward arcs between s and $(A \setminus A_\emptyset) \cap S$ if $s \in S$, maintained in a min-heap.

4. Forward arcs between $(B \setminus B_\emptyset) \cap S$ and t if $t \notin S$, maintained in a min-heap.

5. Empty 2-paths from s to vertices of $(B \setminus B_\emptyset) \setminus S$, if $s \in S$, maintained in a BCP data structure $D_\emptyset(A)$.

6. Empty 2-paths from vertices of $(A \setminus A_\emptyset) \cap S$ to t, if $t \notin S$, maintained in a BCP data structure $D_\emptyset(B)$.

7. Empty 3-paths, from s to t, if $s \in S$ and $t \notin S$, maintained in a BCP data structure $D_\emptyset(A, B)$.

Observe that (1-3) queries all arcs leaving S to non-empty vertices, and (4-6) queries the length 2 or 3 paths leaving S through empty vertices. **⟨⟨correctness statement here?⟩⟩**

In the next few lemmas, we bound the number of relaxations of each type. This will provide a time bound for the Hungarian search in terms of the number of relaxations, each of which take $O(1)$ BCP queries and updates.

Lemma 4.10. *There are $O(k)$ non-empty relaxations in* HUNGARIAN-SEARCH2 *before a deficit vertex is reached.*

Proof. Let $E_{>0}(f) = \{(v, w) \in E \mid f(v, w) > 0\}$. Each edge relaxation adds a new vertex to S. 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. By the non-emptiness assumption, it 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 $O(|E_{>0}(f)|)$. **⟨⟨need a formal statement that that $|E_{>0}(f)| = O(k)$⟩⟩** ☐

Lemma 4.11. *There are $O(k)$ empty 2- and 3-path relaxations in* HUNGARIAN-SEARCH2 *before a deficit vertex is reached.*

Proof. There is only one empty 3-path relaxation, since t can only be added to S once. This is also the case for the empty 2-paths surrounding a $B_\emptyset$ vertex.

On the other hand, the relaxation of an empty 2-path surrounding an $A_\emptyset$ vertex adds some non-empty $b \in B \setminus B_\emptyset$ into S. By definition, b 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 $E_{>0}(f)$ edge is charged more than twice, therefore the total number of empty 2-path relaxations surrounding $A_\emptyset$ vertices is $O(|E_{>0}(f)|)$. **⟨⟨need a formal statement that that $|E_{>0}(f)| = O(k)$⟩⟩** ☐

Lemma 4.12. *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. Like for matchings, we use a BCP to find the argmin quickly; we maintain $P = (S \cap A)$ and $Q = (B \setminus V)$ using weights $\omega(v) = \pi(v) - \delta$, and increase δ in lieu of increasing the potential of all S vertices. 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.

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 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.

For potential updates, we use the same trick as in Lemma 3.3 to lazily update potentials after vertices leave S. To remind, the number of potential updates outside of empty vertices was bounded by the number of relaxations performed by the most recent invocation of Hungarian search, and thus $O(k)$ by Lemma 4.10 and 4.11.

Potentials for the empty vertices must be updated at the end of a scale and whenever they stop being empty, i.e. when an augmentation sends flow through one of its surrounding empty paths. In both cases, we fix them to $\pi(a) \leftarrow \pi(s)$ for $a \in A_\emptyset$ and $\pi(b) \leftarrow \pi(t)$ for $b \in B_\emptyset$. **⟨⟨probably move correctness of this choice into its own lemma; "feasible choice"⟩⟩** The number of empty vertex potential updates is proportional to the size of the blocking flows, which is $O(k\sqrt{k})$ in total for the scale. **⟨⟨point to a lemma for blocking flow sizes⟩⟩** □

4.4.2 Depth-first search

The depth-first search is similar to HUNGARIAN-SEARCH2 in that it relies on the relaxation of a minimum-reduced cost edge.

Lemma 4.13. *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. □

4.5 Preprocessing for $C = O(n^2)$

Lemma 4.14 (Sharathkumar, Agarwal). *In $O(n \log n)$ time, we can preprocess A, B by partitioning into $(A_1, B_1), \ldots, (A_\ell, B_\ell)$ such that*

1. *each (A_i, B_i) has $|A_i| = |B_i|$,*

2. *the union of optimal matching solutions on (A_i, B_i) is an optimal matching for A, B, and*

3. *the spread of (A_i, B_i) is $O(n^2)$.*

Algorithm 7 Depth-first search

```
function DFS($H = (V, E), f, \pi$)
    $f' \leftarrow 0.$
    $S \leftarrow \{v \in V \mid e_f(v) > 0\}$
    $P \leftarrow \emptyset$
    repeat
        $v' \leftarrow$ POP($P$)
        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 $P$
            $P \leftarrow \emptyset$

        $w' \leftarrow \arg\min\{c_\pi(v', w) \mid w \in V \setminus S\}$
        $\gamma \leftarrow c_\pi(v', w')$

        if $\gamma \leq 0$ then                  ▷ if $(v', w')$ is admissible, extend the current path
            $S \leftarrow S \cup \{w'\}$
            $P \leftarrow$ PUSH(P,w')
        if $e_f(w') < 0$ then                    ▷ $w'$ is a deficit
            return $(f, \pi)$
    until $S = \emptyset$
    return failure
```

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.