plain

Geometric Partial Matching and Unbalanced Transportation

Pankaj K. Agarwal Allen Xiao

November 17, 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 *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 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 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 from Euclidean distances to any L_p distances.

2 Preliminaries

2.1 Matching

Let $G = (A \cup B, E)$ be a bipartite graph between vertex sets A and B, with costs $c(v, w)$ for each $e \in E$. 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 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 the size k matching M^* of minimum cost.

2.2 Minimum-cost flow

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 all $u(v, w) = 1$. Let $\phi : V \to \mathbb{N}_{\geq 0}$ be a supply-demand function on V, where positive values of $\phi(v)$ we refer to as *supply*, negative values of $\phi(v)$ are *demand*, and $\sum_{v \in V} \phi(v) = 0$. We augment G_0 to make it *symmetric* (for every arc $(v, w) \in E_0$ its reverse $(v, w)^R = (w, v)$ is also an arc) and the costs *antisymmetric* ($c(v, w) = -c(w, v)$): for each $(v, w) \in E_0$, create an arc (w, v) and define $u(w, v) = 0$ and $c(w, v) = -c(v, w)$. Let this set of new *reverse arcs* be E^R. From here forward, we work with the symmetric multigraph $G = (V, E = E_0 \cup E^R)$. The combination of graph, costs, capacities, and supply-demands is a *network* (G, c, u, ϕ).

A *pseudoflow* f is an antisymmetric function on arcs ($f(v, w) = -f(w, v)$) such that for each arc $f(v, w) \leq u(v, w)$. We say that f *saturates* an arc $e \in E$ if $f(v, w) = u(v, w)$. All our algorithms will handle integer-valued pseudoflows, so in the unit-capacity setting an arc has either 0 flow or is saturated. 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*, vertices with $e_f(v) > 0$ we call *excess vertices* and respectively those with $e_f(v) < 0$ *deficit vertices*. If all vertices have $e_f(v) = 0$, the pseudoflow is a *circulation*. The cost of a pseudoflow is $\text{cost}(f) = \sum_{(v,w) \in E} c(v, w) f(v, w)$. The minimum-cost flow problem (MCF) asks to find the circulation f^* of minimum cost.

For each arc $(v, w) \in E$, 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* is $E_f := \{(v, w) \in E \mid u_f(v, w) > 0\}$. The

residual graph is $G_f = (V, E_f)$. A pseudoflow f' in G_f can be "added" or "augmented" to f to produce a new pseudoflow (i.e. the per-arc addition of $f + f'$ is a valid pseudoflow in G). A pseudoflow f' in G_f is an *improving flow* if (1) $0 \leq e_{f'}(v) \leq e_f(v)$ for all excess vertices $v \in V$, (2) $0 \leq -e_{f'}(v) \leq -e_f(v)$ for all deficit vertices $v \in V$, (3) if $e_f(v) = 0$ then $e_{f'}(v) = 0$, and (4) $\sum_{v \in V} |e_{f'}(v)| < \sum_{v \in V} |e_f(v)|$. If improving flow f' is on a simple path (from an excess vertex to a deficit vertex), we call it an *augmenting path flow* and its edges an *augmenting path*. If f' saturates at least one residual arc of every augmenting path in G_f, we call it 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 increases it to size k using *alternating augmenting paths*. Given a non-maximal matching M, an alternating augmenting path P is a path between an unmatched $a \in A$ and unmatched $b \in B$. Then, $M' = M \oplus P$ is a matching of size 1 greater. By restricting alternating augmenting paths to edges which satisfy a certain cost condition (admissibility, defined momentarily), one can prove that each intermediate matching, of size $j \leq k$, is minimum-cost for size j. There is a similar augmentation procedure for flows, which sends improving flows (e.g. augmenting path flows) to gradually reduce the imbalance in a pseudoflow to 0, making it a circulation. By restricting augmentations to residual arcs satisfying a certain cost condition (admissibility), one can prove that the resulting circulation is minimum cost.

Formally, we use the linear programming dual problem to define *potentials* (dual variables) $\pi(v)$ for each $v \in V$. The *reduced cost* of $(v, w) \in E_f$ with respect to π is $c_\pi(v, w) := c(v, w) - \pi(v) + \pi(w)$. The *dual feasibility constraint* is that $c_\pi(v, w) \geq 0$ for all residual arcs; potentials which satisfy this are said to be *feasible*. The linear programming *optimality conditions* state that, for an optimal circulation f^*, there exist feasible potentials π^* which satisfy $c_\pi(v, w) = 0$ on all arcs with $f^*(v, w) > 0$. We can similarly define potentials and reduced cost for matchings, using $c_\pi(a, b) = c(a, b) - \pi(a) + \pi(b)$ for $(a, b) \in A \times B$.

Suppose we relax the dual feasibility constraint to allow for a violation of $\varepsilon > 0$. We say that a pseudoflow f is *ε-optimal* if $c_\pi(v, w) \geq -\varepsilon$ for all $(v, w) \in E_f$ with $u_f(v, w) > 0$. Note that 0-optimality coincides with the optimality conditions. We say that residual arcs with $c_\pi(v, w) \leq 0$ are *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 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 are admissible if $c_\pi(a, b) \geq 0$ (resp. nonmatching edges, if $c_\pi(a, b) \leq 0$), 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 ($c_\pi(v, w) = 0$).

Now, we can concretely state how admissible augmentations lead to a correct algorithm for $\varepsilon > 0$.

Lemma 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.*

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 $u(v, w) = f(v, w) > (f + f')(v, w)$, meaning $f'(w, v) > 0$. By assumption, (w, v) was an ε-admissible arc, thus $c_\pi(w, v) \leq 0$, meaning $c_\pi(v, w) \geq 0$. 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. Finally, we show that ε-optimality is sufficient to prove when a circulation is an approximate MCF solution, when the underlying graph is unit-capacity.

Lemma 2 *Let G be a unit-capacity graph with $|V| = n$ and $|E| = m$, f an ε-optimal circulation in G, and f^* an optimal circulation for G. Then,* $\text{cost}(f) \leq \text{cost}(f^*) + m\varepsilon$.

By the flow decomposition theorem, there exists a residual pseudoflow f' such that $f + f' = f^*$, and f' can be decomposed into a set of unit flows on edge-disjoint cycles. The 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 of telescope. 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 uses a graph where the bound is $6k$, and converts the statement from an additive approximation to a relative approximation. Since our matching algorithm uses a 0-optimal (exact) solution, we do not include a proof for approximation quality of ε-optimal matchings.

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. It maintains the invariant that matching edges are admissible. Since there are k augmentations and each alternating path is length at most $2k - 1$, the total time spent on bookkeeping the matching is $O(k^2)$. This leaves the analysis of the subroutine which updates the potentials and finds an admissible augmenting path, called the *Hungarian search*.

Algorithm 1 Hungarian Algorithm

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

Theorem 4 (Hungarian algorithm time) *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 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: $\forall (a, b) \in M, c_\pi(a, b) = 0$

1: **function** HUNGARIAN-SEARCH($G = (A \cup B, E)$, M, π)
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: $\gamma \leftarrow c_\pi(a', b')$
6: $\pi(v) \leftarrow \pi(v) + \gamma, \forall v \in S$ ▷ make (a', b') admissible
7: $S \leftarrow S \cup \{b'\}$
8: **if** $b' \notin V(M)$ **then** ▷ b' unmatched
9: $\Pi \leftarrow$ alternating augmenting path in S to b'
10: **return** Π
11: **else** ▷ b' is matched to some $a'' \in A \cap V(M)$
12: $S \leftarrow S \cup \{a''\}$
13: **end if**
14: **until** $S = (A \cup B)$
15: **return** failure
16: **end function**

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 *There are $\leq k$ edge relaxations before the Hungarian search finds an alternating augmenting path.*

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

In non-geometric 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 in each Hungarian search — edges must be pushed into the queue even if they are not relaxed. We avoid this problem by finding the minimum edge using a *bichromatic closest pair* (BCP) query on additively weighted Euclidean distances, for which there exist fast (poly-logarithmic query and update) dynamic data structures. The BCP problem is to find, between two point sets $P, Q \subseteq \mathbb{R}^2$, the $p \in P$ and $q \in Q$ minimizing $\|p - q\| - \omega(p) + \omega(q)$, for some real-valued vertex weights $\omega(p)$; a perfect fit for reduced cost. The state of the art for dynamic BCP data structures — from Kaplan, Mulzer, Roditty, Seiferth, and Sharir [3] — inserts and deletes points

in $O(\text{polylog}(n))$ time, and answers queries in $O(\log^2 n)$ time. The following lemma, combined with Theorem 4, completes the proof of Theorem 1.

Lemma 4 *Using a dynamic BCP, we can implement Hungarian search with $T(n,k) = O(k\,\text{polylog}(n))$ and $P(n,k) = O(n\,\text{polylog}(n))$.*

We will maintain a BCP between $P = (S \cap A)$ and $Q = (B \setminus S)$. Changes to the BCP sets are entirely driven by changing S, i.e. updates to S incur BCP insertions/deletions. We first analyze the bookkeeping outside of dual updates, and then show how dual updates can be done efficiently.

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

2. During each Hungarian search, points are added to P (A points added to S) and removed from Q (B points added to S) — at most one of each per edge relaxation. By Lemma 3 the number of relaxed edges is at most k, so the number of these BCP operations is also at most k.

3. To obtain the assumption used in (1), we keep a log of the points added since S_0^t in the last Hungarian search (i.e. those of (2)). After the augmentation, we use this log 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 costs $O(k)$ BCP operations.

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

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. Throughout the course of the Hungarian algorithm, we maintain a value δ (initially 0) which aggregates dual changes. Vertices that are added to S are saved into P with weight $\omega(p) \leftarrow \pi(p) - \delta$. When the points of S have their duals increased in (2), we instead raise $\delta \leftarrow \delta + c_\pi(\hat{a}, \hat{b})$. Thus, the "true" potential for any point in S is $\omega(p) + \delta$. For points outside S (i.e. $B \setminus S$), we simply use the true potential as the BCP weight. Since all $S \cap A$ BCP weights are uniformly offset from their potential by δ, this change does not alter the BCP result. Once a point is removed from S, we update its true potential as $\pi(p) \leftarrow \omega(p) + \delta$.

Obviously, 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 save the true potential is bounded by the number times we remove a point from S. By the previous argument (2), this is $O(k)$ per Hungarian search as well. The total time for potential updates per Hungarian search is therefore $O(k)$. Overall, the time per Hungarian search is $T(n,k) = O(k\,\text{polylog}(n))$.

4 Matching with the Goldberg *et al.* algorithm

The basis of the algorithm in this section is a *cost-scaling* algorithm for unit-capacity min-cost flow from [1]. 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 to partial matching.

4.1 MPM to unit-capacity MCF reduction

For a partial matching problem on $G = (A \cup B, E_0)$ with parameter k, we direct the bipartite edges of E_0 from $(A \to B)$, with costs equal to the original cost $c(a, b)$ and capacity 1. Next, we add a dummy vertex s with arcs (s, a) to every $a \in A$, and respectively a dummy vertex t with arcs (b, t) for each $b \in B$, all with arc 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. Let the resulting graph be $H = (V, E)$.

Observation 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 5 *Let f be an integer pseudoflow in H with $O(k)$ excess, and let $E_{>0}(f) = \{(v, w) \mid f(v, w) > 0\}$. Then, $|E_{>0}(f)| = O(k)$.*

By Observation 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, then, there are $O(k)$ such paths. The maximum length of any path of positive-flow arcs in H is 3, by the capacities in the 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 $M_f \subseteq E$ 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.

For cost approximation, we can improve Lemma 2 on H. Note that for integer-valued (e.g. unit) capacities, there is always an integer-valued optimal circulation.

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

We label the arcs of H_f 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 be reverse arcs. The reverse arcs may either be (i) on one of the M_f edges or else (ii) between $\{s\} \times A$ or (iii) between $B \times \{t\}$. 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 charge per edge of $M_f \cap \Gamma$. We can then charge all arcs of $f' = (f^* - f) = \sum \Gamma_i$ 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.

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 6 into a relative approximation.

Corollary 1 *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 2 *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, ε′)
2:      ε ← kC
3:      f ← 0
4:      π ← 0
5:      repeat
6:          (f, π) ← SCALE-INIT(H, f, π)
7:          (f, π) ← REFINE(H, f, π)
8:          ε ← ε/2
9:      until ε ≤ ε′/6
10:     return f
11: end function
```

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 7 *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. 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

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

Algorithm 4 Scale Initialization

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

  for all $(v, w) \in E_f$ **do**
    if $c_\pi(w, v) < -\varepsilon$ **then**
      $f(v, w) \leftarrow 0$
    end if
  end for
  return (f, π)
end function

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

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 $O(k)$ 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.

Algorithm 5 Refinement

function REFINE($H = (V, E)$, f, π)
  while $\sum_{v \in V} |e_f(v)| > 0$ **do**
    $\pi \leftarrow$ HUNGARIAN-SEARCH2(H, f, π)
    $f' \leftarrow$ DFS(H, f, π) ▷ f' is an admissible blocking flow
    $f \leftarrow f + f'$
  end while
  return (f, π)
end function

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 9 (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 5 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 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 10 *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 13, 14), the total running time for REFINE is ultimately $O((n + k\sqrt{k})\,\mathrm{polylog}(n))$. Combining with Lemmas 7 and 8 completes the proof of Theorem 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 that (s,v,b) is an empty 2-path *surrounding* empty vertex v. Separately, we consider the length 3 s-t paths that

Algorithm 6 Hungarian Search (cost-scaling)

function HUNGARIAN-SEARCH2($H = (V, E)$, f, π)
  $S \leftarrow \{v \in V \mid e_f(v) > 0\}$
  repeat
    $(v', w') \leftarrow \arg\min\{c_\pi(v, w) \mid (v, w) \in S \times (V \setminus S)\}$
    $\gamma \leftarrow c_\pi(v', w')$
    if $\gamma > 0$ **then** ▷ make (v', w') admissible if it isn't
      $\pi(v) \leftarrow \pi(v) + \gamma\lceil\frac{\gamma}{\varepsilon}\rceil, \forall v \in S$
    end if
    $S \leftarrow S \cup \{w'\}$

    if $e_f(w') < 0$ **then** ▷ w' is a deficit
      return (f, π)
    end if
  until $S = (A \cup B)$
  return failure
end function

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 $v_2 \in B_\emptyset$. 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 $t \notin S$.

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

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 $O(|E_{>0}(f)|)$.

Lemma 12 *There are $O(k)$ empty 2- and 3-path relaxations in* HUNGARIAN-SEARCH2 *before a deficit vertex is reached.*

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 with $f(e) > 0$. We charge this relaxation to b if it is deficit, or the adjacent $f(e) > 0$ edge otherwise. No $f(e) > 0$ edge is charged more than twice, therefore the total number of empty 2-path relaxations surrounding $A_\emptyset$ vertices is $O(|E_{>0}(f)|)$.

Lemma 13 *Using a dynamic BCP, we can implement* HUNGARIAN-SEARCH2 *with $T_1(n, k) = O(k \operatorname{polylog}(n))$ and $P_1(n, k) = O(n \operatorname{polylog}(n))$.*

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 4 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.

Unlike matchings, there are now some arcs that are eligible for the minimum but not "captured" by the BCP. Specifically, it makes no sense to place s or t in the BCP (they do not correspond to points in the plane), or the arcs from $B \to A$ (arcs of $(S \cap B) \times (V \setminus S)$ are not residual unless there is positive flow on the respective $A \to B$ arc). We label these special classes of arcs (i) E_s^+ and E_s^-,

residual arcs with s on the head and tail respectively, (ii) E_t^+ and E_t^-, residual arcs with t on the head and tail respectively, and (iii) E_{BA}, $B \to A$ residual arcs. There are $O(r)$ arcs in E_s^+ and E_s^-, $O(n)$ arcs in E_t^+ and E_t^-, and $O(k)$ arcs in E_{BA} (by Lemma 5). We maintain a min-heap for each arc set on the same distances used by the BCP ($c(v,w) - \omega(v) + \omega(w)$). Until s (resp. t) is included into S, we "deactivate" E_s^+ (resp. E_t^+) and do not query it for the minimum. Once s (resp. t) is added, we activate E_s^+ (resp. E_t^+) and deactivate E_s^- (resp. E_t^-). These activations/deactivations occur once per Hungarian search, since vertices do not leave S once they enter. When evaluating the argmin, we take the minimum over all active heaps plus the BCP.

We can ultimately charge all heap operations to BCP operations, and see that there are $O(k)$ total. When a vertex moves into S, it begets a heap operation in addition to the BCP deletion. For example, if $a \in A$ moves into S, (s,a) should be deleted from E_t^-, or (a,s) should be added to E_t^+

For potential updates, we use the same trick as in Lemma 4 to lazily update potentials after vertices leave S. To remind, these potential updates were bounded by the number of times a point left S.

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.

Algorithm 7 Depth-first search

function DFS($H = (V, E)$, f, π)
  $f' \leftarrow 0$.
  $S \leftarrow \{v \in V \mid e_f(v) > 0\}$
  $P \leftarrow \emptyset$
  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
      $P \leftarrow \emptyset$
    end if

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

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

Lemma 14 *Using a dynamic NN, we can implement* DFS *with* $T_2(n,k) = O(k \operatorname{polylog}(n))$ *and* $P_2(n,k) = O(n \operatorname{polylog}(n))$.

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

Lemma 15 (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* $O(n^2)$.

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.