A divide-and-conquer algorithm for min-cost perfect matching in the plane[*]

Kasturi R. Varadarajan[†]

May 4, 1998

Abstract

Given a set V of $2n$ points in the plane, the *min-cost perfect matching* problem is to pair up the points (into n pairs) so that the sum of the Euclidean distances between the paired points is minimized. We present an $O(n^{3/2} \log^5 n)$-time algorithm for computing a min-cost perfect matching in the plane, which is an improvement over the previous best algorithm of Vaidya [21] by nearly a factor of n. Vaidya's algorithm is an implementation of the algorithm of Edmonds [9], which runs in n phases, and computes a matching with i edges at the end of the i-th phase. Vaidya shows that geometry can be exploited to implement a single phase in roughly $O(n^{3/2})$ time, thus obtaining an $O(n^{5/2} \log^4 n)$-time algorithm. We improve upon this in two major ways. First, we develop a variant of Edmonds' algorithm that uses geometric divide-and-conquer, so that in the conquer step we need only $O(\sqrt{n})$ phases. Second, we show that a single phase can be implemented in $O(n \log^5 n)$ time.

[*] Work on this paper has been supported by National Science Foundation Grant CCR-93–01259, by an Army Research Office MURI grant DAAH04-96-1-0013, by a Sloan fellowship, by an NYI award, by matching funds from Xerox Corporation, and by a grant from the U.S.–Israeli Binational Science Foundation.

[†] Department of Computer Science, Box 90129, Duke University, `krv@cs.duke.edu`

1 Introduction

Given a set V of $2n$ points in the plane, we can associate a complete undirected graph $G(V)$ (or simply G) with V as follows. The vertex set of G is the set of points V, and its edge set E consists of all unordered pairs (u, v) such that $u, v \in V$ and $u \neq v$. The *cost* of an edge (u, v) is the Euclidean distance $d(u, v)$ between u and v. A *matching* of G (or of V) is a collection M of edges such that no vertex in V is incident on more than one edge in M. A *perfect* matching of V is a matching M in which every vertex in V is incident on *exactly* one edge M. Note that a perfect matching of V has cardinality n. We define the *cost of a matching* M to be $\sum_{(u,v)\in M} d(u, v)$, the sum of the costs of the edges in M. The *Euclidean min-cost perfect matching problem* (MCPM) is to find a perfect matching of V whose cost is the smallest.

The MCPM problem has applications in operations research, pattern recognition, statistics, and VLSI (see [16]). The problem is used in determining the efficient movement of mechanical plotters, which is a special case of the Chinese postman problem [10]; see the survey by Avis [6]. The fact that MCPM and related problems can be solved in polynomial time for general graphs is a classical and fundamental result due to Edmonds [9]. Lawler [14] gave an $O(|V|^3)$ implementation of Edmonds' algorithm; using this, the MCPM problem in the plane can be solved in $O(n^3)$ time. The question that motivates us is whether we can exploit geometry to do much better. (Note that the complete graph induced by the set of $2n$ points is entirely specified by the co-ordinates of the points.)

Since the min-cost, max-cardinality problem can be solved for sparse graphs in $O(|E||V| \log |V|)$ time (Galil et al. [12]), there have been attempts at showing that the min-cost perfect matching in the plane is a substructure of geometric structures such as the Delaunay triangulation. Counterexamples to several such conjectures were given by Akl [3]. (Note that the Euclidean minimum spanning tree is contained in the Delaunay triangulation [19] and Yao's graph [22].) Vaidya [21] was the first to show that geometry can be exploited to get a sub-cubic algorithm; his $O(n^{5/2} \log^4 n)$-time algorithm is the best known for Euclidean MCPM.

For the bipartite version of this problem, Agarwal et al. [1] have given a near-quadratic algorithm that improves over an earlier sub-cubic algorithm of Vaidya [21]. Attention has been paid to special cases of the Euclidean MCPM, for instance the case when all the points are in convex position; see Marcotte and Suri [16], and Buss and Yianilos [7] where near-linear time algorithms have been given for such problems. There has also been considerable amount of work on approximation algorithms for Euclidean matching; see Junger and Pulleyblank [13], the survey by Avis [6], and the references therein. A lot of this work looks at the case where the points are in a unit sqaure, and aims at producing a matching whose *absolute cost* is small. In contrast, Vaidya [20] gave an algorithm that runs in roughly $O(n^{3/2}/\varepsilon^3)$ time and returns a perfect matching whose cost is at most $(1+\varepsilon)$ times the optimal, for any $\varepsilon > 0$. The recent algorithm of Arora [4] solves the same problem in time that is near-linear in n, but is exponential in $1/\varepsilon$.

The huge literature on matchings in general graphs is outside the scope of this paper. We refer the reader to standard books on combinatorial optimization ([14],[18]) and matching theory ([15]).
Our results. We present an $O(n^{3/2} \log^5 n)$-time algorithm for computing a min-cost perfect matching in the plane, which is an improvement over the previous best algorithm of Vaidya [21] by nearly a factor of n. Vaidya's algorithm is an implementation of the algorithm of Edmonds [9], which runs in n phases, and computes a matching with i edges at the end of the i-th phase. Vaidya shows that geometry can be exploited to implement a single phase in $\tilde{O}(n^{3/2})$ time (we use the $\tilde{O}()$ notation when ignoring log-factors), thus obtaining an $O(n^{5/2} \log^4 n)$-time algorithm. We improve upon this

in two major ways. First, we develop a variant of Edmonds' algorithm that uses geometric divide-and-conquer, so that in the conquer step we need only $O(\sqrt{n})$ phases. (Divide-and-conquer has been used before for special cases of MCPM in the plane, for instance by Marcotte and Suri [16], but these approaches rely heavily on the properties of the special cases.) The geometric tool that we use for divide-and-conquer is based on the technique of Miller et al. [17] for finding geometric separators for overlap graphs. Second, we show a single phase for an n-point set can be implemented in $O(n \log^5 n)$ time. To do this, we interpret the dual variables geometrically and establish certain nice properties that they exhibit. We then exploit these properties to show that to implement a single phase, it suffices to look at a subset of $\tilde{O}(n)$ *candidate edges*, and not all the $n(n-1)/2$ edges. The candidate edges are not known at the beginning of the phase itself, but are generated as the phase unfolds, using a total of $\tilde{O}(n)$ time. Combining this with the data structures of Galil et al. [12] for implementing a phase of the matching algorithm for sparse graphs in $\tilde{O}(|E|)$ time, we obtain an $O(n \log^5 n)$-time implementation of a phase. For generating the candidate edges, we introduce a notion called the *semi-separated decomposition*, which is a relaxation of the *well-separated decomposition* of Callahan and Kosaraju [8].

In Section 2, we present our divide-and-conquer algorithm for MCPM, and show that only $\sqrt{n}$ phases are needed in the conquer step for a set of n points. In Section 3, we describe our approach for implementing a single phase of the algorithm. For lack of space, we have not provided all the proofs and details; we present some of them in an appendix, and have omitted some of them.

2 A divide and conquer framework for matching

In this section, we present a divide-and-conquer approach for min-cost perfect matching of the set of points V in the plane. We assume in the following that we are dealing with the graph $G(V) = (V, E)$ associated with the given set of points V. We say that a subset $Q \subseteq V$ of V is an *odd subset* or an *odd-set* if $|Q|$ is odd and $|Q| \geq 3$. For $Q \subseteq V$, let $\xi(Q)$ denote the subset of edges E with exactly one endpoint in Q, that is, $\xi(Q) = \{(u,v) \in E : |\{u,v\} \cap Q| = 1\}$. Let $S(p,r)$ denote the disk of radius r centered at point p.

Edmonds' algorithm is motivated by duality theory for linear programs; see [9] and [14] for a discussion of linear programming duality. His algorithm associates a 'dual variable' variable ω_v for each $v \in V$ and a dual variable ω_Q for each odd set Q. Sometimes, it will be convenient to denote ω_v by $\omega_{\{v\}}$. Corresponding to edge (u,v), let $\pi_{uv} = \omega_u + \omega_v + \sum_{(u,v)\in\xi(Q)} \omega_Q$. From duality theory, it follows that a perfect matching M is optimal if there exist values ω_v, for each $v \in V$, and ω_Q, for each odd subset Q, such that the following conditions hold:

EDGE-FEASIBILITY: $\pi_{uv} \leq d(u,v)$ for each $(u,v) \in E$.

POSITIVE-DUAL: $\omega_Q \geq 0$ for each odd subset Q.

MATCHING-ADMISSIBILITY: $(u,v) \in M \Rightarrow \pi_{uv} = d(u,v)$.

MAXIMALITY: For each odd subset Q, if $\omega_Q > 0$, then the matching M is maximal within Q, that is, the number of edges in M both of whose endpoints are in Q is $(|Q|-1)/2$. Since M is a perfect matching, this is equivalent to $M \cap \xi(Q) = 1$.

Actually, we can prove this using a direct arguement. We simply note that the EDGE-FEASIBILITY and POSITIVE-DUAL conditions imply that the cost of any perfect matching is at least $\sum_{v\in V} \omega_v + \sum_Q \omega_Q$, while the conditions MATCHING-ADMISSIBILITY and MAXIMALITY imply that the cost of M is exactly $\sum_{v\in V} \omega_v + \sum_Q \omega_Q$.

Like Edmonds' algorithm, our approach also computes a perfect matching and a corresponding set of dual variables such that EDGE-FEASIBILITY, POSITIVE-DUAL, MATCHING-ADMISSIBILITY, and MAXIMALITY are satisfied. The difference is that unlike in Edmonds' algorithm, we use geometric divide-and-conquer for doing this. Before describing our approach, we describe the important notion of blossoms that was introduced by Edmonds. Our description of blossoms and other standard components of the matching algorithm are based on the presentation of Galil et al.[12].

Definition 2.1 For any vertex $v \in V$, let $\lambda(v) = \omega_v + \sum_{v \in Q} \omega_Q$. An edge (u, v) is *feasible* if $\pi_{uv} \leq d(u, v)$. It is *admissible* if $\pi_{uv} = d(u, v)$.

2.1 Blossoms

During the course of our algorithm, certain odd subsets of V are designated as *blossoms*. The algorithm maintains the property that $\omega_Q > 0$ for an odd subset Q only if Q is a blossom. The set of blossoms at any stage have the following *nested* structure: For any two distinct blossoms B and B', either $B \cap B' = \emptyset$, or $B \subset B'$, or $B' \subset B$. Each $v \in V$ is a trivial blossom of size one. A non-trivial blossom B is given by a sequence of blossoms $B_0, \ldots, B_r$, where $r = 2k$, for $k \geq 1$, and a sequence of admissible edges $e_i = (u_{i-1}, v_i)$, for $i = 1, \ldots, r+1$, such that

1. $u_i, v_i \in B_{i \bmod (r+1)}$
2. For $1 \leq i \leq r+1$, $(u_{i-1}, v_i) \in M$ if i is even and $(u_{i-1}, v_i) \notin M$ if i is odd.

The blossoms $B_0, \ldots, B_r$ are referred to as the *subblossoms* of B. A blossom that is not a subblossom of any other blossom is called an *outermost* blossom. Clearly, the outermost blossoms induce a partition of V. It can be shown from the properties above that any blossom B contains an odd number of vertices, and that the matching M is maximal within B. The unique vertex of B that is not matched to any other vertex of B is called its *base*. The base can also be defined by induction on the structure of blossoms as follows. The base of a trivial blossom v is the vertex v itself. The base of a blossom B whose subblossoms are given by the sequence $B_0, \ldots, B_r$ (as above) is the base of B_0.

An *alternating path between vertices* v_0 and v_r is a sequence of admissible edges $e_i = (v_{i-1}, v_i)$, for $i = 1, \ldots, r$, such that for $i = 1, \ldots, r-1$, $e_i \in M$ if and only if $e_{i+1} \notin M$. In other words, it is a path in which alternate edges are in the matching. An *alternating path between outermost blossoms* B_0 and B_r is given by a sequence of admissible edges $e_i = (u_{i-1}, v_i)$, for $i = 1, \ldots, r$, and a sequence of outermost blossoms $B_0, \ldots, B_r$, where $u_i, v_i \in B_i$, and for $i = 1, \ldots, r-1$, $e_i \in M$ if and only if $e_{i+1} \notin M$. We say that a vertex v is *exposed* if no edge of the matching M is incident on v; an outermost blossom B is exposed if no edge of the matching M is incident on the base of B. An alternating path between two exposed vertices is called an *augmenting path*.

Lemma 2.2 *Let u and v be points in different outer blossoms. The edge (u, v) is feasible (that is, $\pi_{uv} \leq d(u, v)$) iff $\lambda(u) + \lambda(v) \leq d(u, v)$. The edge (u, v) is admissible (that is, $\pi_{uv} = d(u, v)$) iff $\lambda(u) + \lambda(v) = d(u, v)$.*

Proof: Follows from the fact that if u and v are in different outer blossoms, $\pi_{uv} = \lambda(u) + \lambda(v)$. □

We show later that throughout our algorithm, $\lambda(v) \geq 0$ for any $v \in V$. We define disk(v), the *disk* of vertex v, to be the disk of radius $\lambda(v)$ centered at v. Since $\lambda(v) \geq 0$, disk(v) is well defined. Lemma 2.2 tells us that if u and v are vertices in different blossoms, feasibility of (u, v) means that disk(u) and disk(v) do not overlap (although they can touch); admissibility of (u, v) means disk(u) and disk(v) do not overlap but touch. This geometric interpretation is due to Junger and Pulleyblank [13].

2.2 The divide-and-conquer algorithm

Let $U \subseteq V$ be a subset of the given set of points, and let $|U| = m$. We will describe our divide-and-conquer scheme for the set U. In our algorithm, we are also specified a *bound* $\mathrm{limit}(u)$ for each $u \in U$. (In the beginning, we set $\mathrm{limit}(v) = \infty$ for each $v \in V$ and call the divide-and-conquer procedure with U set to V.) The goal in the sub-problem for U is to compute a (not necessarily perfect) matching M of U, a set of blossoms in U, and a set of dual variables ω_u for each $u \in U$, and ω_Q for each blossom Q (the dual variables of odd sets that are not blossoms are assumed to be 0), so that (1) the conditions EDGE-FEASIBILITY, POSITIVE-DUAL, MATCHING-ADMISSIBILITY, and MAXIMALITY hold for U, and (2) in addition, the following two conditions are also satisfied:

RADIUS-CONSTRAINT: For each $u \in U$, $\lambda(u) \leq \mathrm{limit}(u)$.

EXPOSED-CONSTRAINT: For each exposed blossom Q of U, there is a $q \in Q$ such that $\lambda(q) = \mathrm{limit}(q)$.

Let us call a blossom Q of U *constrained* if Q is exposed and there is a $q \in Q$ such that $\lambda(q) = \mathrm{limit}(q)$; we say that Q is unconstrained otherwise. Thus the last condition says that every exposed blossom is constrained.

The divide step. Let C be a circle in the plane, and U_1 (resp. U_2) be the subset of U that lies *inside* (resp. *outside*) the circle C. For each $u \in U$, let $\beta(u)$ denote the distance from u to the circle C. We call C a *separating circle* for U if the following conditions hold:

1. $\min\{|U_1|, |U_2|\} \geq |U|/4$.
2. Let $W \subseteq U$ be any subset of points such that the family of disks $\{S(w, \beta(w)) | w \in W\}$ has the property that any two disks in it have disjoint interiors. Then, $|W| = O(\sqrt{m})$.

We argue briefly in the appendix that a separating circle for U exists and can be computed in $O(m)$ time.

If the set U contains more c points, for some constant c, we find a separating circle C for U that partitions U into two non-empty sets U_1 and U_2 as above. We recurse on the set U_1 with the bound for each $u \in U_1$ set to $\mathrm{newlimit}(u) = \min\{\mathrm{limit}(u), \beta(u)\}$. We recurse on the set U_2 with the bound for each $u \in U_2$ set to $\mathrm{newlimit}(u) = \min\{\mathrm{limit}(u), \beta(u)\}$.

Suppose that the recursive calls return a matching, blossoms, and dual variables for U_1 (resp. U_2) satisfying the six conditions for U_1 (resp. U_2). To begin the conquer step for U, we obtain an initial matching, dual variables, and blossoms by combining the matching, dual variables, and blossoms for U_1 and U_2. At this stage, it is easy to see that all the six conditions except the EXPOSED-CONSTRAINT are satisfied for U. We sketch the proof for the most interesting case, which is EDGE-FEASIBILITY for edge (u, u') where $u \in U_1$ and $u' \in U_2$. We have

$$\lambda(u) \leq \mathrm{newlimit}(u) = \min\{\mathrm{limit}(u), \beta(u)\} \leq \beta(u).$$

Similarly $\lambda(u') \leq \beta(u')$. Combining the inequalities, we have $\lambda(u) + \lambda(u') \leq \beta(u) + \beta(u')$. Since u and u' lie on opposite sides of the circle C, we can conclude that $\beta(u) + \beta(u') \leq d(u, u')$. Thus, $\lambda(u) + \lambda(u') \leq d(u, u')$; geometrically, what we have shown is that disk(u) and disk(u') do not overlap. Since u and u' are obviously in different outer blossoms, it follows from Lemma 2.2 that (u, u') is feasible.

Observe that the EXPOSED-CONSTRAINT condition may be violated for a blossom Q of U. The 'conquer' stage of the divide-and-conquer algorithm for U eliminates the violations of the EXPOSED-CONSTRAINT, thus 'solving' the sub-problem for U. The 'conquer' stage consists of a series of phases; in each phase the number of exposed, unconstrained blossoms, is reduced by either one or two.

Base case. The base case for the divide-and-conquer is when $|U| \leq c$. To solve the base case, we initialize the matching on U to be empty, and set all the dual variables to be zero. The only blossoms of U are the trivial blossoms, and these are considered to be exposed and unconstrained. We then execute the algorithm for the 'conquer' stage for U, which we now describe.

2.3 The conquer stage

As we indicated, the conquer stage consists of phases. Each phase begins with the current matching M, a set of dual variables, and a set of blossoms. Some of the exposed blossoms are constrained, and are called c-blossoms. The algorithm always maintains the five conditions EDGE-FEASIBILITY, POSITIVE-DUAL, MATCHING-ADMISSIBILITY, MAXIMALITY, and RADIUS-CONSTRAINT. In each phase, the number of exposed, unconstrained, blossoms is decreased by one or two. Thus, each phase decreases the number of violations of the sixth condition EXPOSED-CONSTRAINT, and so the algorithm terminates after a finite number of phases.

During a phase, some unconstrained outer blossoms are *labelled* as s-blossoms and t-blossoms. (An outer blossom is labelled as either an s-blossom or a t-blossom, but not both.) An unconstrained outer blossom which is not labelled is called a *free* blossom or f-blossom. (s-, t-, and f- prefixes are only for unconstrained blossoms.) A vertex is called an s-vertex, t-vertex, or f-vertex according to whether it belongs, respectively, to an s-blossom, t-blossom, or f-blossom. We let S, T, and F denote, respectively, the set of s-vertices, t-vertices, and f-vertices. For any $v \in V$, let $b(v)$ denote the outermost blossom containing V.

A phase is divided into $O(m)$ *sub-phases.* At the end of each sub-phase, the following invariants hold. An exposed, unconstrained blossom is always an s-blossom. For every s- or t- blossom B, there is an alternating path $\sigma(B', B)$ between an exposed, unconstrained blossom B' and B. If B is an s-blossom, $\sigma(B', B)$ has even length, that is, there are an even number of edges in the alternating path. If B is a t-blossom, $\sigma(B', B)$ has odd length. The s- and t-blossoms, together with the corresponding alternating paths, induce a forest of rooted trees, a tree being rooted at each exposed, unconstrained blossom. The trees are called *alternating trees*, and the forest is called an *alternating forest.* (The c-blossoms are not in the alternating forest.) The leaves of the alternating trees are always s-blossoms.

For every f-blossom B, there is another f-blossom C such that there is an edge in matching M between the bases of B and C. That is, M induces a perfect matching on the bases of all the f-blossoms.

At the start of the phase, we label each exposed, unconstrained blossom as an s-blossom; every other unconstrained outer blossom is an f-blossom. A sub-phase consists of the following loop, which is repeated until a termination condition for the phase is met. The above invariants hold at the end of each iteration of the loop. Let

$$\delta_1 = \min_{Q \text{ a nontrivial } t\text{-blossom}} \omega_Q \qquad \delta_2 = \min_{u \in S, v \in F}(d(u,v) - \pi_{uv})$$
$$\delta_3 = \min_{u,v \in S;\ b(u) \neq b(v)}(d(u,v) - \pi_{uv})/2 \quad \delta_4 = \min_{u \in S, v \text{ a } c\text{-vertex}}(d(u,v) - \pi_{uv})$$
$$\delta_5 = \min_{u \in S}(\text{limit}(u) - \lambda(u))$$

and let $\delta = \min\{\delta_1, \delta_2, \delta_3, \delta_4, \delta_5\}$.

Dual change: Let ω_Q be the dual variable corresponding to the blossom Q. (If Q is a trivial blossom consisting of a vertex v, then $\omega_Q = \omega_v$.) For each s-blossom Q, we increase ω_Q by δ, and for each t-blossom Q, we decrease ω_Q by δ. After the dual change, one of δ_1, δ_2, δ_3, δ_4, or δ_5 becomes zero. (In case of a tie, we pick an arbitarary δ_i that is zero.) We will be terse about some of the following cases, which are standard; see [12]

$\delta_1 = 0$: In this case, the dual variable ω_B corresponding to a (non-trivial) t-blossom B becomes zero. We expand B, that is, we stop regarding it as a blossom and make its subblossoms outer blossoms. Some of these new outer blossoms become s-blossoms, some become t-blossoms, and some f-blossoms.

$\delta_2 = 0$: In this case, an edge (u, v), which is now admissible, between an s-vertex u and an f-vertex v has been discovered. Two f-blossoms are added to the alternating forest, one as a t-blossom and the other as an s-blossom.

$\delta_3 = 0$: An edge (u, v) which is now admissible has been discovered between s-vertices u and v. Either a new s-blossom is formed, or an alternating path between two exposed, unconstraned blossoms is discovered. The latter subcase ends the phase and is handled in a manner similar to the case where $\delta_4 = 0$.

$\delta_4 = 0$*:* An edge (u, v), which is now admissible, has been discovered between an s-vertex u and a c-vertex v. Let A (resp. B) be the s-blossom (resp. c-blossom) containing u (resp. v). Let A' be the exposed, unconstrained, blossom which is the root of the alternating tree containing A, and let $\sigma(A', A)$ denote the corresponding even-length alternating path between A' and A. Note that $\sigma(A', A)$, the edge (u, v), and the blossom B together constitute an alternating path between the exposed blossoma A' and B. We expand this to an alternating path π between the exposed bases of A' and B. We augment the current matching M by excluding all edges of M belonging to π and including the other edges of π. Note that the cardinality of the matching M increases by one, and the number of exposed, unconstrained, blossoms falls by one since A' is now no longer exposed. We also change appropriately the bases of all the blossoms through which the augmenting path passes. This ends the current phase of the algorithm.

$\delta_5 = 0$*:* In this case, $\lambda(u)$ has increased to limit(u) for an s-vertex u. Let A be the s-blossom containing u. Let A' be the exposed, unconstrained, blossom which is the root of the alternating tree containing A, and let $\sigma(A', A)$ denote the corresponding even-length alternating path between A' and A. We expand $\sigma(A', A)$ to an even-length alternating path π between the bases of A' and A. We alter the current matching M by excluding all edges of M belonging to π and including the other edges of π. We change appropriately the bases of all the blossoms through which the augmenting path passes. This ends the current phase of the algorithm. We can show that the cardinality of the matching M remains unchanged, and the number of exposed, unconstrained, blossoms falls by one. Note that in the next phase, A is constrained.

This completes the description of a phase. At the end of the phase, we (recursively) expand all outer blossoms whose dual variable is zero.

This also completes our description of the overall divide-and-conquer scheme for min-cost perfect matching.

Lemma 2.3 *The number of phases in the conquer step for U is $O(\sqrt{m})$.*

Proof: Let $\mathcal{E}$ denote the number of exposed, unconstrained blossoms at the beginning of the conquer step. Since each phase decreases the total number of exposed, unconstrained blossoms by one or two, the number of phases is at most $|\mathcal{E}|$. Hence it suffices to show $|\mathcal{E}| = O(\sqrt{m})$. To do this, we will use the properties of the separating circle C. We first argue that for each $Q \in \mathcal{E}$, there is a $q \in Q$ such that $\lambda(q) = \beta(q)$. Assume, w. l. o. g, that $Q \subseteq U_1$. Since Q is exposed, the condition

EXPOSED-CONSTRAINT for U_1 implies that there is a $q \in Q$ such that

$$\lambda(q) = \text{newlimit}(q) = \min\{\text{limit}(q), \beta(q)\}.$$

Since Q is unconstrained at the beginning of the conquer step for U, $\lambda(q) < \text{limit}(q)$. It follows that $\lambda(q) = \beta(q)$.

Consider the family of disks formed by picking for each $Q \in \mathcal{E}$ a disk $S(q, \beta(q))$ such that $q \in Q$ and $\beta(q) = \lambda(q)$. From Lemma 2.2 and the fact that the EDGE-FEASIBILITY condition holds, we see that this family of disks have pairwise disjoint interiors. The second property of the separating circle C implies that there are only $O(\sqrt{m})$ disks in the family. We conclude that $|\mathcal{E}| = O(\sqrt{m})$. $\square$

A phase of our 'conquer' step is quite similar to a phase in Edmonds' algorithm, except that we also need to deal with constrained blossoms and the RADIUS-CONSTRAINT condition. For a fast implementation of one phase of the conquer algorithm (or of Edmonds' algorithm), we need a mechanism to quickly compute when δ_i becomes zero. As in a phase of Edmonds' algorithm, handling δ_2 and δ_3 seem to be the hard cases. We can easily maintain δ_1 and δ_5 in a total of $\tilde{O}(n)$ per phase, as this involves only the dual variables corresponding to $O(n)$ blossoms. We can maintain δ_5 efficiently using a data-structure for answering closest point queries [5]. Maintaining δ_2 and δ_3 using such an approach is more problematic because of the way the blossoms and the labels change. However, Vaidya [21] showed that geometry can be exploited to maintain δ_i using a total of $\tilde{O}(m^{3/2})$ time per phase (in Edmonds' algorithm), thus obtaining a running time of $\tilde{O}(n^{5/2})$ for MCPM. In Section 3, we show that we can detect when δ_i becomes zero using a total of $O(m \log^5 m)$ time per phase (we can show this for a phase in Edmonds' algorithm as well). The following theorem results from a careful implementation of a phase, similar to the implementation of a phase of Edmonds' algorithm described by Galil et al. [12] or Vaidya [21].

Theorem 2.4 *Suppose that we can detect when δ_i becomes zero using a total of $O(\lambda)$ time in a single phase of the conquer step for U. Then, one phase can be implemented in $O(m \log m + \lambda)$ time, where $m = |U|$.*

Thus, a phase of the conquer step takes $O(m \log^5 m)$ time. As there are $O(\sqrt{m})$ phases, the conquer step takes $O(m^{3/2} \log^5 m)$ time. Since a separating circle for U can be found in $O(m)$ time, we conclude that the time for solving the sub-problem for U, not counting the time for solving the recursive sub-problems U_1 and U_2, is $O(m^{3/2} \log^5 m)$. Since $|U_1|, |U_2| \geq |U|/4$ (first property of separating circle), a standard analysis tells us that the overall time needed to solve the sub-problem for U is $O(m^{3/2} \log^5 m)$. Putting everything together, we conclude:

Theorem 2.5 *A min-cost perfect matching of a set V of $2n$ points in the plane can be computed in $O(n^{3/2} \log^5 n)$ time.*

3 Implementing a phase

In this section, we describe an efficient algorithm for implementing a single phase of the conquer step for U. We begin by making some useful observations about our algorithm. Some other geometric observations needed for the correctness of our algorithm are presented in the appendix. The following lemma uses the triangle inequality for distances in the Euclidean metric.

Lemma 3.1 *For any vertex $v \in V$, $\omega_v \geq 0$ at all stages in the algorithm. Consequently, $\lambda(v) \geq 0$, for all $v \in V$.*

Definition 3.2 The *time* at any point in a single phase of the algorithm is the sum $\sum \delta$ of all the dual changes made by the algorithm since the beginning of the phase. That is, the time at the beginning of the phase is zero, and each dual change step increments the time by δ.

Within a single phase, the dual variables, and the quantities that depend on them, can be regarded as functions of time. Hence, we will denote by $\mu[t]$ the value of a dual variable μ at time t of the algorithm. We will do the same for quantities that depend on the dual variables. The following observation depends on the fact that the algorithm increases the dual variables corresponding to the s-blossoms, decreases the dual variables corresponding to the t-blossoms, and does not change the dual variables corresponding to the f-blossoms. It also expresses a property of the algorithm's labelling scheme.

Fact 3.3 *During a phase, a vertex v may change its status from an f-vertex to a t-vertex (and vice versa) a number of times. In this part of the phase, $\lambda(v)$ can only decrease. However, once v becomes an s-vertex, it remains an s-vertex until the end of the phase. In this part of the phase, $\lambda(v)$ can only increase. If v belongs to a c-blossom, $\lambda(v)$ does not change at all during the phase.*

Recall that we defined disk(v) to be the disk of radius $\lambda(v)$ centered at v. Since $\lambda(v) \geq 0$ (Lemma 3.1), disk(v) is well defined. Lemma 2.2 tells us that if u and v are vertices in different blossoms, disk(u) and disk(v) do not overlap (although they can touch). Thus, the question of detecting when δ_2, δ_3, or δ_4 becomes zero (as a consequence of dual changes) boils down to detecting when disks of points in different blossoms touch.

3.1 Candidates

To detect when δ_2, δ_3, or δ_4 becomes zero during a phase, we could 'monitor' all the edges (u, v) and detect when disk(u) and disk(v) touch. In this section, we show that it is sufficient to monitor a certain set of $\tilde{O}(n)$ *candidate* edges. This is shown in Lemma 3.7, the main result of this section. To prove this result, we use the properties established above. The candidate edges are not known at the beginning of the phase itself, but are generated as the phase progresses. Before we can specify how the candidate edges are generated, we need to introduce a certain cover of the set of edges.

A semi-separated decomposition Let $C(p, r)$ denote the closure of $\mathbb{R}^2 - S(p, r)$, where $S(p, r)$ is the disk of radius r centered at p. We say that two point sets A and B are *semi-separated* if there exists a point p and a real number $r \geq 0$ so that

1. $A \subseteq S(p, r)$, and
2. $B \subseteq C(p, sr)$. Here, s is the *separation constant*, assumed throughout to be fixed to a constant greater than 1. (For this paper, we take $s = 9$.)

A set $\{(A_1, B_1), \ldots, (A_k, B_k)\}$ of pairs is said to be a *semi-separated decomposition* (SSD) of U if

1. For any edge (u, v) of $G(U)$, there is a pair (A_i, B_i) such that either $u \in A_i$ and $v \in B_i$, or $v \in A_i$ and $u \in B_i$.
2. A_i and B_i are semi-separated, for all $i = 1, \ldots, k$. Let p_i denote the point and r_i the radius such that $A_i \subseteq S(p_i, r_i)$, and $B \subseteq C(p_i, sr_i)$.

For the pair (A_i, B_i) of the SSD, we will refer to p_i as the *center* and r_i the *radius* corresponding to (A_i, B_i). The *size* of the semi-separated decomposition is $\sum_i(|A_i| + |B_i|)$. Note that the SSD is similar to the well-separated decomposition of Callahan and Kosaraju [8]. In fact, any well-separated decomposition of U is an SSD of U. Our weaker notion of an SSD is motivated by the fact that the size (according to our definition) of any well-separated decomposition of certain m-point is $\Omega(m^2)$. (See [8].) In contrast, we present in Section 7 a scheme to construct an SSD of U whose size is $O(m \log^4 m)$.

Let $\theta = 2\pi/h$, where h is a sufficiently large integer constant. We refine the SSD as follows. Assume that for the semi-separated pair (A_i, B_i), $A_i \subseteq S(p_i, r)$ and $B_i \subseteq C(p_i, sr)$, for some point p_i and $r > 1$. We subdivide the plane into h cones $c_1, \ldots, c_h$ such that each c_j has p_i has its apex and an angular opening of θ. Let $D(j) = B_i \cap c_j$ denote the set of points in B_i that are contained in the cone c_j. We replace each pair (A_i, B_i) in the original SSD by the set of pairs $(A_i, D(j))$, for $1 \leq j \leq h$ to obtain the *refined semi-separated decomposition* (RSSD) of U. See Figure 1 at the beginning of the appendix for an illustration. We define the center and radius of $(A_i, D(j))$, for $1 \leq j \leq h$, to be the center and radius of (A_i, B_i). We will refer to θ as the *angular constant* of our RSSD. (In this paper, we choose $\theta = 1/18$ radians.) In the description that follows, we assume that we have computed an RSSD of U whose size is $O(m \log^4 m)$. Using the algorithm described in Section 7 in the appendix, we can compute the RSSD in $O(m \log^5 m)$ time.

The event queue. We do not know what the candidate edges are at the beginning of the phase itself. Rather, we generate the candidates as the phase progresses, when certain 'events' occur. We maintain an *event-queue* to detect these events.

Definition 3.4 Let $\{(A_1, B_1), \ldots, (A_k, B_k)\}$ be an RSSD of the given set of points V. Consider a pair (A_i, B_i) with center p_i and radius r_i. We pick an arbitarary point $a_i \in A_i$ as the *representative* of A_i. Let $b_i \in B_i$ be the point in B_i that is closest to p_i, and let $\ell_i = d(p_i, b_i)$ denote the Euclidean distance between b_i and p_i. We let b_i be the representative of B_i.

There can be two entries in the event queue corresponding to the pair (A_i, B_i). The representative a_i is present if a_i is an s-vertex and $\lambda(a_i) \leq 2r_i$. We define the *priority* of a_i to be $2r_i - \lambda(a_i)$. The representative b_i is present if b_i is an s-vertex and $\lambda(b_i) \leq \theta\ell_i + 3r_i$; here, θ is the angular constant of the RSSD. We define the priority of b_i to be $\theta\ell_i + 3r_i - \lambda(b_i)$. (The entry corresponding to a_i (resp. b_i) is there to detect the event when $\lambda(a_i)$ increases to $2r_i$ (resp. $\lambda(b_i)$ increases to $\theta\ell_i + 3r_i$).) Note that the priorities of all the entries in the event queue are non-negative. Also, the priorities decrease uniformly with time, because the disks of s-vertices grow uniformly with time. When the priority of an entry becomes zero, it is removed from the event queue.

Generation of candidates. We now describe how the candidates are generated during a phase. At the beginning of the phase, we use the dual variables to compute $\lambda(v)$, for each point $v \in V$. We generate an initial set of candidate edges by examining each pair (A_i, B_i) of the RSSD as follows. If $\lambda(a_i) \geq 2r_i$, we execute the procedure Generate-candidates(A_i, B_i) described below. If $\lambda(b_i) \geq \theta\ell_i + 3r_i$, we execute the procedure Generate-candidates(B_i, A_i).

As mentioned before, the other candidates are generated as the phase unfolds, when certain events are triggered. Such an event occurs when the priority of some element in the event-queue becomes zero (as a consequence of a change in the dual variables). When this happens, the element is removed from the event queue. Suppose the element corresponds to some pair (A_i, B_i) of the RSSD of U. If the element is the representative of A_i (resp. B_i), we first compute $\lambda(a)$, for each $a \in A_i$ (resp. $\lambda(b)$, for each $b \in B_i$). We then generate a set of candidates by calling the procedure

Generate-candidates(A_i, B_i) (resp. Generate-candidates(B_i, A_i)). To complete the description of our scheme for candidate generation, we now describe the procedure Generate-candidates.

Definition 3.5 For any point p and any $v \in V$, the weighted distance of p from v, denoted $\mathrm{wd}(v, p)$, equals $d(v, p) - \lambda(v)$.

Generate-candidates(X, Y): We assume that $\lambda(x)$ is known for each $x \in X$. For each $y \in Y$, we find the 'closest point' in X, that is, $x \in X$ that minimizes $\mathrm{wd}(x, y) = d(x, y) - \lambda(x)$, and add (x, y) to the list of candidate edges. For an efficient implementation, we compute, in $O(|X| \log |X|)$ time, the weighted Voronoi-diagram of X, where the weight of an element $x \in X$ is $\lambda(x)$. (See [5] for a survey of results on weighted Voronoi diagrams.) For any y, the 'closest point' in X can be found using this data-structure in $O(\log |X|)$ time. Hence, Generate-candidates(X, Y) can be implemented in $O((|X| + |Y|) \log |X|)$ time.

Using Fact 3.3, we can show that the number of candidate edges generated per phase is proportional to the size of the RSSD, which is $O(m \log^4 m)$. The time spent in candidate generation and in maintaining the event-queue is $O(m \log^5 m)$.

Definition 3.6 Let $C(t)$ denote the set of candidate edges generated before time t in the phase. At any time t, we let δ_2^* be the minimum of $(d(u, v) - \pi_{uv})$ over all candidate edges $(u, v) \in C(t)$ such that $u \in S$ and $v \in F$. We let δ_3^* be the minimum of $(d(u, v) - \pi_{uv})/2$ over all candidate edges $(u, v) \in C(t)$ such that u and v are s-vertices not in the same blossom. We let δ_4^* be the minimum of $(d(u, v) - \pi_{uv})$ over all candidate edges $(u, v) \in C(t)$ such that u is an s-vertex and v is in a c-blossom.

Candidate edges are sufficient. Let us suppose that at some time t during the phase, there are two vertices u and v in different maximal blossoms M and N, respectively, such that $\pi_{uv} = d(u, v)$, that is, disk(u) and disk(v) touch. Then the following lemma, proved in Section 6 of the appendix, says that the edge (u, v) is in $C(t)$, the set of candidate edges generated before time t. (If more that one pair of disks from blossoms M and N touch, the lemma guarantees that the edge corresponding to at least one pair is in $C(t)$.)

The main consequence of the lemma is that at any time t, $\delta_2 = 0 \Leftrightarrow \delta_2^* = 0$, $\delta_3 = 0 \Leftrightarrow \delta_3^* = 0$, and $\delta_4 = 0 \Leftrightarrow \delta_4^* = 0$. So it is sufficient for our algorithm to maintain δ_2^*, δ_3^*, and δ_4^* instead of δ_2, δ_3 and δ_4.

Lemma 3.7 *Suppose that at some time t' during the phase, there are two vertices u and v in different maximal blossoms M and N, repectively, such that $\pi_{uv} = d(u, v)$. Then there is a candidate edge $(x, y) \in C(t')$ such that $x \in M$, $y \in N$, and $\pi_{xy}[t'] = d(x, y)$.*

3.2 Data structures

In their algorithm for matching in general graphs, Galil et al. [12] give a method for maintaining δ_2 and δ_3 using a total of $O(|E| \log |V|)$ time per phase, where $|V|$ and $|E|$ are, respectively, the number of vertices and edges in the graph. Using their approach along with our procedure for generating candidate edges, we can maintain δ_2^*, δ_3^*, and δ_4^* using a total of $O(m \log^5 m)$ time per phase. We omit here the other details of implementing a phase; many of these are quite similar to their approach. We conclude that a phase of the conquer step for U can be implemented in $O(m \log^5 m)$ time.

References

[1] P. K. Agarwal, A. Efrat, and M. Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 39–50, 1995.

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA, 1974.

[3] S. G. Akl. A note on Euclidean matchings, triangulations, and spanning trees. *Journal of Combinatorics, Information and System Sciences*, 8(3):169–174, 1983.

[4] S. Arora. Nearly linear time approximation schemes for Euclidean TSP and other geometric problems. In *Proc. 38th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 554–563, 1997.

[5] F. Aurenhammer. Voronoi diagrams: A survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23:345–405, 1991.

[6] D. Avis. A survey of heuristics for the weighted matching problem. *Networks*, 13:475–493, 1983.

[7] S. Buss and P. Yianilos. Linear and $O(n \log n)$ time minimum-cost matching algorithms for quasi-convex tours. In *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, pages 65–76, 1994.

[8] P. B. Callahan and S. R. Kosaraju. A decomposition of multi-dimensional point-sets with applications to k-nearest-neighbors and n-body potential fields. In *Proc. 24th Annu. ACM Sympos. Theory Comput.*, pages 546–556, 1992.

[9] J. Edmonds. Maximum matching and a polyhedron with (0,1) vertices. *J. Res. National Bureau of Standards*, ??:125–130, 1965.

[10] J. Edmonds and E. J. Johnson. Matching, euler tours, and the chinese postman. *Mathematical Programming*, 5:88–124, 1973.

[11] D. Eppstein, G. L. Miller, and S.-H. Teng. A deterministic linear time algorithm for geometric separators and its applications. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 99–108, 1993.

[12] Z. Galil, S. Micali, and H. N. Gabow. Priority queues with variable priority and an $o(ev \log v)$ algorithm for finding a maximal weighted matching in general graphs. In *Proc. 22nd Annual IEEE Symposium on Foundations of Computer Science*, pages 255–261, 1982.

[13] M. Jünger and W. Pulleyblank. New primal and dual matching heuristics. *Algorithmica*, 13(4):357–380, 1995.

[14] E. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart & Winston, New York, 1976.

[15] L. Lovasz and M. D. Plummer. *Matching Theory*, volume 29 of *Ann. Discrete Math.* North-Holland, 1986.

[16] O. Marcotte and S. Suri. Fast matching algorithms for points on a polygon. In *Proc. 30th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 60–65, 1989.

[17] G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis. Geometric separators for finite element meshes. *SIAM J. Sci. Comput.*, ??, 1995. Submitted.

[18] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice Hall, Englewood Cliffs, NJ, 1982.

[19] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, NY, 1985.

[20] P. M. Vaidya. Approximate minimum weight matching on points in k-dimensional space. *Algorithmica*, 4:569–583, 1989.

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

[22] A. C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM J. Comput.*, 11:721–736, 1982.

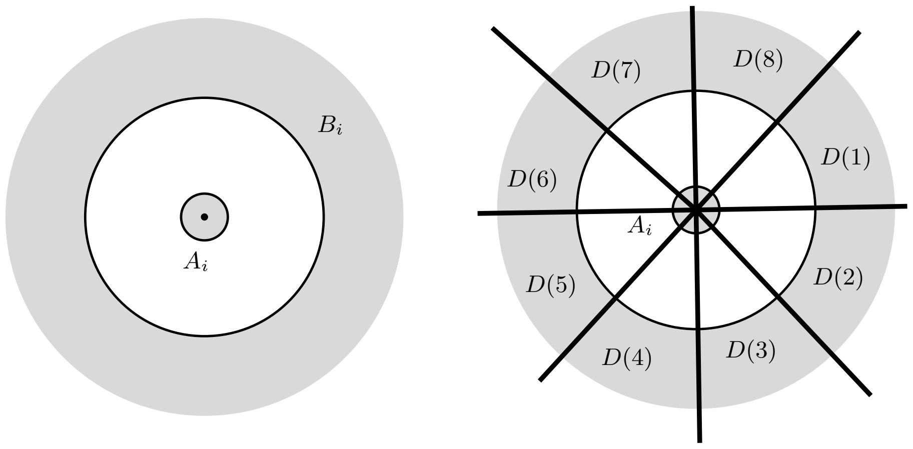

Figure 1: A pair (A_i, B_i) in the SSD; and its refinement to get the RSSD

4 Appendix: Computing the separating circle

We briefly describe our algorithm for computing a separating circle for a set U of m points in the plane. Our algorithm is based on the approach of Miller et al. [17] for computing geometric separators. They show that there is a continous, bijective map Π from the plane (plus a special point at 'infinity') to the unit sphere S_2 in $\mathbb{R}^3$, that has the following properties:

1. Π maps disks (or the complement of disks) on the plane to spherical caps on the unit sphere S_2, where a spherical cap is defined to be the intersection of S_2 with any halfspace in $\mathbb{R}^3$.
2. Let $U^* = \Pi(U)$ be the points on S_2 that Π maps U into. Any hemi-sphere of S_2 contains at least $n/4$ points of U^*.

Moreover, such a map Π can be computed in $O(m)$ time. Given such a map, we find a great-circle C^* on S_2 with the following *separation* property:

> Let $Y^* \subseteq U^*$ be the set of points whose distance (along S_2) from C^* is smaller than $1/\sqrt{m}$. Then $|Y^*| = O(\sqrt{m})$

Actually, we can show that a random great-circle of S_2 has the separation property with probability at least 1/2. We can also check in $O(m)$ time if a given great-circle has the separation property. This immediately gives us a randomized, $O(m)$ expected-time, Las-Vegas algorithm for computing a great-circle with the separation property. Using the techniques of Eppstein et al. [11], we can in fact compute a great-circle with the separation property in $O(m)$ time using a deterministic algorithm.

The pre-image of C^* under Π is a circle C in the plane. We return C as our circle separator for U. Let U_1 (resp. U_2) be the subset of U lying inside (resp. outside) the circle C. The first condition for a circle separator, that is, $\min\{|U_1|, |U_2|\} \geq |U|/4$, follows from property 2 of the map Π. The second condition is shown by a packing arguement for spherical caps on S_2 that uses the properties of the map Π and the fact that C^* has the separation property.

5 Implementing a phase: properties of the algorithm

In these section, we state some additional properties of the matching algorithm that are exploited in the efficient implementation of a phase. The following lemma is easily proved by (structural) induction over the recursive structure of the algorithm.

Lemma 5.1 *Let $U \subseteq V$ be a set of points in a sub-problem of the overall recursive divide-and-conquer procedure. Throughout the conquer step for U, the conditions* EDGE-FEASIBILITY, POSITIVE-DUAL, MATCHING-ADMISSIBILITY, MAXIMALITY, *and* RADIUS-CONSTRAINT *are satisfied.*

The following is a useful corollary of Lemma 2.2 and Lemma 3.1.

Corollary 5.2 *For any $u, v \in U$, if $d(u, v) < \lambda(u)$ in the conquer step for U, then u and v are in the same outer blossom.*

Corollary 5.3 *For any $u, v \in U$, $\lambda(u) \leq \lambda(v) + d(u, v)$ in the conquer step for U.*

Proof: From EDGE-FEASIBILITY, we have

$$\lambda(u) + \lambda(v) \leq d(u, v) + 2 \sum_{u,v \in Q} \omega_Q.$$

Since all dual variables are non-negative,

$$\sum_{u,v \in Q} \omega_Q \leq \sum_{v \in Q} \omega_Q \leq \lambda(v).$$

Combining the two inequalities, we get the statement of the corollary. □

We will now state some properties about a single phase in the conquer step for U.

Lemma 5.4 *There are at most $O(m)$ s-blossoms, f-blossoms, t-blossoms, c-blossoms, and sub-phases in each phase of the algorithm.*

Lemma 5.5 *Let u and v be two vertices such that $b(u) = b(v)$ at all times between t'' and t' in a phase. Then for any t, $t'' \leq t \leq t'$,*

$$\lambda(u)[t] - \lambda(v)[t] = \lambda(u)[t''] - \lambda(v)[t''].$$

Proof: The lemma follows from the fact that if u and v are vertices in the same blossom, the dual change step changes $\lambda(u)$ and $\lambda(v)$ by the same amount. □

6 Implementing a phase: Proof of the main lemma

We present here the proof of Lemma 3.7.
Lemma 3.7 Suppose that at some time t' during the phase, there are two vertices u and v in different maximal blossoms M and N, repectively, such that $\pi_{uv} = d(u, v)$. Then there is a candidate edge $(x, y) \in C(t')$ such that $x \in M$, $y \in N$, and $\pi_{xy}[t'] = d(x, y)$.

Proof: There is a pair (A, B) in the RSSD of the points V such that either $u \in A$ and $v \in B$ or $v \in A$ and $u \in B$. Assume, w. l. o. g, that $u \in A$ and $v \in B$. Let p and r be the center and radius corresponding to (A, B). Let a and b be the representatives of A and B, respectively. Note that A is contained in a disk $S(p, r)$ that is centered at p and has radius r. B is contained in a cone K with apex at p and whose angular opening is θ, the angular constant of the RSSD. Note that $\ell = d(p, b) \geq sr$, where s is the separation constant, and $d(p, b') \geq \ell$ for any $b' \in B$.

Since $\pi_{uv} = d(u, v)$, and u and v lie in different maximal blossoms, it follows from Lemma 2.2 that

$$\lambda(u)[t'] + \lambda(v)[t'] = d(u, v) \tag{1}$$

To prove the lemma, we consider two cases: either $\lambda(u)[t'] \geq 4r$, or $\lambda(u)[t'] < 4r$.

Case 1 $\lambda(u)[t'] \geq 4r$: Since $d(u, a) \leq 2r$, we can conclude from Corollary 5.3 that $\lambda(a)[t'] \geq 2r$. Let $t'' \leq t'$ be the earliest time such that $\lambda(a) \geq 2r$ at all times between t'' and t'. (Possibly, $t'' = 0$.) From Corollary 5.2, we can conclude that at any given time between t'' and t', all points in A belong to the same maximal blossom.

At time t'', our procedure for generating candidate edges finds an $a' \in A$ that minimizes $\mathrm{wd}(c, v)$, over all $c \in A$, and introduces (a', v) as a candidate edge. Thus, $\mathrm{wd}(a', v)[t''] \leq \mathrm{wd}(u, v)[t'']$. This implies, by Lemma 5.5, that $\mathrm{wd}(a', v)[t'] \leq \mathrm{wd}(u, v)[t']$. We conclude that $d(a', v) - \pi_{a'v}[t'] \leq d(u, v) - \pi_{uv}[t']$. Since $\pi_{uv}[t'] = d(u, v)$, EDGE-FEASIBILITY implies that $\pi_{a'v}[t'] = d(a', v)$. Hence the lemma holds with $x = a'$ and $y = v$.

Case 2 $\lambda(u)[t'] < 4r$: In this case, the lemma follows from a series of claims, whose proofs we provide later. From the triangle inequality,

$$d(u, v) + d(u, p) \geq d(p, v).$$

Using equation 1 and the fact that $u \in A$,

$$\lambda(u)[t'] + \lambda(v)[t'] + r \geq d(p, v).$$

Since $\lambda(u)[t'] < 4r$, we obtain

$$\lambda(v)[t'] \geq d(p, v) - 5r.$$

Let $\mathrm{pr}(v)$ denote the 'projection' of v onto the disk $S(p, \ell)$, that is, the point of intersection of the segment pv with the circle of radius ℓ centered at p. We can write

$$d(p, v) = d(p, \mathrm{pr}(v)) + d(\mathrm{pr}(v), v) = \ell + d(\mathrm{pr}(v), v).$$

Since B lies within the cone K with apex at p and angular opening θ, $d(\mathrm{pr}(v), b) \leq \theta\ell$.

Claim 6.1 *At time t', (1) b and v are in the same blossom, and (2) $\lambda(b) \geq \theta\ell + 3r$.*

Let t'' be the earliest time such that $\lambda(b) \geq \theta\ell + 3r$ at all times between t'' and t'. (Possibly, $t'' = 0$.) At time t'', our procedure for generating candidate edges finds a $b' \in B$ that minimizes $\mathrm{wd}(c, u)$ over all $c \in B$, and introduces (u, b') as a candidate edge.

Claim 6.2 *The points b' and v belong to the same blossom at all times between t'' and t'.*

We have $\mathrm{wd}(b', u)[t''] \leq \mathrm{wd}(v, u)[t'']$. By Lemma 5.5, this implies that $\mathrm{wd}(b', u)[t'] \leq \mathrm{wd}(v'u)[t']$. We conclude that $d(b', u) - \pi_{b'u}[t'] \leq d(v, u) - \pi_{vu}[t']$. Since $\pi_{vu}[t'] = d(v, u)$, EDGE-FEASIBILITY implies that $\pi_{b'u}[t'] = d(b', u)$. Hence the lemma holds with $x = u$ and $y = b'$. □

Proof of Claim 6.1. We have

$$\begin{aligned}
\lambda(v) - d(v, b) &\geq (d(p, v) - 5r) - (d(v, \mathrm{pr}(v)) + d(\mathrm{pr}(v), b)) \\
&\geq (\ell + d(v, \mathrm{pr}(v)) - 5r) - (d(v, \mathrm{pr}(v)) + \theta\ell) \\
&= \ell(1 - \theta) - 5r \\
&\geq \theta\ell + 3r,
\end{aligned}$$

since $\theta = 1/18$ and $\ell \geq 9r$. Since $\theta\ell + 3r > 0$, part (1) of the claim follows from Corollary 5.2. Part (2) of the claim follows from Corollary 5.3. □

Proof of Claim 6.2. We prove the claim in two parts: (1) At any time between t'' and t', the points b and v belong to the same blossom, and (2) at any time between t'' and t', the points b' and b belong to the same blossom. Clearly, the claim is proved if we prove (1) and (2).

We first argue that

$$\mathrm{wd}(v, \mathrm{pr}(v))[t'] - \mathrm{wd}(b, \mathrm{pr}(v))[t'] \leq 2r \tag{2}$$

Assume the contrary, that is, $d(\mathrm{pr}(v), v) - \lambda(v)[t'] > d(\mathrm{pr}(v), b)[t'] - \lambda(b) + 2r$

$$\begin{aligned}
\mathrm{wd}(v, u)[t'] &= d(u, v) - \lambda(v)[t'] \\
&\geq d(p, v) - d(p, u) - \lambda(v)[t'] \\
&= d(p, \mathrm{pr}(v)) + d(\mathrm{pr}(v), v) - d(p, u) - \lambda(v)[t'] \\
&= d(p, \mathrm{pr}(v)) - d(p, u) + d(\mathrm{pr}(v), v) - \lambda(v)[t'] \\
&> d(p, \mathrm{pr}(v)) - d(p, u) + d(\mathrm{pr}(v), b) - \lambda(b)[t'] + 2r \\
&\geq d(p, \mathrm{pr}(v)) - d(p, u) + d(\mathrm{pr}(v), b) - \lambda(b)[t'] + 2d(p, u) \\
&= d(p, \mathrm{pr}(v)) + d(p, u) + d(\mathrm{pr}(v), b) - \lambda(b)[t'] \\
&\geq d(u, b) - \lambda(b)[t'] \\
&= \mathrm{wd}(b, u)[t']
\end{aligned}$$

This is a contradiction, since the setting of the lemma and the fact that b and v belong to the same blossom at t' (part (1) of Claim 6.1) imply that $\mathrm{wd}(v, u)[t'] \leq \mathrm{wd}(b, u)[t']$.

We are now ready to prove part (1) of the claim. Assume that part (1) of the claim is false, that is, there is a time t, where $t'' \leq t \leq t'$, so that b and v belong to different blossoms at time t. Also suppose that t is the largest such time. This means that at any time after t and upto t', b and v belong to the same blossom. From the inequality 2 and Lemma 5.5, we conclude that

$$\mathrm{wd}(v, \mathrm{pr}(v))[t] - \mathrm{wd}(b, \mathrm{pr}(v))[t] \leq 2r \tag{3}$$

Now, since $d(\mathrm{pr}(v), b) \leq \theta\ell$, and $\lambda(b)[t] \geq \theta\ell + 3r$ at all times, $\mathrm{wd}(b, \mathrm{pr}(v))[t] \leq -3r$. Using inequality 3, we conclude that $\mathrm{wd}(v, \mathrm{pr}(v))[t] \leq -r$.

Since both $\mathrm{wd}(v, \mathrm{pr}(v))[t]$ and $\mathrm{wd}(b, \mathrm{pr}(v))[t]$ are negative, we have $d(\mathrm{pr}(v), v) < \lambda(v)[t]$, and $d(\mathrm{pr}(v), b) < \lambda(b)[t]$. By Lemma 2.2, v and b belong to the same blossom at time t. This contradicts the assumption that b and v were in different blossoms at time t. This completes the proof of part (1).

To prove part (2), we first argue that $\mathrm{wd}(\mathrm{pr}(b'), b')[t''] - \mathrm{wd}(\mathrm{pr}(b'), b)[t''] \leq 2r$. Assuming the contrary, we get $\mathrm{wd}(b', u)[t''] > \mathrm{wd}(b, u)[t'']$ as above. This contradicts the fact that (b', u) was chosen as the candidate edge at time t''. To complete the proof of part (2), we proceed exactly as in the proof of part (1). Only, we proceed in the 'opposite' direction, from t'' to t'. $\square$

7 Computing the semi-separated decomposition

In this section, we describe our algorithm for computing a semi-separated decomposition (SSD) of a given set of m points U. We first construct a range-tree on the set of points U, which can report the subset of U contained in a query rectangle as a union of few subsets of U. The range-tree is a 2-level partition tree, each of whose nodes is associated with a so-called *canonical subset* of U. The total size of all the canonical subsets in the tree is $O(m \log^2 m)$. For a query rectangle, the query procedure selects $O(\log^2 m)$ canonical subsets whose union consists of exactly those points contained in the rectangle.

Let $A_1, \ldots, A_k$ denote the canonical subsets in the range-tree of U. We now describe a procedure that 'assigns' a subset B_i of U to each A_i. Our semi-separated decomposition of U is simply $\{(A_i, B_i) | B_i \neq \emptyset\}$. The assignment is accomplished by a recursive procedure. At any level of the recursion, we have a subset X of U and its minimum spanning tree $\mathrm{MST}(X)$. (We start off with $X = U$.) If X contains only one point, there is nothing to do, and we return. Otherwise, let e be the longest edge in $\mathrm{MST}(X)$, and ℓ be the length of e. We remove e from $\mathrm{MST}(X)$, thus splitting it into two sub-trees whose vertex sets are, say, Y and Z. Note that the two sub-trees are the minimum spanning trees $\mathrm{MST}(Y)$ and $\mathrm{MST}(Z)$ of Y and Z respectively.

Assume, w. l. o. g, that $|Y|$ is no bigger than $|Z|$. We repeat the following the procedure for each $y \in Y$, and for each $0 \leq i \leq \lfloor \log n \rfloor$. We cover the annulus

$$\mathrm{ann}(y, i) = S(y, 2^{i+1}\ell) - S(y, 2^i l)$$

using $O(1)$ squares of side $2^i * (\ell/2s)$. (Notice that the side-length is chosen so that the points within any square are semi-separated from y.) For each such square, we query the range-tree to obtain $O(\log^2 m)$ canonical subsets of U, and assign y to each of these canonical subsets. (Notice that each of these subsets is semi-separated from y.) This completes the procedure at the current level of recursion, and we recurse with Y and Z.

We first bound the size of the resulting SSD of U. It is clear that each point in Y is assigned to $O(\log^3 m)$ canonical subsets at one level of recusrion. From the fact that Y is smaller than Z, it follows from a standard arguement that any point is assigned to $O(\log^4 m)$ canonical subsets over the entire procedure. Hence $\sum_i |B_i|$ is $O(m \log^4 m)$. Since $\sum_i |A_i|$ is $O(m \log^3 m)$, this means that the size of the SSD of U is $O(m \log^4 m)$.

To show that we indeed compute an SSD, we establish the following lemma:

Lemma 7.1 *Let $C_i \subseteq Y$ be the set of points assigned to A_i at the current level of recursion. For any $y \in Y$ and $z \in Z$, there is an A_i such that $z \in A_i$ and $y \in C_i$.*

Proof: For any $y \in Y$, let $\mathrm{ann}(y) = S(y, nl) - S(y, l)$. It is clear that for any $u \in U$ that lies in $\mathrm{ann}(y)$, the procedure assigns y to a canonical subset A_i that contains u. Moreover, A_i and $\{y\}$ are semi-separated.

Hence, it suffices to show that $z \in \operatorname{ann}(y)$ for any $z \in Z$. From the properties of a minimum spanning tree, it follows that the distance between any point in Y and any point in Z is at least ℓ, the length of e. Hence, $d(y, z) \geq \ell$, and $z \notin S(y, \ell)$.

Now, $\mathrm{MST}(X)$ contains a path between y and z that uses at most $(n-1)$ edges. The longest edge in this path is e, which has length ℓ, so the overall path length is at most $(n-1)\ell$. It follows that $d(y, z) \leq (n-1)\ell$, and so $z \in S(y, n\ell)$. Thus, $z \in \operatorname{ann}(y)$. □