Algorithmica (2004) OF1–OF33
DOI: 10.1007/s00453-004-1124-z

Algorithmica

New Approximation Results for the Maximum Scatter TSP[1]

Yi-Jen Chiang[2]

Abstract. We consider the following *maximum scatter traveling salesperson problem* (TSP): given an edge-weighted complete graph $\mathcal{G} = (S, E)$, find a Hamiltonian path or cycle such that the length of a *shortest* edge is *maximized*. In other words, the goal is to have each point far away (most "scattered") from the points that are visited just before or just after it in the path/cycle. This is also referred to as the *max-min* 1*-neighbor TSP*. More generally, in the *max-min m-neighbor TSP* problem, the goal is to maximize the minimum distance between any point and all of its "m-neighbors" along the path/cycle. An *m-neighbor* of p is a point that is at most m points away from p in the path/cycle. These problems are closely related to the bottleneck TSP, and are motivated by applications in manufacturing (e.g., sequencing of rivet operations) and medical imaging.

In this paper we give $O(n^{2.5})$-time approximation algorithms for the max-min 2-neighbor TSP with the triangle inequality. We achieve an approximation factor of 12 for the path version, and a factor of 18 for the cycle version of the problem, improving the previous best factors of 32 and 64, respectively [2], for all cases of n. Moreover, we present an $O(mn^{2.5})$-time algorithm that achieves a *constant* approximation factor of 16 for the path version of the max-min m-neighbor TSP with the triangle inequality when $n = (m + 1)k$ or when $n = (m + 1)k + m$, where $m \in (2, n)$ is *part of the input*. This significantly improves the previous best approximation factor of $4 \cdot 8^{\lceil m/2 \rceil}$ [2], from exponential in m to a small constant.

Key Words. Traveling salesperson problem (TSP), Maximum scatter TSP, Bottleneck TSP, Hamiltonian cycle/path, Approximation algorithms, Matching, Optimization.

1. Introduction.

We consider the following *maximum scatter traveling salesperson problem* (TSP): given an edge-weighted complete graph $\mathcal{G} = (S, E)$, find a Hamiltonian path or cycle such that the length of a *shortest* edge is *maximized*. In other words, the goal is to have each point far away (most "scattered") from the points that are visited just before or just after it in the path/cycle. This is also referred to as the *max-min* 1*-neighbor TSP*. More generally, in the *max-min m-neighbor TSP* problem the goal is to maximize the minimum distance between any point and all of its "m-neighbors" along the path/cycle. An *m-neighbor* of p is a point that is at most m points away from p in the path/cycle. The problem, then, is to find a path/cycle, $(p_1, \ldots, p_n)$, on the n points of S in order to maximize

$$\min_{i \in \{1,\ldots,n\}} \min_{j \in \{i-m,\ldots,i-1,i+1,\ldots,i+m\}} d(p_i, p_j),$$

[1] This research was supported in part by NSF CAREER Grant CCR-0093373, NSF Grant ACI-0118915, and NSF ITR Grant CCR-0081964.

[2] Department of Computer and Information Science, Polytechnic University, Brooklyn, NY 11201, USA. yjc@poly.edu.

Received August 3, 2003; revised March 24, 2004. Communicated by J.-D. Boissonnat.
Online publication November 15, 2004.

where $d(p_i, p_j)$ denotes the length of edge (p_i, p_j) (*not* the length along the path/cycle linking p_i and p_j). The indices in the minimization should be modified appropriately to account for wrap-around effects in cycles or endpoint effects in paths.

Motivation. As described in [2], this problem arises in some manufacturing processes where one is given sheets of metal to be fastened together. After necessary alignment, the top metal plate has a set of pre-specified points at which one should apply riveting operations to join the plates together. To prevent nonuniform deformation of the sheet metal plates, it is important to sequence the riveting process so that the distance between one rivet and the next one is large; i.e., the riveting operations should be *scattered*. This problem was first brought to our attention by Boeing [15]–[19]. In operations that involve heating the workpiece, it is important not just to have each point well separated from its immediate predecessor and successor, but also from its m-neighbors, in order to allow for cooling time in the vicinity of each operation [2], [15]–[19]. This motivates the study of the more general max-min m-neighbor TSP with $m > 1$ [2].

The maximum scatter TSP also arises in some medical imaging applications, as described and studied in [2] and [14]. When imaging physiological functions using a Dynamic Spatial Reconstructor (DSR), the radiation sources are placed along the top half of a circular ring, with sensors placed directly opposite, in the bottom half of the ring. The "firing sequence" determines the order in which the sources and their partnered sensors are activated, usually in a periodic pattern [2], [14]. The sensors collect intensity data of the energy that passes through the patient, who is placed in the center of the ring. When source i is activated, some amount of scattering occurs, so it is important not to activate sensors of nearby sources (e.g., $i + 1, i - 1, i + 2, \ldots$) soon after i is activated [2], [14]. This motivates the study of the cycle version of the problem [2].

Previous Work. Previously Arkin et al. [2] gave the first algorithmic study of the problem. It was shown that the maximum scatter TSP is, in general, NP-complete and that no constant-factor approximation algorithm exists unless P = NP. In the case that distances obey the triangle inequality, they proposed factor-2 (best factor) approximation algorithms for the max-min 1-neighbor TSP, for both the path and cycle versions. The methods also extended to give a factor-2 (best factor) approximation for the max-min 2-neighbor TSP, for the cycle version and some cases ($n \neq 3k + 1$) of the path version of the problem. However, these extensions are highly impractical—the underlying *Regularity Lemma* [1], [11] needed is valid only for $n > Q$ where $\log^* Q \approx 2^{20}$ [1], [2]! They also gave practical algorithms (i.e., without using the *Regularity Lemma*) for the max-min 2-neighbor TSP with the triangle inequality, for both the path (factor 32) and cycle (factor 64) versions. These methods extended to yield an approximation factor of $4 \cdot 8^{\lceil m/2 \rceil}$ for the path version of the max-min m-neighbor TSP, for any *constant* $m > 2$, when $n = (m + 1)k$. We remark that although not explicitly stated in [2], this max-min m-neighbor TSP algorithm can be easily extended to the case of $n = (m + 1)k + m$ in the path version, with the same approximation factor. For the max-min 1-neighbor TSP with Euclidean distances, they gave linear-time (optimal) exact algorithms for the case of points on a line or on a circle, for both path and cycle versions [2].

Fekete [6] (and its extensively expanded journal version by Barvinok et al. [3]) proved that the Euclidean maximum scatter TSP is NP-hard in three or more dimensions. Recently, Barvinok et al. [3] also proved, as extensions to their NP-hardness results of the

corresponding settings for the *MAX TSP* problem (see below), that the maximum scatter TSP is NP-hard under the following settings: on the $(d-1)$-dimensional surface of the d-dimensional unit sphere S^{d-1} under shortest distances (for $d \geq 3$), on the $(d-1)$-dimensional surface of a d-dimensional convex polytope with an unbounded number of facets under geodesic distances (for $d \geq 3$), and for points in d-dimensional space under L_p norms where $1 < p \leq \infty$ and d is part of the input. They also showed that the maximum scatter TSP is Max-SNP-hard for points in d-dimensional space under L_∞-norms when d is part of the input. It still remains open whether the Euclidean maximum scatter TSP is NP-hard in two dimensions.

A closely related problem is the *bottleneck TSP* (BTSP) in which the goal is to *minimize* the length of a *longest* edge in a Hamiltonian cycle. (See [7] and [12].) The BTSP is known to be NP-complete, and there exists no constant-factor approximation algorithm unless P = NP. Assuming the edge lengths satisfy the triangle inequality, there exists an approximation algorithm to produce a tour whose longest edge has length at most twice the optimal, and this is best possible [13]. For an excellent survey on the many variants of BTSP, see [10], which also surveys the previous results of our maximum scatter TSP problem. Another variant of the TSP that is potentially related to our work is the *MAX TSP*, in which the goal is to find a Hamiltonian cycle whose total length is as long as possible. As mentioned above, Fekete [6] and Barvinok et al. [3] extended their NP-hardness results from MAX TSP to maximum scatter TSP for several settings. Many other algorithmic studies on the MAX TSP and its variations appeared in the literature; we refer to [4] for an excellent survey.

Summary of Our Results. We summarize our results of this paper as follows:

- We give $O(n^{2.5})$-time approximation algorithms for the max-min 2-neighbor TSP with the triangle inequality, for all cases of n, which do not use the *Regularity Lemma* and are practical:
  - — We give a factor-12 approximation algorithm for the path version, improving the previous best factor of 32 [2] that do not use the *Regularity Lemma*.
  - — We give a factor-18 approximation algorithm for the cycle version, improving the previous best factor of 64 [2] that do not use the *Regularity Lemma*.

  As mentioned above, the factor-2 approximation algorithms of [2] do not work for the case of $n = 3k + 1$ in the path version, and are valid only when n is much larger than the estimated number of atoms in the entire universe(!) because of the use of the *Regularity Lemma*, and thus should not be taken into account for any practical purposes.

  Our results include new algorithms for the base case of $n = 3k$ in the path version, new algorithms for all cases of n in the cycle version, and new algorithms for the remaining cases of $n \neq 3k$ in the path version. We remark that the case of $n = 3k + 1$ in the path version, as also observed in [2], is surprisingly difficult to handle; we give two novel algorithms to deal with this case. Both algorithms, in addition to improving the approximation factor given in [2], are much simpler than the method of [2], by exploiting some nice and elegant structural properties of the point set S under the distance metric, and may be of independent interest.
- We present an $O(mn^{2.5})$-time algorithm that achieves a *constant* approximation factor of 16 for the path version of the max-min m-neighbor TSP with the triangle inequality

when $n = (m+1)k$ or when $n = (m+1)k+m$, where $m \in (2, n)$ is *part of the input*. This significantly improves the previous best approximation factor of $4 \cdot 8^{\lceil m/2 \rceil}$ [2], from exponential in m to a small constant.

Our algorithm utilizes novel, combined matching and swapping techniques such that the approximation factor stays the same (a fixed constant) throughout the $\Theta(m)$ phases of the construction process, and may be interesting by its own.

Notation. We consider a complete graph, $\mathcal{G} = (S, E)$, on a set of points S, with the full edge set E. Throughout the paper we assume that the distances obey the triangle inequality. Also, without loss of generality, we assume that the edge lengths in E are *distinct* (achieved by symbolic perturbations); this simplifies our presentation. We let $d(p, q)$ denote the distance between two points p and q, and $diam(R)$ denote the *diameter* of the set $R \subseteq S$; i.e., $diam(R) = \max_{p,q \in R} d(p, q)$. The (open) *disk* D of radius r, centered at a point $p \in S$ is defined to be the set of points in S that are at a distance less than r from p; the boundary of D is the *circle* consisting of points of S that are at a distance exactly r from p.

An algorithm is said to yield a *factor α approximation* for a maximization problem if the algorithm is guaranteed to produce a solution whose objective function value is at least $(1/\alpha)$ times the value of an optimal solution.

Paper Organization. The rest of the paper is organized as follows. In Section 2 we present our approximation algorithms for the path and cycle versions of the max-min 2-neighbor TSP, including the base algorithm which readily works for the case of $n = 3k$ in the path version (Section 2.1), the algorithms for all cases of n in the cycle version (Section 2.2), and the algorithms for the remaining cases of $n \neq 3k$ in the path version (Section 2.3) with main focus on the two algorithms that handle the case of $n = 3k + 1$. Finally, we describe our approximation algorithm for the max-min m-neighbor TSP in Section 3.

2. Approximating the Max-Min 2-Neighbor TSP.

In this section we describe our approximation algorithms for the path and cycle versions of the max-min 2-neighbor TSP, including the base algorithm which readily works for the case of $n = 3k$ in the path version, the algorithms for all cases of n in the cycle version, and the algorithms for the remaining cases of $n \neq 3k$ in the path version. Our starting point is the following lemma proved in [2]:

LEMMA 1 [2]. *For any integer $m \geq 1$, let $R \subset S$ be a subset of nodes with $|R| > \lfloor n/(m+1) \rfloor$ (resp., $|R| > \lceil n/(m+1) \rceil$). Then, in any Hamiltonian cycle (resp., path) on S, there must exist two nodes of R that are m-neighbors.*

Lemma 1 immediately gives upper bounds on the objective values of the max-min 2-neighbor TSP, which are used in [2] and are re-stated below:

COROLLARY 2 [2]. *Let r (resp., r') be the radius of a smallest disk, $\mathcal{C}$ (resp. $\mathcal{C}'$), centered at one of the n points such that it contains exactly $\lfloor n/3 \rfloor$ (resp., $\lceil n/3 \rceil$) points of S (including the center point) in its interior, and one additional point on its boundary.*

Then the max-min distance in the cycle (*resp.*, *path*) *version of the* 2*-neighbor TSP is at most* $2r$ (*resp.*, $2r'$).

Observe the subtle differences between the path and cycle versions of the problem. First, for the cycle version, we have to take care of the additional wrap-around effect, which is not needed for the path version. Secondly, when n is a multiple of 3, $\lfloor n/3 \rfloor = \lceil n/3 \rceil = n/3$, and thus $r = r'$, i.e., we have the *same* upper bound for both versions; however, when n is not a multiple of 3, $\mathcal{C}'$ contains one more point than $\mathcal{C}$ in the interior, and thus $r' > r$ and the upper bounds are different. It turns out that when n is not a multiple of 3 (in particular when $n = 3k + 1$), the path version is surprisingly far more difficult to handle than the cycle version; we give two novel algorithms to deal with the path version when $n = 3k+1$, which exploit some nice structural properties of the point set S under the distance metric and are technically quite interesting.

In the following we first present in Section 2.1 our base algorithm, which readily works for the path version when n is a multiple of 3. In this case we do not need to consider the wrap-around effect of the cycle version, nor do we need to deal with the complication of the path version of n not being a multiple of 3. We then present in Section 2.2 the algorithm for the cycle version for all cases of n, including how to handle the wrap-around effect and how to deal with the case of n not being a multiple of 3. Finally, we present the algorithms for the remaining cases of $n \neq 3k$ in the path version, in particular the two major technical algorithms for handling the case of $n = 3k + 1$, in Section 2.3.

2.1. *Base Algorithm*: *Path Version When n Is a Multiple of* 3. We describe in this section the base algorithm, which readily handles the path version of the max-min 2-neighbor TSP when n is a multiple of 3. We assume for now that n is a multiple of 3; other cases are handled in Section 2.3. The base algorithm is also used for the cycle version of the problem, to be discussed in Section 2.2.

The overall strategy of our algorithm consists of the following steps:

1. Construct $n/3$ vertex-disjoint triangles whose vertices are points in S and whose edges are "long," meaning that the edges are of length at least some constant fraction of the upper bound, $2r'$ (as given in Corollary 2).
2. Concatenate these $n/3$ triangles to create a Hamiltonian path such that any 2-neighbors on the path are far apart.

Recall from Corollary 2 that $2r'$ is an upper bound on the objective value, where r' is the radius of a smallest disk $\mathcal{C}'$ centered at one of the n points such that it contains exactly $n/3$ points (including the center point) in its interior, plus one additional point on its boundary. (Since n is a multiple of 3, $\lfloor n/3 \rfloor = \lceil n/3 \rceil = n/3$, and thus $r = r'$ and $\mathcal{C} = \mathcal{C}'$ in Corollary 2; we still use r' and $\mathcal{C}'$ here to stress that we are currently working on the path version.) Using the idea of [2], we can compute C' and r' in $O(n^2)$ time as follows: First, for each point, we find the $(n/3)$th shortest edge among all edges incident on the point in $O(n)$ time, using the linear-time median-selection algorithm; the overall time for this step is $O(n^2)$. Then, among the n selected edges we choose the shortest one, which defines r' and the center of $\mathcal{C}'$. The overall computation takes $O(n^2)$ time.

After computing r' and $\mathcal{C}'$, we use Algorithm *Large-Triangles* below to construct $n/3$ vertex-disjoint triangles using n points of S as vertices, such that the triangle edges are of length at least $r'/3$.

Algorithm *Large-Triangles*

There are three phases in the algorithm.

1. Complete matching phase
Create a bipartite graph $G = (V_1, V_2, E)$, where the nodes of V_1 are all $n/3$ points in the *interior* of $\mathcal{C}'$, and the nodes of V_2 are all remaining $(2n)/3$ points of S, which are outside or on the boundary of $\mathcal{C}'$. Edge set E consists of "long" edges from V_1 to V_2, between two nodes whose distance is at least r'. Perform a maximum cardinality matching on G. As proved in [2], there exists a complete matching, and hence all nodes of V_1 are matched. For each matched pair, create a segment e by connecting the two points. At the end, there are $n/3$ segments, each of length at least r'.

2. Maximal matching phase
Create another bipartite graph $H = (U_1, U_2, \mathcal{E})$, where the nodes of U_1 are the $n/3$ segments created in Phase 1, and the nodes of U_2 are the remaining $n/3$ points of S that are *not* matched in Phase 1. The edge set $\mathcal{E}$ consists of edges from U_1 to U_2, between two nodes $e \in U_1$ and $p \in U_2$ such that p is away from *both* endpoints of e by distances at least $r'/3$. Perform a *maximal* (rather than maximum cardinality) matching on H. For each matched pair e and p, create a triangle by connecting p to both endpoints of e. If the matching is a perfect matching, then stop—there are already $n/3$ triangles whose edge lengths are at least $r'/3$; otherwise, continue to the next phase.

3. Swapping phase
Let P be the set of points of S that are not yet in any triangle (called *left-over points*), and let E' be the set of segments (created in Phase 1) that are not yet in any triangle (called *left-over segments*). The primitive operation in this phase is to pick a left-over point, a left-over segment, and a triangle, and create two desirable triangles using the six points involved as the triangle vertices (see Lemma 6). At the end, all left-over points and segments are picked up, resulting in $n/3$ vertex-disjoint triangles whose edge lengths are at least $r'/3$. Notice that the bound on the shortest triangle-edge length ($r'/3$) stays the same as that in Phase 2; this is achieved by interesting insights in the combined maximal matching and swapping operations.

LEMMA 3. *Let $e = (v_1, v_2) \in E'$ be an arbitrary left-over segment. The set P of left-over points is either partitioned into two disjoint nonempty sets P_1 and P_2 (called a* 2-partition *of P), where each point of P_i is within distance less than $r'/3$ to v_i for each $i = 1, 2$ and any pair of two points from different sets are at a distance larger than $r'/3$ from each other, or each point of P is within distance less than $r'/3$ to the* same *endpoint v_1 or v_2 (called a* 1-partition *of P). In other words, each segment $e \in E'$ induces either a 1-partition or a 2-partition of P. Moreover, two segments in E' may induce* different

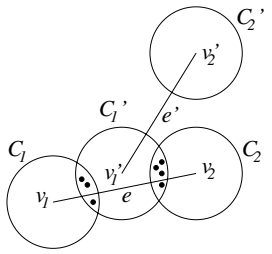

Fig. 1. An example of two different partitions of P induced by two segments of E': $e = (v_1, v_2)$ induces a 2-partition while $e' = (v_1', v_2')$ induces a 1-partition. The points of P are shown as small filled circles.

partitions of P. In particular, it is possible that one segment induces a 2-partition while another induces a 1-partition.

PROOF. First we show that $e = (v_1, v_2)$ induces either a 1-partition or a 2-partition. Consider two disks C_1, C_2, centered respectively at v_1 and v_2 with the same radius $r'/3$. Because e is of length at least r' by the construction of Phase 1, C_1 and C_2 are *disjoint* and any point in the interior of C_1 is away from any point in the interior of C_2 by a distance larger than $r' - r'/3 - r'/3 = r'/3$. Now consider any point $p \in P$. Since p is not matched with e in Phase 2, p is within distance $r'/3$ to either v_1 or v_2, i.e., p is in the *interior* of either C_1 or C_2, but not both. If all points of P are in the interior of the same disk (C_1 or C_2), then e induces a 1-*partition* of P; otherwise e induces a 2-*partition* (and P_i is the set of points in the interior of C_i for each $i = 1, 2$).

Now we refer to Figure 1 for an example of two different partitions of P induced by two segments $e = (v_1, v_2)$ and $e' = (v_1', v_2')$. The disks C_i and C_i' are of radius $r'/3$ centered at v_i and v_i', respectively, for each $i = 1, 2$. Since the interiors of C_1 and C_2 are apart by a shortest distance larger than $r'/3$ only but $diam(C_1') = (2r')/3$, it is possible that the interior of C_1' has nonempty intersections with the interiors of C_1 and C_2. If P consists only of points in the interiors of $C_1 \cap C_1'$ and $C_2 \cap C_1'$, then there is no point in the interior of C_2', meaning that e induces a 2-partition while e' induces a 1-partition. □

We pick one arbitrary left-over segment $e = (v_1, v_2) \in E'$, and construct disks C_i, D_i, R_i centered at v_i with radii $r'/3$, $(2r')/3$, and r', respectively, for each $i = 1, 2$. By Lemma 3, e induces either a 2-partition (P_1, P_2) with points of P_i in the interior of C_i for each $i = 1, 2$, or a 1-partition. Without loss of generality, we assume that in the case of a 1-partition, C_1 contains all points of P in the interior, and we use $(P_1, \emptyset)$ to denote the 1-partition, where $P_1 = P$.

For each *nonempty* point set P_i, $i = 1, 2$, we call P_i *cluster* i. We emphasize that cluster i always refers to the point set P_i which is *nonempty*. For each cluster i, we call a triangle *good* for cluster i if all the triangle vertices are outside or on the boundary of R_i, and *bad* for cluster i otherwise.

LEMMA 4. *Each left-over segment in E' must have at least one endpoint in the interior of D_i, for any cluster i, $i = 1, 2$.*

PROOF. Consider any left-over point p in cluster i, and any left-over segment $e' \in E'$. Since p is not matched with e' in Phase 2, p is within distance less than $r'/3$ to one of the endpoints of e', say v', i.e., $d(p, v') < r'/3$. However, p is in the interior of C_i, i.e., $d(v_i, p) < r'/3$, and thus we have $d(v_i, v') \leq d(v_i, p) + d(p, v') < (2r')/3$, meaning that v' is in the interior of D_i. □

LEMMA 5. *Let ℓ_i be the number of left-over points in cluster i, $i = 1, 2$. Then the number of good triangles for cluster i is at least ℓ_i.*

PROOF. Let T be the number of triangles, and let $T_{i,\text{good}}$ and $T_{i,\text{bad}}$ be the numbers of triangles good and bad for cluster i, respectively. Our goal is to show that $T_{i,\text{good}} \geq \ell_i$. Clearly, $T_{i,\text{good}} + T_{i,\text{bad}} = T$. Also, we have $n/3 - T$ left-over segments, since at the end of Phase 1 there are $n/3$ segments, and each time the number of triangles is increased by one, the number of left-over segments is decreased by one—this is true for triangles obtained from maximal matching in Phase 2 as well as those from the swapping operations in Phase 3 (see Lemma 6 below: a good triangle, a left-over point, and a left-over segment are picked up and swapped to produce two triangles). Now consider the number $|R_i|$ of points in the *interior* of the disk R_i: these are ℓ_i left-over points in cluster i, at least one endpoint from each of the $n/3 - T$ left-over segments (see Lemma 4), and at least one vertex from each of the $T_{i,\text{bad}}$ bad triangles for cluster i (by the definition of bad triangles). Therefore, we have $|R_i| \geq \ell_i + (n/3 - T) + T_{i,\text{bad}}$. Recall that R_i is of radius r'; by the definition of r', R_i has no more than $n/3$ points in the interior (see Corollary 2 and the discussion at the beginning of Section 2.1), namely, $n/3 \geq |R_i|$. Putting these inequalities together, we have $\ell_i \leq T - T_{i,\text{bad}} = T_{i,\text{good}}$. □

Now we are ready to describe the major operations in Phase 3—the *swapping operations*, in the next lemma.

LEMMA 6 (The Swapping Operations). *For each cluster i, $i = 1, 2$, we can repeatedly pick a left-over point in cluster i, a* good *triangle for cluster i, and a left-over segment, and construct two triangles using the six points involved as the triangle vertices, so that at the end of Phase* 3 *there are $n/3$ vertex-disjoint triangles whose shortest edge length is at least $r'/3$.*

PROOF. We proceed in one or two stages. In Stage 1 we put all left-over points in cluster 1 to triangles by the swapping operations. If $e = (v_1, v_2)$ induces a 1-partition, then we are done; otherwise, in the case of a 2-partition, we continue to Stage 2, putting all left-over points in cluster 2 to triangles. In the following we describe the more general case of a 2-partition.

In Stage 1 we first identify all triangles (which are constructed in Phase 2) that are *good* for cluster 1 and put them to a queue. By Lemma 5, the number of such good triangles is at least ℓ_1, the number of left-over points in cluster 1. We then repeatedly perform the following operation until the left-over points in cluster 1 are exhausted: pick up a left-over point x from cluster 1, a left-over segment (p, q), and the next available good triangle $\triangle uvw$ from the queue, and swap to obtain two vertex-disjoint triangles using

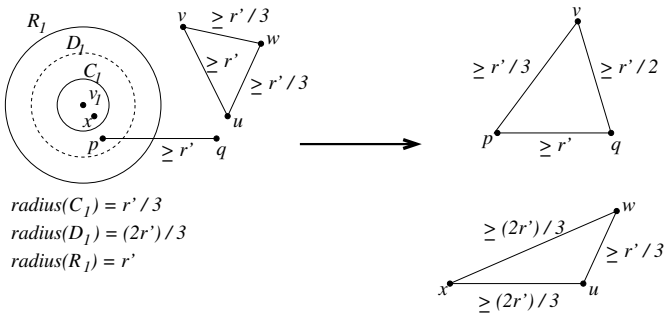

Fig. 2. The swapping operation in Stage 1 described in Lemma 6: the left-over segment (p, q) (left) is grown into a triangle $\triangle vpq$ (right) by including v as its new vertex, and the triangle $\triangle uvw$ (left), good for cluster 1, is modified to become the triangle $\triangle uwx$ (right), by *swapping out* the vertex v and *swapping in* the left-over point x in cluster 1 as its new vertex.

the six points x, p, q, u, v, and w. Notice that there are always enough good triangles to be swapped with.

We now describe how to perform the swapping in detail (refer to Figure 2). Recall that one edge of triangle $\triangle uvw$, say (u, v), is constructed in Phase 1, and the remaining two edges are constructed in Phase 2. Therefore, (u, v) is of length at least r' and the other two edges, (u, w) and (v, w), are of length at least $r'/3$. By Lemma 4, at least one endpoint of (p, q), say p, is in the interior of D_1. Consider q and edge (u, v). Without loss of generality, suppose q is no closer to v than to u, i.e., $d(q, v) \geq d(q, u)$. Then we swap the vertices to construct two triangles $\triangle vpq$ and $\triangle uwx$ (see Figure 2). We have the following key properties.

CLAIM 7. *The triangles $\triangle vpq$ and $\triangle uwx$ have shortest edge length at least $r'/3$. Moreover, $\triangle vpq$ is* bad *for cluster* 2, *and $\triangle uwx$ has the property that the edges (u, x) and (w, x) are both of length at least $(2r')/3$.*

PROOF. We refer to Figure 2 for the proof. We first look at $\triangle vpq$. Recall that $d(q, v) \geq d(q, u)$. We show that the triangle edge (q, v) is of length at least $r'/2$: if $d(q, u) \geq r'/2$, then $d(q, v) \geq d(q, u) \geq r'/2$; else, $d(q, u) < r'/2$, then since (u, v) is of length of at least r', $d(q, v) \geq d(u, v) - d(u, q) > r'/2$. For the triangle edge (p, q), it is easy to see that its length is at least r', since it is a segment constructed in Phase 1. Finally, we show that the edge (v, p) is of length at least $r'/3$. Since v is previously a vertex of the good triangle $\triangle uvw$, v is outside or on the boundary of R_1, i.e., $d(v_1, v) \geq r'$. Also, since p is in the interior of D_1, we have $d(v_1, p) < (2r')/3$, and thus $d(v, p) \geq d(v_1, v) - d(v_1, p) > r'/3$. We conclude that the shortest edge length of $\triangle vpq$ is at least $r'/3$. Moreover, (p, q) is originally a left-over segment, and hence, by Lemma 4, at least one of the endpoints is in the interior of R_2. This means that $\triangle vpq$ is *bad* for cluster 2.

Now look at $\triangle uwx$. The edge (u, w) has length at least $r'/3$ since it is originally an edge of $\triangle uvw$ before the swapping. Also, since $\triangle uvw$ is a good triangle, u and w are

both outside or on the boundary of R_1, i.e., $d(u, v_1) \geq r'$ and $d(w, v_1) \geq r'$. However, x is a left-over point lying in the interior of C_1, namely, $d(x, v_1) < r'/3$. Therefore, we have $d(u, x) \geq d(u, v_1) - d(x, v_1) > (2r')/3$ and similarly $d(w, x) > (2r')/3$. We conclude that $\triangle uwx$ has a shortest edge length of at least $r'/3$, and both edges (u, x) and (w, x) are of length of at least $(2r')/3$. □

The process in Stage 2 is similar to that in Stage 1: we first identify all triangles that are *good* for cluster 2, putting them into a queue, and then repeatedly perform the swapping operation in a similar way, until the left-over points in cluster 2 (and hence all left-over segments as well) are exhausted. Notice that by Lemma 5 again, the number of good triangles for cluster 2 is at least the number of left-over points in cluster 2, and hence we can finish swappings for all left-over points and segments and produce $n/3$ vertex-disjoint triangles in the end.

Now we give the details of the swapping operation in Stage 2, and show that the resulting triangles all have a shortest edge length at least $r'/3$. Consider a left-over point x' in cluster 2, a left-over segment (p', q'), and a triangle good for cluster 2 that are currently being swapped to produce two triangles. Observe that the good triangle in question must be one of the following two types: (1) a triangle produced in Phase 2, without being used for swapping in Stage 1, such as the triangle $\triangle uvw$ considered in Stage 1 above but is good for cluster 2 instead, and (2) a triangle produced by the swapping operation in Stage 1. For type (1), we proceed as in Stage 1, producing two triangles of shortest edge length at least $r'/3$; see Figure 2 and Claim 7. For type (2), by Claim 7, of the two triangles produced by swapping in Stage 1, only the triangle $\triangle uwx$ (i.e., the one that does *not* contain a left-over segment) can be *good* for cluster 2; we thus consider swapping on x', (p', q'), and the triangle $\triangle uwx$ good for cluster 2 (refer to Figure 3).

By Claim 7, the edges (u, x) and (w, x) of triangle $\triangle uwx$ are both of length at least $(2r')/3$; we consider edge (u, x) and the left-over segment (p', q'). Again, By Lemma 4, at least one endpoint of (p', q'), say p', is in the interior of D_2. Without loss

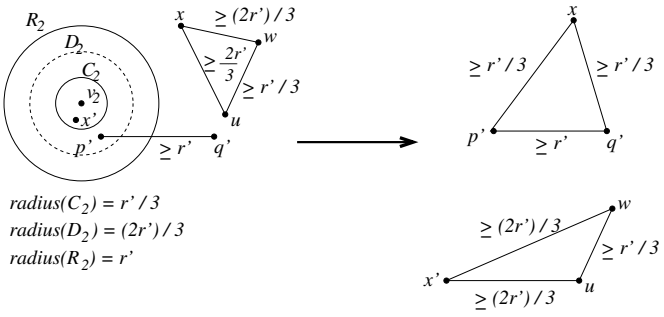

Fig. 3. The swapping operation in Stage 2 described in Lemma 6: the left-over segment (p', q') (left) is grown into a triangle $\triangle xp'q'$ (right) by including x as its new vertex, and the triangle $\triangle uwx$ (left), good for cluster 2, is modified to become the triangle $\triangle uwx'$ (right), by *swapping out* the vertex x and *swapping in* the left-over point x' in cluster 2 as its new vertex.

of generality, suppose q' is no closer to x than to u (i.e., $d(q', x) \geq d(q', u)$), then we swap the vertices to construct two triangles $\triangle xp'q'$ and $\triangle uwx'$ (see Figure 3). By the same proof method for Claim 7, we have that the edges (x, p'), (x, q'), and (p', q') of the triangle $\triangle xp'q'$ are of lengths at least $r'/3$, $r'/3$, and r', respectively, and that the edges (u, w), (u, x'), and (w, x') of the triangle $\triangle uwx'$ are of lengths at least $r'/3$, $(2r')/3$, and $(2r')/3$, respectively. It follows that the triangles constructed in Stage 2 all have shortest edge length at least $r'/3$. Therefore, at the end of Phase 3, there are $n/3$ vertex-disjoint triangles whose edges are of length at least $r'/3$. □

We analyze the running time of Algorithm *Large-Triangles*. At the very beginning, we compute r' and $\mathcal{C}'$ in $O(n^2)$ time, as analyzed before. In Phases 1 and 2 we perform maximum cardinality matching and maximal matching for bipartite graphs, which can be done in time $O(n^{2.5})$ (either by the $O(n^{2.5})$-time algorithm of [9] or by the $O(\sqrt{n}m)$-time algorithm of [5], where $m = O(n^2)$ is the number of edges in the graphs). In Phase 3 we pick an arbitrary left-over segment $e \in E'$ and construct the disks C_i, D_i, and R_i, identify the corresponding good/bad triangles, and repeatedly perform local swappings, in a total of $O(n)$ time.

THEOREM 8. *Given a complete graph on set S of n points satisfying the triangle inequality, where n is a multiple of 3, there exists an algorithm that constructs in $O(n^{2.5})$ time $n/3$ vertex-disjoint triangles from points in S such that the smallest edge length of the triangles is at least $r'/3$, where r' is the radius of a smallest disk centered at a point of S, containing exactly $n/3$ points in its interior and one additional point on its boundary.*

Our next task is to concatenate the $n/3$ vertex-disjoint triangles produced by Algorithm *Large-Triangles* to create a Hamiltonian path such that any 2-neighbors on the path are far apart. We use the method of [2] to carry out the task, which we summarize in the following theorem.

THEOREM 9 [2]. *Given a list of triangles whose minimum edge length is δ, the triangles can be chained into a path such that any two nodes that are 2-neighbors in the path are at a distance at least $\delta/2$ from each other.*

The primitive operation for Theorem 9 is the following lemma given in [2].

LEMMA 10 [2]. *We are given a triangle $\triangle abc$ whose vertices are visited in that order by a path, and a triangle $\triangle xyz$ to be added after $\triangle abc$, such that all (six) edges in both triangles are of length at least δ. Then the order of the vertices of triangle $\triangle xyz$ can be chosen such that, in the resulting path, if $p \in \{a, b, c, x, y, z\}$ and $q \in \{a, b, c, x, y, z\}$ are 2-neighbors, then $d(p, q) \geq \delta/2$.*

To create the path, we start with any triangle, in any order of the vertices, and keep adding new triangles to the path, one at a time, according to Lemma 10, in $O(1)$ time each, until all the triangles are added [2]. The order in which the triangles are placed in

the path is arbitrary, so that any permutation on the triangles is possible [2]. This will be used to complete a Hamiltonian cycle in Section 2.2.

By Theorem 8, the $n/3$ triangles produced have shortest edge length at least $r'/3$. Applying Theorem 9, we create a Hamiltonian path whose shortest 2-neighbor distance in the path is at least $r'/6$. Recall from Corollary 2 that $2r'$ is an upper bound on the optimal objective value, and thus the approximation factor of our algorithm is 12.

THEOREM 11. *Given a complete graph on set S of n points satisfying the triangle inequality, where n is a multiple of 3, there exists an algorithm that constructs in $O(n^{2.5})$ time a Hamiltonian path on S whose objective value for the max-min 2-neighbor problem is at least $\frac{1}{12}$ times the optimal.*

2.2. *Cycle Version for All Cases of n.* In this section we present our algorithm for the cycle version of the max-min 2-neighbor TSP, for *all cases* of n. To apply the base algorithm given in Section 2.1, we need to resolve two additional technical issues: (1) how to handle the case of n not being a multiple of 3, and (2) how to take into account the wrap-around effect when we chain the triangles to create a desirable Hamiltonian cycle. We proceed to address these issues.

Recall from Corollary 2 that $2r$ is an upper bound on the optimal objective value of the cycle version of the problem, where r is the radius of a smallest disk $\mathcal{C}$ centered at a point of S, containing exactly $\lfloor n/3 \rfloor$ points (including the center point) in its interior and one additional point on its boundary. As before, we can compute r and the center point of $\mathcal{C}$ in $O(n^2)$ time. Our overall strategy is similar to the base algorithm in Section 2.1, but for the cycle version instead: first, we modify Algorithm *Large-Triangles* to create $\lfloor n/3 \rfloor$ vertex-disjoint large triangles on the points of S, with one or two points left over if n is not a multiple of 3. In the latter case we perform an additional step to attach the left-over point(s) to one or two triangles to create large clique(s) of four or five vertices, viewed as special "triangle(s)." Finally, we chain the $\lfloor n/3 \rfloor$ triangles, special or not, to create a desirable Hamiltonian cycle.

We slightly modify Algorithm *Large-Triangles* as follows: we let $\mathcal{C}$ and r play the roles of $\mathcal{C}'$ and r', respectively. The disk $\mathcal{C}$ now contains $\lfloor n/3 \rfloor$ points in its interior. In Phase 1 these $\lfloor n/3 \rfloor$ points are grown into $\lfloor n/3 \rfloor$ segments of length at least r, and in Phases 2 and 3 these segments are grown into $\lfloor n/3 \rfloor$ triangles with shortest edge length at least $r/3$.

If n is not a multiple of 3, then there are one or two left-over points not in any triangle. Consider the more general case where there are two such points, p and q. We claim that among the $\lfloor n/3 \rfloor$ triangles, there must be one triangle, say $\triangle uvw$, whose vertices are all at a distance at least r from p. We then construct a special triangle $puvw$, which is a clique of four vertices with each edge length at least $r/3$. We prove the claim as follows. Suppose the claim were not true, that is, each of the $\lfloor n/3 \rfloor$ triangles would have at least one vertex within a distance less than r from p. Then the disk centered at p with radius r would contain at least $\lfloor n/3 \rfloor + 1$ points in its interior, including p, which contradicts the definition of r. Applying the same argument for q, there exists a triangle t, special or not, whose vertices are all at a distance at least r from q. We then include q into t as its new vertex to create a clique, whose edges are of length at least $r/3$. If t is the

special triangle $puvw$, then overall we have one special triangle $pquvw$; otherwise we have two special triangles of four vertices each.

Our next task is to chain the $\lfloor n/3 \rfloor$ triangles, possibly one or two being special triangles, to create a Hamiltonian cycle. Given such triangles with shortest edge length at least δ (here $\delta = r/3$), we use Theorem 12 below to create a Hamiltonian cycle whose shortest 2-neighbor distance is at least $\delta/3$.

THEOREM 12. *Given a list of* $\lfloor n/3 \rfloor$ *triangles* (*with possibly one or two being special triangles*) *whose shortest edge length is at least* δ, *we can chain the triangles in* $O(n^2)$ *time to create a Hamiltonian cycle such that any two nodes that are* 2*-neighbors in the cycle are at a distance of at least* $\delta/3$.

PROOF. First, consider the case in which there is no special triangle. Our algorithm proceeds in two steps. In Step 1 we check whether there exists a pair of triangles $\triangle abc$ and $\triangle xyz$ such that two vertices of $\triangle abc$, say a and b, are each at a distance of at least $\delta/3$ from each of x, y, and z. If so, we chain the triangles starting with $\triangle abc$, visiting in that order, and end with $\triangle xyz$, visiting in the order decided by its predecessor triangle, by applying Lemma 10. The order of other triangles placed in the cycle is arbitrary. Clearly, the resulting cycle has the desired property. Also, finding such pair of triangles takes $O(n^2)$ time.

Now suppose no such pair of triangles $\triangle abc$ and $\triangle xyz$ exists, and we proceed to Step 2. We pick an arbitrary triangle, $\triangle abc$, and construct disks A, B, and C of the same radius $\delta/3$ centered at a, b, and c, respectively. Since the search in Step 1 is not successful, A, B, and C must together contain (in their *interiors*) at least two vertices of each of the remaining triangles. Also, each disk contains at most one vertex of a triangle, since each triangle edge has length at least δ while the diameter of each disk is only $(2\delta)/3$. Therefore, each triangle can be classified into one of the following four types: ABC, ABx, AxC, and xBC, where A (resp. B and C) means the corresponding vertex is in the interior of disk A (resp. B and C), and x means the corresponding vertex is *not* in the interior of any disk. Observe that there is always a triangle of type ABC, namely, $\triangle abc$, but other types of triangles may or may not exist. It is easy to see that any two points in different disks are at a distance larger than $\delta/3$ from each other, since triangle $\triangle abc$ has shortest edge length at least δ. Moreover, for type ABx, the vertex x is at a distance larger than $\delta/3$ from any point in A (resp. B). (Suppose $\triangle a'b'x$ is one such triangle with $a' \in A$. For any point $a'' \in A$, $d(a'', a') < diam(A) = (2\delta)/3$, but $d(x, a') \geq \delta$, and thus $d(x, a'') \geq d(x, a') - d(a'', a') > \delta/3$.) Similar properties hold for types AxC and xBC.

We construct the Hamiltonian cycle by repeatedly traversing triangles of the same type (visiting the vertices in the order specified) until the triangles of that type are exhausted, and then going to the next type (or going back to the starting point), in the order given as follows:

$$ABC \rightarrow xBC \rightarrow AxC \rightarrow ABx \rightarrow \text{starting point.}$$

We refer to the above order as the *canonical order*. In cases where not all four types of triangles exist, we skip the missing type(s) and go to the next type in the canonical order.

We verify that the resulting Hamiltonian cycle has the claimed property. Recall that type ABC always exists. If this is the only type, then clearly the cycle is desirable. Suppose there are more than one type. It is easy to see that for all types, the vertex in A is always visited first, the vertex in B second, and the vertex in C third, so that vertices in disks are taken care of (any pair of vertices in disks that might possibly be 2-neighbors in the cycle are in *different* disks, separated by a distance larger than $\delta/3$). We therefore focus on vertex x (of types other than ABC) that is not in any disk. For type xBC (if it exists), the 2-neighbors *before* x from the predecessor type are vertices in B and C (the predecessor type is ABC, which always exists), which are away from x by a distance larger than $\delta/3$ since this is type xBC; the 2-neighbors *after* x do not go beyond the current type, and thus we need not worry about its successor type. For type AxC, the 2-neighbor before x from the predecessor type is in C (the predecessor type is either xBC, if it exists, or ABC otherwise), and the 2-neighbor after x from the successor type is in A (the successor type is either ABx, if it exists, or ABC otherwise—by the wrap-around effect). However, x is away from any point in A or in C by a distance larger than $\delta/3$, since this is type AxC. A similar argument applies to type ABx. We conclude that the resulting cycle is as desired.

Now we consider the case in which there are one or two special triangles. During the chaining process, we treat each special triangle, say $uvwp$ or $uvwpq$, as if it were a triangle. In order to concatenate it with its predecessor triangle, we treat it as triangle $\triangle uvw$. Applying Lemma 10, the predecessor will fix a specific ordering of points u, v, and w to be visited; we then visit the remaining points of the special triangle (p or both p and q). In order to concatenate a triangle after it, we consider only the last three points visited as a triangle, and apply Lemma 10.

We now consider the wrap-around effect. If there is a special triangle with five vertices, say $pquvw$, then at least two vertices, say p and q, must each be "far away" from all vertices of $\triangle abc$ (since A, B, and C together can contain at most three vertices of $pquvw$ in their interiors). Thus we can complete the chaining of triangles in Step 1 by starting with $pquvw$ (in that order) and ending with $\triangle abc$ (in the order required by its predecessor). Consider therefore special triangles K with four vertices. If K has less than three vertices in the interiors of A, B, and C, then again K has at least two vertices each far away from all the vertices of $\triangle abc$, and hence Step 1 can be completed. We thus only need to consider special triangles of type $ABCx$, where x (by the same argument) is at a distance larger than $\delta/3$ from any point in A (resp. in B and C). Our canonical order now becomes the following (note that types $ABCx$ and ABC always exist, and that the new canonical order is the same except for starting with the special triangles):

$$ABCx \rightarrow ABC \rightarrow xBC \rightarrow AxC \rightarrow ABx \rightarrow \text{starting point.}$$

As analyzed before, Step 1 takes $O(n^2)$ time. In Step 2, classifying triangles into four types and visiting the triangles according to the canonical order takes $O(n)$ time. The overall process thus takes $O(n^2)$ time. □

We remark that the bound $\delta/3$ in Theorem 12 is tight for the method. Suppose in Step 1 we search for a pair of triangles such that two vertices of one triangle are each far away from all three vertices of the other triangle by a distance at least z. This distance

z then decides the radius (which is z) of the disks A, B, and C in Step 2. Any pair of points in different disks are at a distance of at least $\delta - 2z$. Also, for type ABx (and similarly for types AxC and xBC), x is at a distance of at least $\delta - 2z$ from any point in A (resp. in B). The objective value of the resulting cycle is thus $\min\{z, \delta - 2z\}$, whose maximum occurs when $z = \delta - 2z$, i.e., when $z = \delta/3$, with the maximum value $\delta/3$.

After producing $\lfloor n/3 \rfloor$ triangles with smallest edge length at least $r/3$ ($= \delta$), possibly one or two triangles being special triangles, we apply Theorem 12 to create a Hamiltonian cycle whose shortest 2-neighbor distance is at least $\delta/3 = r/9$. Recall that $2r$ is an upper bound on the optimal objective value. The approximation factor of our algorithm is therefore 18.

THEOREM 13. *Given a complete graph on set S of n points satisfying the triangle inequality, for all cases of n, there exists an algorithm that constructs in $O(n^{2.5})$ time a Hamiltonian cycle on S whose objective value for the max-min 2-neighbor problem is at least $\frac{1}{18}$ times the optimal.*

2.3. *Path Version When n Is Not a Multiple of* 3. In this section we present our algorithms for the path version of the max-min 2-neighbor TSP when n is not a multiple of 3. Recall that the base algorithm described in Section 2.1 achieves an approximation factor of 12 for the path version when n is a multiple of 3 (Theorem 11); here we achieve the same factor for the remaining cases. As also observed in [2], the case of $n = 3k + 2$ is quite easy to handle, while the case of $n = 3k + 1$ is surprisingly difficult. We give two novel algorithms to deal with the latter case, which, in addition to improving the approximation factor given in [2], are much simpler than the method of [2], by exploring novel structural properties of the point set S under the distance metric. It is also worth noting that in the last case of both algorithms, we actually produce an *optimal* Hamiltonian path.

Recall from Corollary 2 that $2r'$ is an upper bound on the optimal objective value, where r' is the radius of a smallest disk C' centered at a point of S, containing exactly $\lceil n/3 \rceil$ points (including the center point) in its interior and one additional point on its boundary. Since now $n = 3k + i$, $i = 1$ or 2, C' contains in its interior $\lceil n/3 \rceil = k + 1$ points, as opposed to k points in the case of $n = 3k$. To apply the base algorithm, the idea is to add one or two *fake points* to make the total number of points a multiple of 3, described next.

When $n = 3k + 2$, we add one fake point Q far away from all n points of S so that Q does not affect the max-min 2-neighbor distance at all, and then apply Algorithm *Large-Triangles* on this new set of $n' = 3k + 3$ points. The disk C' (with radius r') now contains $n'/3 = k + 1$ points in the interior, as desired, and Algorithm *Large-Triangles* produces $n'/3 = k+1$ triangles of shortest edge length at least $r'/3$, where $2r'$ is a correct upper bound on the optimal objective value. Suppose $\triangle Qyz$ is the triangle containing Q produced by Algorithm *Large-Triangles*. Then in applying Theorem 9 to chain the $k + 1$ triangles into a Hamiltonian path, we start with $\triangle Qyz$, visiting the vertices in that order, and chain the remaining triangles in an arbitrary order. Finally, we remove Q and obtain a Hamiltonian path on S with approximation factor 12.

When $n = 3k + 1$, we apply the same trick, adding two fake points Q and Q' far away from all n points in S as well as from each other, and apply our algorithm on this

new set of $n' = 3k + 3$ points. Again disk $\mathcal{C}'$ and its radius r' are as desired. If we can make sure that Q and Q' are in the same triangle, say $\triangle QQ'z$, among the $k + 1$ triangles produced by Algorithm *Large-Triangles*, then we can again chain the triangles by visiting Q, Q', and z first (in that order), followed by the remaining triangles, and finally remove Q and Q' from the resulting path to obtain a Hamiltonian path on S with the same approximation factor.

In the rest of this section we show how to enhance Algorithm *Large-Triangles* so that we can construct $k + 1$ triangles with shortest edge length at least $r'/3$, such that Q and Q' are in the same triangle. This task poses a major technical challenge, since we have to perform swapping operations in Phase 3. A natural attempt to achieve this task is first to construct some triangle $\triangle xQQ'$, and then to try to show, by modifying the argument of Lemma 5, that there are still enough good triangles to perform the swappings, even *without using* $\triangle xQQ'$ during the swappings. Unfortunately this method does not work, and thus we have to employ more delicate techniques as presented in our two new algorithms.

We remark that in [2] an algorithm is proposed to construct $k+1$ triangles with shortest edge length at least $r'/8$ such that Q and Q' are in the same triangle. That algorithm, however, is extremely complicated, and would need a highly nontrivial extension, if not impossible, to improve the shortest edge bound from $r'/8$ to $r'/3$. Such an extension, if possible, would require many more additional cases to be analyzed and thus would make the already complicated algorithm even more complex. Both of our new algorithms, on the other hand, are much simpler. In particular, a case similar to our 1-*partition* in [2] is extremely difficult to handle in the algorithm of [2], but this case is trivial for both of our new algorithms, which employ a new direction of the construction process that gives an extra flexibility and consequently enables us to handle the case trivially.

Now we proceed to present our two new algorithms, denoted Algorithm I and Algorithm II.

2.3.1. *Algorithm I for the Case* $n = 3k + 1$. We assume that the two fake points Q and Q' have been added, and we are working on the new point set of $n' = 3k + 3$ points. Let a be the center point of $\mathcal{C}'$. Recall that in Phase 1 of Algorithm *Large-Triangles*, we grow each of the $k + 1$ points in the interior of $\mathcal{C}'$ into a segment so that we have $k + 1$ segments in total, each of length at least r'.

Now, in Algorithm I, we still produce $k+1$ segments of length at least r' in Phase 1, but one of them is the special segment (Q, Q'), so we must leave out one point s (specified later) in the interior of $\mathcal{C}'$ that is *not* grown into a segment in Phase 1. Moreover, the special segment (Q, Q') does *not* participate in the maximal matching in Phase 2, so that in the swapping operations in Phase 3 we have the flexibility of pairing up the segment (Q, Q') with a left-over point at our convenience, to produce a desired triangle with both Q and Q' staying together at the end.

We describe the algorithm in detail. In Phase 1 we construct a bipartite graph $G = (V_1, V_2, E)$ and perform a maximum cardinality matching as before, but modify the vertex sets V_1 and V_2 as follows: V_1 consists of the $k + 1$ points in the interior of $\mathcal{C}'$ (including the center point a), *except for* one point s (to be specified later); V_2 consists of all points outside or on the boundary of $\mathcal{C}'$, *except for* Q and Q'. Note that $|V_1| = k$ and $|V_2| = 2k$. We now specify how to choose the point s. Among the $k + 1$ points in

the interior of C', if there exists a point that is at distance at least $r'/3$ from a, then we choose one such point as s; if no such point exists, then we pick as s an arbitrary point such that $s \neq a$.

The edge set E of the graph G, as before, consists of edges from V_1 to V_2, between two nodes whose distance is at least r'. We perform a maximum cardinality matching on G. By Lemma 14 below, there exists a complete matching, and we produce k segments of length at least r' from the matching as before. We also produce the special segment (Q, Q'), whose length is of course larger than r'. Note that s is the only point in the interior of C' that is not in any segment.

LEMMA 14. *There exists a complete matching on the bipartite graph G.*

PROOF. We use Hall's theorem [8], which states: a complete matching of V_1 into V_2 exists if and only if for every subset A of V_1, $|A| \leq |\Gamma(A)|$, where $\Gamma(A)$ is a subset of the vertices of V_2 that are adjacent to at least one vertex in A.

Suppose that the lemma is not true, then there exists a set $A \subseteq V_1$ such that $|A| > |\Gamma(A)|$; we want to show a contradiction. Let p be a point in A, then the set $V_2 \backslash \Gamma(A)$ consists of all points in V_2 that are at a distance less than r' from p. Now consider how many points are in $V_2 \backslash \Gamma(A)$. Since $|V_1| = k$ and $|V_2| = 2k$, we have $|\Gamma(A)| < |A| \leq |V_1| = k$, i.e., $|\Gamma(A)| \leq k - 1$, and thus $|V_2 \backslash \Gamma(A)| \geq 2k - (k - 1) = k + 1$. This means that the disk centered at p with radius r' contains at least $k + 2$ points in the interior (at least $k + 1$ points from $V_2 \backslash \Gamma(A)$, plus one more point, p), but any such disk with radius r' can contain at most $k + 1$ points in the interior by the definition of r', a contradiction. □

In Phase 2 we construct a bipartite graph $H = (U_1, U_2, \mathcal{E})$ and perform a maximal matching on H as before, with the following modifications: the nodes of U_1 are all segments constructed in Phase 1 except for the special segment (Q, Q'), and the nodes of U_2 are all left-over points, including the point s in the interior of C'. The edge set $\mathcal{E}$, as before, consists of edges from U_1 to U_2, between two nodes $e \in U_1$ and $p \in U_2$ such that p is away from *both* endpoints of e by distances of at least $r'/3$. After the maximal matching, we construct a triangle of edge lengths at least $r'/3$ from each matched pair as before. Note that the segment (Q, Q') does *not* participate in the maximal matching and is thus a left-over segment. If (Q, Q') is the only left-over segment, then we pair up (Q, Q') with the only left-over point to construct a triangle and we are done; otherwise we leave (Q, Q') untouched and proceed to Phase 3.

In Phase 3 there are left-over segments other than (Q, Q') as well as left-over points. By Lemma 3, every left-over segment other than (Q, Q') induces either a 1-partition or a 2-partition on the set P of left-over points. Recall from Lemma 6 that for each cluster i, $i = 1, 2$, only triangles *good* for cluster i will be involved in the swapping operations in swapping stage i. Our main strategy is thus trying either to make the triangle T containing Q and Q' *bad* for cluster i in stage i, or, if T is possibly good for cluster i, to perform a swapping on a left-over point in cluster i without consuming any good triangle for cluster i, so that T is not needed in any remaining swappings and is left untouched. The algorithm proceeds by considering the following cases.

Case I: *There exists a left-over segment $e = (v_1, v_2)$ that induces a 1-partition.* We use e and the 1-partition $(P_1, \emptyset)$, where $P_1 = P$. We pick an arbitrary left-over point x and construct triangle $\triangle x Q Q'$, and proceed with the swapping operations as before. Notice that $\triangle x Q Q'$ is *bad* for cluster 1 (since x is in the interior of R_1, the disk centered at v_1 with radius r'), and hence will never be involved in any swapping operations, as desired—the case of 1-partition is trivially handled.

Case II: *Each left-over segment other than (Q, Q') induces a 2-partition.* We pick an arbitrary left-over segment (b, d) other than (Q, Q'), and use its 2-partition (P_1, P_2) by constructing disks C_1, D_1, R_1 centered at b with radii $r'/3$, $(2r')/3$, and r', respectively, and disks C_2, D_2, R_2 centered at d with radii $r'/3$, $(2r')/3$, and r'. Recall that exactly one of b, d is in the interior of the disk $\mathcal{C}'$ (by the construction in Phase 1); we call such an endpoint b and the other d. Also recall that a is the center point of $\mathcal{C}'$, and that s is the only point in the interior of $\mathcal{C}'$ that is left untouched in Phase 1. We consider the following cases.

Case II.1: $d(a, s) \geq r'/3$. We have the following nice properties.

LEMMA 15. *In Case* II.1 *the point a must be in a triangle, namely, a is neither a left-over point nor in a left-over segment. Moreover, this triangle is bad for cluster* 1, *with a in the interior of R_1.*

PROOF. By Lemma 14, we have a complete matching in Phase 1, and thus a is not a left-over point. Now, if a is in a left-over segment, say (a, v), then since each left-over segment other than (Q, Q') induces a 2-*partition*, there must be a left-over point x at a distance less than $r'/3$ from a. However, any left-over point is either outside or on the boundary of $\mathcal{C}'$ (at a distance at least r' from a), or possibly the point s (with $d(a, s) \geq r'/3$ by the condition of Case II.1), and thus no such x exists, a contradiction. We conclude that a must be in a triangle. Moreover, since b is in the interior of $\mathcal{C}'$, $d(a, b) < r'$, and hence a is in the interior of R_1, meaning that the triangle containing a is bad for cluster 1. □

The following lemma holds in general and will be used in several places, not just for Case II.1.

LEMMA 16. *Suppose (Q, Q') is already paired up with a left-over point so that (Q, Q') is in some triangle and not a left-over segment. Let ℓ_i be the number of left-over points in cluster i, $i = 1, 2$. If the disk R_i contains less than $k + 1$ points in its interior, or some bad triangle for cluster i has more than one vertex in the interior of R_i, then the number of good triangles for cluster i is at least $\ell_i + 1$.*

PROOF. This is a refinement of the proof of Lemma 5. After Phase 2 and putting (Q, Q') into some triangle, let T be the number of triangles, and let $T_{i,\text{good}}$ and $T_{i,\text{bad}}$ be the numbers of triangles good and bad for cluster i, respectively, where $T_{i,\text{good}} + T_{i,\text{bad}} = T$. Also, there are $(k + 1) - T$ left-over segments. The points in the *interior* of the disk R_i include ℓ_i left-over points in cluster i, at least one endpoint from each of the $(k + 1) - T$

left-over segments, and t_i vertices from bad triangles for cluster i, where $t_i \geq T_{i,\text{bad}}$. When R_i contains less than $k+1$ points in its interior, we have $k+1 > \ell_i + [(k+1) - T] + T_{i,\text{bad}}$. When some bad triangle has more than one vertex in the interior of R_i, we have $t_i \geq T_{i,\text{bad}} + 1$, and hence $k + 1 \geq \ell_i + [(k + 1) - T] + (T_{i,\text{bad}} + 1)$. In both cases we have $T_{i,\text{good}} = T - T_{i,\text{bad}} \geq \ell_i + 1$, as desired. □

Let the triangle containing a be $\triangle auv$, which is bad for cluster 1. We consider two subcases.

Case (a): *$\triangle auv$ has more than one vertex in the interior of R_1.* We pick any left-over point x in cluster 2, construct triangle $\triangle xQQ'$, and proceed to complete swappings on the left-over points in cluster 1 without touching $\triangle xQQ'$, and then complete swappings on the left-over points in cluster 2 without touching $\triangle xQQ'$. Note that $\triangle xQQ'$ may be good for cluster 1, but since the bad triangle $\triangle auv$ has more than one vertex in the interior of R_1, by Lemma 16 the number of good triangles for cluster 1 is at least one more than the number of left-over points in cluster 1, and hence we can complete swappings on cluster 1 without touching the possibly good triangle $\triangle xQQ'$. Also note that $\triangle xQQ'$ is bad for cluster 2 (by the choice of x), and hence $\triangle xQQ'$ is again untouched when we perform swappings on cluster 2. This completes the case.

Case (b): *$\triangle auv$ has exactly one vertex (namely a) in the interior of R_1 (and u and v are both outside or on the boundary of R_1).* We do the following: pick an arbitrary left-over point x in cluster 2 and construct triangle $\triangle xQQ'$ (which may be good for cluster 1 and is bad for cluster 2); pick an arbitrary left-over point y in cluster 1 and construct triangle $\triangle ayd$ (which is bad for both clusters); construct triangle $\triangle buv$ (which is bad for cluster 1 but may be good for cluster 2); finally, complete swappings on cluster 1, and then complete swappings on cluster 2. By Lemma 18 below, we can complete all swappings without touching $\triangle xQQ'$, and all resulting triangles have shortest edge length at least $r'/3$, as desired.

To prove the correctness, we first show a property of the primitive swapping operations in the following lemma, which holds in general and will be used in several places.

LEMMA 17. *Let $\triangle xyz$ be a good triangle for cluster i ($i = 1$ or 2) with shortest edge length at least $r'/3$. If $\triangle xyz$ has one edge with length at least $(2r')/3$, then we can swap $\triangle xyz$ with an arbitrary left-over point t in cluster i and an arbitrary left-over segment (p, q) to produce two triangles using the six points involved as triangle vertices, such that the two resulting triangles both have shortest edge length at least $r'/3$ and both are bad triangles for cluster i.*

PROOF. Without loss of generality, assume that edge (x, y) has length at least $(2r')/3$, and the other two edges of $\triangle xyz$ have length at least $r'/3$. By Lemma 4, at least one endpoint of (p, q), say q, is in the interior of R_i. Consider p and (x, y); suppose p is closer to x than to y (the other case is symmetric). We construct triangles $\triangle ypq$ and $\triangle xzt$. To verify that both triangles have the desired shortest edge length, we see that (p, q) has length at least r' from the construction of Phase 1. Also, we have $d(y, q) > r'/3$: Since y is outside or on the boundary of R_i (because $\triangle xyz$ is good for cluster i) and q is in

the interior of D_i by Lemma 4, where R_i and D_i are centered at the same point with radii r' and $(2r')/3$, respectively, we have $d(y, q) > r' - (2r')/3 = r'/3$. Moreover, we have $d(y, p) \geq r'/3$: if $d(x, p) \geq r'/3$, then $d(y, p) \geq d(x, p) \geq r'/3$; otherwise, $d(x, p) < r'/3$, and $d(x, y) \geq (2r')/3$, and hence $d(y, p) \geq d(x, y) - d(x, p) > r'/3$. Therefore $\triangle ypq$ has shortest edge length at least $r'/3$. Note that q is in the interior of R_i, and thus $\triangle ypq$ is bad for cluster i. As for $\triangle xzt$, edge (x, z) is an original edge of triangle $\triangle xyz$ and hence has length at least $r'/3$. Also, we have $d(x, t) > (2r')/3$, since x is outside or on the boundary of R_i and t is in the interior of C_i, where C_i has the same center point as R_i with radius $r'/3$, and thus $d(x, t) > r' - r'/3 = (2r')/3$. Similarly, we have $d(z, t) > (2r')/3$. Therefore $\triangle xzt$ has the desired shortest edge length. Since t is in the interior of R_i, $\triangle xzt$ is bad for cluster i. □

Now we are ready to show the correctness of the algorithm for this case.

LEMMA 18. *In Case* (b) *we can complete all swappings without touching* $\triangle xQQ'$, *and all resulting triangles have shortest edge length at least* $r'/3$.

PROOF. We first show that we can complete all swappings without touching $\triangle xQQ'$. In swappings on cluster 1, note that we effectively use the *bad* triangle $\triangle auv$, the left-over point y, and the left-over segment (b, d) to perform a swapping to create two triangles $\triangle ayd$ and $\triangle buv$, decreasing the number of left-over points in cluster 1 by one while keeping the number of good triangles for cluster 1 unchanged. Therefore, we can complete the swappings on cluster 1 without touching the possibly good triangle $\triangle xQQ'$. Also, $\triangle xQQ'$ is bad for cluster 2, and hence is untouched in swappings on cluster 2.

Now we show that all the resulting triangles have shortest edge length at least $r'/3$. For $\triangle ayd$, we have $d(a, y) \geq r'/3$ (since y is a left-over point, either $y = s$ or y is outside or on the boundary of C', i.e., $d(a, y) \geq r'/3$ or $d(a, y) \geq r'$), $d(a, d) \geq r'$ (since d is outside or on the boundary of C'), and $d(y, d) > (2r')/3$ (since $d(y, b) < r'/3$ and the left-over segment (b, d) has length at least r', we have $d(y, d) \geq d(b, d) - d(y, b) > r' - r'/3 = (2r')/3$). Also, $\triangle ayd$ is bad for both clusters 1 and 2, and will not participate in the remaining swappings. For $\triangle buv$, we have $d(b, u) \geq r'$ and $d(b, v) \geq r'$ (since u and v are outside or on the boundary of R_1 by the condition of Case (b) and R_1 is centered at b with radius r'), and $d(u, v) \geq r'/3$ (since (u, v) is originally an edge of $\triangle auv$ whose shortest edge length is at least $r'/3$). Note that $\triangle buv$ may be *good* for cluster 2 and hence may be needed for swappings on cluster 2. However, $\triangle buv$ has two edges (b, u) and (b, v) with length at least r'; by Lemma 17, the resulting triangles using $\triangle buv$ in the swapping have shortest edge length at least $r'/3$. Therefore, all resulting triangles after all swappings have edge length at least $r'/3$. □

Case II.2: $d(a, s) < r'/3$. By the way we choose s in Phase 1 (among the $k + 1$ points in the interior of C', if there exists a point that is at distance at least $r'/3$ from a, then we choose one such point as s; if no such point exists, then we pick as s an arbitrary point such that $s \neq a$), this condition ($d(a, s) < r'/3$) means that all the $k + 1$ points in the interior of C' are within a distance less than $r'/3$ from a.

Recall the definitions of the left-over segment (b, d) and of the disks R_i, D_i, C_i, $i = 1, 2$, from the beginning of Case II. Also note that in this case a is either in a triangle or in a left-over segment (compare Lemma 15 for Case II.1), and that it is possible to have $a = b$.

LEMMA 19. *Cluster* 1 *contains exactly one left-over point. Moreover, this left-over point is* s.

PROOF. By the construction in Phases 1 and 2, each constructed triangle has a vertex v in the interior of $\mathcal{C}'$. Such a vertex v is within distance $(2r')/3$ from b ($d(v, b) \leq d(v, a) + d(a, b) < r'/3 + r'/3$, by the fact that all points in the interior of $\mathcal{C}'$, including v and b, are within a distance less than $r'/3$ from a), meaning that v is in the interior of R_1 and hence each constructed triangle is *bad* for cluster 1. However, cluster 1 is *nonempty*, since the left-over segment (b, d) induces a 2-*partition*. This means that cluster 1 contains at least one left-over point, which must correspond to one *good* triangle for cluster 1 according to Lemma 5. The only triangle possibly good for cluster 1 is the triangle $\triangle x Q Q'$ for some left-over point x from *cluster* 2. If we instead construct a triangle $\triangle y Q Q'$ for some left-over point y in *cluster* 1, then every triangle is *bad* for cluster 1 and there cannot be any more left-over point in cluster 1 (by Lemma 5) other than y. Therefore, cluster 1 has exactly one left-over point y.

Now we show that $y = s$. Observe that $d(y, a) < (2r')/3$ (y is a left-over point in cluster 1, i.e., $d(y, b) < r'/3$; also, $d(a, b) < r'/3$, and hence $d(y, a) \leq d(y, b) + d(b, a) < (2r')/3$), namely, y is in the interior of $\mathcal{C}'$. However, all left-over points are outside or on the boundary of $\mathcal{C}'$ except for possibly s. Therefore, the only left-over point y in cluster 1 must be s. □

To take care of Case II.2 we first construct $\triangle s Q Q'$ and we are done with cluster 1 by Lemma 19. Note that $\triangle s Q Q'$ is possibly *good* for cluster 2. We want to complete swappings on the left-over points of cluster 2 without touching $\triangle s Q Q'$ whenever possible.

Case II.2.1: R_2 (*centered at* d *with radius* r') *contains less than* $n'/3$ $(= k + 1)$ *points in its interior, or some triangle bad for cluster* 2 *has more than one vertex in the interior of* R_2. By Lemma 16, the number of good triangles for cluster 2 is at least one more than the number of left-over points in cluster 2, and hence we can complete swappings on cluster 2 without touching the possibly good triangle $\triangle s Q Q'$.

Case II.2.2: R_2 *contains exactly* $n'/3 = k + 1$ *points in its interior, and each triangle bad for cluster* 2 *has exactly one vertex in the interior of* R_2. *In addition, there exists a point* t *in the interior of* R_2 *with* $d(t, d) \geq r'/3$. Note that t cannot be a left-over point, since we are already done with cluster 1—any left-over point x must be in cluster 2, with $d(x, d) < r'/3$, but $d(t, d) \geq r'/3$ and thus $t \neq x$. We thus consider two subcases: (a) t is in a left-over segment and (b) t is in a triangle.

Case (a): t *is an endpoint of some left-over segment* (t, p).

CLAIM 20. *In Case* (a), *we can carry out the desired swapping operations*.

PROOF. If t is in the interior of $\mathcal{C}'$ (then p is outside or on the boundary of $\mathcal{C}'$), we swap $\triangle sQQ'$ to $\triangle tQQ'$, construct a new left-over segment (s, p) (this effectively swaps (t, p) to (s, p)), and proceed to complete swappings on the left-over points in cluster 2 without touching $\triangle tQQ'$ (since $\triangle tQQ'$ is *bad* for cluster 2). Note that the new segment (s, p) has length at least $(2r')/3$ (since p is outside or on the boundary of $\mathcal{C}'$, i.e., $d(p, a) \geq r'$, but $d(s, a) < r'/3$ (by the condition of Case II.2), and thus $d(s, p) \geq d(p, a) - d(s, a) > (2r')/3)$, so the swapping on a left-over point in cluster 2 using segment (s, p) still results in triangles with shortest edge length at least $r'/3$ (recall from Lemma 6 and Figures 2 and 3), as desired.

Otherwise, t is outside or on the boundary of $\mathcal{C}'$ (and p is in the interior of $\mathcal{C}'$). We pick up any left-over point x in cluster 2 and two left-over segments (t, p) and (b, d), and construct $\triangle tbd$ and a new left-over segment (p, x). Note that this reduces the number of left-over points in cluster 2 by one while keeping the number of good triangles for cluster 2 unchanged, and hence we can proceed to complete swappings on all left-over points in cluster 2 without touching the possibly *good* triangle $\triangle sQQ'$. Moreover, $\triangle tbd$ has shortest edge length at least $r'/3$: $d(t, b) > (2r')/3$ (since $d(t, b) \geq d(t, a) - d(b, a) > r' - r'/3$), $d(t, d) \geq r'/3$ (by the condition of Case II.2.2), and $d(b, d) \geq r'$ ((b, d) is a left-over segment). Also, $\triangle tbd$ is *bad* for cluster 2 and is untouched during the swappings on cluster 2. The new left-over segment (p, x) has length larger than $(2r')/3$ (since x is a left-over point with $x \neq s$, x is outside or on the boundary of $\mathcal{C}'$, i.e., $d(x, a) \geq r'$, but p, being in the interior of $\mathcal{C}'$, is within a distance less than $r'/3$ from a, and hence $d(p, x) \geq d(x, a) - d(p, a) > (2r')/3$), and thus the swapping on a left-over point in cluster 2 using segment (p, x) still results in triangles with shortest edge length at least $r'/3$, as desired. □

Case (b): *t is a vertex of some triangle $\triangle tuv$.*

CLAIM 21. *In Case* (b), *we can carry out the desired swapping operations.*

PROOF. Since t is in the interior of R_2, $\triangle tuv$ is *bad* for cluster 2 and thus both u and v are outside or on the boundary of R_2 (since exactly one vertex of any bad triangle is in the interior of R_2).

If t is in the interior of $\mathcal{C}'$, then we swap $\triangle sQQ'$ to $\triangle tQQ'$, construct $\triangle suv$, and complete swappings on the left-over points in cluster 2 without touching the *bad* triangle $\triangle tQQ'$. (It is possible that $t = s$ and $\triangle tuv = \triangle sQQ' = \triangle tQQ'$, but this case is trivial: since $\triangle sQQ'$ is already *bad* for cluster 2 ($t = s$ is in the interior of R_2), we can just complete swappings on the left-over points in cluster 2 without touching $\triangle sQQ'$.) Note that $\triangle suv$ has shortest edge length at least $r'/3$: $d(u, s) > (2r')/3$ (since t is in the interior of $\mathcal{C}'$, u must be outside or on the boundary of $\mathcal{C}'$, i.e., $d(u, a) \geq r'$, but $d(s, a) < r'/3$ by the condition of Case II.2, and thus $d(u, s) \geq d(u, a) - d(s, a) > (2r')/3)$, $d(v, s) > (2r')/3$ (same reason), and $d(u, v) \geq r'/3$ ((u, v) is an edge of the original triangle $\triangle tuv$). Also, $\triangle suv$ may be *good* for cluster 2 and hence may be used during the swappings on cluster 2, but $\triangle suv$ has two edges with length at least $(2r')/3$, and thus by Lemma 17 the resulting triangles of the swapping have shortest edge length at least $r'/3$, as desired.

Otherwise, t is outside or on the boundary of $\mathcal{C}'$. We pick up any left-over point x in cluster 2, the left-over segment (b, d) and the *bad* triangle $\triangle tuv$, and swap to construct two triangles $\triangle tbd$ and $\triangle xuv$. This reduces the number of left-over points in cluster 2 by one while keeping the number of good triangles for cluster 2 unchanged, and hence we can continue to finish swappings on all left-over points in cluster 2 without touching the possibly *good* triangle $\triangle sQQ'$. (In this case $s \neq t$, since s is in the interior of $\mathcal{C}'$ but t is outside or on the boundary of $\mathcal{C}'$.) Note that $\triangle tbd$ and $\triangle xuv$ are both *bad* for cluster 2 and are not touched in the remaining swappings on cluster 2. Also, both triangles have shortest edge length at least $r'/3$. For $\triangle tbd$, the same reason as in Case (a) applies (see the proof of Claim 20); for $\triangle xuv$, we have $d(x, u) > (2r')/3$ (u is outside or on the boundary of R_2, i.e., $d(u, d) \geq r'$, and x is a left-over point in cluster 2, i.e., $d(x, d) < r'/3$, and hence $d(x, u) \geq d(u, d) - d(x, d) > (2r')/3$), $d(x, v) > (2r')/3$ (same reason), and $d(u, v) \geq r'/3$ ((u, v) is an edge of the original triangle $\triangle tuv$), as desired. □

Case II.2.3: *R_2 contains exactly $n'/3 = k + 1$ points in its interior, and each triangle bad for cluster 2 has exactly one vertex in the interior of R_2. In addition, there is no point t in the interior of R_2 with $d(t, d) \geq r'/3$, i.e., all the $k + 1$ points (including d) in the interior of R_2 are within distance less than $r'/3$ from d.*

LEMMA 22. *We can construct an optimal max-min 2-neighbor Hamiltonian path on the original point set S.*

PROOF. From now on, we disregard the two fake points Q and Q' and consider S only. Recall that all $k + 1$ points (including a) in the interior of $\mathcal{C}'$ are within a distance less than $r'/3$ from a. Let S_1 be the set of these $k + 1$ points, and let S_2 be the set of the $k + 1$ points (including d) in the interior of R_2. Observe that the centers a of $\mathcal{C}'$ and d of R_2 are at a distance at least r' from each other, since d is outside or on the boundary of $\mathcal{C}'$. Therefore S_1 and S_2 are *disjoint*, and for any pair of points $x \in S_1$ and $y \in S_2$ we have $d(x, y) > r' - r'/3 - r'/3 = r'/3$. Let $S' = (S \backslash S_1) \backslash S_2$; we have $|S'| = k - 1$. Also, the diameters of S_1 and S_2 are both less than $(r'/3) \cdot 2 = (2r')/3$.

Let $d(S_1, S_2)$ denote the *shortest distance* between any pair of points, one from S_1 and the other from S_2. We claim that $d(S_1, S_2)$, $d(S_1, S')$, and $d(S', S_2)$ are all larger than $(2r')/3$. To see that $d(S_1, S_2) > (2r')/3$, observe that any point $s_2 \in S_2$ is outside or on the boundary of $\mathcal{C}'$, i.e., $d(s_2, a) \geq r'$, but any point $s_1 \in S_1$ has $d(s_1, a) < r'/3$, and hence $d(s_2, s_1) \geq d(s_2, a) - d(s_1, a) > (2r')/3$. Similarly, any point of S' is outside or on the boundary of $\mathcal{C}'$ and thus $d(S_1, S') > (2r')/3$. As for $d(S', S_2)$, any point $s' \in S'$ is outside or on the boundary of R_2, i.e., $d(s', d) \geq r'$, but any point $s_2 \in S_2$ has $d(s_2, d) < r'/3$, and thus $d(s', s_2) > (2r')/3$. We depict the structure of the point sets S_1, S_2, and S' in Figure 4.

Without loss of generality, assume that $diam(S_1) < diam(S_2)$. We first show that $diam(S_2)$ is an upper bound for d^*, the max-min 2-neighbor distance of an optimal path. To avoid having any pair of 2-neighbors both in S_1, the Hamiltonian path starts from a point in S_1, and visits two points outside S_1 (one in S_2 and one in S'), then repeats the process. After repeating the process $(k - 1)$ times, S_1 and S_2 both have two points left, and S' is empty. The only possible way to avoid having any pair of 2-neighbors both in

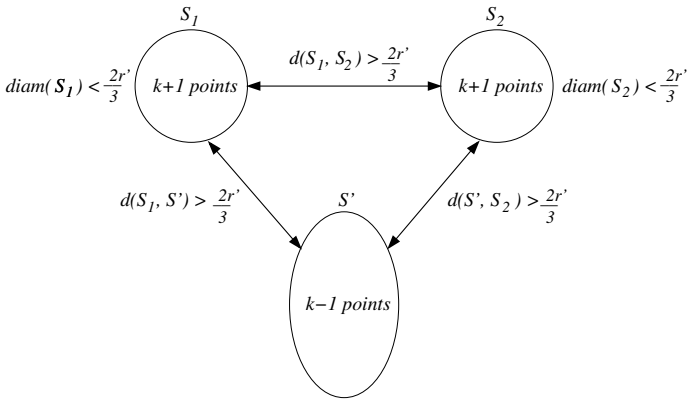

Fig. 4. Proof of Lemma 22: the structure of the point sets S_1, S_2, and S'.

S_1 is to visit the kth point of S_1, then the last two points of S_2, and finally the last point of S_1. This means that there must be a pair of points both in S_2 that are 2-neighbors, i.e., $diam(S_2)$ is an upper bound for d^*. We can achieve this upper bound by constructing the path $(S_1S_2S')^{k-1}S_1S_2S_2S_1$, where each occurrence of S_1 means visiting a point in S_1, and similarly for the occurrences of S_2 and S', and the last two points in S_2 to be visited are the pair in S_2 defining $diam(S_2)$. Clearly, all other 2-neighbors in this path have distances larger than $(2r')/3$, which is larger than $diam(S_2)$, and the minimum 2-neighbor distance is $diam(S_2)$, the upper bound for d^*. Therefore $d^* = diam(S_2)$ and the path is optimal. □

This completes Algorithm I for the case of $n = 3k + 1$. It is easy to see that the running time is the same as before.

THEOREM 23. *For the case of* $n = 3k+1$, *Algorithm* I *either constructs* $k+1$ *triangles with shortest edge length at least* $r'/3$ *while keeping the two fake points* Q, Q' *in the same triangle*, *or constructs an optimal max-min* 2*-neighbor Hamiltonian path on* S, *in* $O(n^{2.5})$ *time*.

2.3.2. *Algorithm* II *for the Case* $n = 3k + 1$. In this section we present Algorithm II for the case $n = 3k + 1$. Algorithm II simplifies Algorithm I of the last section; it is the same as Algorithm I from the beginning to Case II.1 (including Cases (a) and (b)), and starting from Case II.2 Algorithm II works as follows.

Case II.2: *Each left-over segment other than* (Q, Q') *induces a* 2*-partition*, *and* $d(a, s) < r'/3$. Recall from Lemma 19 that cluster 1 has exactly one left-over point s. Again we construct $\triangle sQQ'$ and we are done with cluster 1. We want to complete swappings on the left-over points of cluster 2 without touching the possibly *good* triangle $\triangle sQQ'$ whenever possible. In the following the disk D_2 centered at d with radius $(2r')/3$ will play a similar role to R_2, and thus we denote D_2 by R'_2 from now on.

Case II.2.1: *In the interior of R'_2 there exists an endpoint t of some left-over segment (t, p) such that $d(t, d) \geq r'/3$.*

CLAIM 24. *In Case* II.2.1, *we can carry out the desired swapping operations.*

PROOF. Before constructing $\triangle s\,Q\,Q'$ (i.e., when s is still a left-over point), (t, p) induces a 2-partition, and hence there exists a left-over point x with $d(x, t) < r'/3$. We pick x and two left-over segments (t, p) and (b, d) to construct $\triangle tbd$ and a new left-over segment (p, x) (the latter effectively swaps (t, p) to (p, x)). If x is not s, then x is a left-over point in cluster 2 (by Lemma 19); the operation reduces the number of left-over points in cluster 2 by one while keeping the number of good triangles for cluster 2 unchanged, and thus we can complete swappings on cluster 2 without touching the possibly *good* triangle $\triangle s\,Q\,Q'$. If x is s, then $(p, x) = (p, s)$; we pick any left-over point y in cluster 2 to construct $\triangle y\,Q\,Q'$, and complete swappings on cluster 2 without touching the *bad* triangle $\triangle y\,Q\,Q'$. Note that $\triangle tbd$ is *bad* for cluster 2 and will not be touched in the remaining swappings on cluster 2. Also, $\triangle tbd$ has shortest edge lengths at least $r'/3$: $d(b, d) \geq r'$ (length of the left-over segment (b, d)), $d(d, t) \geq r'/3$ (condition of Case II.2.1), and $d(b, t) > r'/3$ ($d(b, d) \geq r'$, $d(t, d) < (2r')/3$ since t is in the interior of R'_2, and $d(b, t) \geq d(b, d) - d(t, d)$). Finally, (p, x) has length at least $(2r')/3$ (since (p, t) is a segment of length at least r' and $d(x, t) < r'/3$), and thus the swapping on cluster 2 using (p, x) results in triangles of desired shortest edge length. □

Case II.2.2: *All segment endpoints lying in the interior of R'_2 are within a distance less than $r'/3$ from d.* This means that all left-over points in cluster 2 and all segment endpoints in the interior of R'_2 are lying in the interior of the same disk C_2 (centered at d with radius $r'/3$). From now on, we *redefine* a triangle to be good/bad for cluster 2 using R'_2 (rather than R_2), that is, a triangle is *good* for cluster 2 if all its vertices are outside or on the boundary of R'_2, and *bad* otherwise. Observe that we can still pick a left-over point in cluster 2, a left-over segment, and a good triangle for cluster 2 to perform a swapping to obtain two triangles of shortest edge length at least $r'/3$, since the radius difference between R'_2 and C_2 is $r'/3$. In addition, we have the same property that the number of good triangles for cluster 2 is at least the number of left-over points in cluster 2.

Case 1: *R'_2 contains less than $n'/3 = k+1$ points in its interior, or some bad triangle for cluster* 2 *has more than one vertex in the interior of R'_2.* It is easy to see that Lemma 16 carries over for R'_2, namely, the number of good triangles for cluster 2 is at least one more than the number of left-over points in cluster 2, and thus we can complete swappings on cluster 2 without touching the possibly good triangle $\triangle s\,Q\,Q'$.

Case 2: *R'_2 contains exactly $k+1$ points in its interior, and each bad triangle for cluster* 2 *has exactly one vertex in the interior of R'_2. In addition, there exists a point t in the interior of R'_2 such that $d(t, d) \geq r'/3$.*

LEMMA 25. *In Case* 2, *t must be in a triangle, namely, t is neither a left-over point nor in a left-over segment. Moreover, t cannot be s, and thus this triangle is not $\triangle s\,Q\,Q'$.*

PROOF. We first show that t cannot be a left-over point. Since t is in the interior of R_2' (i.e., $d(t,d) < (2r')/3$) and (b,d) is a left-over segment of length at least r', we have $d(b,t) \geq d(b,d) - d(t,d) > r'/3$. Also, we have $d(t,d) \geq r'/3$ by the condition of Case 2. Therefore, t is at a distance at least $r'/3$ from *both* endpoints of (b,d) and thus cannot be a left-over point (otherwise t and (b,d) would have been matched in Phase 2 and would not be a left-over point and segment). It is easy to see that t cannot be an endpoint of a left-over segment: by the condition of Case II.2.2, all segment endpoints in the interior of R_2' are within a distance less than $r'/3$ from d, but t is in the interior of R_2' with $d(t,d) \geq r'/3$. Therefore t must be in a triangle. To see that $t \neq s$, observe that $d(s,b) < r'/3$ (recall that s is the only left-over point in cluster 1 by Lemma 19) and (b,d) is a left-over segment of length at least r', and hence we have $d(s,d) > (2r')/3$, namely, s is outside R_2'. (This also shows that $\triangle s\,Q\,Q'$ is in fact *good* for cluster 2.) Since t is in the interior of R_2', t cannot be s. □

CLAIM 26. *In Case* 2, *we can carry out the desired swapping operations*.

PROOF. Let the triangle containing t be $\triangle tuv$. Since t is in the interior of R_2', $\triangle tuv$ is *bad* for cluster 2, and thus u and v are both outside or on the boundary of R_2' (by the condition of Case 2, each bad triangle for cluster 2 has exactly one vertex in the interior of R_2'). We pick any left-over point x in cluster 2, the left-over segment (b,d), and $\triangle tuv$ (bad for cluster 2) to construct two triangles $\triangle tbd$ and $\triangle xuv$. This reduces the number of left-over points in cluster 2 by one while keeping the number of good triangles for cluster 2 unchanged, and hence we can continue to complete swappings on cluster 2 without touching the good triangle $\triangle s\,Q\,Q'$. Note that $\triangle tbd$ has shortest edge length at least $r'/3$ (t is away from both b and d by distance at least $r'/3$ as shown in the proof of Lemma 25, and $d(b,d) \geq r'$), and that $\triangle xuv$ has shortest edge length at least $r'/3$ too (since u and v are both outside or on the boundary of R_2' and x is in the interior of C_2, $d(u,x)$ and $d(v,x)$ are both larger than $r'/3$, and $d(u,v) \geq r'/3$ since (u,v) is an edge of the original triangle $\triangle tuv$). Finally, $\triangle tbd$ and $\triangle xuv$ are both *bad* for cluster 2 and will not be touched in the remaining swappings on cluster 2, and therefore all resulting triangles at the end have shortest edge length at least $r'/3$, as desired. □

Case 3: R_2' *contains exactly* $n'/3 = k+1$ *points in its interior, and each bad triangle for cluster* 2 *has exactly one vertex in the interior of* R_2'. *In addition, there is no point* t *in the interior of* R_2' *with* $d(t,d) \geq r'/3$, *i.e., all the* $k+1$ *points* (*including* d) *in the interior of* R_2' *are within distance less than* $r'/3$ *from* d.

LEMMA 27. *We can construct an optimal max-min* 2*-neighbor Hamiltonian path on the original point set* S.

PROOF. By the definition of r', the disk R_2 centered at d with radius r' has no more than $n'/3 = k+1$ points in its interior, but R_2' centered at d with radius $(2r')/3$ has exactly $k+1$ points in its interior, meaning that R_2 actually contains exactly $k+1$ points in its interior. Moreover, these $k+1$ points (including d) are within a distance less than

$r'/3$ from d. This is exactly the same condition as that of the last case (Case II.2.3) of Algorithm I, and Lemma 22 carries over. □

THEOREM 28. *For the case of $n = 3k+1$, Algorithm* II *either constructs $k+1$ triangles with shortest edge length at least $r'/3$ while keeping the two fake points Q, Q' in the same triangle, or constructs an optimal max-min* 2*-neighbor Hamiltonian path on S, in $O(n^{2.5})$ time.*

Algorithm I (Theorem 23) and Algorithm II (Theorem 28) provide two alternatives for handling the difficult case of $n = 3k + 1$. Combining the algorithms for handling other cases, we have

THEOREM 29. *Given a complete graph on set S of n points satisfying the triangle inequality, for all cases of n, we can construct in $O(n^{2.5})$ time a Hamiltonian path on S whose objective value for the max-min* 2*-neighbor problem is at least $\frac{1}{12}$ times the optimal.*

3. Max-Min *m*-Neighbor TSP. In this section we present our *constant-factor* approximation algorithm for the max-min m-neighbor TSP, for any *input parameter* $m \in (2, n)$. For simplicity, we only consider the path version where n is a multiple of $m + 1$ or when $n = (m + 1)k + m$. The method of "closing" a path into a cycle in Theorem 12 and the techniques of handling the case in which n is not a multiple of $m + 1$ (except for $n = (m + 1)k + m$) for the path version in Section 2.3 are not easily extended to the case in which $m > 2$; we pose them as open questions.

We first consider the case where n is a multiple of $m + 1$. The overall strategy is the following:

1. Create $n/(m + 1)$ vertex-disjoint $(m + 1)$-cliques whose vertices are the points in S and whose edges are "long," meaning that the edges are of length at least some *constant* fraction of an upper bound of the optimal objective value. An $(m+1)$-clique K_{m+1} consists of $m + 1$ vertices that are points in S; the *clique edges* of K_{m+1} are implicitly defined by its vertices.
2. Concatenate these $(m + 1)$-cliques together to create a Hamiltonian path such that any m-neighbors on the path are far apart.

Let r be the radius of a smallest disk $\mathcal{C}$ centered at one of the n points such that it contains exactly $n/(m + 1)$ points of S (including the center point) in its interior, and one additional point on its boundary. By Lemma 1 (and analogous to Corollary 2), $2r$ is an upper bound on the optimal objective value of the path version of the max-min m-neighbor TSP (since n is a multiple of $m + 1$). Again, $\mathcal{C}$ and r can be easily computed in $O(n^2)$ time.

After computing $\mathcal{C}$ and r, we use Algorithm *Large-$(m + 1)$-Cliques* below to grow the $n/(m + 1)$ points in the *interior* of $\mathcal{C}$ into $n/(m + 1)$ vertex-disjoint $(m + 1)$-cliques. The process consists of m phases, where in phase i we grow each i-clique into an $(i + 1)$-clique.

Algorithm *Large-*$(m+1)$*-Cliques*

The algorithm consists of m phases, classified into two parts.

Part 1. Complete matching phases

This part consists of $\lfloor m/2 \rfloor$ phases. For each phase $i = 1, 2, \ldots, \lfloor m/2 \rfloor$, perform the following.

Phase i:
Construct a bipartite graph $G = (V_1, V_2, E)$, where the nodes V_1 represent the i-cliques constructed in phase $i-1$ (if $i = 1$, then the nodes V_1 are the points in the interior of $\mathcal{C}$), and V_2 are the left-over points. Note that $|V_1| = n/(m+1)$ and $|V_2| = n - i \cdot (n/(m+1)) = (m+1-i)(n/(m+1))$. Edge set E consists of "long" edges from V_1 to V_2, between two nodes $K_i \in V_1$ and $p \in V_2$ where p is at a distance at least r from *all* vertices of K_i. Perform a maximum cardinality matching on G. As proved in [2], all nodes of V_1 are matched. For each matched pair $K_i \in V_1$ and $p \in V_2$, grow the i-clique K_i into an $(i+1)$-clique by including point p as its additional vertex. Clearly, all clique edges of all resulting $(i+1)$-cliques have lengths at least r.

Part 2. Maximal matching and swapping phases

This part consists of the remaining $\lceil m/2 \rceil$ phases. For each phase $i = \lfloor m/2 \rfloor + 1, \ldots, m$, perform the following.

Phase i:
Construct a bipartite graph $G = (V_1, V_2, E)$ in the same way as in Part 1, except that the edge set E is specified as follows: there is an edge between two nodes $K_i \in V_1$ and $p \in V_2$ if and only if p is at a distance at least $r/2$ (defined as δ) from *all* vertices of K_i. Now perform the following steps.

1. Maximal matching step.
   Perform a *maximal* matching (*not* maximum cardinality matching) on G. For each matched pair K_i and p, grow K_i into an $(i+1)$-clique by including p as its new vertex. If all nodes in V_1 are matched, stop phase i; otherwise go to the next step.
2. Swapping step.
   The primitive operation here is to take a left-over point p, a left-over i-clique K_i ("left-over" from the matching in Step 1), and an $(i+1)$-clique K_{i+1} grown from Step 1, and *swap* them to obtain two "large" $(i+1)$-cliques by replacing some vertex x of K_{i+1} with p, and growing K_i into an $(i+1)$-clique by including x. (See Lemma 31 and Figure 5 below for the detailed operation.) Repeat this operation on each left-over i-clique so that all left-over i-cliques are grown into $(i+1)$-cliques.

The major achievement of Algorithm *Large-*$(m+1)$*-Cliques* lies in Part 2, where we are able to keep the shortest clique-edge length at least some *fixed constant* fraction of r *throughout all phases*, thus achieving an *exponential improvement* on the approximation factor over the previous best method [2] (whose factor is exponential in m). Our

algorithm achieves this improvement by exploiting some nice structural properties of the point set S under the distance metric, together with delicate matching and swapping techniques.

Now we describe the process of Part 2 more globally, across different phases. Let phase h be the first phase that its Step 1 does not give a complete matching. We choose an arbitrary left-over h-clique K_h not matched in Step 1 as the *base clique*. For each vertex v_j of K_h, $j = 1, \ldots, h$, we construct a pair of disks C_j and R_j centered at v_j with radii $r/2$ and r, respectively. By Lemma 30 below, all left-over points are clustered in the interior of some disks C_j's; there are at most h clusters, and two different clusters are *not* necessarily disjoint. (Note that this is a major difference from Algorithm *Large-Triangles* in Section 2.1 for the max-min 2-neighbor TSP where the clusters are *disjoint*.)

From now on, disks C_j and R_j ($j = 1, \ldots, h$) are *fixed* (in terms of the center points and radii) across all the remaining phases, and, by Corollary 32 below, the left-over points in each remaining phase are again in the interior of some disks C_j's. We refer to the *nonempty* set of left-over points in the interior of C_j as *cluster* j. In each phase i, $i \geq h$, we say that an $(i+1)$-clique K_{i+1} is *good* for cluster j if all vertices of K_{i+1} are outside or on the boundary of R_j, and *bad* otherwise.

In Step 2 of each phase i, we grow each left-over i-clique into an $(i+1)$-clique by picking a left-over point, which must belong to some cluster j, and an $(i+1)$-clique *good* for cluster j, and performing the swapping operation of Lemma 31 below. By Lemma 33 below, there are always *enough* good $(i+1)$-cliques to be used for swapping, so that all left-over i-cliques can be grown into $(i+1)$-cliques, for each phase $i \geq h$.

We first prove some properties mentioned above.

LEMMA 30. *In phase h all left-over points right after Step* 1 *are clustered in the interior of some disks C_j's. There are at most h clusters, and two different clusters are* not *necessarily disjoint.*

PROOF. Suppose some left-over point p were *not* in the interior of any of the disks C_j's, then p would be at distances at least $r/2$ from *all* vertices of K_h, and hence p and K_h would have been matched in Step 1 of this phase and would not be a left-over point and clique, a contradiction. Also, since the edge lengths of K_h are only at least $r/2$, the disks C_j's may overlap, and hence two different clusters are not necessarily disjoint. □

LEMMA 31 (The Swapping Operation). *Let K be an $(i+1)$-clique good for cluster j, with edge lengths at least $\delta = r/2$. We can pick any left-over point p in cluster j, any left-over i-clique k and swap them into two $(i+1)$-cliques K' (modified from K) and k' (grown from k), such that the shortest edge length of K' is still at least δ and the new vertex x in k' is away from all original vertices of k by a distance of at least $\delta/2$. This swapping operation takes $O(i^2)$ time.*

PROOF. Our swapping operation consists of the following two steps: (i) find a vertex x of the $(i+1)$-clique K that is away from all vertices of the left-over i-clique k by a distance of at least $\delta/2$; (ii) grow the i-clique k into an $(i+1)$-clique k' by including x as its new vertex, and modify the $(i+1)$-clique K into a new $(i+1)$-clique K' by replacing

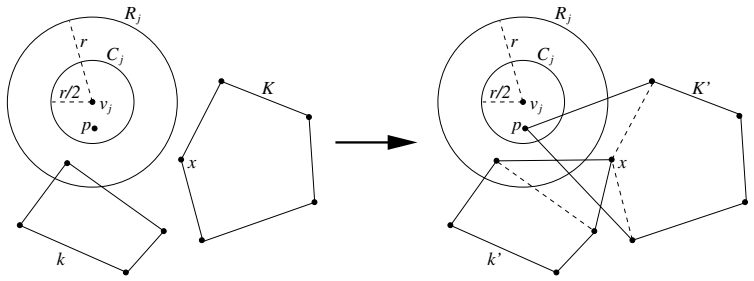

Fig. 5. The swapping operation of Lemma 31: the left-over 4-clique k (left) is grown into a 5-clique k' (right) by including x as its new vertex, and the 5-clique K (left), which is good for cluster j, is modified to become the 5-clique K' (right) by swapping out its original vertex x and swapping in a left-over point p in cluster j as its new vertex.

its vertex x with the left-over point p, i.e., by *swapping out* vertex x and *swapping in* p as a new vertex. See Figure 5.

We first prove that the shortest edge of K' still has length at least δ. We only need to show that the new vertex p is away from all other vertices of K' by a distance of at least $\delta = r/2$. Since p is in the *interior* of C_j (i.e., $d(p, v_j) < r/2$) and originally K is *good* for cluster j, i.e., any vertex y has $d(y, v_j) \geq r$, we have $d(y, p) \geq d(y, v_j) - d(p, v_j) > r/2$. Now we show that in step (i) we can always find such a vertex x. Initially, all $i + 1$ vertices of K are candidates for x. We consider the i vertices $u_1, \ldots, u_i$ of k one by one. We claim that u_1 can be *too close* (within a distance less than $\delta/2$) to at most one vertex of K: if there is a vertex v_t of K that is too close to u_1 (i.e., $d(u_1, v_t) < \delta/2$), then since all edges of K have length at least δ, we have $d(v, v_t) \geq \delta$ for any vertex $v \neq v_t$ of K, and thus $d(v, u_1) \geq d(v, v_t) - d(u_1, v_t) > \delta/2$. Therefore, u_1 can only remove at most one candidate v_t from the pool of $i + 1$ candidates for x. Now consider u_2 and the remaining i candidates. Again u_2 can only remove at most one candidate by the same argument. Repeating this process for all i vertices $u_1, \ldots, u_i$, there are i constraints, removing at most i candidates. However, we have $i + 1$ candidates for x to start with, and hence we can always find x. This process clearly takes $O(i^2)$ time. □

COROLLARY 32. *The properties of Lemma* 30 *hold for* all *phases* $i \geq h$ *as well. That is, for all phases* i, $i \geq h$, *all left-over points immediately after Step* 1 *are clustered in the* interior *of some of the disks* C_j*'s. There are at most* h *clusters, and two different clusters are* not *necessarily disjoint.*

LEMMA 33. *Consider any phase* i, $i \geq h$, *and any cluster* j, $1 \leq j \leq h$, *at any time when Step* 2 (*the swapping step*) *is being performed. The number of* $(i+1)$*-clusters that are* good *for cluster* j *is always at least the number of the left-over points in cluster* j, *so that the swapping operation of Lemma* 31 *can always be performed.*

PROOF. This is analogous to Lemma 5. Suppose there are T $(i+1)$-cliques (and hence $n/(m+1) - T$ left-over i-cliques), and ℓ_j left-over points in cluster j. Among the T

$(i+1)$-cliques, suppose $T_{j,\text{good}}$ of them are *good* for cluster j, and $T_{j,\text{bad}}$ of them are *bad* for cluster j, where $T_{j,\text{good}} + T_{j,\text{bad}} = T$. We want to show that $T_{j,\text{good}} \geq \ell_j$.

CLAIM 34. *Each left-over i-clique has at least one vertex in the interior of R_j.*

PROOF. We assume, by way of contradiction, that some left-over i-clique K_i has each vertex u outside or on the boundary of R_j (centered at v_j with radius r). Since each left-over point p in cluster j is in the interior of C_j (centered at v_j with radius $r/2$), u is away from p by a distance of at least $r - r/2 = r/2$ and thus K_i can be matched with p in Step 1, contradicting the fact that K_i is a left-over i-clique. □

To prove the lemma, consider the number $|R_j|$ of points in the *interior* of R_j: these are ℓ_j left-over points in cluster j, at least one vertex from each of the $n/(m+1) - T$ left-over i-cliques (by Claim 34), and at least one vertex from each of the $T_{j,\text{bad}}$ bad $(i+1)$-cliques for cluster j. By the choice of r, we have $|R_j| \leq n/(m+1)$, and thus $n/(m+1) \geq |R_j| \geq \ell_j + (n/(m+1) - T) + T_{j,\text{bad}}$. Therefore, we have $T_{j,\text{good}} = T - T_{j,\text{bad}} \geq \ell_j$. □

Now we want to show the key properties that enable the constant-factor approximation. Suppose initially all cliques have shortest edge length at least δ. Recall from Lemma 31 that after swapping, the clique k' grown from the left-over clique k might reduce its shortest edge length to $\delta/2$. If later k' becomes a *good* clique for some cluster and is used for swapping, then the shortest clique-edge length, among all resulting cliques, might be halved again, and hence might be halved in each of the $\lceil m/2 \rceil$ phases in the worst case, resulting in an undesirable fraction of r whose denominator is *exponential* in m. However, this will never happen, as shown in the following key lemmas.

LEMMA 35. *When a clique becomes a left-over clique* (*i.e.*, *cannot be grown by matching in Step* 1) *in some phase i, it remains to be a left-over clique in all subsequent phases. Moreover, such a left-over i-clique in phase i, after growth into an $(i+1)$-clique, is always* bad *for* all *clusters, and remains* bad *for* all *clusters throughout all subsequent phases.*

PROOF. Let k be a left-over i-clique not matched in Step 1 of phase i, and let L be the set of left-over points not matched in Step 1 of phase i. Since k is not matched, each point $p \in L$ is "too close" (i.e., within a distance less than $r/2$) to at least one of the i vertices of k. Also, by Claim 34, k has at least one vertex in the interior of R_j; this applies to *each* cluster j. Now in Step 2 of phase i, k is grown into an $(i+1)$-clique k', which has exactly the same i vertices as k plus one additional vertex x, according to the swapping operation of Lemma 31. This means that the $(i+1)$-clique k' also has at least one vertex in the interior of R_j for each cluster j, and hence is *bad* for all clusters in phase i. Therefore, k' will not participate in any swappings and is left untouched in phase i. In the beginning of phase $i+1$, any left-over point q to be matched in Step 1 must come from L, and hence q is too close to at least one vertex of k' and cannot be matched with k', meaning that k' remains a left-over clique in phase $i+1$. Again, k'

is grown by including an additional vertex and thus remains bad for all clusters. In this way, the original "bad" vertices of k that are in the interior of R_j for all clusters j and that are too close to the left-over points in L to prevent a matching in Step 1 still stay in the clique without being swapped out from phase i to phase $i+1$, and inductively these vertices always stay in the clique in all subsequent phases. Therefore, the clique remains a left-over clique and is bad for all clusters throughout all subsequent phases. □

LEMMA 36. *For any phase i of Part 2, $i \geq h$, any $(i+1)$-clique that is good for some cluster must have edge lengths at least $\delta = r/2$.*

PROOF. Initially, when $i < h$, all $(i+1)$-cliques have edge lengths at least δ. Starting from phase h, we grow the cliques by matching and swapping operations. The matching operation only introduces clique edges of length at least δ, so we only need to consider the swapping operation. By the swapping operation of Lemma 31, the only situation to introduce a clique edge of length less than δ (called a "short edge," whose length is still at least $\delta/2$) is when we grow a left-over i-clique into an $(i+1)$-clique. In other words, any "short edge" is only introduced to a *left-over* clique. However, by Lemma 35, the resulting clique is bad for all clusters, and remains bad for all clusters in all subsequent phases. Therefore, in any time, any "short edge" only occurs in a clique that is bad for all clusters, and any clique that is good for some cluster has edge lengths at least $\delta = r/2$. □

By Lemma 36, any good clique used for swapping has edge lengths at least δ, and thus by Lemma 31 the clique-edge lengths resulting from swappings are at least $\delta/2 = r/4$. Therefore, the edge lengths of all cliques produced by Algorithm *Large-$(m+1)$-Cliques* are at least $r/4$. As for the running time, in Parts 1 and 2 we perform maximum cardinality matching and maximal matching on bipartite graphs (in $O(n^{2.5})$ time in [5] and [9]) for $O(m)$ times, using $O(mn^{2.5})$ time in total. In Part 2 we also perform $O(n/(m+1))$ swappings in each phase i, where each swapping takes $O(i^2) = O(m^2)$ time by Lemma 31, and hence in total the swappings take time $O((m^2 \cdot n/(m+1))m) = O(m^2 n) = O(mn^2)$ (recall that $m < n$). The overall time complexity is $O(mn^{2.5})$.

THEOREM 37. *Algorithm* Large-$(m+1)$-Cliques *produces $n/(m+1)$ vertex-disjoint $(m+1)$-cliques with edge lengths at least $r/4$, using $O(mn^{2.5})$ time.*

Our next task is to concatenate the $n/(m+1)$ vertex-disjoint $(m+1)$-cliques to create a desirable Hamiltonian path. We carry out the task by using the method of [2], which is analogous to Theorem 9 and creates a Hamiltonian path such that any two nodes that are m-neighbors in the path are at distance at least $\beta/2$ from each other, where β is the length of a shortest clique edge [2]. In the process, chaining each new clique takes $O(m^2)$ time [2], and the overall chaining process takes $O(mn)$ time. By Theorem 37, $\beta = r/4$, and recall that $2r$ is an upper bound on the optimal objective value. Therefore, we achieve an approximation factor of 16, with overall running time $O(mn^{2.5})$.

For the case of $n = (m+1)k + m$, we apply the simple technique of Section 2.3 that handles the case of $n = 3k+2$ for the path version of the max-min 2-neighbor TSP. We add a fake point Q far away from all n points of S so that Q does not affect the max-min

m-neighbor distance at all, and then apply Algorithm *Large-*$(m+1)$*-Cliques* on this new set of $n' = (m+1)k + (m+1)$ points. When chaining the resulting $(m+1)$-cliques, we start with the clique containing Q, traversing this clique by visiting Q first. Finally, we remove Q to obtain a Hamiltonian path on S with approximation factor 16.

THEOREM 38. *Given a complete graph on set S of n points satisfying the triangle inequality and an input parameter $m \in (2, n)$, where n is a multiple of $m+1$ or $n = (m+1)k + m$, there exists an algorithm that constructs in $O(mn^{2.5})$ time a Hamiltonian path on S whose objective value for the max-min m-neighbor problem is at least $\frac{1}{16}$ times the optimal.*

References

[1] N. Alon, R.A. Duke, H. Lefmann, V. Rödl, and R. Yuster. The algorithmic aspects of the regularity lemma. In *Proc. IEEE Symposium on the Foundations of Computer Science* (*FOCS* '92), pages 473–481, 1992.

[2] E.M. Arkin, Y.-J. Chiang, J.S.B. Mitchell, S.S. Skiena, and T.-C. Yang. On the maximum scatter traveling salesperson problem. *SIAM J. Comput.*, 29(2):515–544, 1999.

[3] A. Barvinok, S.P. Fekete, D.S. Johnson, A. Tamir, G.J. Woeginger, and R. Woodroofe. The geometric maximum traveling salesman problem. *J. ACM*, 50(5):641–664, 2003.

[4] A. Barvinok, E.Kh. Gimadi, and A.I. Serdyukov. The maximum TSP. In *The Traveling Salesman Problem and Its Variations*, G. Gutin and A. P. Punnen (eds.), Chapter 12, Kluwer Academic, Dordrecht, 2002.

[5] S. Even and R.E. Tarjan. Network flow and testing graph connectivity. *SIAM J. Comput.*, 4:507–518, 1975.

[6] S.P. Fekete. Simplicity and hardness of the maximum traveling salesman problem under geometric distances. *Proc. ACM–SIAM Symposium on Discrete Algorithms*, pages 337–345, 1999.

[7] G. Gutin and A.P. Punnen (eds.). *The Traveling Salesman Problem and Its Variations*, Kluwer Academic, Dordrecht, 2002.

[8] P. Hall. On representatives of subsets. *J. London Math. Soc.*, 10:26–30, 1935.

[9] J.E. Hopcroft and R.M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2:225–231, 1973.

[10] S.N. Kabadi and A.P. Punnen. The bottleneck TSP. In *The Traveling Salesman Problem and Its Variations*, G. Gutin and A.P. Punnen (eds.), Chapter 15, Kluwer Academic, Dordrecht, 2002.

[11] J. Komlós, G.N. Sárközy, and E. Szemerédi. On the square of a Hamiltonian cycle in dense graphs. *Random Structures Algorithms*, 9(1–2):193–211, 1996.

[12] E.L. Lawler, J.K. Lenstra, A.H.G. Rinooy Kan, and D.B. Shmoys (eds.). *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley, New York, 1985.

[13] R.G. Parker and R.L. Rardin. Guaranteed performance heuristics for the bottleneck traveling salesman problem. *Oper. Res. Lett.*, 2(6):269–272, 1984.

[14] K. Penavic. Optimal firing sequences for CAT scans. Manuscript, Department of Applied Mathematics, SUNY, Stony Brook, NY, 1994.

[15] F. Scholz. Coordination hole tolerance stacking. Technical Report BCSTECH-93-048, Boeing Computer Services, November, 1993.

[16] F. Scholz. Coordination hole tolerance stacking panel/panel/stringer join. Technical Report BCSTECH-94-017, Boeing Computer Services, May, 1994.

[17] F. Scholz. Cylinder section tolerance stacking issues. Technical Report BCSTECH-94-018, Boeing Computer Services, May, 1994.

[18] F. Scholz. Tolerance stack analysis methods a critical review. Technical Report BCSTECH-95-021, Boeing Computer Services, November, 1995.

[19] F. Scholz. Tolerance stack analysis methods. Technical Report BCSTECH-95-030, Boeing Computer Services, December, 1995.