MAP LABELING WITH CIRCLES

by

Minghui Jiang

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

Doctor of Philosophy

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

April 2005

APPROVAL

of a dissertation submitted by

Minghui Jiang

This dissertation has been read by each member of the dissertation committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the College of Graduate Studies.

Dr. Binhai Zhu

Approved for the Department of Computer Science

Dr. Michael Oudshoorn

Approved for the College of Graduate Studies

Dr. Bruce R. McLeod

STATEMENT OF PERMISSION TO USE

In presenting this dissertation in partial fulfillment of the requirements for a doctoral degree at Montana State University, I agree that the Library shall make it available to borrowers under rules of the Library. I further agree that copying of this dissertation is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U. S. Copyright Law. Requests for extensive copying or reproduction of this dissertation should be referred to Bell & Howell Information and Learning, 300 North Zeeb Road, Ann Arbor, Michigan 48106, to whom I have granted "the exclusive right to reproduce and distribute my dissertation in and from microform along with the non-exclusive right to reproduce and distribute my abstract in any format in whole or in part."

Minghui Jiang

To Whay

ACKNOWLEDGMENTS

A childhood by this fountain wondering
Would leave impress of circle-mysteries:
One would have faith that the unjustest thing
Had geometric grace past what one sees.

—Richard Wilbur, *Caserta Garden*

I thank my advisor, Dr. Binhai Zhu, for guiding me into this wonderful field of theoretical computer science, for opening my mind to the mysterious joy in creative research, and for supporting me through the past three years of my metamorphosis.

I thank Dr. Rocky Ross, Dr. Brendan Mumey, and Dr. Denbigh Starkey, for all the help, and the encouragement.

I thank my mom. I thank my wife.

TABLE OF CONTENTS

LIST OF FIGURES vii

1. INTRODUCTION 1

2. PRELIMINARIES 7

Maximal Feasible Region 7
Distance Graph 12
3-Diameter 14

3. APPROXIMATIONS FOR MLUC 17

$(3+\epsilon)$-Approximation for MLUC 17
$(2.98+\epsilon)$-Approximation for MLUC 26

4. APPROXIMATIONS FOR MLUCP 38

$(1.5+\epsilon)$-Approximation for MLUCP 38
$(1.491+\epsilon)$-Approximation for MLUCP 40

5. LOWER BOUNDS 46

NP-Hardness Results 46
Inapproximability Results 52

6. IMPLEMENTATION AND EXPERIMENTS 59

Preprocessing Phase 59
Searching Phase 61
Improving Phase 62
Graphical User Interface 64
Experimental Results 66

7. CONCLUSION 68

REFERENCES CITED 69

APPENDICES 74

APPENDIX A – SOURCE CODE FOR NUMERICAL CHECKING 75
APPENDIX B – SOURCE CODE FOR MLUC IMPLEMENTATION 79

LIST OF FIGURES

Figure Page

1. Maximal Feasible Regions 8
2. A Circle Packing Property 9
3. From 1 to $1 - \delta$ 13
4. From $1/\delta$ to $1/\delta^2$ 14
5. $R^* \leq (2 + \sqrt{3})D_3$ 15
6. $R^* \geq D_3/8$ 16
7. Shrink and Rotate (1) 19
8. From r to $r/3$ 22
9. An Example of Seven Circles 25
10. Shrink and Rotate (2) 29
11. Interferences Between Single-site Clusters 32
12. Interferences Involving Multi-site Clusters 34
13. At Least $2r$ Between MLUCP Sites 39
14. From r to $2r/3$ 39
15. Interferences Between MLUCP Sites 42
16. Variable Gadget 47
17. Clause Gadget 48
18. Variable Segment and Literal Leg 49

19. Planar Layout of Construction 50

20. Rotation in Variable Gadget 53

21. Rotation in Clause Gadget 55

22. Graphical User Interface of AMLUC 65

ABSTRACT

We study two geometric optimization problems motivated by cartographic applications: Map Labeling with Uniform Circles (MLUC) and Map Labeling with Uniform Circle Pairs (MLUCP). We show that the decision problems of both MLUC and MLUCP are NP-hard, and that the related optimization problems for maximizing the label sizes are NP-hard to approximate within factor 1.0349. We design approximation algorithms with constant performance guarantees for the two problems: for MLUC, we present a $(3+\epsilon)$-approximation and a $(2.98+\epsilon)$-approximation; for MLUCP, a $(1.5+\epsilon)$-approximation and a $(1.491+\epsilon)$-approximation. We also describe the implementation of AMLUC, a software system for automated map labeling with uniform circles. The system is based on our approximation algorithms for MLUC and uses an effective shake-and-grow heuristic to find near-optimal label placements.

CHAPTER 1

INTRODUCTION

Cartography is an important application domain of the design and analysis of computer algorithms. In maps and diagrams, feature objects such as points, lines, and polygons are often annotated by textual or graphical labels to convey information. Aesthetic standards and practical concerns demand that each label on the map associates clearly and unambiguously with its feature object, that no labels overlap or obscure the feature objects, and that all labels have large enough size to be legible [24, 31, 20]. Manually labeling a map to meet these restrictive and often conflicting requirements is a notoriously tedious task that takes about 50% of the total map production time [32]. Since the 1970s, cartographers and computer scientists have been interested in designing efficient computer algorithms to automate this task [46]. With the rapid growth of information to be visualized, the interest in map labeling automation algorithms has been steadily increasing. The ACM Computational Geometry Impact Task Force report [6] identifies automated label placement as an important research area in computational geometry [10]; a number of recent Ph.D. theses [2, 14, 33, 19, 25, 43, 37, 39] are devoted to this topic.

Early research efforts on automating label placement experimented with a large variety of heuristic methods such as force simulation [23], expert systems [3, 15], log-

ical programming [28], mathematical programming [48], simulated annealing [7, 16], and constraint satisfaction [42]. These heuristics may be very effective in practice, but they seldom offer any theoretical guarantees on the quality of their label placements.

Theoretical computer scientists formulate the task of automated label placement as a geometric optimization problem: Given a set of feature objects in the plane, and a label to be placed near each object, the goal of *Map Labeling* is to label the maximum number of objects with the maximum label size such that no labels overlap. The two quality measures, the number of labeled objects and the label size, lead to three different optimization goals:

Size: Maximize the label size such that all objects are labeled;

Number: Maximize the number of labeled objects with a given label size;

Bi-criteria: Maximize the number of labeled objects with a label size close to (not necessarily equal to) a given size.

As a geometric optimization problem, Map Labeling is closely related to famous NP-hard problems such as Maximum Independent Set [22] and Geometric Packing [21]. When the objects are point sites and the labels are uniform squares tangent to their respective sites, Map Labeling is NP-hard in general [20, 29] and admits approximation algorithms that maximize either the label size [34, 35] or the number of labeled points [1, 40, 41] or both [11, 47].

While textual labels can be conveniently modeled as axis-parallel rectangular

boxes framing the texts, labels conveying graphic information are more suitably modeled as circles. Furthermore, the circular shape, with its close relation to the Euclidean metric in planar geometry, is the most natural label shape for the map labeling model of sliding labels [40, 41], where each label is not restricted to a few pre-specified *discrete* label positions but can slide *continuously* around its site while keeping tangent to it. In this sliding model, a circular label always remains at the same distance to its site as it slides around the site—this uniformity of label positions makes map labeling with circular labels especially interesting for theoretical study. In this dissertation, we study two hard computational problems motivated by map labeling with circular labels, and focus on designing approximation algorithms and proving lower bounds for the optimization goal of maximizing the label size such that all objects are labeled.

We study two closely related geometric problems: *Map Labeling with Uniform Circles (MLUC)* and *Map Labeling with Uniform Circle Pairs (MLUCP)*. We first define the decision versions of the two problems:

Definition 1.1 (MLUC/MLUCP Decision Problems)

Instance: *Given n point sites $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ in the plane, and a label size $r > 0$.*

MLUC: *Is there a placement for n uniform open circles of radii r, one circle for each input site $P_i \in \mathcal{P}$, such that each site is on the boundary of its labeling circle and no circles intersect?*

MLUCP: *Is there a placement for* $2n$ *uniform open circles of radii* r*, two circles for each input site* $P_i \in \mathcal{P}$*, such that each site is on both boundaries of its two labeling circles and no circles intersect?*

Note that, by requiring each site to be on the boundary (boundaries) of its labeling circle (circles), we enforce a clear association between the label and the site. We next define the optimization versions of the two problems:

Definition 1.2 (MLUC/MLUCP Optimization Problems)

Instance: *Given* n *point sites* $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ *in the plane.*

MLUC: *Find a placement for* n *uniform open circles of radii* r*, one circle for each input site* $P_i \in \mathcal{P}$*, such that each site is on the boundary of its labeling circle, no circles intersect, and the label size* r *is maximized.*

MLUCP: *Find a placement for* $2n$ *uniform open circles of radii* r*, two circles for each input site* $P_i \in \mathcal{P}$*, such that each site is on both boundaries of its two labeling circles, no circles intersect, and the label size* r *is maximized.*

When the MLUC problem was first studied by Doddi *et al.* [11], they proposed a 29.86-approximation for maximizing the label size such that all sites are labeled, and a polynomial-time approximation scheme (PTAS) for the bi-criteria optimization problem. For maximizing the number of labeled sites with a given label size, Erlebach *et al.* [18] also presented a PTAS. Maximizing the label size such that all sites are labeled seems to be the most difficult optimization goal for approximation

algorithms, and it has attracted the most research effort. The initial approximation factor of 29.86 was improved to 19.35 by Strijk and Wolff [38], who also proved the NP-hardness of this problem. Later, Doddi *et al.* [12, 13] introduced the concept of *feasible region*, and improved the approximation factor to $3.6 + \epsilon$. Still later, Jiang *et al.* [27] presented a conceptually simple $(3 + \epsilon)$-approximation. This was achieved by proving a combinatorial lemma on labeling disjoint unit circles with points, and by generalizing the concept of feasible region further to *maximal feasible region*. Recently, Jiang *et al.* [26] presented a $(2.98 + \epsilon)$-approximation for this problem; this algorithm built on the important concept of maximal feasible region, and handled the previous difficult cases with new algorithmic techniques.

The MLUCP problem was first studied by Zhu and Poon [47] as part of the research effort in multi-label map labeling. They proposed a 2-approximation [47] for maximizing the label size. This bound was subsequently improved to 1.96 [34], 1.686 [36], and 1.5 [44, 45]. Recently, Jiang *et al.* [26] designed a remarkably simple $(1.5+\epsilon)$-approximation for MLUCP using the concept of maximal feasible region, then improved the approximation factor to $1.491 + \epsilon$ with new algorithmic techniques.

For the hardness of approximation, Strijk and Wolff [38] proved that the MLUC decision problem is NP-hard; their proof also implied that it is NP-hard even to approximate the MLUC optimization problem (maximizing the label size) within factor $1 + O(1/n)$. For the MLUCP optimization problem, a 1.37 inapproximability result [34] was claimed (no details were given); it was later found to be actually

$1 + O(1/n)$. In the recent work of Jiang *et al.* [26], a completely new reduction was designed to obtain the first nontrivial lower bound of 1.0349 for both MLUC and MLUCP.

This dissertation focuses on our recent theoretical work on MLUC and MLUCP. We organize the rest of this dissertation as follows: In Chapter 2, we introduce some preliminary concepts. In Chapter 3, we present our approximation algorithms for MLUC. In Chapter 4, we present our approximation algorithms for MLUCP. In Chapter 5, we present our NP-hardness and inapproximability results for MLUC and MLUCP. In Chapter 6, we present an efficient MLUC implementation based on our approximation algorithms with additional heuristics. In Chapter 7, we conclude the dissertation.

CHAPTER 2

PRELIMINARIES

In this chapter, we introduce some preliminary concepts used by our approximation algorithms: maximal feasible region, distance graph, and 3-diameter.

We first define a few common terms: A *unit circle* is a circle of unit radius. The *distance between a circle and a point* is the distance between the circle center and the point; the *distance between two circles* is the distance between the two circle centers. Two sites *interfere* each other if their labels intersect. An approximation algorithm A for a maximization problem Π has a *performance guarantee* $\rho > 1$ if, for every instance I of Π, the solution of A has a value that is at least $1/\rho$ of the optimal value for I; the factor ρ is also called the *approximation factor* of A.

Maximal Feasible Region

In the MLUC problem, a site can be anywhere on the boundary of its labeling circle; from a different perspective, the circle *slides* around the site while keeping tangent to it; or, the site is a fixed pivot on the boundary of the circle, and the circle *rotates* around the site. The position of a circle, given its radius, is determined by its *direction*: the vector from the site to the circle center. If a circle contains a site in its interior, then it always intersects the circle of the contained site; therefore, a circle

at a *feasible position* contains no sites in its interior. As a circle rotates continuously from one feasible position to another, the corresponding direction vector sweeps a cone, or, a *feasible region*. We consider only *maximal feasible regions*, in the sense that no maximal feasible region is a proper subset of another feasible region; this concept was introduced by Jiang *et al.* [27].

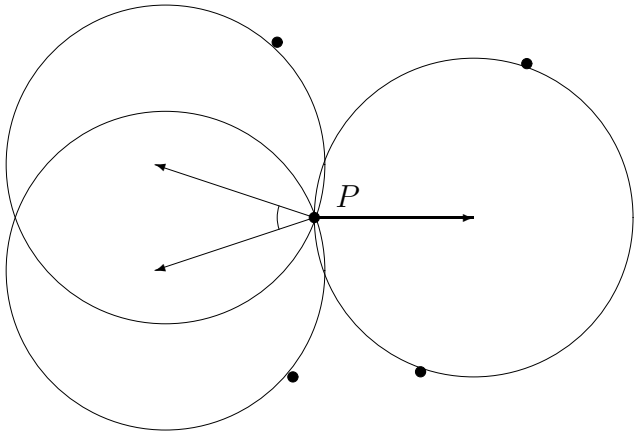


Figure 1. Maximal Feasible Regions.

We refer to Figure 1 for an example. With the label size as shown in the figure, the maximal feasible regions for the site P include a cone and a single vector (a degenerated cone).

Because a site is always on the boundary of its circle, which, at a feasible position, contains no sites in its interior, each site is a closest site to its circle in a label placement. Maximal feasible regions are naturally related to the Voronoi diagram: the center of a labeling circle is within the Voronoi cell of its site if and only if the circle is at a feasible position.

The MLUCP problem can be considered as a special case of the MLUC problem where each MLUCP site becomes two coinciding MLUC sites. To avoid overlapping, the two circles of each site must always be labeled in opposite directions, and the site is at the midpoint of its two circle centers. The concept of maximal feasible region can be defined also for MLUCP, but we first need to prove a circle packing property stated in the following lemma.

Lemma 2.1 *Given three disjoint unit circles and one point on the boundary of each circle, if two points have distance* $d \leq \sqrt{3} - 1$*, then the distance from the third point to the two points is at least* $D(d) = \sqrt{5 - 4\cos(\frac{\pi}{3} - \phi)} - 1$*, where* $\phi = \cos^{-1} \frac{5-(1+d)^2}{4}$.

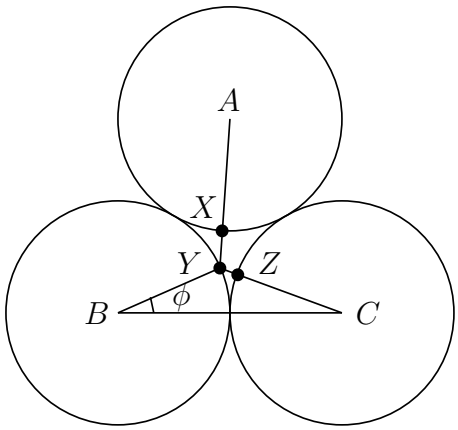


Figure 2. A Circle Packing Property.

Proof. We refer to Figure 2. Points X, Y, and Z are on the boundaries of three circles centered at A, B, and C, respectively. Y and Z have distance $d \leq \sqrt{3} - 1$. Without loss of generality, we assume that $XY < XZ$, that is, the distance between points X and Y is smaller than the distance between points X and Z. We show that

XY has the lower bound $D(d)$.

Imagine that a spring connects X and Y, and that a rigid rod of length d connects Y and Z. The three circles may not intersect but can move freely otherwise. When the spring shrinks till XY reaches the minimum, the three points A, X, and Y become collinear; the three points C, Z, and Y become collinear; and the three circles are tightly packed in an equilateral triangular formation.

In $\triangle YBC$, $BY = 1$, $BC = 2$, $YC = 1 + d$, and $\angle YBC = \phi$. Therefore, we have

$$\phi = \cos^{-1} \frac{BY^2 + BC^2 - YC^2}{2 \cdot BY \cdot BC} = \cos^{-1} \frac{5 - (1+d)^2}{4}.$$

For $d \leq \sqrt{3} - 1$, we have $\phi \leq \frac{\pi}{3}$.

In $\triangle YBA$, $BY = 1$, $BA = 2$, $AY = 1 + D(d)$, $\angle ABY = \frac{\pi}{3} - \phi$. A similar calculation shows that

$$D(d) = \sqrt{5 - 4\cos\left(\frac{\pi}{3} - \phi\right)} - 1.$$

□

The function $D(d)$ is monotonically decreasing in $[0, \sqrt{3} - 1]$; it achieves the maximum $\sqrt{3} - 1$ at $d = 0$, and the minimum 0 at $d = \sqrt{3} - 1$. For completeness, we define $D(d) = 0$ for $d > \sqrt{3} - 1$. A calculation shows that the equation

$$D(d) = d \tag{2.1}$$

has solution $d_0 \simeq 0.24 < \sqrt{3} - 1$. For any distance d such that $0 \leq d < d_0$, the decreasing property of function $D(d)$ guarantees that $D(d) > d$. This implies the following corollary.

Corollary 2.1 *If an MLUC instance with unit label size has a label placement, then each site in the instance has at most one neighbor within distance* d_0.

We are now ready to define maximal feasible region for MLUCP, which is implied by the following corollary.

Corollary 2.2 *In an MLUCP instance with unit label size, the two circles of a site labeled at a* feasible position *have distances at least* $\sqrt{3}$ *to the other sites.*

In the MLUC problem, maximal feasible regions based on the classic definition [27] exclude obviously infeasible label positions where the circles contain other sites in their interiors. The following corollary, also immediate from Lemma 2.1, defines more restricted maximal feasible regions that exclude a broader range of infeasible positions.

Corollary 2.3 *In an MLUC instance with unit label size, the circle of a site labeled at a* feasible position *has distance at least* $D(PQ) + 1$ *to any two other sites* P *and* Q *in the instance.*

We now revise our definitions of feasible position and maximal feasible region for MLUC based on Corollary 2.3. We use the term *revised maximal feasible region* to distinguish this new concept with the *classic* maximal feasible region. As we will see later, this concept of revised maximal feasible region is crucial for our improvement of the approximation factor for MLUC from $3 + \epsilon$ to $2.98 + \epsilon$.

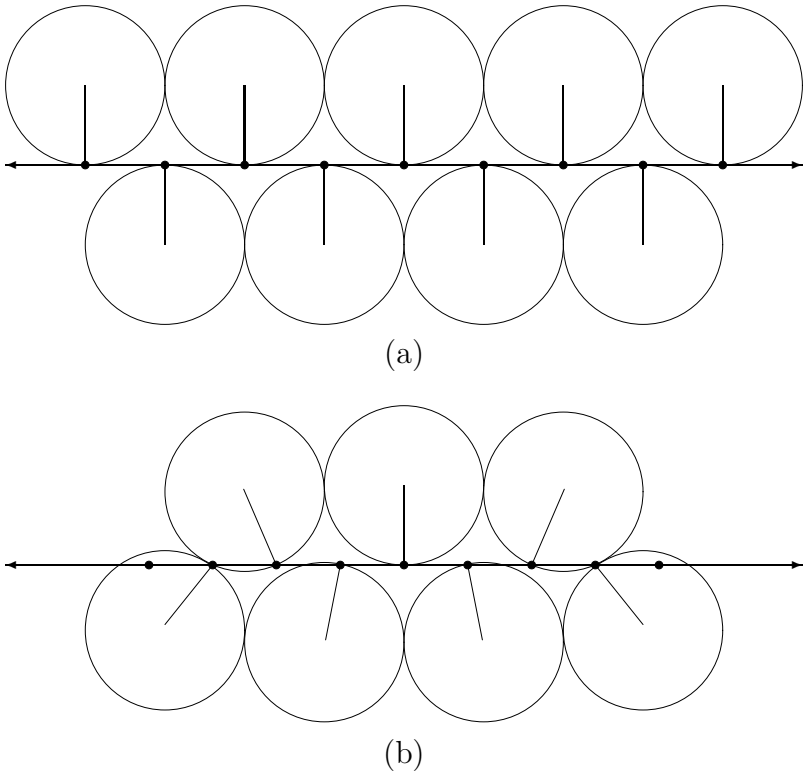
Distance Graph

Given a set $\mathcal{P}$ of point sites and a threshold distance d, we define the *distance graph* $G_{\mathcal{P}}(d)$ to be the graph where each site is represented by a vertex, and where an edge connects two vertices if and only if their corresponding sites have distance less than d. We use the abbreviated notation $G(d)$ when the set of sites is clear from context.

When the set $\mathcal{P}$ of sites has a label placement with a label size larger than the threshold distance d, the distance graph $G_{\mathcal{P}}(d)$ has an interesting combinatorial property. This important property was discovered by Jiang *et al.* [27] (Lemma 1); we rephrase it here using the concept of distance graph.

Lemma 2.2 *Given $\mathcal{P}$, a set of sites labeled with disjoint unit circles, and a threshold distance $1-\delta$, where $0 < \delta < 1$, the maximum size of a connected component in the distance graph $G_{\mathcal{P}}(1-\delta)$ is bounded by a constant $O(1/\delta^2)$.*

For the proof of this property, we first observe that every simple path in the distance graph $G(1-\delta)$ has length bounded by $O(1/\delta)$. An extreme case happens when the path has unit edge lengths, and each site is labeled by a unit circle directed to alternating side along the path; as shown in Figure 3(a), this path can be infinitely long. If we shrink the edge lengths from 1 to $1-\delta$, as shown in Figure 3(b), while keeping the circles at the same size, every circle has to rotate a little to compensate for the lost space of δ between consecutive sites. Since the rotation adjustment is



(a)

(b)

Figure 3. From 1 to $1 - \delta$.

accumulative, a simple calculation shows that the path can only contain $O(1/\delta)$ sites.

The $O(1/\delta^2)$ bound follows immediately by an area argument. We refer to Figure 4. Take an arbitrary site from a connected component; every other site in the component is within distance $O(1/\delta)$ from this site, since every other site is connected to this site by a simple path of length $O(1/\delta)$. All the unit circles labeling the sites of the component are enclosed in a large circle of radius $O(1/\delta)$. Since each disjoint unit circle occupies at least a constant area of the large circle, there are at most $O(1/\delta^2)$ unit circles labeling $O(1/\delta^2)$ sites in the connected component. (A tighter bound of $O(1/\delta)$ for the connected component size was recently proved by Bereg [4].)

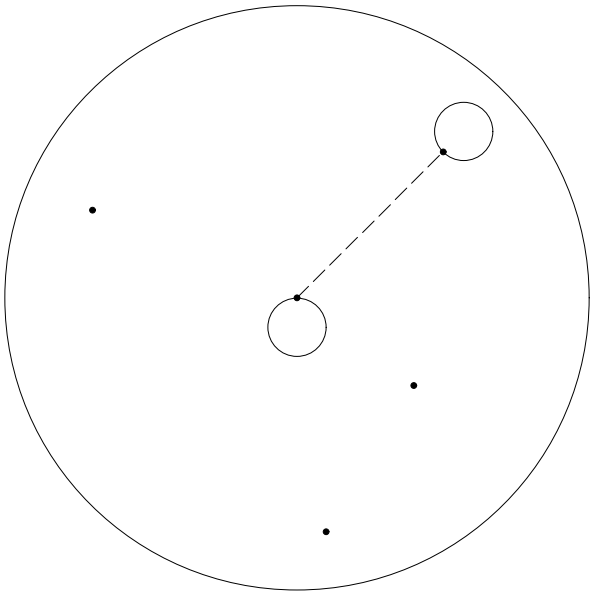


Figure 4. From $1/\delta$ to $1/\delta^2$.

As we will see later, Lemma 2.2 provides a natural way for us to apply the divide-and-conquer strategy in our approximation algorithms: first divide the sites into components of manageable size, next obtain a label placement independently within each component, then *combine* the partial results for all components to obtain a complete label placement.

3-Diameter

Given three sites X, Y, and Z, we define their 3-diameter $D_3(X, Y, Z)$ as the maximum value among the three distances XY, YZ, and ZX. For a set $\mathcal{P}$ of at least three sites, its 3-diameter $D_3(\mathcal{P})$ is the smallest value of the 3-diameter among all size-3 subsets of $\mathcal{P}$. We use the simpler notation D_3 instead of $D_3(\mathcal{P})$ when there is

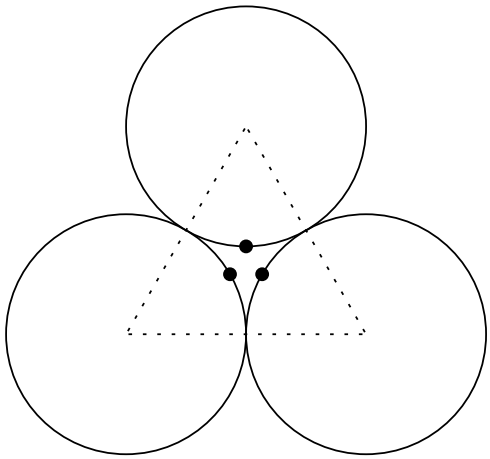


Figure 5. $R^* \leq (2+\sqrt{3})D_3$.

no ambiguity. We have the following lemma about the 3-diameter.

Lemma 2.3 *Given an MLUC instance of at least three sites, the optimal label size* R^* *satisfies* $D_3/8 \leq R^* \leq (2+\sqrt{3})D_3$.

Proof. Given three sites with a fixed 3-diameter, their uniform label size reaches the maximum in the label placement where both the three sites and their three circles are arranged in equilateral triangular formations as shown in Figure 5. From this extreme case, a calculation shows that $R^* \leq (2+\sqrt{3})D_3$.

To prove that $R^* \geq D_3/8$, we construct a label placement by labeling each site with label size $D_3/8$ in the opposite direction to its nearest neighbor (if it has two or more nearest neighbors, choose one arbitrarily). We refer to Figure 6. Take an arbitrary site P from $\mathcal{P}$, and draw a dashed circle of radius $D_3/2$ and a dotted circle of radius $D_3/4$ centered at P.

For those sites outside the dashed circle, their labeling circles can never intersect

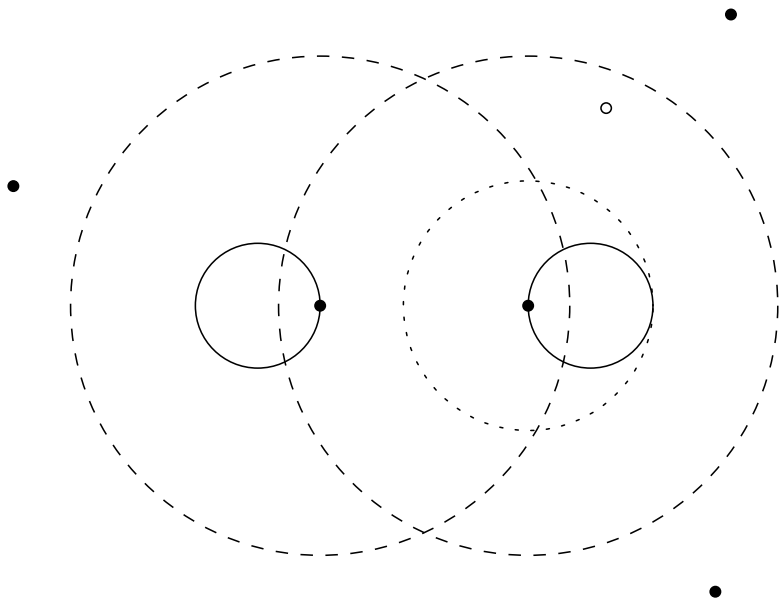


Figure 6. $R^* \geq D_3/8$.

the labeling circle of P, since their labeling circles, of radii $D_3/8$, are always outside the dotted circle, while the labeling circle of P stays inside regardless of its direction. Furthermore, we observe that P has at most one neighbor Q within distance $D_3/2$ (in the dashed circle); otherwise there will be three sites in the dashed circle of diameter D_3, contradicting our definition of the 3-diameter. By labeling P in the opposite direction to Q and, symmetrically, labeling Q in the opposite direction to P, we can ensure that their two labeling circles do not intersect. □

CHAPTER 3

APPROXIMATIONS FOR MLUC

In this chapter, we present two approximation algorithms for MLUC. We first present a simple $(3+\epsilon)$-approximation algorithm, then present a more sophisticated $(2.98+\epsilon)$-approximation algorithm.

$(3+\epsilon)$-Approximation for MLUC

Our first approximation algorithm for MLUC achieves approximation factor $3+\epsilon$, where the tunable parameter ϵ is a positive constant that can be arbitrarily close to zero.

Let R^* be the optimal label size. Lemma 2.3 suggests that R^* is in the range $[R^-, R^+]$, where $R^- = D_3/8$ and $R^+ = (2+\sqrt{3})D_3$. Given the parameter ϵ, our algorithm finds a label placement with label size at least $\frac{R^*}{3+\epsilon}$ by a binary search. The binary search has range $[R^-, R^+]$ and minimum interval δR^-, where $\delta = \frac{\epsilon}{2(3+\epsilon)}$, and employs a decision procedure that, given a tentative label size r, either decides that r exceeds $R = (1-\delta)R^*$ and aborts, or finds a label placement with label size $\frac{r}{3}$. When the binary search converges, we have a label placement with label size

$$\frac{r}{3} \geq \frac{(1-\delta)R^* - \delta R^-}{3} \geq \frac{(1-2\delta)R^*}{3} = \frac{R^*}{3+\epsilon}.$$

We now describe the decision procedure.

1. Compute the distance graph $G(r)$ and group the sites into *components* corresponding to the connected components in $G(r)$. If the size of the largest component exceeds the bounding constant $O(1/\delta^2)$, abort.

2. With label size r, first compute the maximal feasible regions for all sites, then label each component independently (ignoring possible interferences from other components):

   (a) Partition the maximal feasible regions of each site into cones of maximum angle θ_δ, where $\theta_\delta = \frac{d_0^2}{16}\delta$.

   (b) If any site has more than one neighbor within distance $d_0 r$, abort. If two sites P and Q have distance less than $d_0 r$, and if a cone C of P contains an internal (non-boundary) vector $\vec{v}$ in the opposite direction to a boundary vector of a cone of Q, divide C into two smaller cones along $\vec{v}$.

   (c) Limit the label direction of each site to the boundary vectors of its cones. Enumerate all possible combinations to find a label placement for this component. If no label placement can be found, abort.

3. Shrink each circle to radius $\frac{r}{3}$.

The correctness of step 1 of the decision procedure is immediate from Corollary 2.2. It remains to prove that, if $r \leq R$, then step 2 always finds a partial label placement within each component, and step 3 always transforms the output of step 2

into a complete label placement for all sites.

We first prove the following lemma.

Lemma 3.1 *Given two sites of distance d and labeled by two disjoint unit circles, the two circles can shrink to radii $1-\delta$, where $0<\delta<1$, then rotate independently for an angle up to $\theta=\frac{d^2}{16}\delta$, either clockwise or counterclockwise, without intersecting each other.*

Proof. Let P and Q be the two sites. The two circles labeling P and Q are centered at A and B before shrinking, and at C and D after shrinking. Without loss of generality, we assume that the two circles are tangent before shrinking.

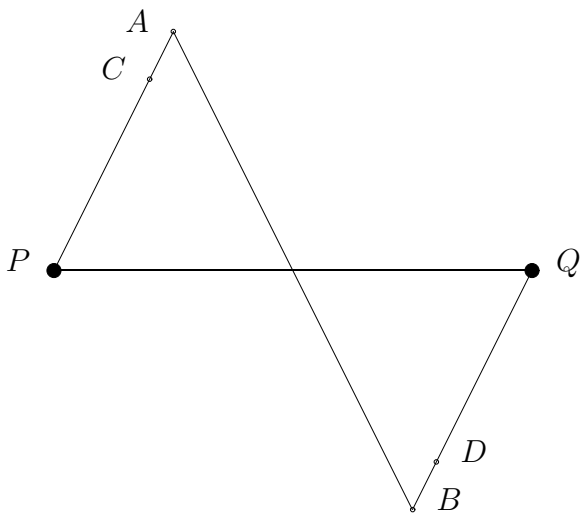


Figure 7. Shrink and Rotate (1).

We refer to Figure 7. We have $PQ=d$, $PA=QB=1$, $AC=BD=\delta$, and $AB=2$. Let $\gamma=\max\{\angle PAB,\angle QBA\}$. It is easy to see that γ reaches the minimum $2\sin^{-1}\frac{d}{4}$ when the midpoint of $\overline{AB}$ coincides with the midpoint of $\overline{PQ}$.

Assume that $\gamma = \angle PAB$. The distance between C and B along $\overline{AB}$ is at least $AB - AC\cos\gamma = 2 - \delta\cos\gamma$, so we have

$$CD \geq CB - BD \geq (2 - \delta\cos\gamma) - \delta.$$

The two small circles of radii $1 - \delta$ at C and D are disjoint and separated by a gap of at least

$$g = CD - 2(1-\delta) \geq (2 - \delta\cos\gamma - \delta) - 2(1-\delta) = \delta(1 - \cos\gamma).$$

Each small circle can rotate for an angle $g/2$ without closing the gap, so we have

$$\theta = \frac{\delta}{2}\left(1 - \cos\left(2\sin^{-1}\frac{d}{4}\right)\right) = \frac{d^2}{16}\delta.$$

□

We now prove the correctness of step 2.

Lemma 3.2 *If $r \leq R$, then the brute-force search in step 2 always finds a label placement for each component.*

Proof. We prove the lemma by construction. Given any *continuous* label placement $\mathcal{L}$ with label size R^*, we construct a *discrete* label placement $\mathcal{L}'$ with label size R that is included in the search space of step 2.

Since $R = (1-\delta)R^* < R^*$, each radius-R^* circle in $\mathcal{L}$ is directed in one of the cones that comprise the maximal feasible regions of its site at label size R. To obtain $\mathcal{L}'$, we simply shrink each circle in $\mathcal{L}$ to radius R, then relabel it along one of the two boundary vectors of its cone.

If a site has distance at least $d_0 r$ to its nearest neighbor, we choose either boundary vector arbitrarily. The angle between the circle's original continuous direction in $\mathcal{L}$ and its discrete direction in $\mathcal{L}'$ is bounded by the maximum cone angle. Lemma 3.1 implies that, with a rotation angle at most $\theta_\delta = \frac{d_0^2}{16}\delta$, there is no intersection.

If a site has distance less than $d_0 r$ to its nearest neighbor, then according to Corollary 2.1 all the other sites are at least distance $d_0 r$ away from the two sites and do not interfere them. To label these two sites, we examine the four candidate pairs of boundary vectors of their two cones. If the two cones have a pair of opposing boundary vectors, we label the two sites along the opposing vectors. Otherwise, the two cones must have no opposing vectors at all (not even opposing internal, or, non-boundary, vectors) because of the dividing vectors in step 2b. In this case we choose the pair of boundary vectors with the largest spreading angle. □

We next prove the correctness of step 3, that is, if $r \leq R$, then step 3 always transforms the output of step 2 into a complete label placement for all sites. In particular, we need to show that the non-intersection within each component is maintained, and that the possible interferences between different components are eliminated.

Lemma 3.3 *If $r \leq R$, then step 3 obtains a label placement for all sites.*

Proof. Since the circles shrink without changing label directions, the non-intersection within each component is clearly maintained. We consider two sites P and Q from different components. The fact that P and Q are not in the same component implies

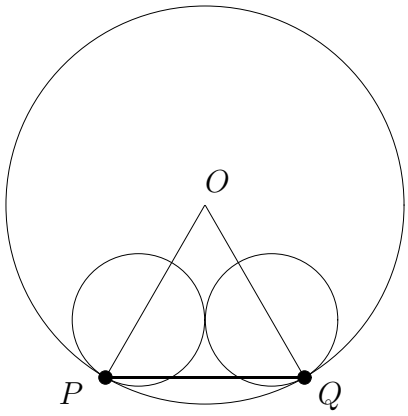


Figure 8. From r to $r/3$.

that the distance PQ is at least the threshold distance r. Before step 3, both P and Q are labeled in their respective maximal feasible regions with label size r. In the worst case, the two circles of P and Q coincide and are tangent to both P and Q; as shown in Figure 8, the three points P, Q, and the circle center O are in an equilateral triangular formation. A calculation shows that the two circles become tangent after shrinking to radii $\frac{r}{3}$. □

We have proved the correctness of the decision procedure. Together with our earlier analysis of the binary search, this result immediately implies that our algorithm does achieve an approximation factor of $3 + \epsilon$. We next analyze the running time of our algorithm.

To speed up the computation, we use a preprocessing step before the binary search to identify the 15 nearest neighbors of each site for all sites in $O(n \log n)$ time [9, 17]. With the nearest neighbors precomputed, we can compute D_3 in linear time,

as implied by the following lemma, and determine the range and interval of the binary search.

Lemma 3.4 *Given three sites O, P, and Q in $\mathcal{P}$, if $D_3(O, P, Q) = D_3(\mathcal{P})$, then O must have two sites X and Y among its six nearest neighbors such that $D_3(O, X, Y) = D_3(\mathcal{P})$.*

Proof. Denote $D_3 = D_3(\mathcal{P})$. We clearly have $OP \leq D_3$ and $OQ \leq D_3$, that is, both P and Q are within distance D_3 to O. If O has at most six neighbors within distance D_3, then both P and Q are among the six nearest neighbors of O.

Suppose that O has at least six neighbors within distance D_3. We can always find two sites X and Y from its six nearest neighbors such that $\angle XOY \leq \frac{360^\circ}{6} = 60^\circ$. Since we have $OX \leq D_3$ and $OY \leq D_3$, this implies that $XY \leq D_3$. Therefore we must have $D_3(O, X, Y) = D_3$. □

With the nearest neighbors precomputed, we can also compute the distance graph, the connected components, and the maximal feasible regions for all sites in linear time, as implied by the following lemma.

Lemma 3.5 *If an MLUC instance with label size r has a label placement, then each site in the instance has at most 15 neighbors within distance $2r$.*

Proof. Both a site and its neighbors within distance $2r$ have their circles of radii r completely enclosed in a large circle of radius $4r$ centered at the site. A simple packing argument completes the proof. □

Given an MLUC instance with label size r, the maximal feasible regions of each site are restricted only by its neighbors within distance $2r$. Lemma 3.5 implies that we can compute the maximal feasible regions of each site considering only its 15 nearest neighbors. To compute the connected components in the distance graph, it is also sufficient to consider only the edges between each site and its 15 nearest neighbors, because we have the threshold distance $r < 2r$. Let $d_{15}(P)$ be the distance from a site P to its 15th nearest neighbor in $\mathcal{P}$, and denote $d^*_{15} = \min\{d_{15}(P) \mid P \in \mathcal{P}\}$. Lemma 3.5 implies that $R^* \leq d^*_{15}/2$. With the nearest neighbors precomputed, we can compute d^*_{15} in linear time and ensure that the upper bound of the binary search does not exceed $d^*_{15}/2$.

In each run of the decision procedure, all computation except the brute-force search in step 2 takes only linear time with the nearest neighbors precomputed. The brute-force search takes $O(1/\epsilon)^{O(1/\epsilon^2)}$ time for each component because each component contains at most $O(\frac{1}{\delta^2}) = O(\frac{1}{\epsilon^2})$ sites and each site has $O(\frac{1}{\theta_\delta}) = O(\frac{1}{\delta}) = O(\frac{1}{\epsilon})$ discrete label positions. There are at most n components; the binary search calls the decision procedure at most

$$\log \frac{R^+ - R^-}{\delta R^-} = O\left(\log \frac{1}{\delta}\right) = O\left(\log \frac{1}{\epsilon}\right)$$

times. Therefore the overall complexity of the binary search is

$$O\left(n\left(\frac{1}{\epsilon}\right)^{O(1/\epsilon^2)} \log \frac{1}{\epsilon}\right) = O\left(n\left(\frac{1}{\epsilon}\right)^{O(1/\epsilon^2)+1}\right) = O\left(n\left(\frac{1}{\epsilon}\right)^{O(1/\epsilon^2)}\right).$$

In summary, we have the following theorem.

Theorem 3.1 *Our algorithm approximates MLUC with factor* $3+\epsilon$ *and runs in* $O(n \log n + n(1/\epsilon)^{O(1/\epsilon^2)})$ *time.*

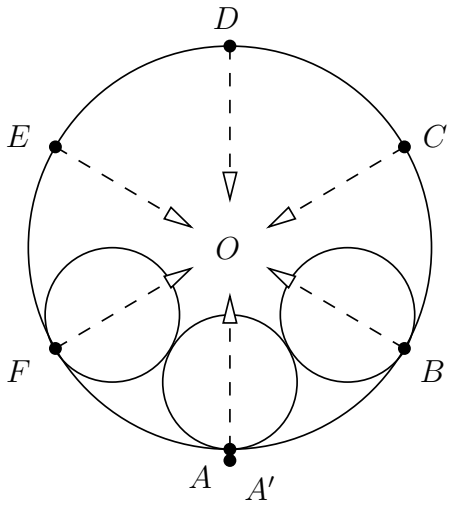


Figure 9. An Example of Seven Circles.

With the classic concept of maximal feasible region, it is difficult to improve the approximation factor $3+\epsilon$ further. We refer to Figure 9. The unit circle centered at O has six sites A, B, C, D, E, and F on its boundary in a regular hexagonal formation. A' is another site outside the circle and is very close to A. In our $(3+\epsilon)$-approximation algorithm, these seven sites are grouped into six different components at unit threshold distance, and each component is labeled independently. A and A' are in the same component and have to be labeled along vectors $\overrightarrow{AO}$ and $\overrightarrow{OA'}$ since this is the only choice in their maximal feasible regions. If B, C, D, E, and F are labeled along $\overrightarrow{OB}$, $\overrightarrow{OC}$, $\overrightarrow{OD}$, $\overrightarrow{OE}$, and $\overrightarrow{OF}$, respectively, we can obtain a label placement with unit label size. However, the vectors $\overrightarrow{BO}$, $\overrightarrow{CO}$, $\overrightarrow{DO}$, $\overrightarrow{EO}$, and $\overrightarrow{FO}$,

respectively, are feasible positions of B, C, D, E, and F; in the worst case these sites may be labeled along these vectors. When the label size shrinks to $\frac{1}{3}$, we still obtain a label placement, but there is no room for the circles to grow larger. If we can label some sites, say, B and F, along $\overrightarrow{OB}$ and $\overrightarrow{OF}$ instead, then we will have more space for the other sites to maneuver. This is exactly the idea behind our new concept of revised maximal feasible region: B and F are near some closely-clustered sites (A and A'), so their feasible regions should be more restricted.

$(2.98+\epsilon)$-Approximation for MLUC

Our improved algorithm achieves approximation factor $c+\epsilon$, where $c = 2.98$. Similar to the $(3+\epsilon)$-approximation algorithm, our algorithm finds a label placement with label size at least $\frac{R^*}{c+\epsilon}$ by a binary search with a decision procedure. The binary search has range $[R^-, R^+]$ and minimum interval δR^-, where $\delta = \frac{\epsilon}{2(c+\epsilon)}$. The decision procedure, given a tentative label size r, either decides that r exceeds $R = (1-\delta)R^*$ and aborts, or finds a label placement with label size $\frac{r}{c}$. When the binary search converges, we have a label placement with label size

$$\frac{r}{c} \geq \frac{(1-\delta)R^* - \delta R^-}{c} \geq \frac{(1-2\delta)R^*}{c} = \frac{R^*}{c+\epsilon}.$$

The decision procedure of our $(2.98+\epsilon)$-approximation is almost identical to that of the $(3+\epsilon)$-approximation except that revised maximal feasible regions are used instead of classic maximal feasible regions, and that an advanced rotation technique is

used in the third step. We now describe the third step of the new decision procedure:

3. (a) Compute the distance graph $G(d_c r)$, where $d_c = \sqrt{4 - (\frac{c^2-3}{c})^2} \simeq 0.326$, and group the sites into *clusters* corresponding to the connected components in $G(d_c r)$. (We call them *clusters* to avoid confusion with the *components* defined in step 1.) If a cluster contains only one site, we call it a *single-site cluster*; otherwise, a *multi-site cluster*.

   (b) Shrink each circle to radius $\frac{r}{c}$. For each single-site cluster, rotate its circle clockwise for an angle $\alpha = \cos^{-1} \frac{c^2-3}{2c} \simeq 9.37°$. If the resulting label placement has intersection, abort.

The correctness of the first two steps of the decision procedure is already proved in the previous section. We still need to prove the correctness of step 3, that is, if $r \leq R$, then step 3 always transforms the output of step 2 into a complete label placement for all sites. In particular, we need to show that the non-intersection within each component is maintained, and that the possible interferences between different components are eliminated.

We need to prove the following lemma first.

Lemma 3.6 *Given two sites of distance $d > d_c$ and labeled with two disjoint unit circles, the two circles can shrink to radii $\frac{1}{c}$, then rotate independently for an angle up to $\theta = \cos^{-1} \frac{d_c}{4} - \cos^{-1} \frac{cd_c}{4}$, without intersecting each other.*

Proof. This is a stronger form of Lemma 3.1. Let P and Q be the two sites. The

two circles labeling P and Q are centered at A and B before shrinking, and at C and D after shrinking and rotation. Without loss of generality, we assume that the two circles are tangent both before and after the shrinking and rotation. We have $PQ = d$, $PA = QB = 1$, $AB = 2$, $PC = QD = \frac{1}{c}$, and $CD = \frac{2}{c}$.

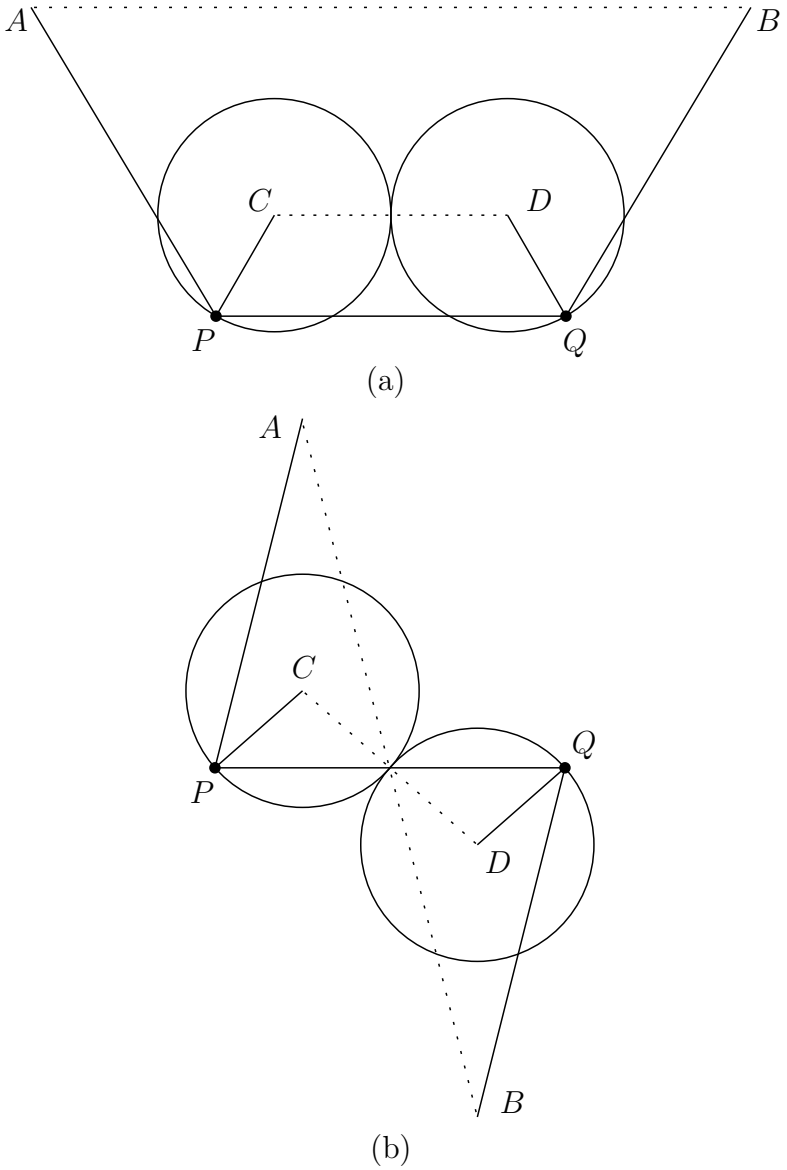


Figure 10. Shrink and Rotate (2).

We refer to Figure 10 for the two extreme cases. If the two circles rotate in opposite directions, then the configuration in Figure 10(a) is the worst case that gives the minimum rotation angle

$$\theta_1 = \sin^{-1}\left(1 - \frac{d}{2}\right) - \sin^{-1}\left(1 - \frac{cd}{2}\right).$$

If the two circles rotate in the same direction, then the configuration in Figure 10(b) is the worst case that gives the minimum rotation angle

$$\theta_2 = \cos^{-1}\frac{d}{4} - \cos^{-1}\frac{cd}{4}.$$

We prove that $\theta_2 \le \theta_1$, which is equivalent to

$$\cos^{-1}\frac{d}{4} - \sin^{-1}\left(1 - \frac{d}{2}\right) \le \cos^{-1}\frac{cd}{4} - \sin^{-1}\left(1 - \frac{cd}{2}\right).$$

This reduces to proving that

$$\frac{\mathrm{d}}{\mathrm{d}x} f(x) \ge 0 \quad \text{for} \quad \frac{d}{4} \le x \le \frac{cd}{4},$$

where

$$f(x) = \cos^{-1} x - \sin^{-1}(1 - 2x).$$

Here is the proof:

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}x}(\cos^{-1} x - \sin^{-1}(1-2x)) &\ge 0 \\
\frac{\mathrm{d}}{\mathrm{d}x}\cos^{-1} x &\ge \frac{\mathrm{d}}{\mathrm{d}x}\sin^{-1}(1-2x) \\
\frac{-1}{\sqrt{1-x^2}} &\ge \frac{-2}{\sqrt{1-(1-2x)^2}} \\
4(1-x^2) &\ge 1-(1-2x)^2 \\
x &\le 1
\end{aligned}$$

To see why the last inequality is satisfied, we note that, to impose any restriction on the rotation angles after the shrinking, we must have $d \leq \frac{4}{c}$, that is, the distance between the two sites must be at most four times the circle radius; on the other hand, we only need to prove the inequality for $\frac{d}{4} \leq x \leq \frac{cd}{4}$. □

We now show that the non-intersection within each component is maintained.

Lemma 3.7 *If $r \leq R$, then step 3 maintains the non-intersection within each component.*

Proof. The circles labeling multi-site clusters do not rotate after shrinking, so there is no interference between multi-site clusters in the same component. A site from a single-site cluster is at least distance d_c away from its nearest neighbor in the same component; its circle shrinks to radius $\frac{r}{c}$ and rotates for an angle $\alpha = \cos^{-1}\frac{c^2-3}{2c}$. Lemma 3.6 guarantees that this rotation causes no intersection because d_c is the solution to the following equation:

$$\cos^{-1}\frac{c^2-3}{2c} = \cos^{-1}\frac{d_c}{4} - \cos^{-1}\frac{cd_c}{4}. \tag{3.1}$$

We show that d_c is indeed the solution:

$$\begin{aligned}
\cos^{-1}\frac{c^2-3}{2c} &= \cos^{-1}\frac{d}{4} - \cos^{-1}\frac{cd}{4}. \\
\frac{c^2-3}{2c} &= \frac{d}{4}\frac{cd}{4} + \sqrt{\left(1-\frac{d^2}{16}\right)\left(1-\frac{c^2d^2}{16}\right)} \\
\left(\frac{c^2-3}{2c} - \frac{cd^2}{16}\right)^2 &= \left(1-\frac{d^2}{16}\right)\left(1-\frac{c^2d^2}{16}\right) \\
\left(\frac{c^2-3}{2c}\right)^2 - \frac{c^2-3}{16}d^2 + \frac{c^2d^4}{256} &= 1 - \frac{c^2+1}{16}d^2 + \frac{c^2d^4}{256}
\end{aligned}$$

$$\begin{aligned} \frac{1}{4}d^2 &= 1 - \left(\frac{c^2-3}{2c}\right)^2 \\ d &= \sqrt{4 - \left(\frac{c^2-3}{c}\right)^2} \end{aligned}$$

The last equation exactly matches our definition for d_c. □

We next show that the interferences between different components are eliminated. We examine two cases in the following two lemmas.

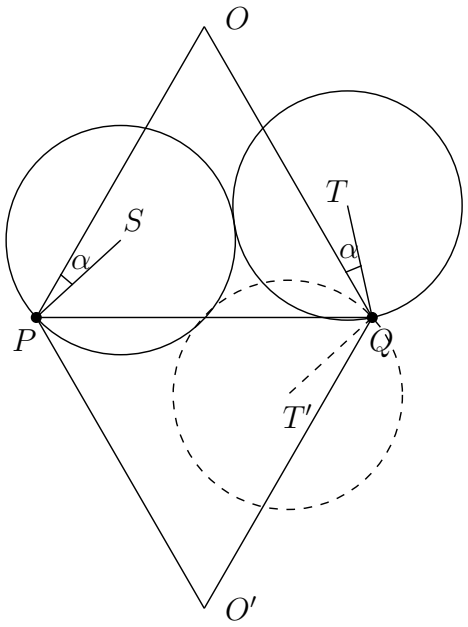


Figure 11. Interferences Between Single-site Clusters.

Lemma 3.8 *If $r \leq R$, then step 3 eliminates interferences between single-site clusters from different components.*

Proof. We refer to Figure 11. Sites P and Q are from two single-site clusters of different components. $PQ = e \geq r$; $OP = OQ = O'P = O'Q = r$.

One extreme case happens when $PQ = r$ and when P and Q are labeled along $\overrightarrow{PO}$ and $\overrightarrow{QO}$ before the rotation. Even in this extreme case, the two circles do not intersect after shrinking to radii $\frac{r}{3}$, as we have shown in our $(3+\epsilon)$-approximation. To improve the approximation factor, we observe that, if both circles rotate in the same direction for an angle α, a gap will appear between them, which allows both circles to grow a little larger, from radii $\frac{r}{3}$ to radii $\frac{r}{c}$, without causing intersection. Indeed, the α-rotation in our algorithm is motivated by this observation.

Let $\angle OPQ = \angle OQP = \angle O'PQ = \angle O'QP = \beta$. We have

$$\beta = \cos^{-1}\frac{e}{2r}.$$

The two circles centered at S and T have coordinates:

$$\left(\frac{r}{c}\cos(\beta-\alpha), \frac{r}{c}\sin(\beta-\alpha)\right) \quad \text{and} \quad \left(e - \frac{r}{c}\cos(\beta+\alpha), \frac{r}{c}\sin(\beta+\alpha)\right).$$

It follows that

$$ST = \frac{e}{c}\sqrt{c^2 - 2c\cos\alpha + 1}.$$

Since $e \geq r$ and $\alpha = \cos^{-1}\frac{c^2-3}{2c}$, a calculation shows that $ST \geq \frac{2r}{c}$. In fact, the parameter α is deliberately defined as such in order to eliminate the interference in this case.

Another extreme case happens when Q is labeled along $\overrightarrow{QO'}$ instead before the rotation. By symmetry, we only need to show that the circle centered at S does not include the midpoint of PQ. This leads to the constraint $\angle SPQ \geq \gamma$, where

$$\gamma = \cos^{-1}\frac{ce}{4r}.$$

Because we also have $\angle SPQ = \beta - \alpha$, we need to show that $\alpha + \gamma \leq \beta$. Here is the proof:

$$\begin{aligned}
\alpha + \gamma &\leq \beta \\
\cos(\alpha + \gamma) &\geq \cos\beta \\
\cos\alpha\cos\gamma - \cos\beta &\geq \sin\alpha\sin\gamma \\
\frac{c^2-3}{2c}\frac{ce}{4r} - \frac{e}{2r} &\geq \sqrt{1 - \frac{(c^2-3)^2}{4c^2}}\sqrt{1 - \frac{c^2e^2}{16r^2}} \\
(c^2-3)ce - 4ce &\geq \sqrt{-c^4+10c^2-9}\sqrt{16r^2-c^2e^2} \\
(c^2-7)^2c^2e^2 &\geq (-c^4+10c^2-9)(16r^2-c^2e^2) \\
(c^4-14c^2+49)c^2e^2 + (-c^4+10c^2-9)c^2e^2 &\geq (-c^4+10c^2-9)16r^2 \\
4(10-c^2)c^2e^2 &\geq (10-c^2)c^216r^2 - 144r^2 \\
(10-c^2)c^2\left(4 - \frac{e^2}{r^2}\right) &\leq 36
\end{aligned}$$

Because $e \geq r$, we have $4 - \frac{e^2}{r^2} \leq 3$. The last inequality reduces to $(10-c^2)c^2 \leq 12$, which is confirmed by calculation. □

To satisfy Lemma 3.8, we need a constant rotation angle $\alpha \geq \cos^{-1}\frac{c^2-3}{2c}$ for any constant factor $c < 3$. On the other hand, the maximum *safe* rotation angle at each site is determined by the distance d from the site to its nearest neighbor. According to Lemma 3.6, we must have $\alpha \leq \cos^{-1}\frac{d}{4} - \cos^{-1}\frac{cd}{4}$ (note that $\lim_{d\to 0}\alpha = 0$). As a compromise of these two conflicting constraints, we choose the threshold distance d_c to be the solution of Equation (3.1), and handle the difficult case of multi-site clusters

differently from single-site clusters.

Lemma 3.9 *If $r \leq R$, then step 3 eliminates inter-component interferences involving multi-site clusters.*

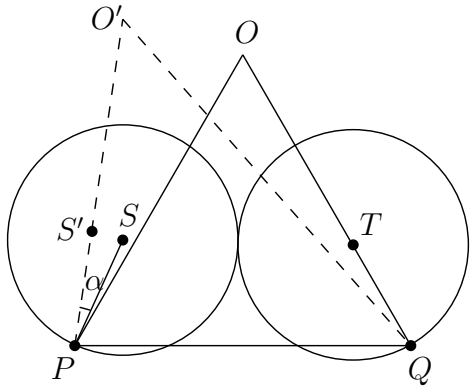


Figure 12. Interferences Involving Multi-site Clusters.

Proof. We refer to Figure 12. Sites P and Q are from two different components; Q is from a multi-site cluster. $PQ = e \geq r$, $OP = OQ = O'P = r$, and $O'Q = (D(d_c) + 1)r$. In the worst case, Q's circle (centered at T) is labeled along $\overrightarrow{QO}$, P's circle (centered at S') is labeled along $\overrightarrow{PO'}$ (because of our definition of *revised* maximal feasible region) and rotates from $\overrightarrow{PO'}$ for an angle α toward Q (its center moves from S' to S). We show that P and Q do not interfere even in this worst case, that is, $ST \geq \frac{2r}{c}$. To simplify the analysis, we prove a stronger claim: let $D_x(X, Y)$ denote the distance along $\overline{PQ}$ between points X and Y; we show that $D_x(S, T) \geq \frac{2r}{c}$.

Let $\beta = \angle O'PQ$. $\overline{S'S}$ is parallel to $\overline{PQ}$ when $\beta = \frac{\pi}{2} + \frac{\alpha}{2}$. We first consider the

case when $\beta \geq \frac{\pi}{2} + \frac{\alpha}{2}$. In this case, we have

$$D_x(P,S) \leq \frac{r}{c} \sin \frac{\alpha}{2}.$$

Since we also have

$$D_x(P,T) = e - \frac{e}{2c} \geq r - \frac{r}{2c},$$

it follows that

$$\begin{aligned} D_x(S,T) &= D_x(P,T) - D_x(P,S) \\ &\geq r - \frac{r}{2c} - \frac{r}{c} \sin \frac{\alpha}{2} \\ &\geq \frac{2r}{c}. \end{aligned}$$

The last inequality is confirmed by calculation.

We next consider the case when $\beta < \frac{\pi}{2} + \frac{\alpha}{2}$. $D_x(S,T)$ is a function of e. As e increases, β becomes smaller, and segment $\overline{S'S}$ tilts farther away from the horizontal position; this implies that

$$\frac{\mathrm{d}}{\mathrm{d}e} D_x(S',S) \leq 0.$$

We know that

$$D_x(P,S') = \frac{r}{c} \cos \beta = \frac{r^2 + e^2 - (D(d_c)+1)^2 r^2}{2ce};$$

therefore we have

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}e} D_x(P,S') &= \frac{1}{2c} + \frac{(D(d_c)+1)^2 - 1}{2c} \cdot \frac{r^2}{e^2} \\ &\leq \frac{1}{2c} + \frac{(\sqrt{3}-1+1)^2 - 1}{2c} \\ &= \frac{3}{2c}. \end{aligned}$$

We also know that

$$D_x(P,T) = e - \frac{e}{2c};$$

therefore we have

$$\frac{\mathrm{d}}{\mathrm{d}e} D_x(P,T) = 1 - \frac{1}{2c}.$$

Combining all the pieces, we have

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}e} D_x(S,T) &= \frac{\mathrm{d}}{\mathrm{d}e} D_x(P,T) - \frac{\mathrm{d}}{\mathrm{d}e} D_x(P,S') - \frac{\mathrm{d}}{\mathrm{d}e} D_x(S',S) \\ &\geq 1 - \frac{1}{2c} - \frac{3}{2c} \\ &> 0. \end{aligned}$$

The inequality above implies that the worst case happens when $e = r$. In this worst case, we have

$$D_x(S,T) = D_x(P,T) - D_x(P,S) = r - \frac{r}{2c} - \frac{r}{c}\cos(\beta - \alpha),$$

where

$$\beta = \cos^{-1} \frac{PO'^2 + PQ^2 - O'Q^2}{2 \cdot PO' \cdot PQ} = \cos^{-1} \frac{2 - (D(d_c) + 1)^2}{2}.$$

The equation $D_x(S,T) = \frac{2r}{c}$ simplifies to

$$\cos(\beta - \alpha) = c - \frac{5}{2}. \tag{3.2}$$

To ensure $D_x(S,T) \geq \frac{2r}{c}$ in this worst case, we choose the parameter $c = 2.98$ to be an approximate solution to Equation (3.2). Note that the two other important parameters, α and d_c, are closely related to c according to Equation (3.1) (Page 31).

Due to the complexity of these two equations, we have to resort to a computer program to search for the solution instead of solving the equations explicitly. □

The proof of Lemma 3.9 also explains why the revised maximal feasible region needs to be introduced in place of the classic maximal feasible region. It is exactly the difference between the two regions, the cone $\angle O'PO$ in Figure 12, that allows us to handle the difficult case of multi-site clusters.

The running time analysis of our $(2.98 + \epsilon)$-approximation algorithm is similar to that of our $(3 + \epsilon)$-approximation algorithm. In summary, we have proved the following theorem.

Theorem 3.2 *Our algorithm approximates MLUC with factor* $2.98 + \epsilon$ *and runs in* $O(n \log n + n(1/\epsilon)^{O(1/\epsilon^2)})$ *time.*

CHAPTER 4

APPROXIMATIONS FOR MLUCP

In this chapter, we present approximation algorithms for MLUCP. We first show a very simple algorithm that achieves a 1.5-approximation, then present a more sophisticated $(1.491 + \epsilon)$-approximation algorithm.

$(1.5 + \epsilon)$-Approximation for MLUCP

We first show a very simple algorithm for MLUCP that achieves a $(1.5 + \epsilon)$-approximation. Again, our algorithm is based on a binary search with a decision procedure. The decision procedure, given tentative label size r, either decides that r exceeds R^* and aborts, or finds a label placement with label size $\frac{2}{3}r$:

1. Compute maximal feasible regions for all sites at label size r. If any two sites have distance less than $2r$, or if any site has no feasible position, abort; otherwise, label each site with a circle pair of radii $\frac{2}{3}r$ in an arbitrary feasible position.

The following lemma claims the correctness of the decision procedure.

Lemma 4.1 *If $r \leq R^*$, then the decision procedure always finds a label placement with label size $\frac{2}{3}r$.*

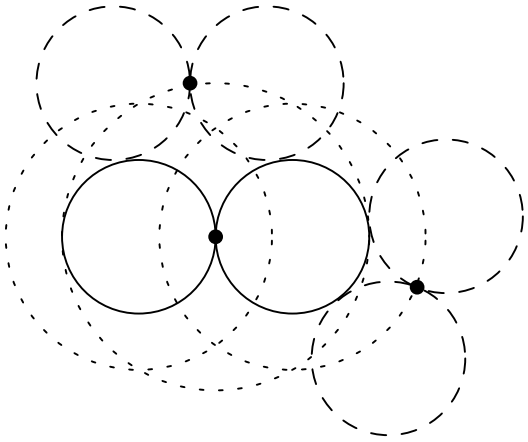


Figure 13. At Least $2r$ Between MLUCP Sites.

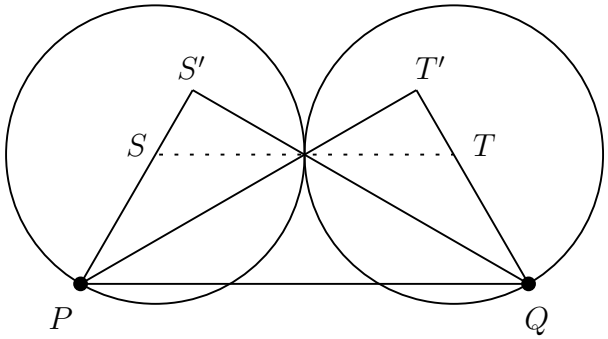


Figure 14. From r to $2r/3$.

Proof. We refer to Figure 13. The solid and dashed circle pairs have radii r, and each labels a site. The large dotted circle has radius $2r$ and is centered at the site labeled with the solid circle pair; the two small dotted circles have radii $\sqrt{3}r$ and are concentric with the two solid circles. All neighbors of the *solid* site must be outside the union of the three dotted circles; therefore, any two sites must have distance at least $2r$.

To show that the radii-$\frac{2}{3}r$ labels do not intersect as long as all sites are labeled

in feasible positions, we refer to Figure 14 for the extreme case. Sites P and Q have distance $2r$. The two circles centered at S and T are from the two circle pairs labeling P and Q. $S'P = T'Q = r$, $SP = TQ = \frac{2}{3}r$, and $S'Q = T'P = \sqrt{3}r$. Corollary 2.2 implies that, in the worst case, the labels of P and Q are directed along $\overline{PS'}$ and $\overline{QT'}$, respectively. Even in this worst case, a calculation shows that $ST = \frac{4}{3}r$; the two radii-$\frac{2}{3}r$ circles centered at S and T do not intersect. □

Similar arguments as in the previous chapter show that a binary search on the decision procedure yields a $(1.5 + \epsilon)$-approximation algorithm. After the initial $O(n \log n)$ time on preprocessing, each run of the decision procedure takes only linear time. The binary search takes $O(\log \frac{1}{\epsilon})$ steps. The total running time of our algorithm is therefore $O(n \log n + n \log \frac{1}{\epsilon})$. In summary, we have the following theorem.

Theorem 4.1 *Our algorithm approximates MLUCP with factor* $1.5 + \epsilon$ *and runs in* $O(n \log n + n \log \frac{1}{\epsilon})$ *time.*

$(1.491 + \epsilon)$-Approximation for MLUCP

Our $(1.491 + \epsilon)$-approximation algorithm also uses a binary search on a decision procedure; in fact it is almost identical to our $(1.5+\epsilon)$-approximation algorithm except that the more sophisticated rotation technique is used in the decision procedure, as in our $(2.98 + \epsilon)$-approximation algorithm for MLUC. We now describe the decision procedure:

1. Compute maximal feasible regions for all sites at label size r. If any two sites have distance less than $2r$, or if any site has no feasible position, abort; otherwise, label each site with a circle pair of radii $\frac{r}{c}$, where $c = 1.491$, in an arbitrary feasible position, then rotate the circle pair clockwise for an angle $\alpha \simeq 8.89°$.

Note that the two parameters, c and α, are defined differently in the previous chapter; we reuse them here without the risk of confusion.

The following lemma for MLUCP is analogous to Lemma 3.8 for MLUC.

Lemma 4.2 *If $r \leq R^*$, then the decision procedure always finds a label placement with label size $\frac{r}{c}$.*

Proof. We refer to Figure 15 for the two extreme cases. Sites P and Q have distance $PQ = e \geq 2r$. The two circles centered at S and T are from the two circle pairs labeling P and Q. $S'P = T'Q = r$, $S'Q = T'P = \sqrt{3}r$, $SP = TQ = \frac{r}{c}$, $\angle S'PS = \angle T'QT = \alpha$, and $\angle S'PQ = \angle T'QP = \beta$, where

$$\beta = \cos^{-1} \frac{e^2 - 2r^2}{2re}.$$

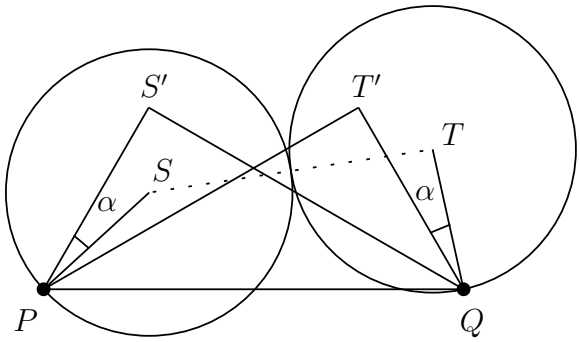
We need to show in both cases that $ST \geq \frac{2r}{c}$.

We first consider the case of Figure 15(a). Since S and T have coordinates

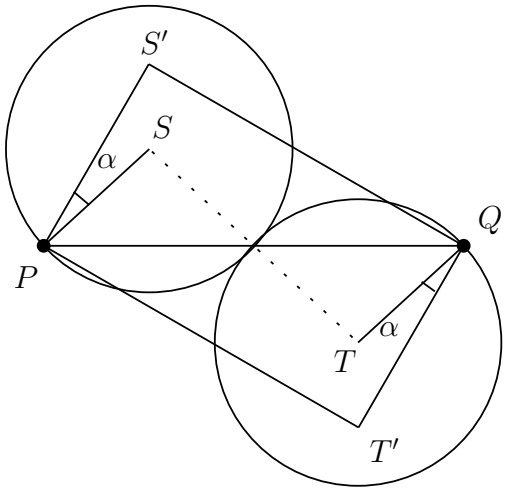
$$\left(\frac{r}{c}\cos(\beta - \alpha),\ \frac{r}{c}\sin(\beta - \alpha)\right) \quad \text{and} \quad \left(e - \frac{r}{c}\cos(\beta + \alpha),\ \frac{r}{c}\sin(\beta + \alpha)\right),$$

we have

$$ST^2 \ = \ e^2 + \frac{4r^2}{c^2}\cos^2\beta - \frac{4re}{c}\cos\alpha\cos\beta$$



(a)



(b)

Figure 15. Interferences Between MLUCP Sites.

$$
\begin{aligned}
&= e^2 + \frac{4r^2}{c^2}\left(\frac{e^2-2r^2}{2re}\right)^2 - \frac{4re}{c}\cos\alpha\frac{e^2-2r^2}{2re} \\
&= \left(1+\frac{1}{c^2}-\frac{2}{c}\cos\alpha\right)e^2 + \frac{4r^4}{c^2}\frac{1}{e^2} - \frac{4r^2}{c^2} + \frac{4r^2}{c}\cos\alpha
\end{aligned}
$$

When $e = 2r$ (assuming this is the worst case for now), we have

$$
\begin{aligned}
ST^2 &= \left(1+\frac{1}{c^2}-\frac{2}{c}\cos\alpha\right)4r^2 + \frac{4r^4}{c^2}\frac{1}{4r^2} - \frac{4r^2}{c^2} + \frac{4r^2}{c}\cos\alpha \\
&= \frac{r^2}{c^2}(4c^2 - 4c\cos\alpha + 1).
\end{aligned}
$$

To ensure that $ST \geq \frac{2r}{c}$, we must have

$$\alpha \geq \cos^{-1} \frac{4c^2 - 3}{4c}. \tag{4.1}$$

We next consider the case of Figure 15(b). S and T are symmetric. If $ST \geq \frac{2r}{c}$, then the distance from S to the midpoint of PQ must be at least $\frac{r}{c}$. So we have the constraint $\angle SPQ \geq \gamma$, where

$$\gamma = \cos^{-1} \frac{e/4}{r/c} = \cos^{-1} \frac{ce}{4r}.$$

Since $\angle SPQ = \beta - \alpha$, it follows that $\alpha \leq \beta - \gamma$. To bound α, we need to find the minimum value of $\beta - \gamma$, which reduces to solving the equation $\frac{\mathrm{d}}{\mathrm{d}e}(\beta - \gamma) = 0$. We have

$$\begin{aligned}
\cos\beta &= \frac{e^2 - 2r^2}{2re}, \\
\cos\gamma &= \frac{ce}{4r}, \\
\frac{\mathrm{d}}{\mathrm{d}e}\cos\beta = -\sin\beta\frac{\mathrm{d}}{\mathrm{d}e}\beta &= \frac{1}{2r} + \frac{r}{e^2}, \\
\frac{\mathrm{d}}{\mathrm{d}e}\cos\gamma = -\sin\gamma\frac{\mathrm{d}}{\mathrm{d}e}\gamma &= \frac{c}{4r}.
\end{aligned}$$

We now solve the equation $\frac{\mathrm{d}}{\mathrm{d}e}(\beta - \gamma) = 0$:

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}e}\beta &= \frac{\mathrm{d}}{\mathrm{d}e}\gamma \\
\frac{1}{\sin\beta}\left(\frac{1}{2r} + \frac{r}{e^2}\right) &= \frac{1}{\sin\gamma}\left(\frac{c}{4r}\right) \\
\frac{1}{c}\left(2 + \frac{4r^2}{e^2}\right) &= \frac{\sin\beta}{\sin\gamma} \\
\frac{1}{c^2}\left(2 + \frac{4r^2}{e^2}\right)^2 &= \frac{1 - \cos^2\beta}{1 - \cos^2\gamma}
\end{aligned}$$

$$\begin{aligned}
\frac{1}{c^2}\left(2+\frac{4r^2}{e^2}\right)^2 &= \frac{1-\frac{e^4-4r^2e^2+4r^4}{4r^2e^2}}{1-\frac{c^2e^2}{16r^2}} \\
\text{Substitute } d=c^2 \text{ and } f=\frac{e^2}{r^2}: & \\
\frac{1}{d}\left(2+\frac{4}{f}\right)^2 &= \frac{1-\frac{f^2-4f+4}{4f}}{1-\frac{df}{16}} \\
\frac{4}{d}\left(1+\frac{2}{f}\right)^2 &= 4\frac{8f-f^2-4}{16f-df^2} \\
\left(1+\frac{2}{f}\right)^2(16f-df^2) &= (8f-f^2-4)d \\
16f\left(1+\frac{2}{f}\right)^2-(f+2)^2d &= (8f-f^2-4)d \\
16f\left(1+\frac{2}{f}\right)^2 &= (8f-f^2-4+f^2+4f+4)d \\
16f\left(1+\frac{2}{f}\right)^2 &= 12fd \\
1+\frac{2}{f} &= \frac{\sqrt{3d}}{2} \\
\frac{1}{f} &= \frac{\sqrt{3d}-2}{4} \\
\frac{r^2}{e^2} &= \frac{\sqrt{3}c-2}{4} \\
\frac{e}{r} &= \frac{2}{\sqrt{\sqrt{3}c-2}}
\end{aligned}$$

The constraint

$$\begin{aligned}
\alpha &\le \beta-\gamma \\
&= \cos^{-1}\frac{e^2-2r^2}{2re}-\cos^{-1}\frac{ce}{4r} \\
&= \cos^{-1}\frac{\frac{e^2}{r^2}-2}{2\frac{e}{r}}-\cos^{-1}\frac{ce}{4r}
\end{aligned}$$

simplifies to

$$\alpha \le \cos^{-1}\frac{\lambda^2-2}{2\lambda}-\cos^{-1}\frac{c\lambda}{4}, \quad \text{where } \lambda=\frac{2}{\sqrt{\sqrt{3}c-2}}. \tag{4.2}$$

To satisfy both inequalities (4.1) and (4.2), we find an approximate solution to the equation

$$\cos^{-1}\frac{4c^2-3}{4c} = \cos^{-1}\frac{\lambda^2-2}{2\lambda} - \cos^{-1}\frac{c\lambda}{4}, \quad \text{where } \lambda = \frac{2}{\sqrt{\sqrt{3}c-2}}$$

and determine $c = 1.491$ and $\alpha \simeq 8.89°$.

With our choices of c and α, a calculation shows that, for the case in Figure 15(a), we have

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}e}ST^2 &= 2\left(1+\frac{1}{c^2}-\frac{2}{c}\cos\alpha\right)e - \frac{8r^4}{c^2}\frac{1}{e^3} \\
&\geq 2\left(1+\frac{1}{c^2}-\frac{2}{c}\cos\alpha\right)2r - \frac{8r^4}{c^2}\frac{1}{8r^3} \\
&= \left(4+\frac{3}{c^2}-\frac{8}{c}\cos\alpha\right)r \\
&\geq 0.
\end{aligned}$$

This implies that the worst case for the case of Figure 15(a) indeed happens when $e = 2r$. □

We immediately have the following theorem.

Theorem 4.2 *Our algorithm approximates MLUCP with factor* $1.491+\epsilon$ *and runs in* $O(n\log n + n\log\frac{1}{\epsilon})$ *time.*

CHAPTER 5

LOWER BOUNDS

In this chapter, we study the computational hardness of MLUC and MLUCP. We first prove the NP-hardness of the MLUC and MLUCP decision problems, then extend the proof to obtain a lower bound of 1.0349 for their optimization problems for maximizing the label sizes.

NP-Hardness Results

We first show that the MLUCP decision problem is NP-hard. Our proof is based on a polynomial-time reduction from the NP-hard Planar-3SAT problem [30] to MLUCP.

The Planar-3SAT problem is to decide whether a given Planar-3SAT instance is satisfiable. A 3SAT instance is Boolean formula in the form of a conjunction of clauses, where each clause is a disjunction of exactly three Boolean literals. A Planar-3SAT instance is a 3SAT instance with an additional graph property: the bipartite graph associated with the instance, where each clause is represented as a clause vertex, each variable as a variable vertex, and an edge connects a clause vertex to a variable vertex if and only if a literal of the variable appears in the clause—this graph can be embedded in the plane without crossing edges.

Given an input Planar-3SAT instance, our reduction constructs an MLUCP decision problem instance that "simulates" the Planar-3SAT instance, that is, the input Planar-3SAT instance is satisfiable if and only if the constructed MLUCP instance has a label placement. Our MLUCP instances have unit label size and are built from small groups of MLUCP sites called variable gadgets and clause gadgets.

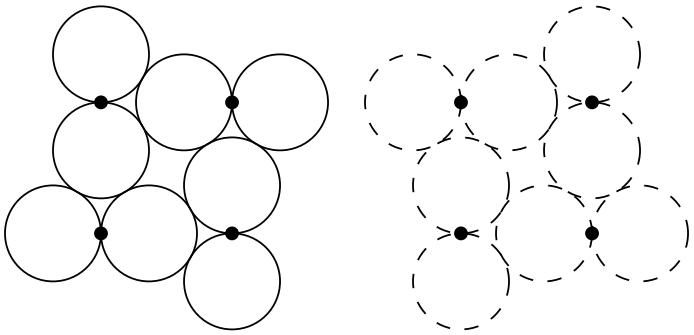


Figure 16. Variable Gadget.

A variable gadget is a four-site cluster as shown in Figure 16. The four *variable sites* in the cluster are placed in a square formation with side length $a_v = 1 + \sqrt{3}$. There are exactly two ways to label the variable gadget with unit circle pairs, thereby encoding the value of a Boolean variable.

A clause gadget is shown in Figure 17. Each clause has a *clause circle* C (the dotted circle in the figure) centered at a *clause site* S, with three *variable sites* O, P, and Q on the boundary of C in an equilateral triangular formation. A variable site X is labeled in a *critical* position if its two circles are directed along the line through $\overline{XS}$; a label position perpendicular to a critical position is a *perpendicular* position.

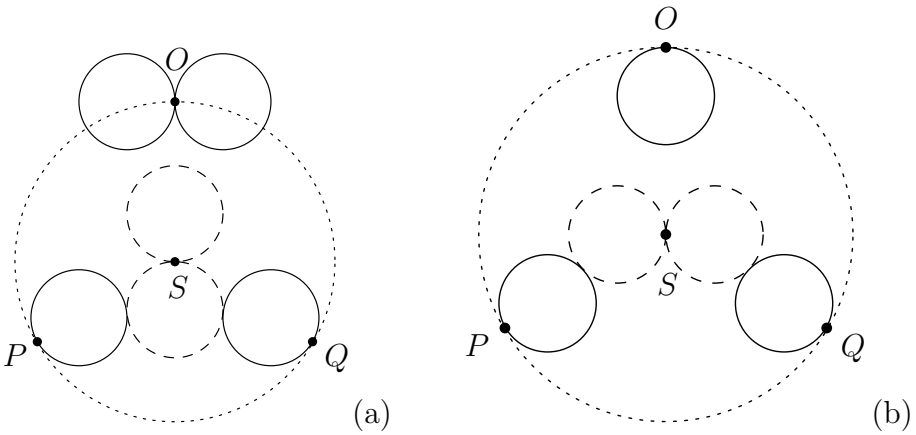


Figure 17. Clause Gadget.

In our construction, the variable gadgets are connected to the clause gadgets in such a way that every variable site on the clause circles has exactly two labeling choices, either in a critical position or in a perpendicular position. To complete the description of the clause gadget, we need to specify the radius of the clause circle, the choice of which is restricted by the following lemma.

Lemma 5.1 *In order to label a clause gadget, the radius of the clause circle must be at least* $\frac{3+\sqrt{13}}{2} \simeq 3.303$ *if exactly two variable sites on the clause circle are labeled in critical positions, and at least* $\frac{2+\sqrt{3}+\sqrt{15}}{2} \simeq 3.802$ *if all three variable sites are labeled in critical positions.*

Proof. We refer to Figure 17(a) for the case where P and Q are labeled in critical positions and O is labeled in a perpendicular position. The radius of C is minimum when the label of S is perpendicular to $\overline{PQ}$ and tangent to the labels of P and Q. A calculation shows that the radius of C is $\frac{3+\sqrt{13}}{2}$ in this extreme case.

We refer to Figure 17(b) for the case where all three variable sites are labeled in critical positions. The radius of circle C is minimum when the label of S is parallel to $\overline{PQ}$ and tangent to the labels of P and Q. A calculation shows that the radius of C is $\frac{2+\sqrt{3}+\sqrt{15}}{2}$ in this extreme case. □

We choose $r_c = \frac{3+\sqrt{13}}{2}$ as the radius of the clause circle in our clause gadget. Lemma 5.1 implies that, with such a radius for the clause circle, a clause gadget has a label placement if and only if at least one of its three variable sites is labeled in a perpendicular position, which simulates the property of a Planar-3SAT clause that the clause is satisfiable if and only if at least one of its three literals is true.

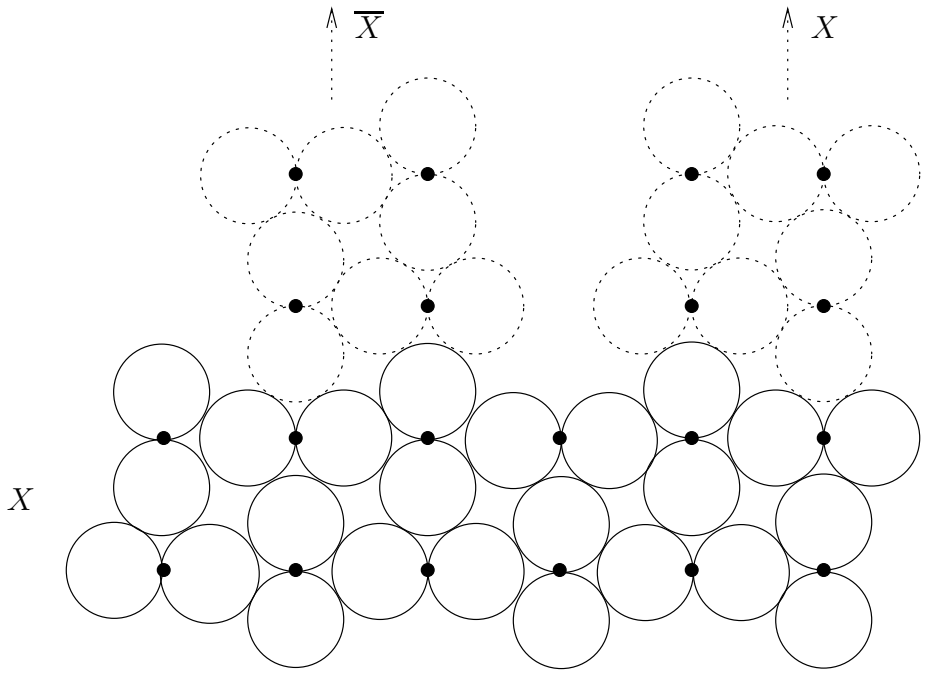


Figure 18. Variable Segment and Literal Leg.

We refer to Figure 18. Similar to other reductions from Planar-3SAT [20, 29],

our reduction models each variable as a *variable segment*—geometrically, a sequence of variable gadgets in the plane. All variable gadgets in the same variable segment must be labeled in the same formation; this ensures the consistency of the variable segment's Boolean value. For each literal of a variable in a clause of the input Planar-3SAT instance, we draw a *literal leg* to connect the corresponding variable segment and clause gadget. Similar to a variable segment, a literal leg is a sequence of variable gadgets all labeled in the same formation, either true or false. By drawing a literal leg at appropriate locations from a variable segment, we can choose the literal to be either a positive variable or its negation.

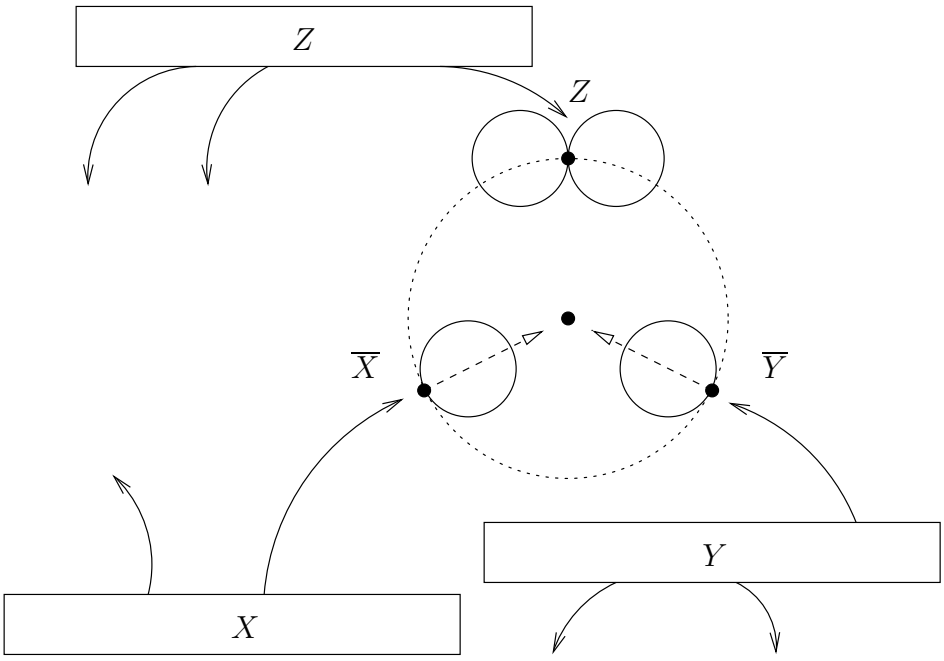


Figure 19. Planar Layout of Construction.

The planar layout of the clause gadgets, the variable segments, and the literal

legs connecting them is shown in Figure 19. The planarity of the input Planar-3SAT instance guarantees no crossing literal legs in our construction. A technical detail that our construction has to attend to is that the three variable sites on the clause circle must be placed in an equilateral triangular formation and labeled in either critical or perpendicular positions. This is achieved by *twisting* each literal leg, that is, shifting and rotating the variable gadgets in the literal leg, by very small constant amounts (the size of our construction is inversely related to this constant), until it eventually reaches the clause gadget at the required position.

By assigning a true (respectively, false) value to a literal if and only if its corresponding literal leg has its variable site on the clause circle labeled in a perpendicular (respectively, critical) position, we ensure that each satisfying Boolean assignment for the input Planar-3SAT instance corresponds to a label placement for the constructed MLUCP instance, and vice versa. We have the following lemma.

Lemma 5.2 *Our constructed MLUCP instance with unit label size has a label placement if and only if the input Planar-3SAT instance is satisfiable.*

An input Planar-3SAT instance with n clauses has $O(n)$ variables and $O(n)$ literals. We need $O(n)$ sites to model the n clause gadgets; we also need $O(n)$ variable gadgets to model each literal leg and each variable segment because of the planarity of the input instance. In total, we need $O(n^2)$ sites to encode an input Planar-3SAT instance as an MLUCP instance; the reduction consequently takes polynomial time.

We have proved the following theorem.

Theorem 5.1 *The MLUCP decision problem is NP-hard.*

By replacing every site in an MLUCP instance with a pair of coinciding (or sufficiently close) sites in an MLUC instance, we obtain a linear-time reduction from MLUCP to MLUC. From Theorem 5.1, this reduction immediately leads to the following corollary.

Corollary 5.1 *The MLUC decision problem is NP-hard.*

Inapproximability Results

To obtain the 1.0349 lower bound for the MLUC and MLUCP optimization problems, we show that our reduction from Planar-3SAT to MLUCP still works even if we shrink the label size of the MLUCP instance by a small amount.

In a label placement for a variable gadget, the labels, at the original unit label size, are tightly packed in a rigid formation. If we shrink the labels to slightly smaller sizes, gaps appear between the labels, which allow each label to rotate for a small angle. Let θ be the maximum angle that a label can rotate after we shrink the labels to radii $r = 1/1.0349$. We now calculate θ.

We refer to Figure 20. P and Q are two sites of distance $a_v = 1+\sqrt{3}$ from the same variable gadget; they are labeled in directions parallel and perpendicular, respectively, to $\overline{PQ}$ before the rotation. The rotation angle is the maximum when the two labels

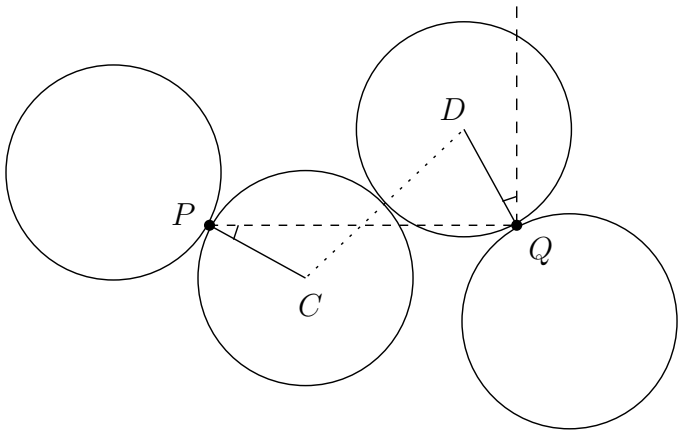


Figure 20. Rotation in Variable Gadget.

of P and Q rotate in opposite directions until they become tangent. In Figure 20, the label of P rotates clockwise and the label of Q rotates counterclockwise until the two circles centered at C and D become tangent. We have $PQ = a_v$, $PC = QD = r$, $CD = 2r$, $\angle QPC = \theta$, and $\angle PQD = \frac{\pi}{2} - \theta$. Let δ_x and δ_y be the distances between C and D along directions parallel and perpendicular, respectively, to $\overline{PQ}$. We have

$$\delta_x = a_v - r\sin\theta - r\cos\theta = a_v - rf_\theta$$

and

$$\delta_y = r\sin\theta + r\cos\theta = rf_\theta,$$

where

$$f_\theta = \sin\theta + \cos\theta = \sqrt{2}\sin\left(\theta + \frac{\pi}{4}\right).$$

The tangency condition $CD = \sqrt{\delta_x^2 + \delta_y^2} = 2r$ leads to the equation

$$2r^2 \cdot f_\theta^2 - 2a_v r \cdot f_\theta + a_v^2 - 4r^2 = 0.$$

Solving this equation, we have

$$f_\theta = \frac{1 - \sqrt{8(r/a_v)^2 - 1}}{2(r/a_v)}$$

and

$$\theta = \sin^{-1} \frac{1 - \sqrt{8(r/a_v)^2 - 1}}{2\sqrt{2}(r/a_v)} - \frac{\pi}{4}. \tag{5.1}$$

A calculation shows that $\theta \simeq 31.6°$. Because $\theta < 45°$, and because the angle between a label's two directions in the true and false formations is 90°, it is impossible for the label placement of a variable gadget to morph continuously from a true formation to a false one; the correctness of the variable gadget's Boolean encoding is therefore maintained. It is easy to check that, with the smaller label size r, the consistency of Boolean value propagation in the variable segments and the literal legs is also maintained.

We next examine the effect of smaller labels on the clause gadgets. With the smaller label size r, the label of each variable site on the clause circles can rotate for an angle at most θ away from its original position, that is, each variable site on the clause circles is now labeled *near* either a critical or a perpendicular position. The following lemma shows that, at the smaller label size r, an MLUCP clause gadget still has the necessary property to simulate a corresponding Planar-3SAT clause correctly.

Lemma 5.3 *If a clause gadget has a label placement with label size $r = 1/1.0349$, then at least one of its three variable sites must be labeled near a perpendicular position.*

Proof. We prove the lemma by contradiction. Assume that all three variable sites

are labeled near critical positions in a label placement for a clause gadget, as shown in Figure 17(b) (Page 48). Among the three intruding circles from the labels of the variable sites, at least two of them must have centers in the same half-plane bounded by the line through the two circle centers of S's label. Without loss of generality, assume that these two variable sites are P and Q. By symmetry, the extreme case happens when the label of S is parallel to $\overline{PQ}$, and the labels of both P and Q rotate for an angle θ away from their critical positions to make room for the label of S, that is, the label of P rotates clockwise and the label of Q rotates counterclockwise.

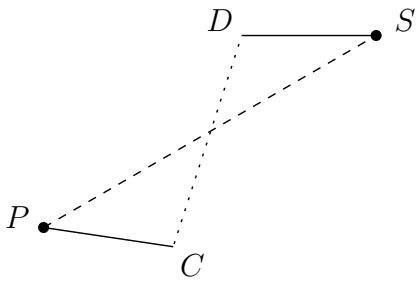


Figure 21. Rotation in Clause Gadget.

Because of the symmetry of P and Q, we only need to consider the interference between P and S. We refer to Figure 21. The intruding circle of P is centered at C; the left circle of S is centered at D. We have $PS = r_c = \frac{3+\sqrt{13}}{2}$, $PC = SD = r = 1/1.0349$, $\angle SPC = \theta \simeq 31.6°$, and $\angle PSD = \gamma = 30°$. Let δ_x and δ_y be distances between C and D along directions parallel and perpendicular, respectively, to $\overline{PS}$. We have

$$\delta_x = r_c - r\cos\theta - r\cos\gamma$$

and

$$\delta_y = r\sin\theta + r\sin\gamma.$$

Therefore, we have

$$\begin{aligned} CD &= \sqrt{\delta_x^2 + \delta_y^2} \\ &= \sqrt{r_c^2 + 2r^2(1+\cos(\theta-\gamma)) - 2r_c r(\cos\theta + \cos\gamma)}. \end{aligned}$$

A calculation shows that the distance CD is less than $2r$; the two labels of P and S still intersect even in this extreme case. In fact, the label size $r = 1/1.0349$ is carefully chosen to satisfy the constraint $CD < 2r$ in this case while balancing Equation 5.1 (Page 54). □

We are ready to prove that our constructed MLUCP instance indeed simulates the input Planar-3SAT instance correctly.

Lemma 5.4 *Our constructed MLUCP instance with label size $r = 1/1.0349$ has a label placement if and only if the input Planar-3SAT instance is satisfiable.*

Proof. The "if" direction is easy to prove. If the input Planar-3SAT instance is satisfiable, our constructed MLUCP instance has a label placement even with the original unit label size, as we have shown earlier in the NP-hardness proof for the MLUCP decision problem, so it obviously has a label placement with a smaller label size.

We next prove the "only if" direction. If our constructed MLUCP instance has a label placement with label size $r = 1/1.0349$, then according to Lemma 5.4 at

least one of the three variable sites in each clause gadget must be labeled near a perpendicular position. For the variable sites labeled near perpendicular positions, we assign true values to their corresponding literals; for those near critical positions, false values. Because of the consistency of Boolean value propagation in the literal legs and the variable segments, the resulting Boolean assignment for the variables satisfies the corresponding Planar-3SAT instance. □

We have the following theorem and corollary.

Theorem 5.2 *It is NP-hard to approximate the MLUCP optimization problem within factor 1.0349.*

Corollary 5.2 *It is NP-hard to approximate the MLUC optimization problem within factor 1.0349.*

Finally, we note that a problem is NP-complete [22] if it is both NP-hard and in NP. So far our discussion has been focused on only the aspect of NP-hardness because we don't know whether the MLUC and MLUCP decision problems are in NP. In fact, for a lot of geometric problems, including MLUC and MLUCP, whether they belong to NP remains open due to a rather curious technical difficulty with high precision arithmetic [5]. Let's take the famous Traveling Salesman Problem (TSP) as an example. The decision problem of Euclidean TSP asks whether a tour of length at most k exists, where the length of a tour is a sum of n square roots. To check whether a tour has a length no more than k, we need to compare the integer k to

the sum of n square roots of integers. For this task, no polynomial-time algorithm is known; a straightforward algorithm that removes the square roots by repeated squaring may generate very large numbers with encoding lengths exponential to the input length, and therefore requires super-polynomial time. We refer to this difficulty with high precision arithmetic as a *technical* difficulty because it affects not only NP-hard problems such as Euclidean TSP but also the supposedly easy problems such as Euclidean Minimum Spanning Tree. Researchers in computational geometry often avoid this precision issue by assuming exact real arithmetic in the model of computation, and discuss only the NP-hardness (instead of the NP-completeness) of geometric problems.

CHAPTER 6

IMPLEMENTATION AND EXPERIMENTS

In this chapter, we describe our implementation of a software system for automated map labeling. The name of the software system is AMLUC, abbreviated from "Automated Map Labeling with Uniform Circles," and pronounced as "am-luck." Our implementation is based on our approximation algorithms for MLUC; it also includes an effective heuristic that improves the quality of the label placements.

Given a set of input sites, AMLUC finds a label placement by computing in three phases: preprocessing, searching, and improving. In the following three sections, we examine each of these three phases in detail. In subsequent sections, we introduce the Graphical User Interface of AMLUC and present our experimental results.

Preprocessing Phase

In the preprocessing phase of AMLUC, we first compute the distances between every pair of sites, and store them in a look-up table, then compute the 15 nearest neighbors for each site, and finally determine the lower and upper bounds of the binary search for the next phase.

For the computation of the distance look-up table, a straight-forward $O(n^2)$ algorithm is used. To compute the 15 nearest neighbors for each site, we use a randomized

selection algorithm [8] that takes expected $O(n)$ time on each site.

After we have computed the nearest neighbors, we compute $d_2(P)$ and $d_{15}(P)$ for each site $P \in \mathcal{P}$, where $d_2(P)$ and $d_{15}(P)$, respectively, are the distances from P to its 2nd and 15th nearest neighbors. We then compute $d_2^* = \min\{d_2(P) \mid P \in \mathcal{P}\}$ and $d_{15}^* = \min\{d_{15}(P) \mid P \in \mathcal{P}\}$. We also compute $D_3(\mathcal{P})$ by finding the smallest D_3 among all combinations of three sites consisting of one site and two of its 15 nearest neighbors.

We choose $R^- = D_3/8$ and $R^+ = \min\{(2+\sqrt{3})D_3, d_{15}^*/2, d_2^*/d_0\}$ as the lower and upper bounds of the binary search for the next phase. The $D_3/8$ and $(2+\sqrt{3})D_3$ bounds come from Lemma 2.3 (Page 15). The $d_{15}^*/2$ bound is implied by Lemma 3.5 (Page 23). The d_2^*/d_0 bound is implied by Corollary 2.1 (Page 11), where d_0 is the solution to Equation 2.1 (Page 10).

The expected time complexity of our preprocessing phase is $O(n^2)$. We are aware of faster $O(n \log n)$ time algorithms [9, 17] for nearest neighbors computation, but decided against them for practical reasons. While in designing approximation algorithms we need to apply the most rigorous analysis to obtain the tightest approximation factors and time bounds, AMLUC is designed as an experimental and proof-of-concept system with only moderate input size. For the ease of implementation, we choose simple algorithms such as the randomized selection algorithm, and compensate for the possible loss in performance by careful software engineering. For example, we precompute the distance look-up table to reduce the floating-point computation; we

also try to avoid expensive numerical functions such as square root by using squares of distances whenever possible instead of distances.

Searching Phase

In the searching phase of AMLUC, we run a binary search that repeatedly calls a decision procedure with tentative label sizes. If the decision procedure succeeds, we record the label directions, update the lower search bound to the label size, and call the decision procedure again with a larger label size, that is, the average of the updated lower and upper search bounds; if the decision procedure fails, we update the upper search bound and try a smaller label size. When the search converges, that is, the difference between the lower and the upper search bounds becomes a small fraction of their average, we copy back the most recent successful label directions from the record, and return $\frac{1}{3}$ of the final lower search bound as the label size.

The decision procedure has three steps. First, we compute the maximal feasible regions for all sites. Second, we compute the distance graph and the connected components in it. Third, we try to label each component by brute force.

To compute the maximal feasible regions of a site, we discretize the 360° continuous direction range into 72 evenly spaced directions. For each direction, we check whether a circle labeled in that direction contains any of the site's 15 nearest neighbors, and record the result in an array associated with the site.

To compute the distance graph, we use the tentative label size as the threshold

distance, and consider only edges between each site and its 15 nearest neighbors; this results in an undirected graph with $O(n)$ edges. To compute the connected components within the distance graph, we use Tarjan's disjoint set algorithm with union-by-rank and path-compression heuristics [8]. This algorithm does not run in linear time in the strict sense, but it is extremely simple and its amortized time complexity is almost linear. To obtain guaranteed linear running time instead, we could cast our undirected distance graph into a directed graph, by replacing every undirected edge with two directed edges in both directions, and compute the strongly connected components in the directed graph using depth-first search [8]. However, as we discussed before, we prefer simple algorithms for the ease of implementation.

After we have obtained the connected components, we try to find a label placement for each component by brute force. Our brute-force implementation is a simple branch-and-bound variant, with the label directions of each site limited to the previously computed discrete feasible directions.

Improving Phase

The improving phase of AMLUC employs a *shake-and-grow* heuristic which is motivated by our $(2.98 + \epsilon)$-approximation algorithm for MLUC. In our presentation of that approximation algorithm, we proved that by rotating the circles of single-site clusters by small angles, we can break the barrier imposed by the worst case as shown in Figure 8, and grow the circles to larger sizes. To improve the approximation factor

from $3+\epsilon$ to $2.98+\epsilon$, and at the same time to simplify our analysis and proof, we had all circles rotate in the same direction in our algorithm, which was indeed very unintuitive. A question naturally comes to mind: if we allow every circle to rotate in both directions, will this result in a better algorithm? This simple question proved to be quite puzzling: it is intuitively true that we can find better label placements with the freedom of rotating the circles in both directions, but we haven't been successful in designing algorithms with provable better approximation factors.

With the belief that rotating the circles in both directions leads to better label placements, we designed the shake-and-grow heuristic for the improving phase of AMLUC. Our heuristic tries to improve the label placement output by the searching phase using several shake-and-grow rounds; each shake-and-grow round includes a large number of shake steps followed by a single grow step.

In a *shake* step, we randomly choose a site, compute the two maximum angles that the circle of this site can rotate continuously clockwise and, respectively, counterclockwise, without intersecting the circles of other sites, then relabel the site in the middle of the two extreme positions. When computing the maximum rotation angles for a site, we only consider the interferences from its 15 nearest neighbors, so each shake step takes only constant time.

In a *grow* step, we increase the label size to the maximum radii that the circles can grow to at their current label directions. For each site, and for each of its 15 nearest neighbors, we compute the maximum radii their two circles can grow to without

intersection; we then set the label size to the minimum value among all the examined pairs. Each grow step clearly takes linear time. Given n input sites, we run $3n$ shake steps and one grow step in each shake-and-grow round, so each round also takes linear time.

In a later section, we will show our experimental results on the effectiveness of the shake-and-grow heuristic.

Graphical User Interface

The Graphical User Interface (GUI) of AMLUC is designed to be simple and intuitive. From Figure 22, which contains a screenshot of AMLUC in action, we can see that the GUI of AMLUC has a single window; a canvas occupies most of the window's central space, and a control panel resides near the bottom of the window.

The canvas serves three purposes. First, it displays the map, which includes the point sites contained in a square frame, and a circle labeling each site. Second, it displays an information string near its top-left corner. The information string informs the number of sites in the map, the label size obtained by the searching phase, the label size obtained by the improving phase, and the ratio of the two sizes. Third, it is a scratch pad where the user can add or remove sites by directly clicking on the map.

The control panel contains two groups of controls. The *input* group is on the left side of the panel. It contains two checkboxes for "Add" and "Remove," a textfield,

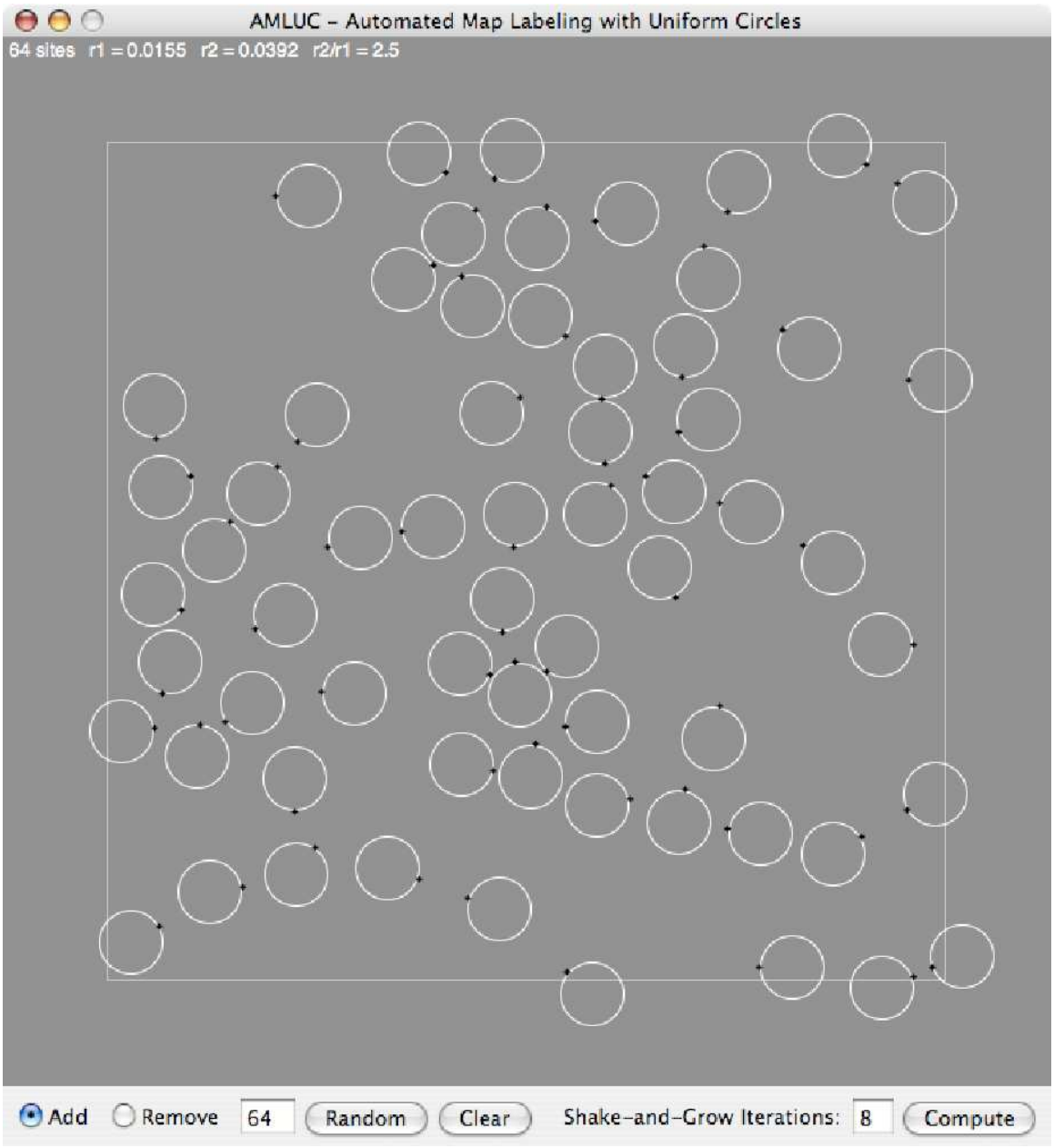


Figure 22. Graphical User Interface of AMLUC.

a "Random" button, and a "Clear" button. The "Add" and "Remove" checkboxes control the manipulation mode of input sites: when set to "Add," the user can add a site anywhere in the map by clicking in the square frame; when set to "Remove," the user can remove a site from the map by clicking on or near the site. When the "Random" button is pressed, depending on the current manipulation mode, some random sites will be either added to or removed from the map, the exact number of which is controlled by the number entered in the textfield to the left of the button. When the "Clear" button is pressed, all sites are removed from the map.

The *compute* group is on the right side of the control panel. It contains a textfield where the user can specify the number of shake-and-grow rounds in the improving phase of AMLUC, and a "Compute" button that, when pressed, starts the three-phase computation of a label placement for the input sites.

Experimental Results

AMLUC is implemented in the Java programming language, using standard JDK packages including the Java Collections Framework and the Abstract Windows Toolkit. Our development and testing system is an Apple iBook with a 1.07 GHz PowerPC G4 CPU and 256 MB RAM running Mac OS X Version 10.3.8 and Java 2 Platform Standard Edition 1.4.2.

Let r_1 and r_2, respectively, be the label sizes of the label placements output by the searching phase and the improving phase, respectively, of AMLUC. The ratio r_2/r_1

can used as a rough measure for the effectiveness of our shake-and-grow heuristic in the improving phase.

We tested AMLUC on randomly generated MLUC instances of 64 input sites. The number of shake-and-grow iterations in the improving phase is set to 8. In our experiment, AMLUC spent only a couple of seconds on average on each MLUC instance; the ratio r_2/r_1 was typically in the range between 2.0 and 2.7.

Since the label placements with label sizes r_1 are found by the searching phase of AMLUC, which is adapted from our $(3+\epsilon)$-approximation algorithm, we know that r_1 is approximately at least one third of the optimal label size. Our experimental results of the r_2/r_1 ratio show that AMLUC often finds near-optimal label placements: our shake-and-grow heuristic is very effective on random MLUC instances.

CHAPTER 7

CONCLUSION

In this dissertation, we studied two geometric optimization problems motivated by cartographic applications: Map Labeling with Uniform Circles (MLUC) and Map Labeling with Uniform Circle Pairs (MLUCP). We showed that the decision problems of both MLUC and MLUCP are NP-hard, and that the related optimization problems for maximizing the label sizes are NP-hard to approximate within factor 1.0349. We designed approximation algorithms with constant performance guarantees for the two problems: for MLUC, we presented a $(3 + \epsilon)$-approximation and a $(2.98 + \epsilon)$-approximation; for MLUCP, a $(1.5 + \epsilon)$-approximation and a $(1.491 + \epsilon)$-approximation. We also described the implementation of AMLUC, a software system for automated map labeling with uniform circles. The system is based on our approximation algorithms for MLUC and uses an effective shake-and-grow heuristic to find near-optimal label placements.

In conclusion, this dissertation shows that rigorous mathematical reasoning and analysis, intuitive and effective heuristics, and sound software engineering principles are three indispensable elements in deploying successful attacks on hard computational problems.

REFERENCES CITED

[1] Pankaj K. Agarwal, Marc J. van Kreveld, and Subhash Suri. Label placement by maximum independent set in rectangles. *Computational Geometry: Theory and Applications*, 11(3-4):209–218, 1998.

[2] John Ahn. *Automatic Map Name Placement System*. Ph.D. thesis, Rensselaer Polytechnic Institute, 1984.

[3] John Ahn and Herbert Freeman. AUTONAP - an expert system for automatic map name placement. In *Proc. International Symposium on Spatial Data Handling (SDH'84)*, pages 544–569, 1984.

[4] Sergey Bereg. Personal communication, 2004.

[5] Marshall Bern and David Eppstein. Approximation algorithms for geometric problems. In Dorit Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*, pages 296–345, PWS Publishing, 1996.

[6] Bernard Chazelle and 36 co-authors. The computational geometry impact task force report. *Advances in Discrete and Computational Geometry*, vol. 223, pages 407–463, American Mathematical Society, 1999.

[7] Jon Christensen, Joe Marks, and Stuart Shieber. An empirical study of algorithms for point-feature label placement. *ACM Transactions on Graphics*, 14(3):203–232, 1995.

[8] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[9] Amitava Datta, Hans-Peter Lenhof, Christian Schwarz, and Michiel Smid. Static and dynamic algorithms for k-point clustering problems. *Journal of Algorithms*, 19(3):474–503, 1995.

[10] Mark de Berg, Marc J. van Kreveld, Mark H. Overmars, and Otfried Schwarzkopf. *Computational Geometry*. Springer-Verlag, 1997.

[11] Srinivas Doddi, Madhav V. Marathe, Andy Mirzaian, Bernard M. E. Moret, and Binhai Zhu. Map labeling and its generalizations. In *Proc. 8th ACM-SIAM Symposium on Discrete Algorithms (SODA'97)*, pages 148–157, 1997.

[12] Srinivas Doddi, Madhav V. Marathe, and Bernard M. E. Moret. Point set labeling with specified positions. In *Proc. 16th Annual ACM Symposium on Computational Geometry (SoCG'00)*, pages 182–190, 2000,

[13] Srinivas Doddi, Madhav V. Marathe, and Bernard M. E. Moret. Point set labeling with specified positions. *International Journal of Computational Geometry & Applications*, 12(1-2):29–66, 2002.

[14] Jeffrey S. Doerschler. *A Rule-Based System for Dense-Map Name Placement*. Ph.D. thesis, Rensselaer Polytechnic Institute, 1987.

[15] Jeffrey S. Doerschler and Herbert Freeman. A rule-based system for dense-map name placement. *Communications of the ACM*, 35:68–79, 1992.

[16] Shawn Edmondson, Jon Christensen, Joe Marks, and Stuart Shieber. A general cartographic labeling algorithm. *Cartographica*, 33(4):13–23, 1997.

[17] David Eppstein and Jeff Erickson. Iterated nearest neighbors and finding minimal polytopes. *Discrete & Computational Geometry*, 11:321–350, 1994.

[18] Thomas Erlebach, Klaus Jansen, and Eike Seidel. Polynomial-time approximation schemes for geometric graphs. In *Proc. 12th ACM-SIAM Symposium on Discrete Algorithms (SODA'01)*, pages 671–679, 2001.

[19] Michael Formann. *Algorithms for Geometric Packing and Scaling Problems*. Ph.D. thesis, Fachbereich Mathematik und Informatik, Freie Universität Berlin, 1992.

[20] Michael Formann and Frank Wagner. A packing problem with application to lettering of maps. In *Proc. 7th Annual ACM Symposium on Computational Geometry (SoCG'91)*, pages 281–288, 1991.

[21] Robert J. Fowler, Michael S. Paterson, and Steven L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Information Processing Letters*, 12(3):133–137, 1981.

[22] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[23] Stephen A. Hirsch. An algorithm for automatic name placement around point data. *The American Cartographer*, 9(1):5–17, 1982.

[24] Eduard Imhof. Positioning names on maps. *The American Cartographer*, 2(2):128–144, 1975.

[25] Claudia Iturriaga. *Map Labeling Problems*. Ph.D. thesis, University of Waterloo, 1999.

[26] Minghui Jiang, Sergey Bereg, Zhongping Qin, and Binhai Zhu. New bounds on map labeling with circular labels. In *Proc. 15th Annual International Symposium on Algorithms and Computation (ISAAC'04)*, LNCS 3341, pages 606–617, 2004.

[27] Minghui Jiang, Jianbo Qian, Zhongping Qin, Binhai Zhu, and Robert Cimikowski. A simple factor-3 approximation for labeling points with circles. *Information Processing Letters*, 87(2):101–105, 2003.

[28] Christopher Jones. Cartographic name placement with Prolog. *IEEE Computer Graphics & Applications*, 9(5):36–47, 1989.

[29] Donald E. Knuth and Arvind Raghunathan. The problem of compatible representatives. *SIAM Journal on Discrete Mathematics*, 5(3):422–427, 1992.

[30] David Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11(2):329–343, 1982.

[31] Joe Marks and Stuart Shieber. The computational complexity of cartographic label placement. Technical Report TR-05-91, Harvard CS, 1991.

[32] Joel L. Morrison. Computer technology and cartographic change. In D.R.F. Tayler, editor, *The Computer in Contemporary Cartography*, Johns Hopkins University Press, 1980.

[33] James E. Mower. *The Selection, Implementation, and Evaluation of Heuristics for Automated Name Placement*. Ph.D. thesis, The State University of New York at Buffalo, 1989.

[34] Zhongping Qin, Alexander Wolff, Yinfeng Xu, and Binhai Zhu. New algorithms for two-label point labeling. In *Proc. 8th European Symposium on Algorithms (ESA'00)*, LNCS 1879, pages 368–379, 2000.

[35] Zhongping Qin, Binhai Zhu. A factor-2 approximation for labeling points with maximum sliding labels. In *Proc. 8th Scandinavian Workshop on Algorithm Theory (SWAT'02)*, LNCS 2368, pages 100–109, 2002.

[36] Michael J. Spriggs and J. Mark Keil. A new bound for map labeling with uniform circle pairs. *Information Processing Letters*, 81(1):47–53, 2002.

[37] Tycho Strijk. *Geometric Algorithms for Cartographic Label Placement*. Ph.D. thesis, Utrecht University, 2001.

[38] Tycho Strijk and Alexander Wolff. Labeling points with circles. *International Journal of Computational Geometry & Applications*, 11(2):181–195, 2001.

[39] Steven van Dijk. *Genetic Algorithms for Map Labeling*. Ph.D. thesis, Utrecht University, 2001.

[40] Marc van Kreveld, Tycho Strijk, and Alexander Wolff. Point set labeling with sliding labels. In *Proc. 14th Annual ACM Symposium on Computational Geometry (SoCG'98)*, pages 337–346, 1998.

[41] Marc van Kreveld, Tycho Strijk, and Alexander Wolff. Point labeling with sliding labels. *Computational Geometry: Theory and Applications*, 13:21–47, 1999.

[42] Frank Wagner and Alexander Wolff. A combinatorial framework for map labeling. In *Proc. Symposium on Graph Drawing (GD'98)*, LNCS 1547, pages 316–331, 1998.

[43] Alexander Wolff. *Automated Label Placement in Theory and Practice*. Ph.D. thesis, Fachbereich Mathematik und Informatik, Freie Universität Berlin, 1999.

[44] Alexander Wolff, Michael Thon, and Yinfeng Xu. A better lower bound for two-circle point labeling. In *Proc. 11th Annual International Symposium on Algorithms and Computation (ISAAC'00)*, LNCS 1969, pages 422–431, 2000.

[45] Alexander Wolff, Michael Thon, and Yinfeng Xu. A simple factor-2/3 approximation algorithm for two-circle point labeling. *International Journal of Computational Geometry and Applications*, 12(4):269–281, 2002.

[46] Pinhas Yoeli. The logic of automated map lettering. *The Cartographic Journal*, 9:99–108, 1972.

[47] Binhai Zhu and Chung Keung Poon. Efficient approximation algorithms for multi-label map labeling. In *Proc. 10th Annual International Symposium on Algorithms and Computation (ISAAC'99)*, LNCS 1741, pages 143–152, 1999.

[48] Steven Zoraster. The solution of large 0-1 integer programming problems encountered in automated cartography. *Operations Research*, 38(5):752–759, 1990.

APPENDICES

APPENDIX A

SOURCE CODE FOR NUMERICAL CHECKING

```
/*
 *    mluc.c
 *
 *    Minghui Jiang
 */

#include <stdio.h>
#include <math.h>

double D(double d) {
    if (d > sqrt(3.0) - 1.0)
        return 0.0;
    else {
        double phi = acos((5.0 - (1.0 + d) * (1.0 + d)) / 4.0);

        return sqrt(5.0 - 4.0 * cos(M_PI / 3.0 - phi)) - 1.0;
    }
}

int main() {
    double c, d;

    for (c = 2.9; c < 3.0; c += 0.001) {
        double tmp = (c * c  - 3.0) / c;
        double d_c = sqrt(4.0 - tmp * tmp);
        double D_ = D(d_c);
        double alpha = acos((c * c - 3.0) / (2.0 * c));
        double beta = acos((2.0 - (D_ + 1.0) * (D_ + 1.0)) / 2.0);

        if (cos(beta - alpha) <= c - 2.5) {
            printf("mluc:  c = %f  d_c = %f  alpha = %f\n",
                    c, d_c, alpha * 180.0 / M_PI);
            break;
        }
    }

    for (d = 0.5; d > 0; d -= 0.0001)
        if (D(d) > d) {
            printf("       d_0 = %f\n", d);
            break;
        }
}
```

```
/*
 *    mlucp.c
 *
 *    Minghui Jiang
 */

#include <stdio.h>
#include <math.h>

int main() {
    double c;
    double alpha, beta, gamma;

    for (c = 1.0; c < 1.5; c += 0.001) {
        double lambda = 2.0 / sqrt(sqrt(3.0) * c - 2.0);

        if (lambda < 2.0 || lambda > 4.0 / c)
            continue;

        alpha = acos((4.0 * c * c - 3.0) / (4.0 * c));
        if (alpha < 0.0 || alpha > M_PI / 4.0)
            continue;

        beta = acos((lambda * lambda - 2.0) / (2.0 * lambda));
        if (beta < 0.0 || beta > M_PI / 3.0)
            continue;

        gamma = acos(c * lambda / 4.0);
        if (gamma < 0.0 || gamma > M_PI / 3.0)
            continue;

        if (alpha > beta - gamma)
            continue;

        printf("mlucp:  c = %f  alpha = %f\n",
                c, alpha * 180.0 / M_PI);
        break;
    }
}
```

```
/*
 *    gadget.c
 *
 *    Minghui Jiang
 */

#include <stdio.h>
#include <math.h>

int main() {
    double a_v = 1.0 + sqrt(3.0);
    double r_c = (3.0 + sqrt(13.0)) / 2.0;
    double gamma = M_PI / 6.0;
    double c;

    for (c = 1.05; c > 1.0; c -= 0.0001) {
        double r = 1.0 / c;
        double x = r / a_v;
        double theta = asin((1.0 - sqrt(8.0 * x * x - 1.0))
                        / (2.0 * sqrt(2.0) * x)) - M_PI / 4.0;
        double diff = r_c * r_c + 2.0 * r * r * (1.0 + cos(theta - gamma))
                        - 2.0 * r_c * r * (cos(theta) + cos(gamma))
                            - 4.0 * r * r;    /* CD^2 - 4 r^2 */

        if (diff < 0) {
            printf("gadget:  c = %f  theta = %f\n",
                    c, theta * 180.0 / M_PI);
            break;
        }
    }
}
```

APPENDIX B

SOURCE CODE FOR MLUC IMPLEMENTATION

```
/*
 *    MLUC.java
 *
 *    Minghui Jiang
 */

import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.util.List;
import java.util.*;

public
class MLUC extends Applet
{
    static String title =
        "AMLUC - Automated Map Labeling with Uniform Circles";
    static String copyright = "Copyright 2005 Minghui Jiang";
    static String version = "Sun Apr 10 13:34:20 MDT 2005";

    public void init() {
        MyWindow window = new MyWindow();
        window.setLayout(new BorderLayout());
        window.setResizable(false);

        MyMap map = new MyMap();
        map.addMouseListener(map);
        map.addMouseWheelListener(map);
        window.add("Center", map);

        Panel panel = new Panel();
        map.registerPanel(panel);

        CheckboxGroup cbg = new CheckboxGroup();

        Checkbox cb = new Checkbox("Add", cbg, true);
        cb.addItemListener(map);
        panel.add(cb);

        cb = new Checkbox("Remove", cbg, false);
        cb.addItemListener(map);
        panel.add(cb);

        TextField tf = new TextField("32", 3);
```

```
        map.registerRandomField(tf);
        panel.add(tf);

        Button button = new Button("Random");
        button.addActionListener(map);
        panel.add(button);

        button = new Button("Clear");
        button.addActionListener(map);
        panel.add(button);

        panel.add(new Label("   Shake-and-Grow Iterations:"));

        tf = new TextField("8", 2);
        map.registerShakeAndGrowField(tf);
        panel.add(tf);

        button = new Button("Compute");
        button.addActionListener(map);
        panel.add(button);

        window.add("South", panel);

        window.pack();
        window.setVisible(true);
    }

    public String getAppletInfo() {
        return title + "\n" + copyright + "\n" + version + "\n";
    }

}  // class MLUC


class MyWindow extends Frame
{
    MyWindow() {
        super(MLUC.title);

        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                e.getWindow().dispose();
            }
        });
    }
```

```
}  // class MyWindow


class MyMap extends Canvas implements ActionListener, ItemListener,
                                      MouseListener, MouseWheelListener
{
    private final static int A = 666;  // square map A^2
    private double zoom = 0.8;  // zoom factor for drawing

    private VolatileImage offscr;  // off-screen for double-buffering
    private Graphics2D g2;  // graphics handle of offscr

    private Panel panel;  // control panel
    private TextField tfRandom;  // # of random actions
    private TextField tfShakeAndGrow;  // # of shake-and-grow iterations

    private List list = new LinkedList();  // list of sites
    private Site[] array;  // array of sites

    private String state = "Add";  // current list-manipulation state

    private double r = 0.0;  // current label size
    private double r_low;  // lower bound of binary search
    private double r_high;  // higher bound of binary search
    private double r1 = 0.0;  // label size after 3-approximation
    private double r2 = 0.0;  // label size after shake-and-grow
    private double[][] D2;  // matrix of distance^2 between two sites

    /*
     *    MyMap-to-Screen and Screen-to-MyMap coordinate transformation
     */

    private int m2s(double xy) {  // map (0.0, 1.0)^2 to screen A^2
        return (int) ((0.5 + (xy - 0.5) * zoom) * A);
    }

    private double s2m(int xy) {  // screen A^2 to map (0.0, 1.0)^2
        return (xy * 1.0 / A - 0.5) / zoom + 0.5;
    }

    /*
     *    Rounding to n digits after decimal point
     */
```

```
private double round(double x, int n) {
    return ((int) (x * Math.pow(10.0, n) + 0.5)) / Math.pow(10.0, n);
}

/*
 *    Map drawing
 */

public void update(Graphics g) { paint(g); }

public void paint(Graphics g) {

    // initialize off-screen image for double buffering
    if (g2 == null) {
        offscr = createVolatileImage(A, A);
        g2 = offscr.createGraphics();
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);
        g2.setFont(new Font(null, Font.PLAIN, 12));
    }

    // draw background
    g2.setColor(Color.gray);
    g2.fillRect(0, 0, A, A);

    // draw map boundary
    drawFrame();

    // drawSite() after drawCircle() so that no site is obscured
    for (Iterator i = list.iterator(); i.hasNext(); ) {
        Site site = (Site) i.next();
        drawCircle(site);
    }
    for (Iterator i = list.iterator(); i.hasNext(); ) {
        Site site = (Site) i.next();
        drawSite(site);
    }

    // draw info string at top-left corner
    String info = list.size() + " sites";

    info += "   r1 = " + round(r1, 4) + "   r2 = " + round(r2, 4);
    if (r1 > 0.0 && r2 > r1)
        info += "   r2/r1 = " + round(r2 / r1, 1);
```

```
    g2.setColor(Color.white);
    g2.drawString(info, 4, 12);

    // show everything when off-screen image is ready
    g.drawImage(offscr, 0, 0, this);
}

private void drawFrame() {
    int delta = (int) ((zoom - 1.0) * 0.5 * A);
    int newA = (int) (zoom * A);

    g2.setColor(Color.lightGray);
    g2.drawRect(-delta, -delta, newA, newA);
}

private void drawSite(Site site) {
    int x = m2s(site.x);
    int y = m2s(site.y);

    g2.setColor(Color.black);
    g2.drawLine(x - 2, y, x + 2, y);
    g2.drawLine(x, y - 2, x, y + 2);
    g2.drawLine(x - 1, y - 1, x + 1, y + 1);
    g2.drawLine(x - 1, y + 1, x + 1, y - 1);
}

private void drawCircle(Site site) {
    int cx = m2s(site.cx());
    int cy = m2s(site.cy());
    int cr = (int) (r * zoom * A);

    g2.setColor(Color.white);
    g2.drawOval(cx - cr, cy - cr, cr + cr, cr + cr);
}

public Dimension getPreferredSize() {
    return new Dimension(A, A);
}

/*
 *    List (of sites) manipulation
 */

private void addSite(double x, double y) {
```

```
    // perturb a little bit for numerical stability
    x += (Math.random() - 0.5) * 0.001;
    y += (Math.random() - 0.5) * 0.001;

    if (x < 1.0 && x > 0.0 && y < 1.0 && y > 0.0)
        list.add(new Site(x, y));
}

private void removeSite(double x, double y) {
    Site s_ = null;
    double min = Double.MAX_VALUE;

    for (Iterator i = list.iterator(); i.hasNext(); ) {
        Site s = (Site) i.next();
        double d2 = s.dist2(x, y);

        if (d2 < min) {
            min = d2;
            s_ = s;
        }
    }

    if (s_ != null)
        list.remove(s_);
}

/*
 *    Control panel interface
 */

void registerPanel(Panel panel) {
    this.panel = panel;
}

void registerRandomField(TextField tf) {
    tfRandom = tf;
}

void registerShakeAndGrowField(TextField tf) {
    tfShakeAndGrow = tf;
}

private int parseRandomField() {
    try {
        return Integer.parseInt(tfRandom.getText());
```

```
    } catch (NumberFormatException e) {
        return 0;
    }
}

private int parseShakeAndGrowField() {
    try {
        return Integer.parseInt(tfShakeAndGrow.getText());
    } catch (NumberFormatException e) {
        return 0;
    }
}

/*
 *    Event handlers for GUI
 */

public synchronized void actionPerformed(ActionEvent e) {
    String command = e.getActionCommand();

    if (command.equals("Random")) {
        int count = parseRandomField();

        for (int i = 0; i < count; i++)
            if (state.equals("Add"))
                addSite(Math.random(), Math.random());
            else if (state.equals("Remove"))
                removeSite(Math.random(), Math.random());
        r1 = r2 = 0.0;

    } else if (command.equals("Clear")) {
        list.clear();
        r = r1 = r2 = 0.0;

    } else if (command.equals("Compute")) {
        panel.setEnabled(false);
        if (list.size() >= 3) {
            array = (Site[]) list.toArray(new Site[0]);
            compute();
        }
        panel.setEnabled(true);
    }

    repaint();
}
```

```
public synchronized void itemStateChanged(ItemEvent e) {
    Checkbox cb = (Checkbox) e.getItemSelectable();
    state = cb.getLabel();
}

public synchronized void mouseWheelMoved(MouseWheelEvent e) {
    int clicks = e.getWheelRotation();

    if (clicks > 0)
        for (int i = 0; i < clicks; i++)
            zoom *= 1.1;
    else
        for (int i = 0; i < -clicks; i++)
            zoom /= 1.1;

    repaint();
}

public synchronized void mouseClicked(MouseEvent e) {
    double x = s2m(e.getX());
    double y = s2m(e.getY());

    if (state.equals("Add"))
        addSite(x, y);
    else if (state.equals("Remove"))
        removeSite(x, y);
    r1 = r2 = 0.0;

    repaint();
}

public void mouseEntered(MouseEvent e) {}
public void mouseExited(MouseEvent e) {}
public void mousePressed(MouseEvent e) {}
public void mouseReleased(MouseEvent e) {}

/*
 *    Compute the distance^2s between sites for later look-up
 */

private void computeD2() {
    D2 = new double[array.length][array.length];

    for (int i = 0; i < array.length; i++)
```

```
        for (int j = i + 1; j < array.length; j++)
            D2[i][j] = D2[j][i] = array[i].dist2(array[j]);
}

/*
 *    Compute nearest neighbors of all sites
 *    (selection algorithm from CLR book)
 */

private int randomPartition(int[] is, int p, int q) {
    int r = (int) (Math.random() * (q - p + 1)) + p;
    double z = D2[i_][is[r]];  // the pivot

    while (true) {
        while (D2[i_][is[p]] < z)
            p++;
        while (D2[i_][is[q]] > z)
            q--;
        if (p < q) {
            int t = is[p];
            is[p] = is[q];
            is[q] = t;
        } else
            return q;
    }
}

private void partition(int[] is, int p, int q, int k) {
    if (p == q)
        return;

    int r = randomPartition(is, p, q);
    int k_ = r - p + 1;

    if (k < k_)
        partition(is, p, r, k);
    else if (k > k_)
        partition(is, r + 1, q, k - k_);
}

private int i_;  // index of current site in computeNearestNeighbors

private void computeNearestNeighbors() {
    int[] is = new int[array.length - 1];  // indices of array
    double min_d2 = Double.MAX_VALUE;
```

```
double min_dn = Double.MAX_VALUE;
double min_d3 = Double.MAX_VALUE;

for (int i = 0; i < array.length; i++) {

    // skip itself
    i_ = i;
    for (int j = 0; j < i; j++)
        is[j] = j;
    for (int j = i + 1; j < array.length; j++)
        is[j - 1] = j;

    // do partition
    int n;

    if (is.length > Site.NN) {
        partition(is, 0, is.length - 1, Site.NN);
        n = Site.NN;
    } else  // too few sites; no need for partition
        n = is.length;

    // copy nearest neighbors
    int[] is_ = new int[n];

    for (int j = 0; j < n; j++)
        is_[j] = is[j];

    // find nearest, 2nd nearest, and farthest
    double d1 = D2[i][is[0]];
    double d2 = D2[i][is[1]];

    if (d1 > d2) {
        double d = d1;
        d1 = d2;
        d2 = d;
    }

    double dn = d2;

    for (int j = 2; j < n; j++) {
        double d = D2[i][is[j]];

        if (d < d1) {
            d2 = d1;
            d1 = d;
```

```
            } else if (d > dn)
                dn = d;
            else if (d < d2)
                d2 = d;
        }
        if (d2 < min_d2)
            min_d2 = d2;
        if (dn < min_dn)
            min_dn = dn;

        // find 3-diameter
        for (int j = 0; j < n; j++)
            for (int k = j + 1; k < n; k++) {
                double d3 = Math.max(D2[is[j]][is[k]],
                                     Math.max(D2[i][is[j]], D2[i][is[k]]));

                if (d3 < min_d3)
                    min_d3 = d3;
            }

        // set neighbors
        array[i].neighbors = is_;
    }

    // determine r_low and r_high
    double d2 = Math.sqrt(min_d2);
    double dn = Math.sqrt(min_dn);
    double d3 = Math.sqrt(min_d3);

    r_high = (2.0 + Math.sqrt(3.0)) * d3;
    r_low = d3 / 8.0;

    if (array.length > Site.NN && r_high > dn / 2.0)
        r_high = dn / 2.0;
    if (r_high > d2 / 0.2393)
        r_high = d2 / 0.2393;
}

/*
 *    Compute connected components with r as threshold distance
 */

private Collection findComponents() {
    Map components = new HashMap();
```

```
    for (int i = 0; i < array.length; i++)
        array[i].makeSet();

    double rr = r * r;

    for (int i = 0; i < array.length; i++) {
        Site s = array[i];

        for (int k = 0; k < s.neighbors.length; k++) {
            int j = s.neighbors[k];
            Site t = array[j];

            if (D2[i][j] < rr) {
                Site u = s.findSet();
                Site v = t.findSet();

                if (u != v)
                    u.linkTo(v);
            }
        }
    }

    for (int i = 0; i < array.length; i++) {
        Site s = array[i];
        Site u = s.findSet();
        List component = (List) components.get(u);

        if (component == null) {
            component = new LinkedList();
            components.put(u, component);
        }
        component.add(s);
    }

    return components.values();
}

/*
 *    Find a label placement for one component with r
 */

private boolean bruteForce(Site[] ss) {
    int[] as = new int[ss.length];  // label directions

    for (int i = 0; i < as.length; i++)
```

```
            as[i] = -1;  // reset position

        int k = 0;  // start with the first site

        while (k >= 0) {
            as[k] = ss[k].getNextFeasible(as[k]);

            if (as[k] == Site.ND) {  // tried all positions; still bad
                as[k] = -1;  // reset position

                k--;
                continue;  // backtrack to previous site
            }

            ss[k].a = as[k];

            boolean bad = false;

            for (int i = 0; i < k; i++)
                if (ss[k].interfere(ss[i])) {
                    bad = true;
                    break;
                }

            if (bad)
                continue;  // try next position for current site

            k++;  // advance to next site

            if (k == ss.length)  // a complete placement!
                return true;
        }

        return false;  // cannot label these sites with r
    }

    /*
     *    Find a label placement for all sites with r
     */

    private boolean labelSites() {
        for (int i = 0; i < array.length; i++)
            if (!array[i].computeMaximalFeasibleRegions())
                return false;
```

```
    Collection components = findComponents();

    for (Iterator i = components.iterator(); i.hasNext(); ) {
        List component = (List) i.next();
        Site[] ss = (Site[]) component.toArray(new Site[0]);

        if (!bruteForce(ss))
            return false;
    }

    return true;
}

/*
 *    Find a label placement for all sites with the maximum r
 */

private void binarySearch() {
    int[] as = new int[array.length];  // label directions

    while ((r_high - r_low) / (r_high + r_low) > 0.005) {
        r = (r_high + r_low) / 2.0;

        if (labelSites()) {
            r_low = r;

            // record the label directions
            for (int i = 0; i < array.length; i++)
                as[i] = array[i].a;
        } else
            r_high = r;
    }

    r = r_low / 3.0;

    // set label directions to the best in history
    for (int i = 0; i < array.length; i++)
        array[i].a = as[i];
}

/*
 *    Find the maximum r with the current label directions
 */

private void grow() {
```

```
    double maxr = Double.MAX_VALUE;

    for (int i = 0; i < array.length; i++) {
        Site s = array[i];

        for (int k = 0; k < s.neighbors.length; k++) {
            int j = s.neighbors[k];
            Site t = array[j];

            double r = s.maxr(t);

            if (r < maxr)
                maxr = r;
        }
    }
    r = maxr;
}

/*
 *    Shake-and-Grow heuristic
 */

private void shakeAndGrow(int n) {
    grow();
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < array.length * 3; j++) {
            int k = (int) (array.length * Math.random());

            array[k].shake();
        }
        grow();
    }
}

/*
 *    Computation starts here
 */

private void compute() {

    // preprocessing
    computeD2();
    computeNearestNeighbors();

    // 3-approximation
```

```
        binarySearch();
        r1 = r;

        // shake-and-grow heuristic
        shakeAndGrow(parseShakeAndGrowField());
        r2 = r;
    }


private class Site
{
    double x;
    double y;
    int a;  // discrete label direction: a * 2 * PI / N

    final static int ND = 72;  // number of discrete label directions
    boolean[] feasible = new boolean[ND];

    final static int NN = 15;  // maximum number of nearest neighbors
    int[] neighbors;  // indices of at most 15 nearest neighbors
    double d2;  // distance^2 to 2nd nearest among 15 neighbors
    double dn;  // distance^2 to farthest among 15 neighbors

    Site(double x, double y) {
        this.x = x;
        this.y = y;
    }

    /*
     *    Maximal feasible regions
     */

    boolean computeMaximalFeasibleRegions() {
        double rr = r * r;
        boolean hasFeasible = false;

        for (a = 0; a < ND; a++) {
            double cx = cx();
            double cy = cy();

            feasible[a] = true;

            for (int k = 0; k < neighbors.length; k++) {
                int j = neighbors[k];
```

```
            if (array[j].dist2(cx, cy) < rr) {
                feasible[a] = false;
                break;
            }
        }

        if (feasible[a])
            hasFeasible = true;
    }

    return hasFeasible;
}

int getNextFeasible(int k) {
    for (int i = k + 1; i < ND; i++)
        if (feasible[i])
            return i;

    return ND;  // feasible positions exausted
}

/*
 *    Convert (int) direction index to (double) direction
 */

double i2d(int i) {
    return i * 2.0 * Math.PI / Site.ND;
}

/*
 *    Distance functions
 */

// x coordinate of circle center
double cx() {
    return x + r * Math.cos(i2d(a));
}

// y coordinate of circle center
double cy() {
    return y + r * Math.sin(i2d(a));
}

// distance^2 from this site to (x, y)
double dist2(double x, double y) {
```

```
    double dx = this.x - x;
    double dy = this.y - y;

    return dx * dx + dy * dy;
}

// distance^2 from this site to another site
double dist2(Site s) {
    return dist2(s.x, s.y);
}

// the maximum r both this site and t can grow to
// with current label directions
double maxr(Site t) {
    double x1 = x;
    double y1 = y;
    double a1 = i2d(a);

    double x2 = t.x;
    double y2 = t.y;
    double a2 = i2d(t.a);

    double x12 = x1 - x2;
    double y12 = y1 - y2;
    double a12 = a1 - a2;

    double a = 2.0 * (1.0 + Math.cos(a12));
    double b = - 2.0 * (x12 * (Math.cos(a1) - Math.cos(a2))
                        + y12 * (Math.sin(a1) - Math.sin(a2)));
    double c = -x12 * x12 - y12 * y12;

    if (a == 0.0 && b == 0.0)
        return Double.MAX_VALUE;

    double delta = b * b - 4.0 * a * c;

    if (delta < 0.0)
        return Double.MAX_VALUE;

    double r;

    if (a == 0.0)
        r = -c / b;
    else
        r = (-b + Math.sqrt(delta)) / (2.0 * a);
```

```
    return r < 0.0 ? Double.MAX_VALUE : r;
}

/*
 *    Check for interference between this site and another site
 */

boolean interfere(Site s) {
    double dx = cx() - s.cx();
    double dy = cy() - s.cy();

    return dx * dx + dy * dy < 4.0 * r * r;
}

/*
 *    Check for interference between this site and its neighbors
 */

boolean interfere() {
    for (int k = 0; k < neighbors.length; k++) {
        int i = neighbors[k];
        Site s = array[i];

        if (interfere(s))
            return true;
    }

    return false;
}

/*
 *    Shake
 */

void shake() {
    int b = a;  // backup the current direction
    int l, r;

    // turn clockwise
    for (l = 0; l < ND; l++) {
        a++;
        if (interfere())
            break;
    }
```

```
    if (l == ND) {  // no restriction on rotation
        a = b + ND / 2;  // turn 180 degrees
        if (a > ND)
            a -= ND;
        return;
    }

    // restore to backup direction
    a = b;

    // turn counter-clockwise
    for (r = 0; r < ND; r++) {
        a--;
        if (interfere())
            break;
    }

    // rotate to middle of the two extremes
    a = b + (l - r) / 2;
    if (a > ND)
        a -= ND;
}

/*
 *    Disjoint set union algorithm from CLR book
 */

Site p;
int rank;

void makeSet() {
    p = this;
    rank = 0;
}

void linkTo(Site site) {
    if (rank >= site.rank) {
        site.p = this;
        if (rank == site.rank)
            rank++;
    } else
        p = site;
}

Site findSet() {
```

```
        if (this != p)
            p = p.findSet();

        return p;
    }

}  // class Site

}  // class MyMap
```