Report

On the Approximation of Unit Disk Graph Coordinates

Thomas Rusterholz
tr@student.ethz.ch
Supervisors: Fabian Kuhn, Roger Wattenhofer
{kuhn,wattenhofer}@inf.ethz.ch
Department of Computer Science, ETH Zurich, 8092 Zurich, Switzerland

October 21, 2003

Abstract

In this paper we study a problem occuring in the context of geometric routing algorithms for mobile ad-hoc networks: Finding unit disk graph coordinates given a graph $G = (V, E)$. Based on a proof that recognition of unit disk graphs is an NP-hard problem, we show that the problem of finding unit disk graph coordinates given a graph $G = (V, E)$ is NP-hard. Subsequently, we show that the proof does not extend to quasi unit disk graphs, a generalization of unit disk graphs. We give an exact formulation of the problem of finding unit disk graph coordinates in terms of a quadratic feasibility problem and explore different approximations in terms of linear programs, quadratic programs and semidefinite programs.

1 Introduction

A *Unit Disk Graph* (UDG) $G = (V, E)$ is a graph where each node $u \in V$ has a position in the plane such that for any two nodes $u, v \in V$ there is an edge $e = (u, v) \in E$ if and only if the Euclidean distance between u and v is less than or equal to 1.

Equivalently, the unit disk graph may be defined as the intersection graph of a set of unit diameter disks in the plane. Each node corresponds to a disk in the plane, and two nodes are adjacent in the graph if the corresponding disks intersect.

One particular application of unit disk graphs occurs in the emerging field of *mobile ad-hoc networks*: In the UDG model for mobile ad-hoc networks, each

node has a *geometric location* and a *transmission range*—a disk centered at this node with fixed *communication radius*. It is assumed that all disks share the same communication radius. Two nodes can communicate directly if and only if they are in mutual transmission range i.e. their Euclidean distance is less than or equal to the communication radius.

As with traditional networks, the *routing problem* arises if two nodes cannot communicate directly. In this case their messages need to be relayed through a series of intermediate nodes. Since the topology of an ad-hoc network is dynamically changing, standard routing schemes which are used in traditional wired networks are not applicable for ad-hoc networks.

Recently, new routing algorithms have been developed to efficiently perform in the UDG model (see [2, 3, 4]). These are so called *geometric routing algorithms* requiring a unit disk graph with associated node positions as input.

In practical mobile ad-hoc networks, however, most often only connectivity information about the network is given but the positions of the nodes are unknown. The target of this project was to develop efficient algorithms to assign coordinates to the nodes in order to realize the network graph. This is equal to the problem of finding a UDG-realization given a graph $G = (V, E)$.

2 Mathematical Preliminaries

2.1 Convex Functions

For an introduction to convexity and its applications in optimization, see [7]. For a more advanced treatment of the topic the reader is referred to [8, 9].

Definition 2.1. (Convex Set) *A (point-)set K is called convex if for any two points p_1 and p_2 with $p_1 \in K,\ p_2 \in K$ all points*

$$\lambda p_1 + (1 - \lambda)p_2 \qquad (0 \leq \lambda \leq 1)$$

belong to K, i.e all points on the line between p_1 and p_2 belong to K.

Definition 2.2. (Convex Function) *A function $f : K \to \mathbb{R}$ from a convex set K to the set $\mathbb{R}$ of real numbers is called convex if for any two points x_1 and x_2 from K the following holds:*

$$f(\lambda x_1 + (1 - \lambda)x_2) \leq \lambda f(x_1) + (1 - \lambda)f(x_2)$$

with $(0 \leq \lambda \leq 1)$, i.e. if the set $\{(x, y) |\ x \in K,\ y \geq f(x)\}$ is convex.

Definition 2.3. (Concave Function) *A function $f : K \to \mathbb{R}$ is called concave if $g = -f$ is a convex function.*

2.2 Optimization Theory

For an introduction to optimization, see [7]. Again, a much deeper discussion may be found in [8, 9] (many practical examples are found in [9]).

Definition 2.4. (Optimization Problem) *The general optimization problem is given by*

$$\begin{array}{rll} \max & f(x_1,\ldots,x_n) & \\ \textit{subject to} & g_i(x_1,\ldots,x_n) \leq 0 & i=1,\ldots,m \\ & x_j \geq 0 & j=1,\ldots,n. \end{array}$$

The f and g_i are functions in the variables $x_1,\ldots,x_n$.

Definition 2.5. (Feasibility Problem) *In the feasibility problem corresponding to an optimization problem the objective function f is equal to 0. Solving the feasibility problem means to find values ("feasible values") for the variables $x_1,\ldots,x_n$ to satisfy all constraints g_i .*

Definition 2.6. (LP) *A LP (Linear Program) is an optimization problem where the f and g_i are linear.*

Definition 2.7. (QP) *A QP (Quadratic Program) is an optimization problem where f is quadratic and the g_i are linear.*

Definition 2.8. (QCQP) *A QCQP (Quadratically Constrained Quadratic Program) is an optimization problem where the f and g_i are both quadratic.*

Definition 2.9. (Convex Optimization Problem) *An optimization problem is called convex if the function f is concave under maximization and convex under minimization, respectively. The functions g_i are required to be convex in both cases.*

The following lemmas are given without proof (A proof may be found in [8]):

Lemma 2.1. *The feasible region of every set of linear constraints is convex.*

Lemma 2.2. *For any convex optimization problem, a* local *optimum of f is the* global *optimum of f.*

3 Unit Disk Graph Realization is NP-hard

In [5] SATISFIABILITY was reduced to the problem of recognizing unit disk graphs, showing that the *decision problem* asking if a given graph has a UDG-realization is NP-hard. It can be shown that the corresponding *optimization problem* of finding such a UDG-realization is NP-hard by reduction.

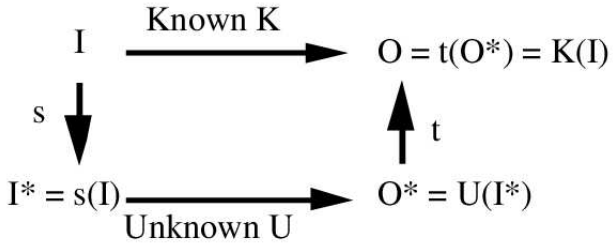

Figure 1: Reduction scheme for deriving a *lower bound* for problem U of unknown complexity by reduction from problem K of known complexity.

Lemma 3.1. *Unit disk graph realization is NP-hard*

Proof. The problem K of known complexity is the decision problem asking if a given graph has a UDG-realization. We know that K is NP-hard. Let the problem U of unknown complexity be the problem of finding a UDG-realization for a given graph.

The input transformation function s does nothing but pass on the input data it receives. $I^* = I$, see Figure 1. Both problem types K and U require the same kind of input, an arbitrary graph $G = (V, E)$.

The output transformation maps the output O^* of U to a yes/no-answer. It checks if the output O^* received from U constitutes a valid UDG-realization. This can be done in time $\mathrm{O}(n^2)$ where $n = |V|$: If for all tuples $(v_i, v_j) \in E$, the distance between v_i and v_j is ≤ 1 and for all tuples $(v_i, v_j) \notin E$, the distance between v_i and v_j is > 1, return yes (the graph has a UDG-realization), else return no (the graph does not have a UDG-realization).

If O^* is a UDG-realization, the input graph $G = (V, E)$ has a UDG-realization. If the output O^* does not realize the graph, the graph does not have a UDG-realization because otherwise, the algorithm solving U (find a UDG-realization for a given graph $G = (V, E)$) would have found it. □

4 The Original Proof

The aim of this section is to show that the proof in [5], although it works well for unit disk graphs, does not work if a graph class with a definition that differs only slightly from the unit disk graph definition is considered. For this reason, we define the notion of the quasi unit disk graph (Quasi-UDG) as introduced in [1]:

Definition 4.1. (Quasi Unit Disk Graph) *Let $V \subset \mathbb{R}^2$ be a set of points in the 2-dimensional plane and $d \in [0, 1]$ be a parameter. The symmetric Euclidean graph (V, E), such that for any pair $u, v \in V$*

- *$(u, v) \in E$ if $|uv| \leq d$ and*
- *$(u, v) \notin E$ if $|uv| > 1$,*

is called a Quasi Unit Disk Graph (Quasi-UDG) with parameter d.

In the following, we give an outline of the proof. We then give some details on key statements of the proof, on the components of a graph called G_C and their building blocks in particular. We show that these building blocks do not work for Quasi-UDGs and that, as a consequence, the components of the proof do not work as expected.

4.1 Proof Outline

This section gives a sketch of the proof in [5] reducing SATISFIABILITY to unit disk graph recognition. For further details we refer the reader to [5].

Given an instance C of SATISFIABILITY, a graph $G_C = (V_C, E_C)$ is constructed such that G_C has a UDG-realization if and only if C is satisfiable. It is assumed that w.l.o.g. each clause in C contains *at most* three literals and each variable appears in *at most* three clauses. The graph G_C is built in several stages. First, a graph G_C^{SAT} is constructed that corresponds closely to the instance C of SATISFIABILITY: The nodes of the graph correspond to the clauses, variables and negated variables. The nodes corresponding to variables and negated variables are called literal nodes. There is an edge between a literal node and a clause node if the literal appears in the clause. A notion of orientability is defined for this graph and it is proved that it is orientable if and only if C is satisfiable.

The next step considers a canonical drawing of this graph on the grid: A grid with polynomial size in the number of variables and clauses of the SATISFIABILITY instance C is constructed. Each of the nodes of the grid is either unused, or is associated with a unique component of the drawing. There are three groups of components: communication components, literals, and clauses. There are three different groups of communication components: wires, corners, and cross-overs. The communication components serve to connect the appropriate literals and clauses. A wire is a unit length line segment passing through a grid node, a corner is two half-length line segments meeting a right angles

at a grid node, and a cross-over is two unit length line segments crossing at right angles on a grid node. Each component in the drawing has one to four points where they can be connected to other components. These connection points are called terminals. The number of terminals is according to the degree of the component in G_C^{SAT}. The terminals on the top, bottom, left and right side are called T, B, L and R terminals respectively. Two components in the drawing are adjacent if they have coincident terminals. A complimentary pair of literals is considered to be a single truth setting component, there are no terminals between them. An *orientation* of a terminal is a *direction* N, S, E, or W. A terminal T (resp. B, L, R) is *directed away* from its component if it is oriented N (resp. S, W, E) and is *directed towards* its component otherwise. A grid drawing is said to be orientable if all terminals can be oriented subject to conditions:

draw1: only terminals adjacent to vertical line segments are directed N or S,

draw2: only terminals adjacent to horizontal line segments are directed E or W,

draw3: every wire, corner, cross-over line segment, and clause has at least one terminal directed away from it,

draw4: every truth setting component has a literal component wit *all* terminals directed away from it.

It is then proved that the grid drawing is orientable if and only if the underlying graph is orientable.

Finally, G_C is formed by simulating components of the grid drawing. A graph component is created for each grid drawing component: wires, corners, cross-overs, truth-setters, and clauses. Each of these graph components has terminals corresponding to the grid drawing terminals T, B, L and R. The graph G_C is constructed by connecting every pair of components adjacent in the grid drawing together by identifying appropriate terminals, by label. Terminal T (resp. B, L, R) should be identified with an adjacent B (resp. T, R, L) terminal.

Subsequently, it is shown that G_C has a UDG-realization if and only if the underlying grid drawing is orientable.

The proof is finished by showing that the entire reduction can be executed in polynomial time.

4.2 Graph Components

In the following, the building blocks are described from which the various graph components are composed: These building blocks consist of cycles ("cages") with additional independent nodes ("beads"). A bead is attached to two nodes (a "hinge") of a cage by a short path through another node ("chain"). Such a cage looks like the one in Figure 2. The maximum number of independent beads that can be embedded inside a cage in any UDG-realization is called *capacity*.

The capacity is constant and depends only on the number of nodes on the cycle. A bead embedded inside a cage diminishes its remaining capacity. It thereby displaces other beads, which may otherwise have been embedded inside the cage. This is the basic mechanism for propagating information in a UDG-realization of G_C.

The following lemma must hold in order for the notion of a node being embedded inside a cage to be well defined:

Lemma 4.1. *Let $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$ be two intersecting edges in a UDG $G = (V, E)$. Then the subgraph induced by $\{u_1, v_1, u_2, v_2\}$ contains K_3 as a subgraph.*

Proof. Suppose that $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$ cross at m. Let $|uv|$ denote the distance between point u and point v. Then

$$\begin{aligned} |u_1u_2| + |v_1v_2| &\leq (|u_1m| + |mu_2|) + (|v_1m| + |mv_2|) \\ &= (|u_1m| + |v_1m|) + (|mu_2| + |mv_2|) \\ &= |u_1v_1| + |u_2v_2| \leq 1 + 1 = 2 \end{aligned}$$

Therefore $|u_1u_2| \leq 1$ or $|v_1v_2| \leq 1$. Similarly, $|u_1v_2| \leq 1$ or $|v_1u_2| \leq 1$. Any of the four possibilities implies a K_3 subgraph. □

However, for Quasi-UDGs with parameter $1/\sqrt{2} \leq d < 1$, Lemma 4.1 does not hold anymore. Instead, the following holds (Lemma 4.2 is proved in [1]):

Lemma 4.2. *Let $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$ be two intersecting edges in a Quasi-UDG G with parameter $d \geq 1/\sqrt{2}$. Then at least one of the edges (u_1, u_2), (u_1, v_2), (v_1, u_2) and (v_1, v_2) exists in G.*

Similar arguments hold for the cage capacities. A *n-cage* is defined as a cage with a capacity for n beads. The capacities of cages with up to 10 nodes can be verified by considering the optimal disk packings in Figure 3: A 6-node cage cannot contain a bead, an 8-node cage cannot contain two beads, a 9-node cage cannot contain three beads, and a 10-node cage cannot contain four beads.

Note however that for Quasi-UDGs and *any* parameter $d < 1$, each of the packings adds capacity at least one.

The graph components do only work as long as beads embedded in one cage force other beads into different cages the way it was intended. For $d < 1$, too

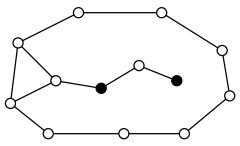

Figure 2: A cage with two embedded independent beads (black nodes) on two chain nodes.

many beads may be embedded in one cage and the graph components do not behave as expected. It may however be possible to overcome this problem by adjusting the cage sizes depending on d.

In Subsection 4.1, conditions $\boldsymbol{draw1-4}$ are described for a grid drawing to be *orientable*. The graph G_C has a UDG-realization if and only if the underlying grid drawing is orientable. In the following example, we show that for Quasi-UDGs with parameter $d = 1/\sqrt{2}$ it is possible for G_C to have a realization while the underlying grid drawing is not orientable because condition $\boldsymbol{draw3}$ cannot be satisfied, thus showing that the proof does not work for Quasi-UDGs with parameter $d = 1/\sqrt{2}$.

Figure 5 shows a wire component which is realized in an incorrect way (see Figure 4 for a correct realization of a wire component): The left terminal has been embedded into a 5-node cage attached to the cycle (so called "mortar"-cage). In the following, we explain why this is a valid realization for a Quasi-UDG with parameter $d = 1/\sqrt{2}$. Let t denote the terminal node, let c denote the chain node through which the terminal node is attached to the hinge. Finally, let m_1 and m_2 denote the two nodes of the shared cage edge. Refer to Figure 6 for details on the region of interest:

According to the cage capacities in Figure 3, a cage with up to 6 nodes cannot contain a bead. However, in a Quasi-UDG with parameter $d = 1/\sqrt{2}$ a 5-node cage can contain a bead. In the left part of Figure 6, the node t has distance $> d$ from all the nodes in the cycle. Thus none of the edges connecting the terminal to the nodes on the 5-node cage are required to exist in G_C according to Definition 4.1.

The edge between the chain node and the terminal crosses the edge shared between the 8-node cage and the 5-node cage. According to Lemma 4.2, at least one of the edges (m_1, c), (c, m_2), (m_1, t) and (t, m_2) exists in G_C. One of these edges is contained in every wire component anyway: the edge (c, m_2). In Figure 6, the nodes are arranged such that the only edge required to exist in G_C is (c, m_2) which readily exists in every wire component.

If wires are realized in this way in the realization of a graph G_C, displaced beads do not properly displace other beads anymore, since some beads are embedded in the surrounding mortar. The basic mechanism for propagating information in a realization of G_C is broken.

In the underlying grid drawing, the corresponding wire component has both terminals directed *towards* itself. Thus condition $\boldsymbol{draw3}$ is not met.

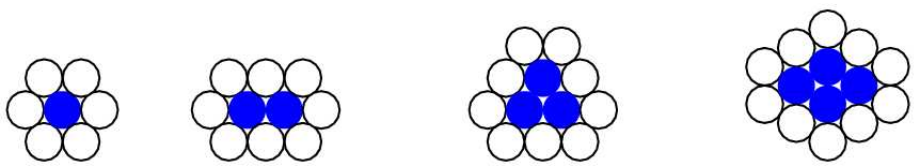

Figure 3: Optimal packings limiting a 0-cage, 1-cage, 2-cage and 3-cage

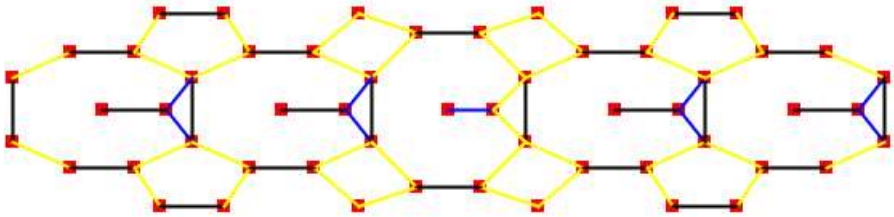

Figure 4: A wire component realized in the correct way (Quasi-UDG with parameter $d = 1/\sqrt{2}$). Lines with length $l = 1$ are plotted in black. Lines with $1/\sqrt{2} < l < 1$ are plotted in yellow. Lines with $l \leq 1/\sqrt{2}$ are plotted in blue.

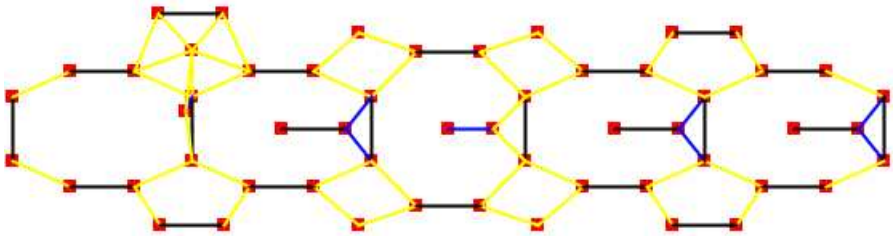

Figure 5: A wire component realized in an incorrect way. The left terminal is embedded into the upper mortar cage.

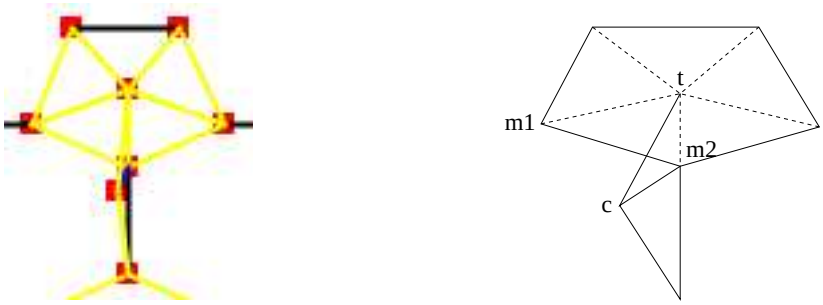

Figure 6: Detail view of a portion of the wire component depicted in Figure 5. Both pictures show the same portion of the wire component. On the left hand side, a detail view of the realization with real coordinates is shown. The right hand side shows a schematic of the same portion of the wire. Solid lines are edges belonging to G_C, dashed lines are edges not belonging to G_C.

5 Exact Formulation of the Problem

In [5] it is proved that deciding if a graph has a UDG-realization is NP-hard. In Section 3 it is shown that the problem of finding such a UDG-realization is NP-hard by reduction. In this section, we give a formal description of the problem of finding a UDG-realization in terms of a quadratic feasibility problem.

First, the UDG-realization is defined by (similar to [5]):

Definition 5.1. (UDG-Realization) *The UDG-realization of a graph $G = (V, E)$ is a function $f : V \to \mathbb{R}^2$ that projects every v_i $(i = 1, \ldots, n)$ to a tuple (x_i, y_i) such that $(v_i, v_j) \in E$ if and only if $d(f(v_i), f(v_j)) \leq 1$ where d is the Euclidean distance between two points.*

In order to obtain a quadratic feasibility problem, the Euclidean distance $d = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ is replaced by its square $d^2 = (x_i - x_j)^2 + (y_i - y_j)^2$. (The right hand side stays 1 since $1^2 = 1$.)

Now, an exact formulation of the problem in terms of a feasibility problem with quadratic constraints can be constructed: Take an arbitrary graph $G = (V, E)$. For nodes v_i, v_j adjacent in the graph, define the constraint set

$$qcqp_{edges} := \{(x_i - x_j)^2 + (y_i - y_j)^2 \leq 1 \mid (v_i, v_j) \in E\},$$

for nodes v_i, v_j not adjacent in the graph, define the constraint set

$$qcqp_{nonedges} := \{(x_i - x_j)^2 + (y_i - y_j)^2 > 1 \mid (v_i, v_j) \notin E\}.$$

The union set

$$qcqp := qcqp_{edges} \cup qcqp_{nonedges} \tag{1}$$

yields constraints for a quadratic feasibility problem. These constraints are quadratic functions in the coordinates x_i, y_i of the nodes v_i.

If coordinate tuples (x_i, y_i) $(i = 1, \ldots, n)$ are found that satisfy Constraints (1), a UDG-realization has been found for the input graph $G = (V, E)$.

By Lemma 3.1 the feasibility problem of finding (x_i, y_i) $(i = 1, \ldots, n)$ to satisfy Constraints (1) is NP-hard. Since the exact problem can probably not be solved in polynomial time, the main interest is in approximation algorithms. In the following sections, ways are explored to find approximations with best possible guarantees on precision and with possibly polynomial running times.

6 Approximation Algorithms

6.1 Linear Programming Formulation of the Problem

In this section, we explore approximations in terms of linear programs. The approximations are found by replacing the quadratic constraints (1) from Section 5 by linear ones.

The objective is that the solutions of the LP formulation are as close as possible to the solutions of the exact problem formulation of Section 5. If two nodes v_i, v_j are not connected by an edge, their distance in the UDG-realization should be > 1. Equivalently, if two nodes v_i, v_j are connected by an edge, their distance in the UDG-realization should be ≤ 1.

We introduce some common norms and a measure of distance in the respective norms:

Definition 6.1. (ℓ_2-Norm) *The ℓ_2-norm (Euclidean distance) of a vector $x \in \mathbb{R}^n$ is defined as $\|x\|_2 = \sqrt{x_1^2 + \cdots + x_n^2}$.*

Definition 6.2. (ℓ_1-Norm) *The ℓ_1-norm (Manhattan distance) of a vector $x \in \mathbb{R}^n$ is defined as $\|x\|_1 = |x_1| + \cdots + |x_n|$.*

Definition 6.3. (ℓ_∞-Norm) *The ℓ_∞-norm of a vector $x \in \mathbb{R}^n$ is defined as $\|x\|_\infty = \max_i |x_i|$.*

Definition 6.4. (Distance $d(x,y)_\sigma$) *Denote by $d(x,y)_\sigma$ the distance between two elements $x, y \in \mathbb{R}^n$ in the norm ℓ_σ: $d(x,y)_\sigma = \|x - y\|_\sigma$ for $\sigma \in \{1, 2, \infty\}$.*

The feasibility problem can be formulated as follows: For nodes v_i, v_j adjacent in the graph, define the set

$$lp_{edges} := \{d(v_i, v_j)_\infty \leq 1 \mid (v_i, v_j) \in E\}. \tag{2}$$

These constraints make sure that the length of any horizontal or vertical line does not exceed 1. Add the constraints

$$lp_{edges} := lp_{edges} \cup \{d(v_i, v_j)_1 \leq \sqrt{2} \mid (v_i, v_j) \in E\}. \tag{3}$$

Both types of constraints are chosen such that a line of length 1 may still be realized at any slope while the maximum length of a line satisfying Constraints (2) and (3) exceeds 1 by as little as possible (the maximum length of a line satisfying the constraints is given by $\sqrt{2}\sqrt{(2-\sqrt{2})} \approx 1.082$, see also Figure 7).

For v_i, v_j not adjacent in the graph, define the set

$$lp_{nonedges} := \{d(v_i, v_j)_1 > 1 \mid (v_i, v_j) \notin E\} \tag{4}$$

These constraints are chosen such that a line of length > 1 may still be realized at any slope while the minimum length of a line satisfying Constraints (4) is smaller than 1 by as little as possible (the minimum length of a line satisfying the constraints is $> 1/2\sqrt{2} \approx 0.707$. See also Figure 8).

Then,

$$lp := lp_{edges} \cup lp_{nonedges} \tag{5}$$

yields constraints for a feasibility problem.

Using the notion of the Quasi-UDG (see Definition 4.1), it can be seen that any solution satisfying Constraints (5) constitutes a Quasi-UDG with parameter $d \approx 0.653$ which is below $1/\sqrt{2} \approx 0.707$ only by a small amount.

Theorem 6.1. *Every set of coordinates* (x_i, y_i) $(i = 1, \ldots, n)$ *satisfying Constraints* (5) *realizes the input graph* $G = (V, E)$ *as a Quasi-UDG with parameter* $d = 1/(2\sqrt{2 - \sqrt{2}}) \approx 0.653$.

Proof. Consider the value $d = 1/2\sqrt{2} \approx 0.707$. Clearly, $d \in [0, 1]$. The distance between every two nodes $(u, v) \notin E$ is $> d$. This implies that all nodes (u, v) with distance $\leq d$ are $\in E$. Thus the coordinates satisfy the first condition in the Quasi-UDG definition: $(u, v) \in E \; if \; |uv| \leq d$.

Edges are realized in the interval $[a, b]$, $a = 0$, $b = \sqrt{2}\sqrt{(2 - \sqrt{2})}$. The largest distance between two nodes $(u, v) \in E$ is b. Since $b > 1$, divide all

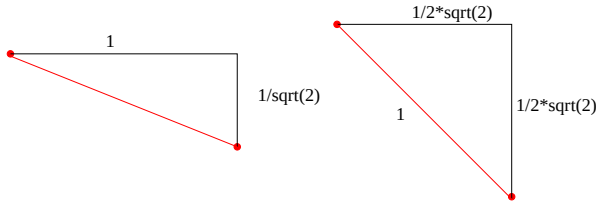

Figure 7: On the left hand side, a configuration is shown where the maximum distance of $\sqrt{2}\sqrt{(2 - \sqrt{2})} \approx 1.082$ for two connected nodes is reached. (There exist other, symmetric configurations, of course). On the right hand side, the well balanced configuration with distance exactly 1 is shown.

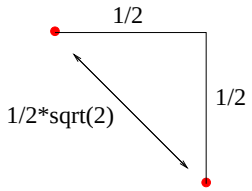

Figure 8: The well balanced configuration with distance $1/2\sqrt{2} \approx 0.707$ which cannot be reached by two unconnected nodes.

coordinates by b. Then,

$$\begin{aligned} a' &= a/b = 0, \\ b' &= b/b = 1, \\ d' &= d/b = 1/2\sqrt{2}/(\sqrt{2}\sqrt{2-\sqrt{2}}) = 1/(2\sqrt{2-\sqrt{2}}) \approx 0.653. \end{aligned}$$

Edges are now realized in the interval $[0, 1]$. Thus the coordinates satisfy the second Quasi-UDG condition: $(u, v) \notin E \; if \; |uv| > 1$. Notice that also $d' \in [0, 1]$. □

In order to use them in an LP formulation, Constraints (2), (3) and (4) have to be reformulated as linear constraints. Constraints (2) and (3) are reformulated as follows (the three lemmas are given without proof):

Lemma 6.2. *A constraint $max_{i=1,\ldots,n}\ a_i \leq c$ may be equivalently expressed by $(a_1 \leq c) \wedge (a_2 \leq c) \wedge \cdots \wedge (a_n \leq c)$*

Lemma 6.3. *A constraint $|x| \leq c$ may be equivalently expressed by two constraints $x \leq c \wedge -x \leq c$.*

Lemma 6.4. *A constraint $|x| + |y| \leq c$ may be equivalently expressed by four constraints $x + y \leq c \wedge x - y \leq c \wedge -x + y \leq c \wedge -x - y \leq c$.*

A single constraint

$$d(v_i, v_j)_\infty = \max\{|x_i - x_j|, |y_i - y_j|\} \leq 1$$

from Constraints (2) may be expressed by two constraints using Lemma 6.2:

$$|x_i - x_j| \leq 1 \wedge |y_i - y_j| \leq 1.$$

Further, these may be expressed by four constraints using Lemma 6.3:

$$x_i - x_j \leq 1 \wedge x_j - x_i \leq 1 \wedge y_i - y_j \leq 1 \wedge y_j - y_i \leq 1.$$

Similarly, a single constraint

$$d(v_i, v_j)_1 = |x_i - x_j| + |y_i - y_j| \leq \sqrt{2}$$

from Constraints (3) may be expressed by four constraints using Lemma 6.4:

$$\begin{aligned} x_i - x_j + y_i - y_j \leq \sqrt{2} \wedge x_i - x_j + y_j - y_i \leq \sqrt{2} \\ \wedge\, x_j - x_i + y_i - y_j \leq \sqrt{2} \wedge x_j - x_i + y_j - y_i \leq \sqrt{2}. \end{aligned}$$

The definition of the ℓ_1-norm involves absolute values. Absolute value expressions can not necessarily be expressed by linear constraints. Although it works for Constraints (2) and (3), it is impossible two model Constraints (4) with linear constraints.

Theorem 6.5. *Constraints* (4) *of the form* $d(v_i, v_j)_1 \geq 1$ *cannot be expressed by linear constraints.*

Proof. The condition $d(v_i, v_j)_1 \geq 1$ is equal to $|x_i - x_j| + |y_i - y_j| \geq 1$. Introduce new variables $dx_{ij} = x_i - x_j$ and $dy_{ij} = y_i - y_j$. The condition becomes $|dx_{ij}| + |dy_{ij}| \geq 1$. This constraint is equivalent to the constraint $|x| + |y| \geq 1$ which is illustrated in Figure 9. The feasible region contains a hole where $|x| + |y| < 1$. A line can be drawn starting in the feasible region, crossing the infeasible hole and ending in the feasible region again. In a convex set however, every line starting and ending in the set only contains points also belonging to the set (by Definition 2.1). This proves that the feasible region of $|x| + |y| \geq 1$ is not convex. However, the feasible region of every set of linear constraints is convex (by Lemma 2.1). Thus the feasible region of $d(v_i, v_j)_1 \geq 1$ cannot be expressed by linear constraints. □

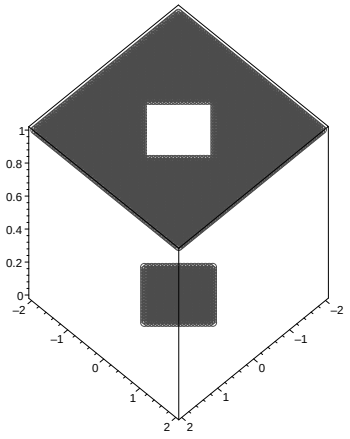

Figure 9: A plot of the function $f(x, y) = 1$ if $|x| + |y| \geq 1$ else 0. Observe the hole where $|x| + |y| < 1$ in the middle of a region where $|x| + |y| > 1$.

6.2 Introducing an Objective Function

Since Constraints (4) cannot be modeled by linear constraints, the constraints are replaced by a linear objective function which is supposed to stretch the graph in several directions in order to pull nodes with no edges between them away from each other. The stretching is done as follows: Choose several pairs of nodes, fix directions between them and maximize the distances between the nodes along these directions. Three nodes a, b and c are chosen and two directions are fixed: the x- and the y-axis.

The algorithm is formulated as follows:

1. Find the nodes a, b with the greatest distance in the graph (e.g. by running Floyd's all pairs shortest path algorithm and taking the nodes with the biggest value).

2. Find the node m which lies exactly in the middle of the path (if the number of nodes on the path is even i.e. there are two middle nodes, take the one that is closer to b).

3. Find a third node c for which the following function is maximum: $(\overline{ac} + \overline{cb})/|\overline{ac} - \overline{cb}|$ where $\overline{ac}$ denotes the length of the shortest path between the nodes a and c. Note the purpose of maximizing $(\overline{ac} + \overline{cb})/|\overline{ac} - \overline{cb}|$ over c: The node c should be as far away as possible from both a and b and, as a heuristic, c should be similarly far away from a and b. Thus the term $|\overline{ac} - \overline{cb}|$ penalizes big differences in the distances $\overline{ac}$ and $\overline{cb}$.

4. Add constraints to set the y-coordinates of the nodes m, a and b to 0 to make sure these nodes are realized on the x-axis.

5. If $m \neq c$, add constraints to set the x-coordinate of m to be equal to the x-coordinate of c to make sure the nodes m and c are realized on an axis perpendicular to the x-axis and parallel to the y-axis.

6. Set the objective function to the following: $f := \max x_b - x_a + y_c$ where x_b denotes the x-coordinate of b. The term y_c is only present for $m \neq c$.

Solve the resulting linear program by obtimizing the objective function $f := \max x_b - x_a + y_c$ subject to Constraints (3) together with the additional constraints for the nodes m, a, b and c introduced above.

For examples on how this performs in practice, see Figures 10 and 11. Figure 10 shows results of the algorithm run on very small graphs. Figure 11 shows that if nodes are attached to the main paths ab and mc, these nodes are collapsed to one single point by the algorithm. The algorithm only stretches nodes along the main paths. It is only suitable for specific classes of graphs e.g. the *path* P_n: $V = \{0, 1, \ldots, n\}$, $E = \{\{i-1, i\} : \ i = 1, 2, \ldots, n\}$. (Consider the path P_2 in Figure 10 for example: the graph is stretched between the two outer nodes a and b.)

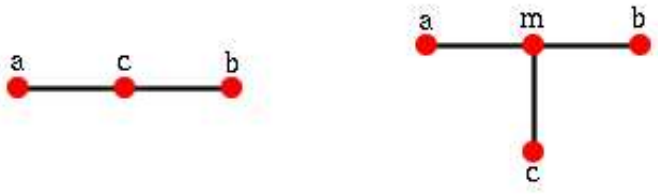

Figure 10: On the left hand side, the path P_2 is shown. To the right, the minimum graph where all nodes m, a, b and c are present and $m \neq c$.

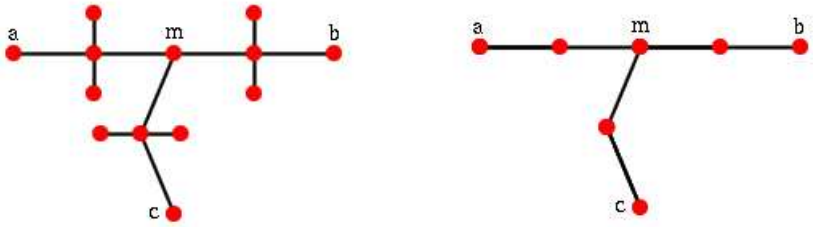

Figure 11: To the left, the graph including small structures attached to the main axes is drawn. To the right, the actual output of the algorithm: The small structures are collapsed to one single point.

6.3 Quadratic Programming Formulation of the Problem

The approach in this section involves a quadratic objective function. For adjacent nodes, we take Constraints (3). We drop Constraints (4) for non-adjacent nodes and add the following quadratic objective function instead:

$$f = \sum_{(v_i,v_j)\notin E} d_2(v_i, v_j)$$

This is the sum of the distances of non-adjacent nodes. The purpose of this objective function is to maximize the distance between non-adjacent nodes, thus stretching the graph similar to the approach in Subsection 6.2 involving a linear objective function.

The optimization problem formulation yields:

$$\begin{aligned} \max \quad & f(x_1, \ldots, x_n) \\ \text{subject to} \quad & lp_{edges} \end{aligned}$$

Since we want a convex optimization problem (which can be solved quickly) and this is a maximization problem, f is supposed to be concave. It turns out, however, that the objective function f as formulated above is not a concave function.

Theorem 6.6. *The function f is not concave.*

Proof. By contradiction. We take a concrete graph $G = (V, E)$. We assume that the function f is concave for this graph. We show how this leads to a contradiction.

Consider the graph $G = (V, E)$ with $V = \{v_1, v_2\}$, $f(v_1) = \{x_1, y_1\}$, $f(v_2) = \{x_2, y_2\}$ and $|E| = 0$. The graph consists of two nodes with no edge. Then,

$$f_1 = (x_1 - x_2)^2 + (y_1 - y_2)^2.$$

W.l.o.g., we set $x_1 = y_1 = y_2 = 0$ (any other configuration may be realized by rotation and translation). Then $f_1 = -x_2^2$. Since f_1 is concave, $g_1 = -f_1$ is convex. Applying the definition of convexity to g_1 yields

$$\begin{aligned} & g_1(\lambda x_1 + (1-\lambda)x_2) \\ & \quad = \lambda^2 x_1^2 + 2\lambda^2 x_1 x_2 - 2\lambda x_1 x_2 \\ & \qquad - x_2^2\lambda^2 + 2x_2^2\lambda - x_2^2 \\ & \quad \leq \lambda g_1(x_1) + (1-\lambda) g_1(x_2) \\ & \quad = \lambda x_1^2 + x_2^2\lambda - x_2^2. \end{aligned} \tag{6}$$

Term cancellation and subtraction of the right hand side yields

$$\begin{aligned} & (\lambda - \lambda^2)(x_1^2 - 2x_1x_2 + x_2^2) \\ & \quad = x_1^2 - 2x_1x_2 + x_2^2 \\ & \quad = (x_1 - x_2)^2 \\ & \quad \leq 0. \end{aligned} \tag{7}$$

The division by $(\lambda - \lambda^2)$ is allowed for $0 < \lambda < 1$. Setting $\lambda = 1$ implies $f(x_1) \leq f(x_1)$. Similarly, $\lambda = 0$ implies $f(x_2) \leq f(x_2)$. For $0 \leq \lambda \leq 1$ we have that Equation (7) is only satisfied if $x_1 = x_2$ or in the cases where both points x_1 and x_2 correspond to one endpoint, but not for all values of x_1 and x_2. The function g_1 is not convex. Contradiction. □

Practical tests have been conducted with simple graphs based on the QP formulation in this section. The optimization tools used were the MOSEK Optimization Tools (see [11]) and LOQO (see [12]). Both were run with their interior-point optimizers turned on. MOSEK refused to solve the problem because of its non-convexity. The solution output by LOQO was unusable.

6.4 Semidefinite Programming Formulation of the Problem

In [10], semidefinite programs are used as convex relaxations of NP-hard quadratic optimization problems. Since the relaxations are for quadratic problems, we are able to apply the technique to our exact problem formulation in Section 5.

In a semidefinite program (SDP) we minimize a linear function of a variable $x \in \mathbb{R}^m$ subject to a matrix inequality:

$$\begin{aligned} \text{min} \quad & c^T x \\ \text{subject to} \quad & F(x) \geq 0 \\ \text{where} \quad & F(x) = F_0 + \sum_{i=0}^{m} x_i F_i. \end{aligned}$$

The problem data are the vector $c \in \mathbb{R}^m$ and $m + 1$ symmetric matrices $F_0, \ldots, F_m \in \mathbb{R}^{n \times n}$. The inequality sign in $F(x) > 0$ means that $F(x)$ is positive semidefinite, i.e., $z^T F(x) z \geq 0$ for all $z \in \mathbb{R}$. We call the inequality $F(x) \geq 0$ a linear matrix inequality (LMI).

Semidefinite programs can be regarded as an extension of linear programming where the componentwise inequalities between vectors are replaced by matrix inequalities. Most interior-point methods for linear programming have been generalized to semidefinite programs. As in linear programming, these methods have polynomial worst-case complexity.

Consider the quadratic optimization problem

$$\begin{aligned} \text{min} \quad & f_0(x) \\ \text{subject to} \quad & f_i(x) \leq 0 \\ \text{where} \quad & f_i(x) = x^T A_i x + 2 b_i^T x + c_i \\ & i = 1, \ldots, L. \end{aligned}$$

The relaxation looks like

$$\begin{array}{rl} \min & \mathbf{Tr}XA_0 + 2b_0^T x + c_0 \\ \text{subject to} & \mathbf{Tr}XA_i + 2b_i^T x + c_i \leq 0 \\ & \begin{bmatrix} X & x \\ x^T & 1 \end{bmatrix} \geq 0 \end{array}$$

$i = 1, \ldots, L$ where the variables are $X = X^T \in \mathbb{R}^{k \times k}$, $x \in \mathbb{R}^k$ and $\mathbf{Tr}X$ is the trace of the matrix X.
Note that the constraint

$$\begin{bmatrix} X & x \\ x^T & 1 \end{bmatrix} \geq 0$$

is equivalent to $X \geq xx^T$.

The target of this relaxation is to find a lower bound for the objective value. Since our problem is a pure feasibility problem, we are not interested in the value of any objective function. Instead, our focus is on the feasible region. Does the relaxation enlarge the feasible region of our original problem in such a way that feasible vectors are reasonable approximations to our problem? Or does the relaxation admit feasible vectors which are not useful? The latter is the case: We show that the solution $x = 0$ is a feasible solution to the SDP-relaxation of our problem for a concrete graph G: Two nodes that are not connected in G are allowed to have distance 0. This is not desirable: The distance between two unconnected nodes should be > 1 for UDGs and $> d$ for Quasi-UDGs. Thus the SDP-relaxation should not be considered for this problem.

Theorem 6.7. *The vector $x = 0$ is a feasible solution to the SDP-relaxation.*

Proof. Let the graph be $G = (V, E)$ with $V = \{v_1, v_2\}$, $f(v_1) = (x_1, y_1)$, $f(v_2) = (x_2, y_2)$ and $\|E\| = 0$. W.l.o.g, we set $x_1 = y_1 = y_2 = 0$.

The quadratic optimization problem looks like

$$\begin{array}{rl} \min & f_0(x) \\ \text{subject to} & f_1(x) = TrXA_1 + 2b_1^T x + c_1 \end{array}$$

where $f_0(x)$ is arbitrary since we look at the feasibility problem.

Setting $A_1 = -1, b_1 = 0, c_1 = 1$ we get $f_1(x) = -x^2 + 1 \leq 0$.

The relaxation is

$$\begin{array}{rl} \min & f_0(x) \\ \text{subject to} & f_1(x) = -X + 1 \leq 0 \\ & \begin{bmatrix} X & x \\ x^T & 1 \end{bmatrix} \geq 0 \end{array} \tag{8}$$

The following lemma is given without proof:

Lemma 6.8. *A matrix is positive definite if the determinants associated with all upper-left submatrices are positive.*

The conditions for the upper-left submatrices are

$$\begin{aligned} \det(X) &= X \\ &\geq 0 \qquad (9) \\ \det(\begin{bmatrix} X & x \\ x^T & 1 \end{bmatrix}) &= X - x^2 \\ &\geq 0 \qquad (10) \end{aligned}$$

Condition (9) is true because of (8). We require $X > x^2$. Choose $X = x^2 + 1$. Then Condition (10) holds for an arbitrary x. This means there exist configurations where the trivial solution $x = 0$ satisfies the constraints. $X = 1, x = 0$ is a particular example. □

7 Conclusion

What is the benefit of an an exact solution to a problem if this solution takes forever to compute? After discovering that a certain problem is NP-hard, the focus can only be on the development of approximation algorithms: We showed that finding a unit disk graph realization given a graph $G = (V, E)$ is NP-hard. We gave an exact formulation of the problem in terms of a quadratic feasibility problem. We developed an LP-based approximation algorithm which is fast and works well on specific classes of graphs. Finally, we proved non-convexity of a QP-based approximation algorithm and showed that the SDP-relaxation of the exact formulation does not provide reasonable approximations.

8 Acknowledgments

I would like to thank Fabian Kuhn and Roger Wattenhofer who guided me through this work for their support.

Additional thanks go to Micha Bröker and Thomas Knecht for fruitful discussions on the subject.

References

[1] Fabian Kuhn, Roger Wattenhofer, and Aaron Zollinger. Ad-Hoc Networks Beyond Unit Disk Graphs. The final version will appear in the Proceedings of the 1st ACM DIALM-POMC Joint Workshop on Foundations of Mobile Computing (DIALM-POMC), San Diego, California, USA, September 2003.

[2] Fabian Kuhn, Roger Wattenhofer, Yan Zhang, and Aaron Zollinger. Geometric Ad-Hoc Routing: Of Theory and Practice. In the Proceedings of the 22nd ACM Symposium on the Principles of Distributed Computing (PODC), Boston, Massachusetts, USA, July 2003.

[3] Fabian Kuhn, Roger Wattenhofer, and Aaron Zollinger. Worst-Case Optimal and Average-Case Efficient Geometric Ad-Hoc Routing. In the Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC), Annapolis, Maryland, USA, June 2003.

[4] Fabian Kuhn, Roger Wattenhofer, and Aaron Zollinger. Asymptotically Optimal Geometric Mobile Ad-Hoc Routing. In the Proceedings of the 6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM), Atlanta, Georgia, September 2002.

[5] Heinz Breu and David G. Kirkpatrick. Unit Disk Graph Recognition is NP-Hard. Technical Report 93-27, University of British Columbia, Department of Computer Science, May 1993.

[6] Heinz Breu and David G. Kirkpatrick. Unit Disk Graph Recognition is NP-Hard. *Computational Geometry*, 9:3-24, January 1998.

[7] Theodor Ellinger, Günter Beuermann und Rainer Leisten. Operations Research: Eine Einführung. 5. durchges. Aufl. Berlin; Heidelberg; New York; Barcelona; Hongkong; London; Mailand; Paris; Singapur; Tokio: Springer, 2001.

[8] Dieter Jungnickel. Optimierungsmethoden: eine Einführung. Berlin; Heidelberg; New York; Barcelona; Hongkong; London; Mailand; Paris; Singapur; Tokio; Springer, 1999.

[9] Steven Boyd and Lieven Vandenberghe. Convex Optimization. URL: `http://www.stanford.edu/~boyd/cvxbook.html`, December 2002.

[10] Steven Boyd and Lieven Vandenberghe. Semidefinite programming relaxations of non-convex problems in control and combinatorial optimization. In Communications, Computation, Control and Signal Processing, 1997.

[11] The MOSEK optimization tools version 2.0 (Build 20). User's manual and reference. URL: `http://www.mosek.com`.

[12] Robert J. Vanderbei. LOQO Users Manual - Version 4.05. Operations Research and Financial Engineering Technical Report, Princeton University, Princeton, New Jersey, October 2000.