Minimizing Potential Energy in Constrained Physical Systems and Applications to Graph Drawing

Bachelor Thesis of

Christian Schnorr

At the Department of Informatics
Institute of Theoretical Computer Science

Reviewers:	Prof. Dr. Dorothea Wagner
	Prof. Dr. Henning Meyerhenke
Advisor:	Dr. Tamara Mchedlidze

Time Period: December 1st, 2016 – March 30th, 2017

KIT – University of the State of Baden-Wuerttemberg and National Laboratory of the Helmholtz Association www.kit.edu

Acknowledgements

Throughout multiple years and lectures, Prof. Dr. Dorothea Wagner got me obsessed with theoretical computer science and graph theory in particular. Thank you for introducing me to such a fascinating field, and for now giving me the opportunity to write my bachelor thesis at your institute.

I would also like to thank my advisor, Dr. Tamara Mchedlidze, for her great support in the form of many suggestions and hours of helpful discussions; and especially for responding to my late-night emails when I was in desperate need of some input.

My thanks also go to all of my proofreaders for helping me make this thesis the best it could be. Specifically, I would like to thank Florian Heininger for many hours of physics-related discussions, and Johannes Hubert for always having a sympathetic ear.

Last but by no means least, I would like to express my sincere gratitude to my parents and grandparents for their unfailing support and encouragement, despite my own limited devotion to staying in touch at times.

I am forever grateful for all of your support.

Statement of Authorship

I hereby declare that this document has been composed by myself and describes my own work, unless otherwise acknowledged in the text.

Karlsruhe, March 30th, 2017

Abstract

Over time, physical systems converge to stable equilibrium states in which their potential energy is at a local minimum. For systems that are not subject to constraints, force-directed algorithms can be used to find equilibrium positions. However, many systems are subject to constraints, restricting the movement of particles in the system. We propose an alternative method to minimize potential energy and thereby find stable equilibrium positions, that can be used for constrained systems.

We then demonstrate how this can be applied in the field of graph drawing. Here the constraints appear in the form of multiple incident edges being drawn using a single circular arc.

Deutsche Zusammenfassung

Physikalische Systeme konvergieren mit der Zeit gegen einen stabilen Gleichgewichtszustand, in dem die potentielle Energie des Systems ein lokales Minimum annimmt. In Systemen, die keinen Zwangsbedingungen unterworfen sind, können sogenannte *Force-Directed Algorithmen* verwendet werden, um Gleichgewichtszustände zu finden. Viele Systeme sind jedoch Zwangsbedingungen unterworfen, die die Bewegungsfreiheit der Partikel im System einschränken. Wir stellen eine alternative Methode zur Minimierung der potentiellen Energie vor, die auch für solche Systeme benutzt werden kann.

Wir zeigen anschließend, wie diese Methode im Bereich des Graphenzeichnens angewendet werden kann. Die Zwangsbedingungen bestehen hierbei darin, dass mehrere inzidente Kanten durch einen einzigen Kreisbogen gezeichnet werden sollen.

Contents

1. Introduction

In graph theory, a graph is an object consisting of a set of elements, called *vertices*, and their pairwise relations, called *edges*, that are expressed as (unordered) pairs of vertices. Graphs are used heavily for modeling routing problems, representing social networks, chip design, and many other fields.

Visualizing graphs is a fundamental aspect in the field of graph drawing and information visualization. A graph's vertices are typically drawn as small circles, with its edges being drawn as curves between their endpoints. When drawing graphs, one of the main goals is making them easy to grasp.

A popular family of algorithms for drawing graphs are so-called force-directed algorithms. In a force-directed algorithm, the graph is regarded as a particle system in which the vertices are particles, and several forces are acting on said particles. One defines these forces such that they act to bring the system into a stable equilibrium position in which its implicitly defined potential energy is at a local minimum and the resulting drawing has some desired features. Typically, one wants adjacent vertices to be close together, with non-adjacent vertices being further apart from each other. The resulting drawings of graphs are generally visually appealing and easy to grasp [Kob13].

1.1 Motivation

In traditional force-directed algorithms, the only features desired in equilibrium are that adjacent vertices are close together and that non-adjacent vertices are further apart from each other. Depending on what one wants to achieve when drawing a graph, one may have additional features — or different features altogether — that make a good drawing. The challenge then is to find appropriate restoring forces that act to put the system into a state of equilibrium that exhibits said features. Some features in graph drawings may be so indispensable that they pose hard constraints that have to be satisfied for the drawing to be acceptable. These constraints effectively restrict the vertices' movement, making traditional force-directed algorithms (in which vertices can move freely in both dimensions) inapplicable for finding a local energy minimum.

Schulz [Sch15] proposed that graph drawings with low visual complexity, i. e. drawings with few geometric entities, are easy to perceive by the viewer. Instead of drawing vertices and edges as their own geometric entity, Schulz aimed to draw multiple incident edges using a single geometric entity; namely, a circular arc. Considering there may not exist a circular arc through four or more arbitrarily placed vertices, this form of hard constraint reduces the number of degrees of freedom in the system.

This thesis serves to discuss an approach to minimizing the potential energy in systems whose constraints reduce its number of degrees of freedom. We will also demonstrate how this approach can be applied to creating drawings of graphs in which circular arcs are used to draw multiple incident edges.

1.2 Related Work

In his research, Schulz dealt with the theoretical lower and upper bounds of the number of required entities and provided an algorithm satisfying said bounds only for very specific classes of graphs [Sch15]. Similar research has been performed for drawing multiple incident edges using straight line segments [DESW07, DM14, IMS15].

Force-directed algorithms have already been used for drawing graphs with additional features desired in equilibrium.

Chernobelskiy et al. [CCG+11], for example, studied so-called Lombardi drawings and used a force-directed approach to optimize angular resolution by introducing a new set of forces. Sugiyama and Misue [SM95] studied force-directed algorithms in which forces exerted by a magnetic field are used to align edges in a certain direction. In both cases, the additional desired feature is dispensable, and therefore only poses a soft constraint affecting the quality of a drawing. Although nice to have, valid drawings can be produced even if these constraints are violated.

Bertault [Ber99] designed a force-directed algorithm called PrEd that preserves edge crossing from an initial layout. The additional desired feature here is indispensable and poses a hard constraint that has to be satisfied for the resulting drawing to be acceptable. Even though arbitrary displacements of the graph's vertices might render some of its drawings invalid, the additional constraint does not affect the number of degrees of freedom in the system.

2. Generalized Coordinates

A particle system's *configuration* encapsulates all positional information of said system, i. e. the position of every particle in the system. Similarly, the *configuration space* of a system models all of its possible configurations. For a system of n particles in two dimensions, we can write the particles' position vectors as

$$\vec{r}_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix} \in \mathbb{R}^2, \qquad i = 1, \dots, n.$$

If all particles can move freely in both dimensions, i. e. all $\vec{r}_i$ can be chosen arbitrarily, the system has $m = 2n$ *degrees of freedom.* In many scenarios, however, the system is subject to *constraints*, restricting the movement of particles by introducing dependencies between the particles' positions. If one were to pick arbitrary positions $\vec{r}_i$, one might end up violating some of these constraints. For now, let us assume constraints to depend only on the particles' positions and to be of the form

$$f(\vec{r}_1, \dots, \vec{r}_n) \overset{!}{\equiv} 0. \tag{2.1}$$

In theoretical physics, constraints of this form are known as *holonomic constraints.* The number of degrees of freedom in a constrained particle system is reduced by independent holonomic constraints [HT94]. Other forms of constraints may not necessarily decrease the number of degrees of freedom in the system.

Generalized coordinates are independent variables uniquely determining a constrained system's configuration that implicitly satisfy all of the system's holonomic constraints. Given a constrained particle system, one can find as many independent variables determining its configuration as there are degrees of freedom in said system. The number of generalized coordinates required to be able to describe all configurations satisfying the constraints is equal to the number of degrees of freedom in the system; however, their concrete choice is generally not uniquely determined [Mü16, Fli09].

Considering the generalized coordinates uniquely determine the system's configuration, each particle's position can be written as a function of the generalized coordinates:

$$\vec{r}_i = \vec{r}_i(q_1, \dots, q_m) \in \mathbb{R}^2$$

Note that for unconstrained particle systems, one could choose Cartesian coordinates $(x_i, y_i) \in \mathbb{R}^2$ as generalized coordinates, but is in no way forced to do so.

Example: Mechanical Pendulum

A mechanical pendulum is a weight hanging from a pivot point by a rigid rod of length l, allowing it to swing. Due to the rod being rigid, the weight's movement is restricted to a circle of radius l around the pivot point. When choosing the coordinate system such that the origin lies in the pivot point, we can express this restriction and dependency between the weight's x and y coordinates as

$$f(x, y) = x^2 + y^2 - l^2 \overset{!}{\equiv} 0. \tag{2.2}$$

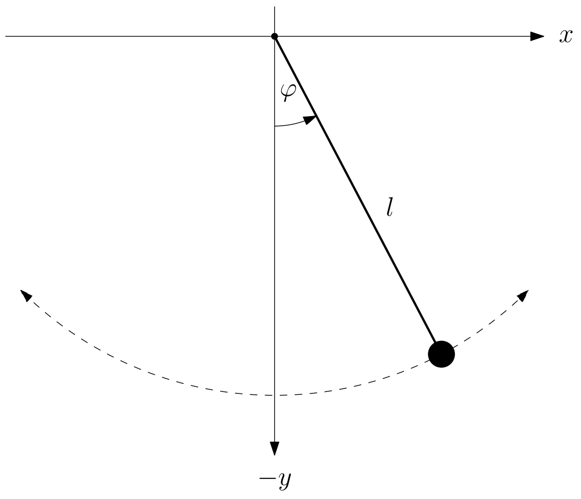

Figure 2.1: A mechanical pendulum.

Introducing a single generalized coordinate φ determining the angular position of the weight relative to the vertical axis allows us to get rid of both x and y, as they can now be written as a function of φ:

$$\vec{r} = \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} l \cdot \sin(\varphi) \\ -l \cdot \cos(\varphi) \end{pmatrix}$$

With only one generalized coordinate φ, we can now express exactly those configurations that satisfy Equation (2.2).

3. Minimizing Potential Energy

Potential energy is the capacity for doing work which results only from a system's configuration, i. e. its particles' positions. Recall that we can express all particle positions $\vec{r_i}$ as a function of the generalized coordinates; hence the potential energy U, too, is a function of the generalized coordinates. In the following, we shall assume the potential energy function to be continuous and locally differentiable.

In conservative systems, mechanical energy is conserved, which is equivalent to the work done by a force being independent of its trajectory. Therefore we can write the forces in terms of the potential energy as

$$\vec{F_i} := -\frac{\partial U}{\partial \vec{r_i}}. \tag{3.1}$$

The forces $\vec{F_i}$ are defined such that the potential energy U is equal to the work one must do *against* the forces to transfer an arbitrary constant reference configuration into the current configuration; hence the negative sign. As a result, the forces are always directed towards a lower energy level and therefore act to reduce the system's potential energy [Nav12].

A particle is said to be in *mechanical equilibrium* if there is no net force on said particle. Similarly, a particle system is in mechanical equilibrium if the net force on all its particles is zero [Ore79]. Using Equation (3.1), we find that a system is in equilibrium if and only if the gradient ∇U is zero for its current configuration. For (local) minima of the potential energy, the system is said to be in *stable equilibria.*

Recall that the forces $\vec{F_i}$ are defined to act to reduce the system's potential energy. If a system has been displaced from an equilibrium configuration, said forces consequently act to return the system to equilibrium and are therefore called *restoring forces.*

Oftentimes a system has multiple stable equilibria in which algorithms looking to minimize the system's potential energy can get stuck. It is a challenging problem to find a configuration in which the system's energy reaches a global minimum.

3.1 Force-Directed Algorithms

As the name suggests, force-directed algorithms define forces between pairs of particles, based on their relative positions, and use these forces to iteratively move the particles, attempting to minimize the system's implicitly defined potential energy [Kob13]. These forces are defined such that they are restoring forces, i. e. they point towards equilibrium.

Using the formula for physical work, we can define the potential energy of a configuration as the work one must do against the restoring forces to transfer an arbitrary constant reference configuration into the current configuration.

$$U := \int -\vec{F}_{\text{res}}(\vec{r})\, d\vec{r} \tag{3.2}$$

For a spring, one could choose the reference configuration such that the spring is relaxed; for two charged particles, one might choose a reference configuration in which they are infinitely far apart. Note that since we are dealing with conservative systems, the concrete trajectory between the endpoints does not matter.

Applying Equation (3.2) to each force allows us to calculate the system's total potential energy without ever explicitly defining it. Equation (3.2) also shows that giving in to a restoring force, i. e. moving a particle by an infinitesimal distance in the direction of the restoring force, decreases the implicitly defined potential energy. Note that there potentially are multiple forces acting on the same particle; therefore one must first calculate the net force acting on each particle as the (vector) sum of the individual restoring forces, and thereby find the direction in which the particles need to be moved in order to reduce the system's potential energy.

When displacing each particle by an infinitesimal distance in the direction of the net force acting on it, it is evident that local energy minima can not be overcome. Depending on the initial configuration, it may not be possible to reach a global energy minimum. Although infinitesimal displacements are not possible in practice, force-directed algorithms with larger displacements generally yield good results [Kob13].

3.1.1 Generalized Forces

Forces do not act on (generalized) coordinates; they act on the particles whose positions are determined by the generalized coordinates. Depending on the constraints the system is subject to, a movement in the direction of the restoring forces may or may not be possible. If it is, one must find the adjustment that needs to be made to the generalized coordinates that results in the desired change in particle positions. Considering the positions $\vec{r}_i$ are functions of the generalized coordinates $q := (q_1, \ldots, q_m)$, we can use the restoring forces to find the so-called *generalized forces* acting on the generalized coordinates [Fit11]:

$$\begin{align} Q_j &:= \sum_{i=1}^{n} \vec{F}_i \cdot \frac{\partial \vec{r}_i}{\partial q_j}, \qquad j = 1, \ldots, m \tag{3.3} \\ &\overset{(3.1)}{=} -\frac{\partial U}{\partial q_j} \tag{3.4} \end{align}$$

Note that the products $Q_j \cdot q_j$ always have the dimension of work. Hence the generalized forces do not necessarily have the dimension of force and instead depend on the dimensions of their corresponding generalized coordinates.

Feasibility

Both generalized coordinates and generalized forces are scalar quantities. The sign of a generalized force Q_j still indicates in which direction its corresponding generalized coordinate q_j needs to be adjusted for it to result in the desired change in particle positions, which in turn reduces the system's potential energy. Whether or not it is feasible to use the signs of the generalized forces as a hint to the direction in which to adjust the generalized coordinates very much depends on the complexity of the $\vec{r}_i(q_1, \ldots, q_m)$ and their partial derivatives with respect to the generalized coordinates q_j.

Note that if the system is not subject to any constraints and one uses Cartesian coordinates as the generalized coordinates, the generalized forces essentially become a resolution of the restoring forces into their x and y components, yielding the traditional force-directed algorithms for systems without constraints.

Forces in Equilibrium

In constrained systems, the constraints implicitly define so-called *constraining forces* which act perpendicularly to the allowed movement to keep all constraints satisfied [Fli09]. These forces need to be taken into account for the net force on all particles to be zero in equilibrium configurations.

Back in the example of a mechanical pendulum, the only exterior force is the gravitational force; and it acts on the pendulum regardless of its position. The net force on the weight only becomes zero in equilibrium when taking the implicit constraining force of the rigid rod into account.

3.2 Explicit Energy Function

Sometimes the restoring forces result in equilibria that do not quite exhibit the desired features, or it is not clear how to choose the restoring forces at all. By instead explicitly assigning each configuration a potential energy, one can easily specify which features are desirable in equilibrium, and which are not. One does not need to provide the restoring forces, i. e. a direction towards equilibrium — a generic optimization algorithm will figure that out.

Let us assume that all generalized coordinates q_j can be (reversibly) transformed to be real-valued. Then we can collect all generalized coordinates in a real-valued vector and write the potential energy function as

$$U' \colon \mathbb{R}^m \to \mathbb{R} \cup \{\infty\}.$$

For invalid configurations, e. g. if two charged particles coincide, we shall use an infinite potential energy instead of leaving the function undefined. The potential energy function U' can then be minimized without any background knowledge of the problem, such as the restoring forces in the system.

3.2.1 Derivative-based Optimization

Many optimization methods require information about partial derivatives of the function to be optimized. These include finding an extremum analytically, but also many numerical optimization techniques such as Newton's method, coordinate descent methods, and conjugate gradient methods. For small systems these methods may be an option; but for larger systems, it is generally infeasible to obtain accurate derivative information.

3.2.2 Derivative-free Optimization

Due to the lack of accurate derivative information, methods that only evaluate function values are often better-suited to minimize the potential energy in larger systems. We shall discuss a simple algorithm in this category in greater detail.

Hill Climbing

Hill climbing is a numeric optimization algorithm that iteratively improves the quality of its solution by adjusting one dimension at a time. In each iteration, the algorithm attempts to adjust a single dimension of its current state and accepts that change if and only if it results in an improvement in value space. This process is repeated until the maximum number of iterations has been performed, or until no further improvements can be found [RN10]. There are three major drawbacks of hill climbing:

1. Local Optima: The algorithm cannot escape a local optimum since adjustments are only accepted if they improve the function evaluation, and may therefore not reach a global optimum.

2. Ridges and Alleys: Considering the algorithm adjusts one dimension at a time, the search tends to zig-zag in non-axis-aligned ridges or alleys, taking an unreasonable amount of time to ascend the ridge or descend the alley.

3. Plateaux: A plateau is an area in which the value function is essentially flat. Depending on the concrete implementation, the algorithm will either not make any improvements at all, or conduct a random walk.

Popular variants of hill climbing include evaluating multiple neighboring states and continuing with the best, and adaptive step sizes for each dimension that change throughout the algorithm. Algorithm 3.1 shows a possible implementation of adaptive hill climbing.

Algorithm 3.1: Adaptive Hill Climbing for Minimization

```
Input: number of iterations $n$,
       value function $f\colon \mathbb{R}^m \to \mathbb{R}$,
       start configuration $\vec{x}_0 \in \mathbb{R}^m$
Output: $\vec{x}_n$ with $f(\vec{x}_n) \leq f(\vec{x}_0)$

acceleration $\leftarrow 1.25$
steps $\leftarrow (1, \ldots, 1)$

for $i \in 1 \ldots n$ do
    $\vec{x}_i \leftarrow \vec{x}_{i-1}$

    for $j \in 1 \ldots m$ do
        $\vec{x}_{i,j} \leftarrow \vec{x}_{i-1}$
        factor $\leftarrow$ acceleration$^{-1}$

        for $k \in -1 \ldots 1$ do
            $\vec{x}_{i,+j,k} \leftarrow \vec{x}_{i-1} + e_j \cdot \text{steps}_j \cdot \text{acceleration}^k$
            $\vec{x}_{i,-j,k} \leftarrow \vec{x}_{i-1} - e_j \cdot \text{steps}_j \cdot \text{acceleration}^k$

            if $f(\vec{x}_{i,k,+j}) < f(\vec{x}_{i,j})$ then
                $\vec{x}_{i,j} \leftarrow \vec{x}_{i,k,+j}$
                factor $\leftarrow$ acceleration$^k$

            if $f(\vec{x}_{i,k,-j}) < f(\vec{x}_{i,j})$ then
                $\vec{x}_{i,j} \leftarrow \vec{x}_{i,k,-j}$
                factor $\leftarrow$ acceleration$^k$

        steps$_j \leftarrow$ factor $\cdot$ steps$_j$

        if $f(\vec{x}_{i,j}) < f(\vec{x}_i)$ then
            $\vec{x}_i \leftarrow \vec{x}_{i,j}$

return $\vec{x}_n$
```

Considering hill climbing adjusts one dimension at a time, for it to converge to a (local) minimum, it should be started from a valid configuration, i. e. from one with finite potential energy. Unlike most randomized optimization algorithms, hill climbing has decent intermediate states, allowing for proper visualization of the optimization process.

4. Drawing Graphs with Circular Arcs

In the style of Schulz [Sch15], we shall now apply the concepts from Chapters 2 to 3 to create drawings of graphs with low visual complexity by drawing multiple incident edges using a single circular arc. The algorithms presented in this chapter allow us to create drawings of arbitrary connected and simple graphs. Potential edge crossings will be disregarded as they are unavoidable in nonplanar graphs and minimizing the number of edge crossings is generally $\mathcal{NP}$-hard.

Let $\Pi = \{P_1, \ldots, P_k\}$ be a set of edge-disjoint paths that may have vertices in common. We shall denote the entirety of vertices and edges of paths $P \in \Pi$ by $V(\Pi)$ and $E(\Pi)$, respectively:

$$\begin{aligned} V(\Pi) &\coloneqq V(P_1) \cup \ldots \cup V(P_k) \\ E(\Pi) &\coloneqq E(P_1) \uplus \ldots \uplus E(P_k) \end{aligned}$$

Given a graph $G = (V, E)$, a *drawing of G with circular arcs* is a drawing of G in which every edge $e \in E$ is drawn as a circular arc Γ_e, such that (i) no vertices coincide, (ii) no edges overlap, and (iii) no edge intersects vertices other than its endpoints. For the purpose of working with circular arcs only, we shall consider straight line segments to be circular arcs with infinite radius and complete circles to be circular arcs, too.

Similarly, given a set of edge-disjoint paths $\Pi = \{P_1, \ldots, P_k\}$, a *drawing of* Π *with circular arcs* is a drawing of the graph $G \coloneqq (V(\Pi), E(\Pi))$ with circular arcs, such that the edges of each path $P \in \Pi$ form a single circular arc Γ_P. Note that this does not violate above condition of every edge being drawn as a circular arc, since subdividing a circular arc yields other, though smaller, circular arcs. The fact that edges may not overlap implies that the order of vertices on Γ_P must be as indicated by P.

The definition of a drawing of Π with circular arcs — and therefore Π itself — implicitly defines the constraints a drawing of Π is subject to. Note that only the constraints of all vertices $v \in V(P)$ on a path $P \in \Pi$ lying on the same circular arc Γ_P can be expressed as holonomic constraints. The constraints of said vertices having the correct order on Γ_P, no vertices coinciding, no edges overlapping, and no edge intersecting vertices other than its endpoints can not be expressed as holonomic constraints. These constraints do not reduce the number of degrees of freedom in the drawing — they simply render some drawings invalid, but need to be accounted for nonetheless.

4.1 Existence of Drawings with Circular Arcs

For an arbitrary set of edge-disjoint paths $\Pi = \{P_1, \dots, P_k\}$ the existence of a drawing of Π with circular arcs is not guaranteed.

A circle — or circular arc for that matter — is uniquely determined by three distinct points. If two paths $P_i \neq P_j$ were to have three or more vertices in common, the circular arcs used to draw them would intersect in at least three points and would therefore overlap. However, in Theorem 1 we show that the resulting necessary condition of two paths $P_i \neq P_j$ having at most two vertices in common is not sufficient in guaranteeing the existence of a (valid) drawing of Π with circular arcs. We start by showing the following lemma:

Lemma 1. A drawing of Π with circular arcs in which edges touch in non-endpoints can be transformed into a drawing of Π with circular arcs in which said edges either cross or do not intersect at all.

Proof. When two edges touch in non-endpoints, then so do the paths they are part of and the circular arcs used to draw these paths. Adjusting the curvature of one of said circular arcs by an infinitesimal amount causes these circular arcs to either cross or to not intersect at all.

Considering this adjustment does not alter the ordering of vertices on said circular arc, it is only left to show that this adjustment can be performed such that the resulting circular arc does not overlap other circular arcs, or intersect vertices not part of the respective path. There's a finite number of vertices and paths in a drawing of Π with circular arcs; hence this adjustment can indeed be performed without violating any of the constraints imposed by Π. $\square$

Theorem 1. There exists a set of edge-disjoint paths Π in which any two paths have at most two vertices in common, i. e. $\forall i \neq j\colon\ |V(P_i) \cap V(P_j)| \leq 2$, for which no (valid) drawing with circular arcs exists.

Proof. An arrangement of pseudocircles is a finite list $\mathcal{C} = (c_1, \dots, c_n)$ of Jordan curves in the plane, such that (i) every two curves intersect in at most two points, and (ii) if two curves meet in a point, they cross in said point. Two arrangements are said to be *equivalent* if there exists a homeomorphism from the plane onto itself that transforms one of the arrangements into the other one. An arrangement of pseudocircles is said to be *circleable* if it is equivalent to an arrangement of proper circles [KM14].

Linhart and Ortner [LO05] showed that the arrangement of pseudocircles in Figure 4.1 is not circleable.

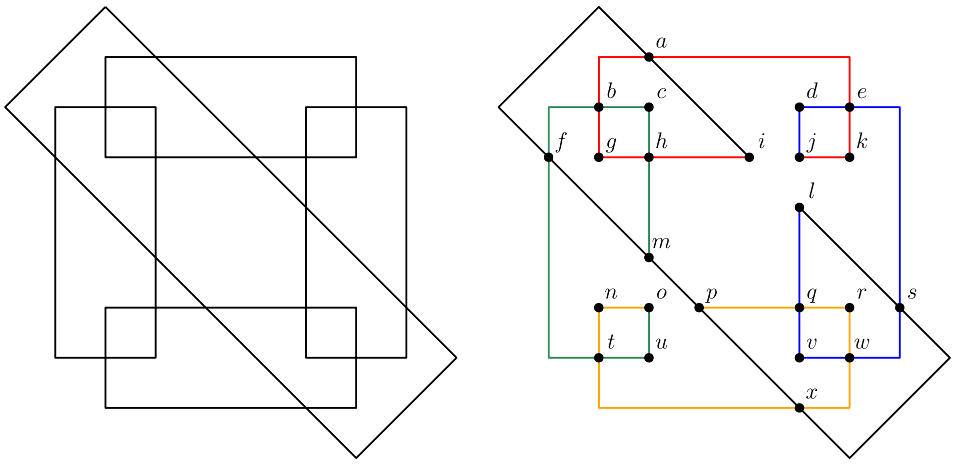

Figure 4.1: An arrangement of pseudocircles that is not circleable (left) and a derived arrangement of pseudo circular arcs (right).

Let us choose a set of edge-disjoint paths $\Pi = \{P_1, \dots, P_n\}$ such that each path represents one of the pseudocircles in Figure 4.1 and their shared vertices completely encode all of said pseudocircles' intersections with each other. A possible choice would be $P_1 = ihgbaekj$ (red), $P_2 = pqrwxtno$ (yellow), $P_3 = mhcbftuo$ (green), $P_4 = lqvwsedj$ (blue), and $P_5 = iafmpxsl$ (black).

Considering circular arcs are slices of complete circles and the arrangement of pseudocircles in Figure 4.1 is not circleable, a drawing of Π with circular arcs in which no two circular arcs touch in non-endpoints can not exist. Lemma 1 then shows that in fact, no drawing of Π with circular arcs can exist at all. □

Theorem 1 shows that the trivial condition of two paths having at most two vertices in common is not sufficient in guaranteeing the existence of a (valid) drawing of Π with circular arcs. Let us now be a little more restrictive and instead consider an ordered sequence of edge-disjoint paths $\Pi = (P_1, \ldots, P_k)$ in which none of a path P_i's internal vertices are contained in an earlier path, i. e. a path P_j with $j < i$:

$$V_{\text{int}}(P_i) \cap V(P_1 \cup \ldots \cup P_{i-1}) \overset{!}{=} \varnothing, \quad i = 2 \ldots k \tag{4.1}$$

Theorem 2. An ordered sequence of edge-disjoint paths $\Pi = (P_1, \ldots, P_k)$ fulfilling Equation (4.1) permits a drawing of Π with circular arcs.

Proof. We will provide a method to construct a drawing of Π with circular arcs by sequentially drawing the paths $P \in \Pi$ in their designated order.

Equation (4.1) ensures that, when drawing a path $P = (v_1, \ldots, v_n)$, its internal vertices $v_2, \ldots, v_{n-1}$ have not yet been drawn. Since the paths are edge-disjoint, its edges have not yet been drawn either. In case an endpoint of P has not yet been drawn, we can assign it an arbitrary position, as long as it is not on any of the circular arcs that have already been drawn.

Considering there's a finite number of vertices and edges, there exists a circular arc Γ_P between P's endpoints that neither intersects any of the vertices that have already been drawn (other than said endpoints), nor overlaps any circular arc that has already been drawn.

It is possible, however, for Γ_P to intersect other circular arcs. Drawing one of P's internal vertices on such an intersection would visually introduce new edges, rendering the drawing of Π invalid. Considering a pair of non-overlapping circular arcs intersects in at most two points, the number of points on Γ_P on which we can not draw P's internal vertices is finite, and there are infinitely many points left on which said vertices could be drawn without rendering the drawing invalid. Therefore we can draw P's internal vertices, in order, on the circular arc and thereby partition Γ_P into smaller circular arcs Γ_e representing the individual edges $e \in E(P)$. $\square$

We have seen that an ordered sequence of edge-disjoint paths $\Pi = (P_1, \ldots, P_k)$ fulfilling Equation (4.1) permits a (greedy) drawing of Π with circular arcs. Therefore we shall also refer to Π as a *greedily realizable sequence of paths.* Vertices $v \in V(\Pi)$ shall be said to be *laid out* by the first path, i. e. the path $P_i \in \Pi$ with the lowest index i, in which they occur. If v is internal to said path, it shall be called *constrained* (with respect to Π); otherwise, it shall be called *unconstrained* (with respect to Π). The constrained and unconstrained vertices form the sets $V_{\text{c}}(\Pi)$ and $V_{\text{u}}(\Pi)$, respectively. Obviously $V_{\text{c}}(\Pi) \cap V_{\text{u}}(\Pi) = \varnothing$ and $V_{\text{c}}(\Pi) \cup V_{\text{u}}(\Pi) = V(\Pi)$.

Note that there are definitely other properties guaranteeing the existence of a (valid) drawing of Π with circular arcs — possibly even less restrictive ones. We use this one because it suggests a straight-forward method to create a drawing, as illustrated in the proof of Theorem 2.

4.2 Graph Decomposition

In this section, we shall discuss a generic approach to decompose a connected and simple graph into a non-trivial greedily realizable sequence of simple paths. We restrain ourselves to connected graphs since paths can not contain isolated vertices. Besides, in a larger graph, the connected components can — and should — be drawn individually. The graph being simple allows it to be decomposed into simple paths, which will be a requirement in a later section.

Algorithm 4.1 greedily assembles the paths one after the other. The `vertices` and `edges` sets keep track of which vertices and edges have already been used and allow us to ensure that the assembled paths both are edge-disjoint and fulfill Equation (4.1).

Algorithm 4.1: Graph Decomposition

Input: Connected and simple graph G
Output: Greedily realizable sequence of simple paths Π, such that $V(\Pi) = V(G)$ and $E(\Pi) = E(G)$

```
paths ← ()
edges ← $\varnothing$
vertices ← $\varnothing$

foreach $u \in V(G)$ do
    foreach $e = \{u, v\} \in E(G)$ do
        ensure $e \notin$ edges else continue

        path ← $(e)$
        head ← $v$
        edges ← edges $\uplus \{e\}$
        vertices ← vertices $\cup \{u\}$

        append:
        while head $\notin$ vertices do
            vertices ← vertices $\uplus$ {head}

            foreach $f = \{$head$, w\} \in E(G)$ do
                ensure $f \notin$ edges else continue
                ensure $w \notin$ path.vertices else continue

                path ← path.append($f$)
                head ← $w$
                edges ← edges $\uplus \{f\}$

                continue append
            break
        paths ← paths.append(path)

return paths
```

Correctness

After the loop in line 6, all edges incident to u are guaranteed to appear in a path. Because this loop is executed for every vertex in the input graph G, all edges in the graph end up being used. By definition the input graph is connected; hence every vertex is incident to at least one edge, meaning that all vertices in the graph are used as well. Considering that edges are only ever used to assemble the working path if they have not been used before, we find that Π is a decomposition of the input graph, i. e. $V(\Pi) = V(G)$ and $E(\Pi) = E(G)$.

It remains to show that Π is a greedily realizable sequence of simple paths. Since the input graph does not contain loops, a single edge always is a simple path. Single-edged paths cannot possibly violate the requirements of a greedily realizable sequence of paths, as they do not have any internal vertices that could have appeared in an earlier path. When appending an edge to the working path, its current head would become an internal vertex. This operation is only performed if the current head can become an internal vertex without violating the requirements in Equation (4.1), i. e. if the current head has not been used beforehand. The additional check in line 20 ensures that edges are only appended to the working path if the resulting path would still be simple, i. e. if appending the edge would not form a cycle. Therefore Π is indeed a greedily realizable sequence of simple paths.

Runtime

The two outermost loops iterate over all vertices and their incident edges. Considering an edge is incident only to its endpoints, each edge is processed exactly twice here. When implemented using an adjacency list, these loop conditions can be checked in $\Theta(|V| + |E|)$.

The condition of the `append` loop in line 14 is checked $\Theta(|V| + |E|)$ times as a result of the containing loop being executed that many times; plus once whenever the algorithm jumps back to the `append` label after appending an edge to the working path, which happens $\mathcal{O}(|E|)$ times. Line 16 ensures that the loop's body, however, is executed at most once per vertex. Using the same argument as above, iterating over the incident edges of $\mathcal{O}(|V|)$ vertices can be done in $\mathcal{O}(|V| + |E|)$.

As a result, the entire algorithm can be executed in $\Theta(|V| + |E|)$ when implemented using an adjacency list.

Adaptation

The sole purpose of above algorithm is to show that it is relatively easy to decompose an input graph G into a non-trivial greedily realizable sequence of paths Π using a greedy algorithm. It does not intend to find a *good* decomposition — quality is a subjective measurement and very much depends on what one wants to achieve when drawing the graph.

Possible tweaks include the choice of vertices in line 5 or the choice of edges in line 6 and line 18. It may even be an option to not greedily append edges as long as possible, and instead start a new path earlier, especially in undirected graphs. In a directed graph, for example, it would make sense to keep track of the number of unused incoming edges for each vertex and pick a vertex with no unused incoming edges in line 5.

It may also be useful to let the user, who possibly has background knowledge about what the graph represents, provide some paths that semantically make sense to emphasize by drawing them as a single circular arc. The user-provided paths can then be completed to a valid decomposition Π of the input graph.

4.3 Circular Arc Geometry

A 3-tuple (P, Q, φ) uniquely determines a circular arc $\Gamma(P, Q, \varphi)$. Here $P, Q \in \mathbb{R}^2$ are its distinct endpoints, and $\varphi \in (-180°, 180°)$ is the (signed) angle from the chord PQ to the arc's tangent in P.

Drawing a circular arc with $\varphi > 0°$ starting in P requires a clockwise motion; hence those arcs shall be called *clockwise*. Similarly, arcs shall be called *counterclockwise* for $\varphi < 0°$. For $\varphi = 0°$ the arc degenerates into the straight line segment connecting P and Q. Figure 4.2 outlines important properties of a circular arc.

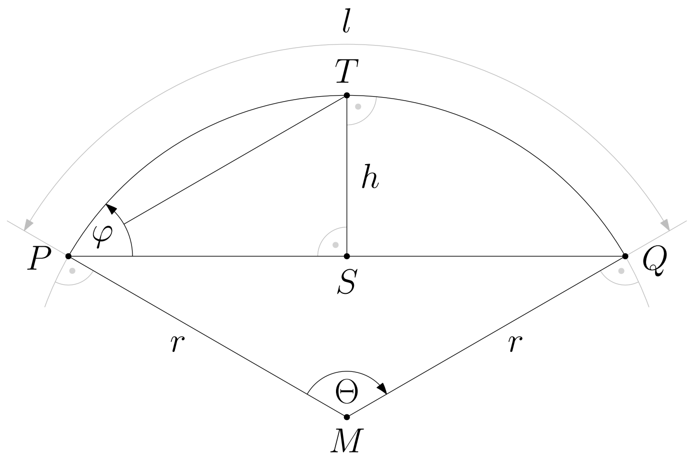

Figure 4.2: Geometry of a clockwise circular arc.

The *central angle* Θ is the (signed) angle which the arc subtends at the center of the circle. For clockwise arcs the central angle is negative; for counterclockwise arcs it is positive:

$$\Theta = -2\varphi$$

The *arc height* h is the (signed) perpendicular distance from the arc's midpoint T to the chord PQ connecting its endpoints. For clockwise arcs the arc height is positive; for counterclockwise arcs it is negative. When cutting Γ in half, the circular arc Γ' with chord PT has $\Theta' = \frac{\Theta}{2}$ and $\varphi' = \frac{\varphi}{2}$. Therefore we find $\angle SPT = \varphi - \frac{\varphi}{2}$, and we can write the arc height as

$$h \;=\; \overline{PS} \cdot \tan(\angle SPT) \;=\; \frac{1}{2}\overline{PQ} \cdot \tan\left(\frac{\varphi}{2}\right).$$

The *arc radius* r and *arc center* M are, respectively, the radius and center of the circle which the arc is a slice of. We find them to be

$$r(\varphi \neq 0°) = \left|\frac{\overline{PS}}{\sin(\varphi)}\right| = \left|\frac{\overline{PQ}}{2\sin(\varphi)}\right|$$

and

$$M(\varphi \neq 0°) = P + R(\varphi - 90°) \cdot \frac{\overrightarrow{PQ}}{2\sin(\varphi)},$$

where $R(\psi)$ is the matrix rotating a column vector by ψ in the counterclockwise direction. Note that for $\varphi = 0°$, both radius r and center M are undefined.

The *arc length* l is the distance between the arc's endpoints P and Q along the arc. It is a portion of the respective circle's circumference:

$$l(\varphi \neq 0^\circ) = \left| \frac{\Theta}{360^\circ} \cdot 2\pi r \right| = \left| \frac{\varphi}{180^\circ} \cdot \frac{\pi \cdot \overline{PQ}}{\sin(\varphi)} \right|$$

For $\varphi = 0^\circ$, above equation for arc length l is undefined. We shall continuously extend it to $\varphi = 0^\circ$ using the limit $\lim_{\varphi \to 0^\circ}$:

$$l(\varphi = 0^\circ) \coloneqq \lim_{\varphi \to 0^\circ} l(\varphi) = \overline{PQ}$$

This extension also matches our intuition of the arc degenerating into a straight line segment for $\varphi \to 0^\circ$.

4.4 Generalized Coordinates

Given a greedily realizable sequence of simple paths $\Pi = (P_1, \dots, P_k)$, we shall now choose generalized coordinates such that the holonomic constraints defined by Π are implicitly satisfied. The generalized coordinates introduced here require a circular arc's endpoints to differ and can therefore not be applied to paths whose endpoints coincide.

Besides its endpoints, a circular arc requires another generalized coordinate encoding its curvature for it to be uniquely determined. This coordinate must be able to encode the arc's direction, i. e. whether it is drawn clockwise or counterclockwise. Considering two circular arcs with the same angle φ, as introduced in the previous section, are guaranteed to be similar, we shall use the angle $\varphi_P \in (-180°, 180°)$ as a generalized coordinate encoding the curvature of the circular arc Γ_P.

Unconstrained vertices $v \in V_{\mathrm{u}}(\Pi)$ can move around arbitrarily and require two degrees of freedom. We shall use Cartesian coordinates $(x_v, y_v) \in \mathbb{R}^2$ to represent their positions.

In contrast, constrained vertices $v \in V_{\mathrm{c}}(\Pi)$ can not move around arbitrarily — they are each being laid out by a path $P(v)$ and are restricted to move along the circular arc $\Gamma_{P(v)}$, while also staying in the order indicated by $P(v)$. We only have one degree of freedom per vertex here: its relative progress along the arc. It can be encoded using a generalized coordinate $p_v \in (0, 1)$ specifying the proportion of the subarc connecting $P(v)$'s tail and v to the entire arc, as illustrated in Figure 4.3.

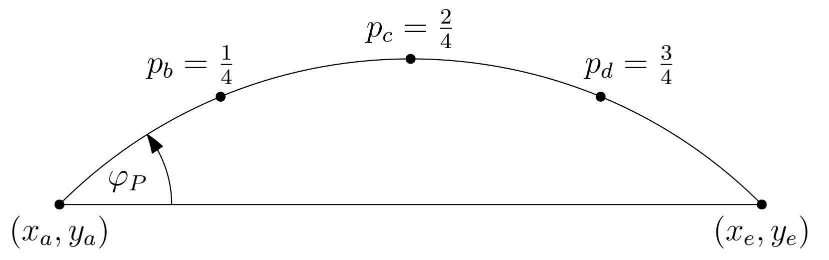

Figure 4.3: Generalized coordinates for a single path $P = abcde$ with equi-length edges.

Collectively, the $\Theta(|V| + |\Pi|)$ generalized coordinates uniquely determine all of the vertices' positions $\vec{r}_v$ and circular arcs Γ_P. We shall write them as a 4-tuple $q_\Pi = (x, y, \varphi, p)$, where

$$\begin{aligned} x\colon & V_\mathrm{u}(\Pi) \to \mathbb{R}, \\ y\colon & V_\mathrm{u}(\Pi) \to \mathbb{R}, \\ \varphi\colon & \quad\ \Pi \quad \to (-180^\circ, 180^\circ), \\ p\colon & V_\mathrm{c}(\Pi) \to (0, 1). \end{aligned}$$

The generalized coordinates chosen here satisfy all holonomic constraints defined by Π, i. e. all vertices $v \in V(P)$ on a path $P \in \Pi$ are guaranteed to lie on the same circular arc Γ_P.

Recall that the configurations that can be expressed only depend on the constraints implicitly satisfied by the choice of generalized coordinates, and that the number of independent generalized coordinates equals the number of degrees of freedom in the system. It is therefore not possible to use fewer generalized coordinates while satisfying the same constraints without also excluding other configurations.

4.5 Transformation of Generalized Coordinates

Although generalized coordinates q_Π are helpful as an internal representation that implicitly satisfies the holonomic constraints, we will need to retrieve vertices' positions $\vec{r}_v$ and circular arcs Γ_P to eventually produce a drawing of the graph $G := (V(\Pi), E(\Pi))$.

In theory, we can write both the position vectors $\vec{r}_v$ and circular arcs Γ_P as an explicit function of the generalized coordinates. However, these expressions get out of hand very quickly when paths are nested within one another, i. e. a path's internal vertex is an endpoint of another path. Instead, we propose the following algorithm to perform the transformation to position vectors sequentially:

Algorithm 4.2: Transformation of generalized coordinates

```
Input: Greedily realizable sequence of simple paths $\Pi$ and corresponding
       generalized coordinates $q_\Pi = (x, y, \varphi, p)$, i. e.
       $x\colon V_\mathrm{u}(\Pi) \to \mathbb{R}$,
       $y\colon V_\mathrm{u}(\Pi) \to \mathbb{R}$,
       $\varphi\colon \Pi \to (-180^\circ, 180^\circ)$,
       $p\colon V_\mathrm{c}(\Pi) \to (0, 1)$
Output: Positions $\vec{r}(v)$ and arcs $\Gamma(P)$ for all vertices and paths in $\Pi$
1  Initialize $\vec{r}\colon V(\Pi) \to \mathbb{R}^2$
2  Initialize $\Gamma\colon \Pi \to$ CircularArc
3
4  foreach $v \in V_\mathrm{u}(\Pi)$ do
5      $\vec{r}(v) \leftarrow (x(v), y(v))$
6
7  foreach $P = (v_1, \dots, v_n) \in \Pi$ do
8      arc $\leftarrow$ CircularArc($\vec{r}(v_1), \vec{r}(v_n), \varphi(P)$)
9
10     foreach $v \in (v_2, \dots, v_{n-1})$ do
11         $\vec{r}(v) \leftarrow$ arc.pointForProgress($p(v)$)
12
13     $\Gamma(P) \leftarrow$ arc
14
15 return $(\vec{r}, \Gamma)$
```

Correctness

In lines 4 to 5, we determine the position vectors for unconstrained vertices $v \in V_{\mathrm{u}}(\Pi)$. For those and only those the generalized coordinates $x(v)$, $y(v)$ are defined, and it is trivial to assemble the vertices' position vectors.

In lines 7 to 13, the position vectors of constrained vertices $v \in V_{\mathrm{c}}(\Pi)$ are determined. When constructing a circular arc Γ_P, we need to know its endpoints' positions. If an endpoint is unconstrained, then it had its position assigned already in lines 4 to 5. If it is not, then it must be constrained, and by definition, it has appeared in an earlier path in whose iteration it had its position assigned. Therefore the endpoints' positions are well-defined at the time of access in line 7. Considering a path's internal vertices v are constrained by Equation (4.1), the $p(v)$ are well-defined, allowing us to compute the vertices' positions on Γ_P. Equation (4.1) also guarantees that vertices do not appear as internal vertices to any more paths once laid out, and therefore have a position assigned only once.

Considering the position vectors are computed for both constrained and unconstrained vertices, all vertices $v \in V(\Pi)$ have their position assigned.

Runtime

The partition of $V(\Pi)$ into $V_{\mathrm{u}}(\Pi)$ and $V_{\mathrm{c}}(\Pi)$ is implicitly given by the domain of x, y, and p. Therefore lines 4 to 5 can be implemented in $\Theta(|V_{\mathrm{u}}(\Pi)|)$. In lines 7 to 13, we construct a circular arc for each path $P \in \Pi$ and use it to compute the positions of P's internal vertices in $\Theta(1)$ each. Considering each vertex $v \in V_{\mathrm{c}}(\Pi)$ has its position assigned once and once only, lines 7 to 13 run in $\Theta(|\Pi| + |V_{\mathrm{c}}(\Pi)|)$, yielding a total runtime of $\Theta(|\Pi| + |V(\Pi)|)$, which is optimal.

Validity of Drawings

The drawing the algorithm produces is not necessarily a (valid) drawing of Π with circular arcs. While the generalized coordinates implicitly satisfy the constraints of all vertices on a path P lying on the same circular arc Γ_P, they do not make any guarantees about vertices not coinciding, edges not overlapping, or edges intersecting no vertices other than their endpoints. Recall that if the order of vertices on Γ_P is not as indicated by P, then there inevitably are overlapping edges. These constraints will be dealt with in the following section.

4.6 Forces and Potential Energy

Let $G = (V, E)$ be a connected and simple input graph. We have seen that one can efficiently decompose G into a greedily realizable sequence of simple paths $\Pi = (P_1, \dots, P_k)$ and choose generalized coordinates that implicitly satisfy the holonomic constraints defined by Π.

For a configuration q_Π to yield a valid drawing of the graph, it is left to ensure that no vertices coincide, that no edges overlap, and that no edge intersects any vertices other than its endpoints. By assigning each configuration a potential energy that is finite if and only if said constraints are satisfied, we can easily distinguish valid from invalid drawings. This potential energy also serves as a measurement of the resulting drawing's quality and can be optimized to get better-looking drawings of G.

When drawing a graph, we want the vertices to be well spaced out with adjacent vertices being close together. Defining both attractive and repulsive forces between each pair of vertices is a standard procedure in many force-directed algorithms [Kob13] and can be applied here as well. In a mechanical system, the forces would act on the vertices and move them around to decrease the implicitly defined potential energy of the system.

In the following, $c_i \in (0, \infty)$ are constants used to scale various physical quantities.

Vertex-Vertex Repulsion

Thinking of the vertices as charged particles pushing each other away allows us to space them out. A repulsive force F_{rep} based on Coulomb's law attempts to push two vertices away from each other. It is exerted along the line connecting the vertices, and its magnitude depends on their distance d:

$$F_{\text{rep}}(d) \coloneqq c_1 \cdot \frac{1}{d^2}$$

We can write the electric potential energy stored in such a pair of charged particles as

$$\begin{aligned} U_{\text{rep}}(u, v) &\coloneqq \int_{\infty}^{d(u,v)} -F_{\text{rep}}(s)\, ds \\ &= c_1 \cdot \frac{1}{d(u, v)}, \end{aligned}$$

where $d(u, v)$ is the Euclidean distance of the vertices u and v's positions. For $d(u, v) = 0$ the constraint of vertices not coinciding is violated, and we shall define $U_{\text{rep}}(u, v) \coloneqq \infty$.

Vertex-Vertex Attraction

In order to keep adjacent vertices close together, we think of every edge as a virtual spring attempting to restore its relaxed length k. Linear springs based on Hooke's law have shown to be too strong when adjacent vertices are far away from each other. Therefore we shall use springs whose restoring forces F_{att} are instead logarithmic in their relative displacement:

$$F_{\text{att}}(l) \coloneqq -c_2 \cdot k \cdot \ln\left(\frac{l}{k}\right)$$

The elastic potential energy stored in such a spring can be written as

$$\begin{aligned} U_{\text{att}}(e) &\coloneqq \int_{k}^{l(\Gamma_e)} -F_{\text{att}}(s)\, ds \\ &= c_2 \cdot kl \cdot \left(\ln\left(\frac{l}{k}\right) - 1\right) + k^2, \end{aligned}$$

where Γ_e is the circular arc used to draw the edge e, and $l(\Gamma_e)$ is its arc length.

Overlapping Edges

Edges can overlap either within a single path or between two distinct paths. For each path $P \in \Pi$ we can define a potential energy that is zero if no edges $e \in E(P)$ overlap each other, and infinite otherwise:

$$U_{\mathrm{ord}}(P) \coloneqq \begin{cases} 0 & \text{if edges } e \in E(P) \text{ do not overlap} \\ \infty & \text{otherwise} \end{cases}$$

If edges within a path overlap, its internal vertices are not ordered correctly on the circular arc Γ_P. In the process of transferring vertices from being ordered correctly to being ordered incorrectly, two vertices $u \neq v$ must coincide, i. e. $d(u,v) = 0$, for which $U_{\mathrm{rep}}(u,v) = \infty$. Therefore U_{ord} does not affect the continuity or local differentiability of the total energy function. Overlapping edges of different paths will be dealt with in the next paragraph.

Vertex-Path Repulsion

Edges of different paths $P_i \neq P_j$ can only overlap if the two circular arcs Γ_{P_i}, Γ_{P_j} overlap. This case only occurs if at least one of the paths' endpoints lies on the other arc. Since the input graph G is simple, this is equivalent to a circular arc intersecting vertices its respective path does not contain.

Therefore it is sufficient to ensure that vertices v only lie on those circular arcs Γ_P where they are part of the respective path, i. e. $v \in V(P)$. For each path $P \in \Pi$ and vertex $v \notin V(P)$, we can define another potential energy that is finite if and only if v does not lie on Γ_P as

$$U_{\mathrm{int}}(v,P) \coloneqq \begin{cases} c_3 \cdot \frac{1}{d(v,\Gamma_P)} & \text{if } d(v,\Gamma_P) \neq 0 \\ \infty & \text{otherwise.} \end{cases}$$

Here $d(v,\Gamma_P)$ is the minimum Euclidean distance from the vertex v's position to any point on the circular arc Γ_P. Note that by defining the potential energy, the corresponding restoring force is implicitly defined as well.

Total Potential Energy

Using the four components described above, we can now calculate the total potential energy of the system determined by a configuration q_Π:

$$U(q_\Pi) \coloneqq \sum_{\{u,v\} \in V^2} U_{\mathrm{rep}}(u,v) + \sum_{e \in E} U_{\mathrm{att}}(e) + \sum_{P \in \Pi} U_{\mathrm{ord}}(P) + \sum_{\substack{v \in V, P \in \Pi \\ v \notin V(P)}} U_{\mathrm{int}}(v,P)$$

The summands — and therefore their sum, too — are finite if and only if all the constraints defined by Π are satisfied. Thus we can easily tell whether or not a configuration yields a (valid) drawing of Π with circular arcs by calculating the system's total potential energy. The energy function can be evaluated in $\mathcal{O}(|V|^2 + |E| + |V| \cdot |\Pi|)$ time.

4.6.1 Minimizing Potential Energy

Considering an explicit function for the $\vec{r}_v$ is not feasible as illustrated in Section 4.5, closed-form expressions for the forces or the system's total potential energy are not feasible either. It is therefore not an option to use generalized forces to minimize the system's potential energy; let alone to perform an analytical optimization. Instead, we shall resort to a generic hill climbing algorithm to find a local energy minimum.

Since hill climbing generally works on real-valued functions of n variables, we need the total energy function to be of the form $U\colon \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$. The following bijective transformations allow all generalized coordinates to be used as real numbers and vice versa:

$$\begin{aligned}(-180^\circ, 180^\circ) \ni \varphi &\mapsto c_4 \cdot \tan\left(\frac{\varphi}{2}\right) \in \mathbb{R}\\ (0,1) \ni p &\mapsto c_5 \cdot \tan(\pi \cdot (p - 0.5)) \in \mathbb{R}\end{aligned}$$

We can then collect all n generalized coordinates in a real-valued vector $q \in \mathbb{R}^n$ determining the system's configuration and evaluate its potential energy in $\mathcal{O}(|V|^2 + |E| + |V| \cdot |\Pi|)$ time.

For hill climbing to converge to a (local) energy minimum, it has to be able to reach a configuration with finite energy by adjusting only a single dimension from its start configuration. Better yet, it should start from a configuration with finite energy, i. e. a configuration that fulfills all constraints and therefore yields a (valid) drawing of Π with circular arcs. Theorem 2 shows that such a configuration exists and its proof suggests a simple algorithm for finding one in $\mathcal{O}(|V|^2 \cdot |\Pi|)$ time.

Considering the potential energy assigned here is always positive, there exists an infimum of the energy function and hill climbing is set to converge.

4.7 Evaluation

Besides this written report, we have implemented the algorithms illustrated in Sections 4.1 to 4.6 in a Mac OS application written in Swift 3. It allows arbitrary input graphs and greedily realizable sequences of paths to be loaded and modified. Unless specified, the initial configuration of the drawing is generated randomly. The user can make manual adjustments to the drawing, but can also use hill climbing to reduce the drawing's energy automatically. Drawings generated by the application can also be exported as vector graphics.

The implementation is open source and can be found on GitHub:

`https://github.com/jenox/bachelor-thesis/`

4.7.1 Results

We tested the algorithms using random graphs of various orders and densities, which were generated according to the Erdős–Rényi model [ER59, Gil59]. The constants in Section 4.6 were chosen as $k = 100$, $c_1 = 10^5$, $c_2 = 1$, $c_3 = 10^4$, $c_4 = 10$, and $c_5 = 10$.

Figure 4.4 shows drawings of graphs with 10 vertices and different densities, each with two different decompositions into greedily realizable sequences of paths: The drawings in the upper row are based on the greedy graph decomposition illustrated in Algorithm 4.1, whereas the drawings in the lower row were created using user-provided decompositions.

For user-provided graph decompositions, Figures 4.5 to 4.7 show the effect additional user adjustments have. The drawings in the upper row were created only by hill climbing from a randomly generated start configuration, whereas in the lower row, few user adjustments have been made.

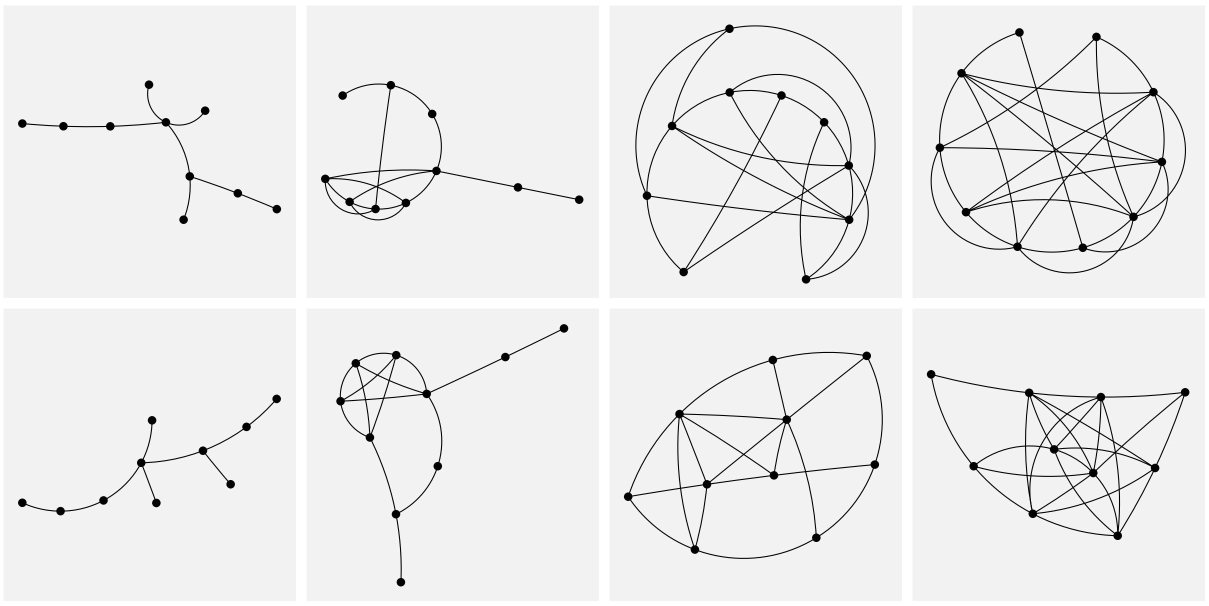

Figure 4.4: Drawings of graphs using decompositions provided by the user (bottom) and greedily determined by Algorithm 4.1 (top).

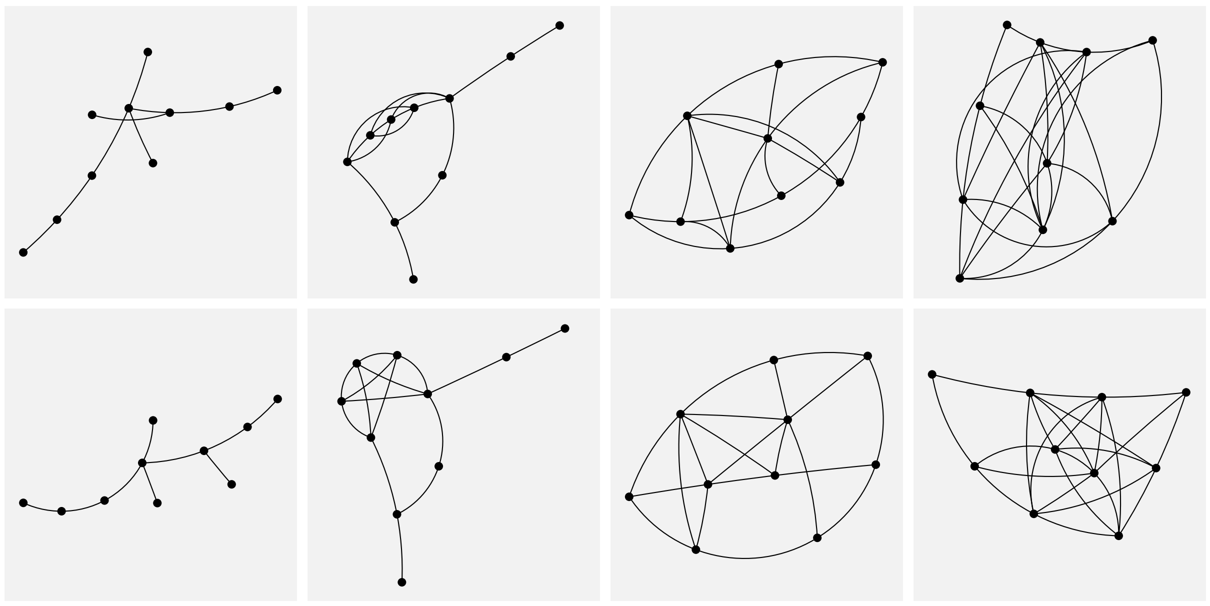

Figure 4.5: Drawings of graphs with 10 vertices and 9/15/20/25 edges; each with (bottom) and without (top) user adjustments.

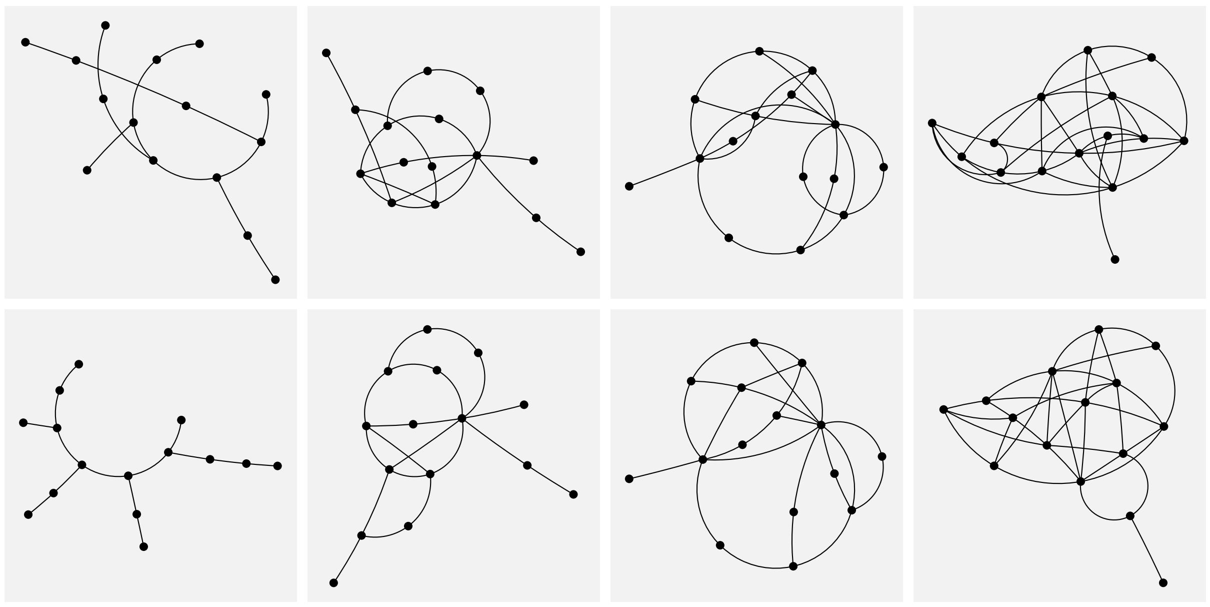

Figure 4.6: Drawings of graphs with 15 vertices and 14/20/25/35 edges; each with (bottom) and without (top) user adjustments.

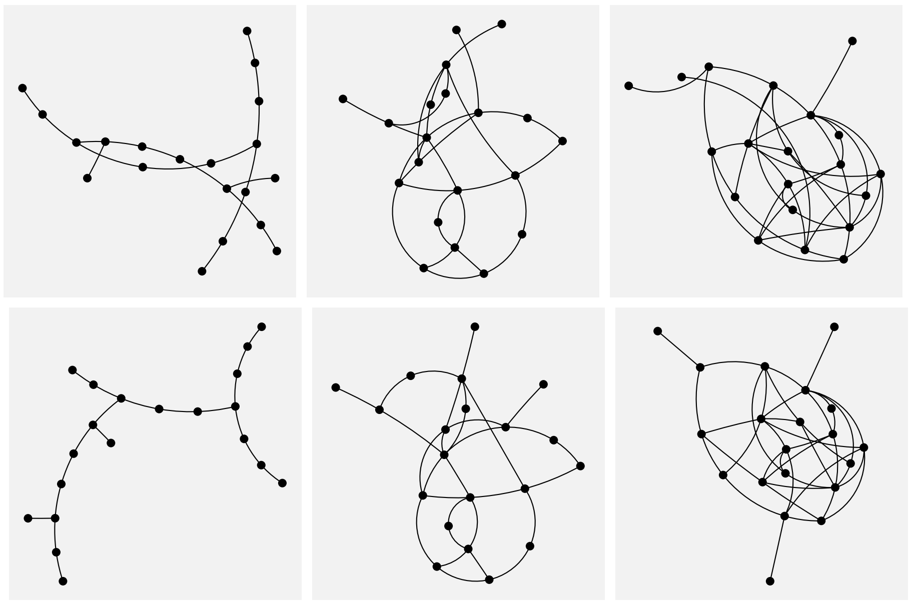

Figure 4.7: Drawings of graphs with 20 vertices and 19/30/40 edges; each with (bottom) and without (top) user adjustments.

4.7.2 Conclusion

The algorithm works nicely with trees and other sparse graphs.

Due to the nature of hill climbing, the algorithm presented here is even more likely to get stuck in local minima than traditional force-directed algorithms. These obstacles are fairly obvious for sparse graphs though, and can easily be overcome using few user adjustments. Even though in theory hill climbing is terribly inefficient, it is highly parallelizable and generally well-suited due to the lack of accurate derivative information of the energy function.

For dense input graphs, on the other hand, the algorithm does not produce satisfying drawings. The hard constraints of vertices on some path P lying on the same circular arc Γ_P take away a lot of freedom in vertex movement that traditional force-directed algorithms have. Possible ways to release tension in the drawing may be prohibited by the constraints, generally yielding drawings with much higher energy values. Figure 4.4 shows that the decomposition of the input graph G into a greedily realizable sequence of simple paths Π is of the utmost importance here. It is very unlikely that a greedy graph decomposition as illustrated in Algorithm 4.1 yields satisfying results.

The choice of generalized coordinates in Section 4.4 and their transformation in Section 4.5 require the path decomposition to be a greedily realizable sequence of paths. As a result, each vertex can be internal to at most one path, meaning that there are most $|V|$ paths that have internal vertices — the rest of the paths, which are $\Theta(|V|^2)$ many for graphs with $\Theta(|V|^2)$ edges, are single-edged. Considering the main motivation for using circular arcs to draw multiple incident edges in the first place was drawing graphs with few geometric entities, the algorithms presented here fail for graphs with large dense components by default.

5. Future Work

The results presented in Section 4.7 were somewhat dissatisfying for dense graphs. The constraints the drawings are subject to play a significant role in this. Recall that the decomposition of the input graph into a greedily realizable sequence of paths implicitly defines said constraints; and that how these constraints are accounted for is irrelevant for the quality of the drawings. Therefore the restrictive choice of precondition in Equation (4.1) leaves much room for improvement.

We have also seen that, depending on how the input graph has been decomposed into paths, the resulting drawings greatly differ in appeal. Possible future research may include finding properties of decompositions that yield appealing drawings, and how such decompositions can be found efficiently. A reference drawing of the graph may allow for a smart choice of paths.

It is left to find out if additional features desired in equilibrium, such as better angular resolution [CCG$^+$11], are compatible with drawing multiple incident edges on the same circular arc.

Although constrained systems are very common in physics, how relevant they really are to graph drawing is yet to be determined.

Optimization-wise, there is still a lot of room for improvement.

The partial derivatives of the energy function with respect to the generalized coordinates may be able to be approximated, allowing for a more natural convergence as all dimensions can be adjusted at once. However, this would introduce other problems of traditional force-directed algorithms, such as possible oscillation around an equilibrium point, or (temporary) decreases in quality.

In systems where the restoring forces are explicitly defined, it may be an option to use a hybrid approach, treating constrained and unconstrained vertices separately. Vertices that can move freely in both dimensions could be displaced using a traditional force-directed approach whereas vertices whose movement is constrained could still be managed by the likes of hill climbing. This approach would be especially beneficial for systems with many unconstrained vertices.

Other optimization techniques, such as genetic optimization algorithms, should also be experimented with. The recombination of two individuals is non-trivial for graph drawings [BBS96] but would allow for local energy minima to be overcome.

Bibliography

[BBS96] Jürgen Branke, Frank Bucher, and Hartmut Schmeck. Using genetic algorithms for drawing undirected graphs. In *The Third Nordic Workshop on Genetic Algorithms and their Applications*, pages 193–206, 1996.

[Ber99] François Bertault. A force-directed algorithm that preserves edge crossing properties. In 7th *International Symposium on Graph Drawing*, pages 351–358, 1999.

[CCG$^+$11] Roman Chernobelskiy, Kathryn I. Cunningham, Michael T. Goodrich, Stephen G. Kobourov, and Lowell Trott. Force-directed Lombardi-style graph drawing. In 19th *International Symposium on Graph Drawing*, pages 320–331, 2011.

[DESW07] Vida Dujmović, David Eppstein, Matthew Suderman, and David R. Wood. Drawings of planar graphs with few slopes and segments. *Computational Geometry*, 38(3):194–212, 2007.

[DM14] Stephane Durocher and Debajyoti Mondal. Drawing plane triangulations with few segments. In *Proceedings of the* 26th *Canadian Conference Computational Geometry*, pages 40–45, 2014.

[ER59] Paul Erdős and Alfréd Rényi. On random graphs I. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.

[Fit11] Richard Fitzpatrick. Newtonian dynamics. `http://farside.ph.utexas.edu/teaching/336k/Newton.pdf`, 2011. Lecture Notes. Accessed: 2017-03-01.

[Fli09] Torsten Fließbach. *Mechanik: Lehrbuch zur Theoretischen Physik I.* Spektrum Akademischer Verlag, 6th edition, 2009.

[Gil59] Edgar Nelson Gilbert. Random graphs. *Annals of Mathematical Statistics*, 30(4):1141–1144, 1959.

[HT94] Marc Henneaux and Claudio Teitelboim. *Quantization of Gauge Systems.* Princeton University Press, 1994.

[IMS15] Alexander Igamberdiev, Wouter Meulemans, and André Schulz. Drawing planar cubic 3-connected graphs with few segments: Algorithms and experiments. In 23rd *International Symposium on Graph Drawing and Network Visualization*, pages 113–124, 2015.

[KM14] Ross J. Kang and Tobias Müller. Arrangements of pseudocircles and circles. *Discrete & Computational Geometry*, 51(4):896–925, 2014.

[Kob13] Stephen G. Kobourov. Force-directed drawing algorithms. In Roberto Tamassia, editor, *Handbook of Graph Drawing and Visualization*, chapter 12, pages 383–408. CRC Press, 1st edition, 2013.

[LO05] Johann Linhart and Ronald Ortner. An arrangement of pseudocircles not realizable with circles. *Contributions to Algebra and Geometry*, 46(2):351–356, 2005.

[Mü16] Milada Margarete Mühlleitner. Moderne Physik für Informatiker. `https://www.itp.kit.edu/~sekulla/MPFI/pdf/VLaktuell.pdf`, 2016. Lecture at Karlsruhe Institute of Technology. Accessed: 2016-09-18.

[Nav12] Carl Rod Nave. Hyperphysics: Potential energy. `http://hyperphysics.phy-astr.gsu.edu/hbase/pegrav.html`, 2012. Accessed: 2017-03-01.

[Ore79] Jay Orear. *Physics*. MacMillan Publishing Company, 1979.

[RN10] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 3^{rd} edition, 2010.

[Sch15] André Schulz. Drawing graphs with few arcs. *Journal of Graph Algorithms and Applications*, 19(1):393–412, 2015.

[SM95] Kozo Sugiyama and Kazuo Misue. A simple and unified method for drawing graphs: Magnetic-spring algorithm. In *Proceedings of the DIMACS International Workshop '94*, pages 364–375, 1995.