A Force Directed Placement Method Including Angular Resolution and Bond Overlap

Ludvig Sundström
Abstract

A Force Directed Placement Method Including Angular Resolution and Bond Overlap

Ludvig Sundström

This study presents a variant of the “spring method” for visualizing simple and undirected graphs. In addition to elementary spring energies, we introduce continuous energy terms for bond angles and bond overlap. To find a good graph layout, the energy of a graph is minimized using an optimization technique including the conjugate gradient method. We strive for a simple, flexible, conceptually intuitive and efficient algorithm. Experiments indicate good results from reducing bond overlap, while optimizing bond angles are more difficult without sacrificing other visualization qualities.
Contents

1 Introduction 3

2 Problem Definition 6

2.1 Overview 6

2.2 Graph Drawing Aesthetics 7

2.3 Force Directed Graph Drawing Algorithms 8

2.4 Angular resolution 9

2.5 Bond Overlap 12

2.6 Some Properties of Our Proposed Energy Function 13

2.7 Scalability 14

2.8 Minimization 15

3 Energy Function Design 17

3.1 Overview 17

3.2 First order term 18

3.3 Second order term 19

3.3.1 Attraction 19

3.3.2 Repulsion 20

3.4 Third order term 20

3.5 Fourth order term 21

4 Implementation and Performance 23

4.1 Overview 23
4.2 Initial placement .................................................. 23
4.3 Local Minimizer .................................................. 25
    4.3.1 Energy function ........................................... 25
    4.3.2 Minimizer .................................................. 27
4.4 Global Minimizer ............................................... 28
4.5 Complexity .................................................... 29
    4.5.1 Local Minimizer ........................................... 29
    4.5.2 Global Minimizer ........................................ 30

5 Experiments and Results ........................................ 31
    5.1 Overview .................................................... 31
    5.2 Sample Visualizations ...................................... 31
    5.3 Global minimization ....................................... 35
    5.4 Bond time complexity ..................................... 36

6 Discussion and Conclusions .................................... 36

7 Future work ..................................................... 38
    7.1 Scalability .................................................. 38
    7.2 Global Minimizer .......................................... 38

8 Acknowledgments ............................................... 39

List of Figures .................................................. 52
1 Introduction

The main purpose of data visualization is to communicate information by encoding it as visual objects contained in graphics [21]. The intent of this work is to visualize graphs, a common data structure used to represent sets of entities and relationships [9]. In this work, the entities are referred to as vertices and the relationships as bonds. In other contexts other terms such as nodes for vertices and edges for bonds are also used.

Graph drawing (or graph visualization) is the process of generating visualizations of the information contained by the graph. This is often done by designing an algorithm describing how to represent the vertices and bonds in a given drawing space, normally as vertex-bond diagrams in two or three dimensions [51]. Since data visualization primarily serves as a human communication method, the goal of graph drawing algorithms is to produce understandable visualizations.

The development of new instrumentation in various fields is generating an increased volume of data in need of visualization. Real-world datasets are often large and complex, and infeasible to draw by hand. To assist in drawing data-sets representable as graphs, computer programs have been developed to implement the algorithms of graph drawing. Fig. 1 contains examples of how information might be visualized by computer programs.

No known implementable algorithm can decide the perfect layout of every graph [8]. There are many different types of graphs, and the interplay of graph theory and geometry poses many theoretical challenges when designing algorithms for graph visualization. Consequently is graph drawing a well studied subject with an extensive literature [17].

Intuitively, drawing graphs “nicely” is a problem open to multiple interpretations. However there are a number of common quality metrics to be acknowledged for any graph drawer. In this study, we mainly refer to these metrics as ‘graph drawing aesthetics’ as in the study of H. Purchase “Which aesthetic has the greatest effect on human understanding?” [46] Most graph drawing algorithms target a limited set of drawing aesthetics.
Figure 1: Examples of data visualization. (a) displays a large set of data visualized with Gephi. (b) displays a protein to protein network visualized with Cytoscape. For image sources, see List of Figures
A common type of algorithm in graph drawing is the force-directed algorithm. As we will see in section 2, these algorithms are often flexible in the way that it allows to be implemented to ensure any set of aesthetic criteria. We recognized the lack of existing implementations of force-directed algorithms directly addressing the issues of narrow bond angles and bond overlap even though considered very important for graph drawing (see section 2.4 and 2.5).

The objective of this work is thus to design and implement a force directed algorithm that accepts a graph representing an arbitrary dataset as input and produces a set of coordinates corresponding to a good graph layout as output. The goal is for the algorithm to be efficient, scalable and produce graph layouts of less narrow bond angles and less bond overlap compared to similar techniques. Like most force-directed algorithms, our algorithm achieves its goal by minimizing the energy-value of a graph, a value that needs to be cautiously defined. This is done by designing an energy function, which determines the energy of a graph of a given state.

Experimenting with an implementation of our algorithm, we obtain pleasing layouts of a large variety of graphs of modest sizes. Reducing bond overlap always contributes to better graph layout qualities. Optimizing bond angles leads to some promising results, but this metric is significantly harder to improve without sacrificing other visualization qualities. We summarize the contributions of this work.

1. A new, simple and flexible formula for calculating the energy of arbitrary graphs, including continuous expressions for the energy from bond angles and bond overlap.

2. Design and implementation of a robust and modular algorithm for minimizing the energy of graphs, using the conjugate gradient of an energy function and global minimization. The source code of the implementation is available at https://github.com/lsund/graph-visualization.
2 Problem Definition

2.1 Overview

The problem of drawing graphs “nicely” is a broad problem and can be interpreted in various ways. In our case, the given conditions are as follows.

1. The data is not restricted to a specific type. Any information that can be encoded as entities and relationships should be treated in the same way.

2. The graph representing the data should be sparse.

Aside from these restrictions, the method of solving the problem is open for analysis. The purpose of this section is to transition from a somewhat vague problem description to one that can be used to formulate technical design goals for a graph visualization approach.

We define a sparse graph $G = (V, B)$ as a pair of sets satisfying $B \subseteq [V]^2 \land |V| \approx |B|$; thus, each element $b_i = (v_i, v_j) \in B$ is a 2-element subset of the set of vertices $V$ and the number of bonds roughly equal the number of vertices $|V|$. A placement of $V$ is defined as a set of points $V_p = \{p_i : p_i \in [\mathbb{R}]^n\}$ of the $n$’th dimension where $p_i \in V_p$ corresponds to $v_i \in V$. Similarly, bond placement is defined as a set of vectors $B_p = \{l_i : l_i = (p_i, p_j), p_i, p_j \in [\mathbb{R}]^n\}$ in the $n$’th dimension where each element $l_i \in B_p$ corresponds to $b_i = (v_i, v_j) \in B$. From $V_p$ and $B_p$, a pictorial representation of $G$ can be obtained by drawing each $v_i$ as a simple figure (circle, rectangle) at $p_i$ and each $b_i$ as a straight line segment along $l_i$. We will refer to the two sets $V_p, B_p$ as a layout $L$. We limit ourselves to drawings in two dimensional space, $n = 2$.

Furthermore, we need to ask the question What makes a visualization of a graph understandable? There is presumably no specific answer to this question, but important metrics have been developed for characterizing graph visualization qualities.
2.2 Graph Drawing Aesthetics

Many quality metrics have been proposed for graph drawings [5]. The goal of these metrics, also known as aesthetics are to find objective means of evaluating the graphs quality [16]. Choosing which aesthetics to consider for a given algorithm often depends on problem specific properties or individual preference. However, some metrics are more frequently used than others. In this study we target a small set of well known drawing aesthetics, supported by verified scientific studies. The quality metrics are stated in italic.

One of the most typical issues arising in graph drawing is bond overlap [15]. According to the influential work of Helen Purchase, bond overlap minimization is in fact the most important graph drawing aesthetic [46, 47]. However, succeeding to eliminate bond overlaps might still result in bonds embedded very close but not crossing. Such configurations might also contribute to bad layouts. One technique that addresses this in addition to other issues is maximizing the angular resolution of a graph. It is known that angular resolution (the smallest angle formed by two bonds meeting at a common vertex) can have significant impact on the readability of graphs visualizations [19], but this metric is not used by popular visualization engines such as GraphViz [24] or Cytoscape [7]. Maximizing angular resolution might have the side effect of placing well connected vertices in a central position of the graph [5]. Since well connected members generally are more important, the placement of well-connected vertices in a central position is a separate quality measure [5]. Furthermore, well connected members are preferably placed close to each other, i.e. minimizing the average bond lengths between vertices [47]. Another regularity measure is enforcing an even vertex distribution helps improve the understandability of the visualization. This is because the human inclination towards regular objects [5]. Lastly and perhaps more intuitively, the graph should be drawn at a central position in the visualization area.

Drawing aesthetics are patterns developed to enhance the quality of graph visualizations. Since computers do not possess the human ability of understanding patterns, we need a more mechanical process of ensuring graph quality metrics. It turns out though, that this isn’t very easy. In fact, obtaining an optimal solution to the graph drawing problem for any set of aesthetics is in
many cases computationally infeasible [8]. The large variety of graphs and the difficulty of satisfying the constraints of multiple drawing aesthetics have traditionally caused researchers to restrict their algorithms to special cases, such as trees, DAG’s or planar graphs [8]. Thankfully, we don’t always need to find an optimal layout in order for the visualization to be understandable.

2.3 Force Directed Graph Drawing Algorithms

Force directed algorithms (FDA) are a class of general and flexible algorithms for calculating graphs layouts. Also known as spring embedders, these algorithms fundamentally aim to model dynamic systems of interconnected springs, where each spring is tending towards its individual minimum potential energy. The force directed layouts, found by minimizing the total energy of the system, often resemble physical objects like the one in Fig. 2. Even though basic FDA are rather simple procedures, they often produce visually pleasing layouts fulfilling many of the aesthetics proposed in section 2.2. [35]. FDA usually operate on one of the most general graph types: the simple, undirected graph and can thus be applied to a vast number of problems.

The principle of FDA is often attributed Tutte, who in 1963 designed an algorithm based on barycentric representations [53]. However, it was Eades [12] that laid the foundation to the standard force directed model, when he in 1984 formulated the important metaphor:

The basic idea is as follows. To embed [lay out] a graph we replace the vertices by steel rings and replace each edge with a spring to form a mechanical system . . . The vertices are placed in some initial layout and let go so that the spring forces on the rings move the system to a minimal energy state.

While most FDA extend upon on this idea, it is important to remember the goal of graph drawing – to efficiently generate of good graph layouts – not to simulate physical reality [22]. Even Eades implementation of the spring forces did not reflect Hooke’s law, but instead followed his own definitions. Additionally, Eades decided that it was important only for a vertex to be near
its immediate neighbors and thus only applied attractive forces to neighboring vertices. Later, Kamada and Kawai proposed a significant alternative to Eades algorithm, calculating the spring forces between all vertex pairs as a function of the shortest distance between a given pair \[31\].

The main value of basic FDA is simplicity; it easy to adapt to and extend the algorithm. For instance, the algorithm of Davidson and Harel(D&H) defined its energy according to a specific set of drawing aesthetics: \textit{Node distribution, Distance to drawing area border, Edge length, Edge crossings and Node-Edge distances} \[8\]. Based mainly on the algorithm of D&H, Fruchterman and Reingold presented a well performing algorithm improving both speed and drawing quality \[22\]. Cruz and Twarog extended the method of D&H to three dimensional euclidean space \[6\]. In contrast, Tamara Munzner emphasized non-euclidean drawing techniques by laying out large graphs in hyperbolic 3D-space. The remarkable result of Munzner’s technique is visualizations of graphs with both uniform bond lengths and uniform vertex distribution \[43\].

Various techniques exists for putting a FDA into action. Stress majorization is often used as a simple and elegant implementation method for minimizing spring energies. Other methods include simulated annealing, genetic algorithms or more conventional global optimization techniques \[54\].

In any case, there is a need of expressing the energy of a graph in a particular state. In this work, the energy is expressed using mathematical function \(f\) whose value depends on the quality metrics of section 2.2. Thus, a good graph layout is found by minimizing \(f\). Of the chosen graph drawing aesthetics, ensuring an even vertex distribution and a even bond length will contribute to the conventional qualities of basic force directed drawings while the metrics for angular resolution and bond overlap are more specialized and should be explored further.

2.4 Angular resolution

In graph drawing, angular resolution refers to the minimum angle formed by any two bonds meeting at a common vertex \[2\]. Prior studies concerning angular resolution mostly seems to apply to special graphs such as trees or planar
Figure 2: A force directed graph layout. With the right parameters, force directed layouts in two dimensions often resemble the projection a three dimensional layout in the plane. For image source, see List of Figures.
graphs. Trees can easily be drawn with perfect angular resolution, if the bond lengths and drawing area are flexible. Planar graphs can be drawn with an angular resolution proportional to $1/d$ if $d$ is the maximum vertex degree of the graph [18]. For non-planar graphs, it is more difficult to reason about angular resolution. Angular resolution is a rather young concept (Formann et. al. 1993) and studies on graph drawing and angular resolution are not very common [14].

According to the existing studies, angular resolution is an important measure for a good visualization [19]. One example of the use of angular resolution for the generation of “nice” graph drawings is “Lombardi drawing”, Fig. 3 which uses bended arcs and a perfect angular resolution between bonds at each vertex [14]. Lombardi drawing is one of the few techniques attempting to optimize one specific criterion. Usually, drawing according to a single aesthetic result in other important aesthetics being unsatisfied. In fact, many graphs cannot be drawn perfectly with Lombardi drawing [19].

Figure 3: Lombardi drawing of a graph, each bond is a circular arc and every vertex has perfect angular resolution. For image source, see List of Figures
2.5 Bond Overlap

A bond overlap occurs in graph drawing when two line segments that represents a pair of bonds in two dimensional space share a point which is not a vertex position. Minimizing bond overlap is one of the most basic and natural problems in graph drawing \[15\]. Addressing the issue of bond overlap is critical for improving the understandability of graph visualizations \[46\]. It is unfortunate though, that bond minimization in general is computationally hard, that is, the problem is solvable but in an excessive amount of time \[51\]. The specific problem:

Given a graph $G$ and a nonnegative integer $K$, decide whether there is a drawing of $G$ with at most $K$ edge crossings.

has been proved NP-complete \[15\]. Given that the problem is hard, overlap minimization has been attempted on specific domains, in hope of finding polynomial time solutions for those cases. The obvious special case is eliminating all bond overlap when the graph is planar \[8\], but special algorithms for other graphs, such as cubic graphs have also been developed \[15\]. Generally though, approaches to overlap minimization is often of heuristic nature. A popular method is the planarization method, introduced by Batini, Talamo, and Tamassia in 1984 \[4\]. It involves solving two subproblems, for which a variety of techniques are available \[15\]:

1. **The maximum planar subgraph problem**: Given a graph $G$ whose bond overlaps to be minimized, compute a planar subgraph $P = (V, B_p)$ of $G$ with as many bonds in $P$ as possible.

2. **The bond insertion problem**: Insert the edges in $B \setminus B_p$ into $P$. During this bond insertion process, bond overlap that occur when inserting a new bond is replaced by a dummy vertex with a degree of four so that the graph remains planar. The objective is to keep the number of the dummy vertices low (and thus the number of bond overlap in the final drawing) as low as possible.

Although planarization method has been successfully incorporated in some drawing algorithms, it does not guarantee good results. It has been shown that even
if the two $NP$-hard sub-problems are solved optimally, the algorithm might still produce arbitrarily bad solutions [15].

Despite being an actively researched subject, bond overlap minimization remains one of the most difficult problem to approach in graph drawing [15]. The planarization method and other general techniques are often hard to implement and they do not scale well with the size of the graph [51]. Moreover, bond overlap minimization might conflict with other qualities of drawing [8]. In a recent study on overlaps in large graphs, Kobourov et al. stated that “... it is a reasonable question to ask to what extent one should try to minimize edge crossings [bond overlap] to justify the cost.” [51].

2.6 Some Properties of Our Proposed Energy Function

Instead of dealing with the issues associated with completely optimizing angular resolution and bond overlap, we conveniently include these metrics in our energy definition so that narrow angles and bond overlap is reduced when the energy of the system is reduced. Even though this approach does not necessarily result in a layout with a minimal number of bond overlap or a perfect angular resolution, the goal of the algorithm was to reduce narrow bond angles and bond overlap, not compute layouts with optimal values of these metrics. D&H made a similar approach, and their method seemed to perform well in practice [8].

Our definition of energy is flexible in the way that it provides an ability of configuring the weights of the terms of the energy function to fit the needs of the users. For instance, if reduced bond overlap is considered more important, increasing the weight for the corresponding term will likely result in fewer bond overlap. However, knowing that attempting to optimize specific aesthetic criteria often results in bad pictures, we should generally aim for a good weight balance. Our method differs from the one of D&H mainly in how bond angles and bond overlap are used during the placement procedure.

Angular resolution Both our algorithm and the one of D&H attempts to minimize a class of bonds that could be described as “close but not quite overlapping”. However, D&H did not actually consider angular resolution. Instead, they implemented the Node-Edge Distance aesthetic particularly
for preventing configurations with individual vertices laid out close to foreign bonds. In our case, we attempt to exploit the side effects of good angular resolution to achieve similar results. Thus, the advantage of our method is good angular resolution itself – which often contributes to regular, good looking pictures [19]. The disadvantage is that the algorithm only prevents bonds connected at a common vertex from being laid out too close. For example, two bonds from two disconnected subgraphs could still be laid out close but not overlapping by our algorithm.

**Bond overlap** While D&H had a constant penalty value for each overlapping bond, we introduce a more advanced term. Constant increments of energy can introduce non-smoothness to the function, something that should be avoided in global optimization. Instead, we introduce a continuous expression treating bond overlap at a local level. Being a function of how much bonds are overlapping, this term enforces overlapping bonds to “slide off” each other during minimization. The disadvantage of this method compared to the one of D&H is computational complexity. Attempting to reduce bond overlap is computationally demanding even with the assumption of sparse graphs.

D&H determined all bonds to be of equal length in visualizations. As it often is important to visualize the similarities between data in real-life graphs, our algorithm will have the option of custom bond lengths. If this alternative is set and user provides the preferred bond lengths with the data, the algorithm makes an attempt to preserve these lengths.

### 2.7 Scalability

While the first FDA were designed for small graphs (less than 100 vertices) [35, 51], many modern real-world datasets (e.g. biological networks) have tens of thousands or even millions of vertices [51]. This is a problem for basic FDA, which often have cubic time complexity [52]. Such drawing algorithms assume force interaction between pairs of every individual member of the graph. A graph with $n$ vertices has $n(n - 1)/2$ pairs, thus finding the energy of such a graph requires $n^2$ calculations, similar to the notorious $n$-body problem. With $n$ iterations, the overall complexity is $O(n^3)$. 
Of course, graph drawing and multi body simulation are not identical problems. Graph drawing is about information visualization while the n-body problem is about predicting the movements of a group of real bodies in space. FDA do not faithfully need to imitate celestials, chemicals or other physical bodies and thus approximation methods are often used to speed up computation. One approximation method, dating back to Fruchterman and Reingold, is approximating forces of distant vertex pairs by not calculating repulsive forces of those pairs, reducing the overall number of calculations per iteration drastically [22]. We follow this approach, reducing complexity by maintaining vertex neighborhoods (further described in section 4).

Apart from optimizing the process of predicting vertex movement, clustering techniques have been used to maintain manageable subsets of data during placement. [37]. The first to present an efficient algorithm for more than 1000 vertices were Hadany and Harel who presented a multi scale technique to deal with large graphs [25]. The algorithm of Hu [26] combines the multilevel approach with Barnes-Hut simulation [29], and is implemented by the GraphViz [24] visualization engine sfdp. Our algorithm does not include multi level drawing, it is listed under section 7 as a potential future improvement.

2.8 Minimization

If we obtain a graph laid out with a minimal energy value, the original problem is solved. Given that we know how to calculate the energy of a specific graph layout, the next step is deciding an optimization technique for finding a layout with minimal energy.

Generally, mathematical (global) optimization is the process of selecting the best element with regard to some criteria from a set of available alternatives [50]. In our case the target set is represented the co-domain of the energy function $f$. A common way of approaching function minimization problems is by using iterative methods. Such a method is a procedure that generates a sequence of approximations, until converging for some minimal value. Well known iterative optimization methods include Newton’s method, quasi-Newton, conjugate gradient and steepest descent [33].
Newton’s method is one of the most famous optimization methods. Originally proposed by Newton to find the roots of polynomials, it is now widely used for various optimization problems. It is remarkable for its efficiency, guessing both direction and length of the required step to the local extremum of a given function. Newton’s method requires that the function is twice differentiable to make this guess, making the method disadvantageous if the Hessian matrix is unavailable or difficult to compute. Furthermore, Newton’s method requires computation of a $n \times n$ system of equations each iteration where $n$ is the number of unknowns of the problem. Finite distance approximations for the Hessian or Jacobian does not always solve the issue of computational complexity.

Quasi-Newton and conjugate gradient are methods that only use first order analytical derivatives directly and often more convenient than Newton’s method, especially when $n$ is big. Moreover, these methods are designed to converge faster than the steepest descent method, known to require many iterations for functions which have long, narrow valley structures. Quasi-newton and conjugate gradient are two similar methods. In fact, it has been shown that the methods are equivalent when exact line search is used and they generally serve the same purpose. In detailed comparison, quasi-Newton seems to be more robust than conjugate gradient. However, conjugate gradient performs no matrix operations and requires relative little storage, making it preferable for large problems. For our minimization algorithm, we chose the conjugate gradient method. The reasons are listed.

1. Since vertices are non-fix members of graph drawings, the number of unknown entities $n$ is equal to the number of vertices in the graph. Because of this, the arity of $f$ should generally be large, making the conjugate gradient method favorable.

2. Conjugate gradient methods are known to perform well on functions that can be written on the quadratic form $g(x) = \frac{1}{2}x^TAx + x^Tb + c$, where $A$ is a $n \times n$ matrix, $b$ is a vector and $c$ is a scalar. As seen in section 3, this is indeed the form of $f$.

3. Expressing the partial derivatives for $f$ is rather complicated. Using Newton’s method and potentially computing and storing the second partial
derivatives of $f$ would imply significant practical and theoretical work.

Regardless of choice of minimization technique, there is a possibility of getting stuck in a non-optimal local minimum. Since the number of local minima grows exponentially with $n^{32}$, the problem is particularly important to address in the case of minimizing the energy of graph layouts. Intuitively, deciding a good initial vertex placement is a crucial step for avoiding suboptimal local minima. To further address the problem, placement algorithms may include iterated local search [36], repetitively guessing new starting points before each search. A variant of this technique is used in this work, see section 4.2.

3 Energy Function Design

3.1 Overview

The energy function $f(G)$ takes a graph $G$ and produces a value $c_e \in \mathbb{R}$. The value $c_e$ depends on how well the layout of $G$ adapts to the metrics established in section 2.2. In summary, these metrics are the following.

1. The graph should have a central position in drawing space.
2. High degree vertices should have at least as central positions as low degree vertices.
3. The lengths of the bonds should be close to the preferred bond length.
   The preferred bond length can be either data defined for each bond or a constant value.
4. Vertices should be evenly distributed spatially over the visualization surface.
5. The angular resolution should be maximal.
6. There should be a minimum number of bond overlap.

Within this section, these rules are referred to using the numbering (1), (2), (3), (4), (5) and (6). The energy function described in this section
\[ f(G) = f_1(G) + f_2(G) + f_3(G) + f_4(G) \]

is the mathematical description of these rules. Each term in \( f \) is designed to have a minimal value if the given layout adapts to the terms corresponding rule(s). Every term also has a weight, which can be adjusted to increase or decrease the influence of a particular rule. \( f_1, f_2, f_3 \) and \( f_4 \) are referred to as the First order term, Second order term, Third order term and Fourth order term because the respective terms performs calculations on sets of singles, pairs, triples and quadruplets of vertices. The gradient \( \nabla f \) of \( f \) is the partial derivatives of \( f \) in the \( x \) and \( y \) dimension. It is defined as \( \nabla f_1 + \nabla f_2 + \nabla f_3 + \nabla f_4 \). \( f \) and \( \nabla f \) is designed to be continuous to provide for a smooth function “landscape” and a good minimization process.

### 3.2 First order term

The first order term

\[ f_1(G) = \sum_{v_i \in V} w_i d_i^2 \]

is the translation of metric (1) and (2). Its value depends on the absolute distance \( d_i = \sqrt{(x_i - p_{cx})^2 + (y_i - p_{cy})^2} \) between individual vertices and the central point \( p_c = [p_{cx}, p_{cy}] \). The weight factor \( w_i \) for vertex \( v_i \) is the product of its mass \( m_i \) and a constant \( c \). The mass of a vertex depends on its degree. Thus, more connected vertices are of higher energy. For each \( v_i \) the two corresponding partial derivatives of \( f_1 \) are

\[ \frac{\partial f_1}{\partial x_i} = 2w_i x_i \]

\[ \frac{\partial f_1}{\partial y_i} = 2w_i y_i \]
3.3 Second order term

The value of the second order term

\[ f_2(G) = f_{2a}(G) + f_{2v}(G) \]

depends on interactions between pairs of vertices. The term is a fundamental part of \( f \), enforcing a regular and balanced graph layout by applying attraction and repulsion between vertices.

3.3.1 Attraction

The attraction energy term

\[ f_{2a}(G) = \sum_{b_{ij} \in B} w_{ij} (d_{ij} - d_{0ij})^2 \]

is the translation of metric (3). Its value depends on how well the actual length \( d_{ij} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} \) of each bond \( b_i \) relates to the preferred length \( d_{0ij} \). The weight factor \( w_{ij} = cm_i m_j \) is the product of the mass of \( v_i \), the mass of \( v_j \) and a constant \( c \). For each bond \( b_i, b_{ij} \), the four corresponding partial derivatives are

\[
\frac{\partial f_{2a}}{\partial x_i} = -2w_{ij}x_i(1 - \frac{d_{0ij}}{d_{ij}})
\]

\[
\frac{\partial f_{2a}}{\partial y_i} = -2w_{ij}y_i(1 - \frac{d_{0ij}}{d_{ij}})
\]

\[
\frac{\partial f_{2a}}{\partial x_j} = 2w_{ij}x_i(1 - \frac{d_{0ij}}{d_{ij}})
\]

\[
\frac{\partial f_{2a}}{\partial y_j} = 2w_{ij}y_i(1 - \frac{d_{0ij}}{d_{ij}})
\]
3.3.2 Repulsion

The repulsion energy term

\[ f_{2r}(G) = \sum_{v_i,v_j \in V} \begin{cases} \omega_{ij}(p - d_{ij})^2, & d_{ij} < p \\ 0, & \text{otherwise} \end{cases} \]

is the translation of metric (4). Its value depends on the closeness of vertices. The vertex pair \((v_i, v_j)\) has closeness \(d_{ij} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}\). The weight factor \(\omega_{ij} = cm_i m_j\) is like for \(f_2\), the product of the mass of \(v_i\), the mass of \(v_j\) and a constant \(c\). \(p\) is a constant for the padding of the vertices, depending on the drawing area and the number of vertices. If \(d_{ij}\) is larger than \(p\), no repulsive force is applied to \(v_i\) and \(v_j\). For each vertex pair \((v_i, v_j)\), the four corresponding partial derivatives are

\[
\begin{align*}
\frac{\partial f_{2a}}{\partial x_i} &= -2\omega_{ij}(x_j - x_i) \\
\frac{\partial f_{2a}}{\partial y_i} &= -2\omega_{ij}(y_j - y_i) \\
\frac{\partial f_{2a}}{\partial x_j} &= 2\omega_{ij}(x_j - y_i) \\
\frac{\partial f_{2a}}{\partial y_j} &= 2\omega_{ij}(y_j - y_i)
\end{align*}
\]

3.4 Third order term

The third order term

\[
f_3 = \sum_{\theta_{ijk} \in B_{adj}} c(\theta_{ijk} - \theta_{0ijk})^2
\]

is the translation of metric (5). Its value depends on the angular resolution of \(G\). Attempting to increase the angular resolution we could (a) identify the smallest angle formed by two bonds at a vertex and attempt to increase it or (b) identify all angles formed by two adjacent bonds at a vertex and attempt to equate them. We chose option (b) since it seemed to be more efficient. With adjacent bonds,
we mean two bonds meeting at a common vertex such that the smallest angle between these bonds does not cross another bond. As an example following this definition, a vertex of degree four has four pairs of adjacent bonds and thus four relevant angles to be equated. Thus, this term depends on the sum of all angles formed by pairs of adjacent bonds in $B_{\text{adj}} \theta_{ijk} = \cos^{-1}(s_{ij} \cdot s_{jk}/d_{ij}d_{jk})$ and the optimal angle $\theta_{0ijk} = 2\pi/n$ between bonds $b_{ij}, b_{jk}$ where $s_{ij}$ displacement vector of $b_{ij}$, $s_{jk}$ is the displacement vector of $b_{jk}$, $d_{ij} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$ is the length of $b_{ij}$, $d_{jk} = \sqrt{(x_k - x_j)^2 + (y_k - y_j)^2}$ is the length of $b_{jk}$ and $n$ is the degree of $v_j$. This factor is weighted by a constant $c$. Its partial derivatives are automatically generated using the Mathematica tool [38].

3.5 Fourth order term

The energy of overlapping bonds seems to be the least intuitive type of energy to define. Our design objectives are to have a resulting expression that is smooth, with no point of singularity on the curve describing the function. Our first attempt of defining this energy was a function of a distance relative to the point of symmetry where the two bonds overlap, similar $g(x) = 1/x$ and has indeed a point of singularity, $x = 0$, which greatly confused the minimizer. The second idea of defining the energy of a certain bond overlap was the scalar product of the cross product of four different vectors depending on the position of the bonds overlapping. This idea seemed promising, but was later discarded in favor of the following solution.

\[
 f_4(G) = \sum_{b_{ij}, b_{kl} \in B} f_4(b_{ij}, l_i) = \begin{cases} 
 f_{4ia}(d_i) = c_{1i}d_i^2, & 0 < d_i < \frac{l_i}{2} \\
 f_{4ib}(d_i, l_i) = -c_{1i}(d_i - \frac{l_i}{2})^2 + c_{2i}, & \frac{l_i}{2} < d_i < \frac{3l_i}{4} \\
 f_{4ic}(d_i, l_i) = c_{1i}(d_i - l_i)^2 & \frac{3l_i}{4} < d_i \leq l_i 
\end{cases}
\]

\[
 c_{1i} = 1/(l_i/2), c_{2i} = f_{4ia}(l_i/2)/2
\]

The above equation expresses the translation of metric (6). The shape of the function determining the value of each summand is a “bump” as it is the composition of three parabolas. There are two values associated with each overlap: (a) the overlapping distance $d_i$ and (b) the combined length of the two bonds $l_i$, the range of $f_4$. The overlapping distance is defined as the combined distance
between the two leftmost vertices of the overlapping bonds and the point of intersection. A visual illustration can be seen in Fig. 4. As seen, the function takes a maximum value when \( d_i = l_i/2 \), that is, when the overlap is symmetrical. In this special case, the gradient is zero, and thus no forces applied to the vertices. However, the vertices are most likely affected by forces from other sources and a configuration of perfect symmetry in this manner should be very rare. The energy from an overlap affects four vertices, each with two partial derivatives. The function has three parts and the number of individual partial derivatives to be computed is \( 4 \times 3 \times 2 = 24 \). The partial derivatives are automatically generated using the Mathematica tool.

Figure 4: (a) displays two overlapping bonds (A, C) and (B, D) as well as the point of intersection X. The overlapping distance in this figure is AX+BX, since A and B are the two leftmost vertices. (b) displays the shape of the function mapping the overlapping distance of a particular overlap to an energy value. The x-axis corresponds to the overlapping distance while the y-axis corresponds to the energy value. Here, we assume the combined length of the two bonds is 2.0 units and the maximum energy 0.5 units.
4 Implementation and Performance

4.1 Overview

We henceforth refer to the implementation of the algorithm as a named procedure \textsc{MINGRAD}(G) where the argument \( G \) contains the data of a graph. The goal is efficiency, modularity and extensibility so that new parts could be added or removed if the underlying design were to be altered. On an abstract level, the algorithm can be described as the composition of four separate procedures.

1. The initial placement procedure
2. The local minimizer
3. The global minimizer
4. The drawing procedure

The original version of \textsc{MINGRAD} was implemented in C \[34\]. Later, the program was translated into Javascript \[28\] using the converting tool Emscripten \[13\] to provide the possibility of running the program locally in a web-browser. The Javascript program includes the drawing procedure of the algorithm described above, producing an actual visualization of a layout generated by the global minimizer. Real world test data was generated from bioinformatics resource DrugBank \[11\] using the Internal Coordinate Mechanics (ICM) language \[27\], and later processed into the JSON \[30\] — the input format of the application. The source code of the implementation is publicly available under \url{https://github.com/lsund/graph-visualization}

4.2 Initial placement

With poor initial configuration, the algorithm most likely settles for a sub-optimal placement regardless of how the algorithm proceeds with minimization. Sub-optimal layouts are sometimes acceptable, bad layouts however, are not. By choosing a smart initial vertex placement, we should be able to avoid bad results. We want an initial placement strategy that efficiently yields a rough approximation of the final layout.
When deciding an initial placement procedure, we keep in mind what we are trying to achieve: a layout adapting to the metrics of section 3.1. Therefore, the chosen procedure should be an attempt of configuring the initial layout to roughly follow the same standards, or at least some of them. By inspection, metric (1), (2) and (4) seems to be solvable with a simple algorithm.

At first, the plan was to place the vertices one by one at the edges at the drawing area, then letting them “slide” towards the middle as if sliding down an elliptic paraboloid. On second thought, this idea seemed complicated and we instead chose to place the vertices as a rectangular spiral with equal spacing between the vertex positions. The center most vertex is the most connected vertex, following the second most connected vertex et cetera. This placement is visualized in Fig. 5 (see also Fig. 8.a). This technique is efficient (as simple iteration of $|V|$ steps assuming the vertices are sorted after degree) and optimally satisfies quality metrics (1), (2) and (4).

![Figure 5: The initial placement and distribution of a sample set of 23 vertices. Black dots represent vertices and numbers represent the indices of the vertices. In this picture, The vertices are ordered so that the vertex with the highest degree has the lowest index. Thus, vertices with many neighbors are placed close to the center. Bonds are not displayed in this picture.](image-url)
4.3 Local Minimizer

4.3.1 Energy function

MINGRAD implements a way of computing the value of the energy function and its gradient according to the specification in section 3. The two procedures \texttt{CalculateEnergy}(G) and \texttt{CalculateGradient}(G) accepts a single argument \texttt{G}, representing a graph of a particular state. \texttt{CalculateEnergy} returns a scalar value representing \( e_c \) and \texttt{CalculateGradient} returns sequence of length \( 2|V| \) representing \( \nabla f \). In the C version of the program, a pointer to this sequence is returned, which the user is responsible for freeing after use. The argument \texttt{G} is a composite data structure containing information about the vertices, the bonds, the drawing area, and other metadata. The vertices and bonds are essentially lists of references to basic structures. During execution, the state of the graph is examined by \texttt{CalculateEnergy} and \texttt{CalculateGradient} by traversing these lists, and values for energy and gradient are accumulated.

An interesting part of the implementation is \( f_{2r} \) – the term for repulsion between vertices. We defined \( f_{2r} \) as a function of a sum of squared distance between pairs of vertices. In the case this distance exceed a certain limit, we approximate the repulsion value for that particular vertex pair to zero. Since summing zero’s is redundant, we rather not consider pairs separated by a large distance at all. Thus, we introduce the notion of vertex neighborhood, giving the algorithm knowledge of which vertices are at a close distance without the need of examining every pair.

To maintain vertex neighborhoods, the drawing area is partitioned into fixed rectangular cells in such a way that if all vertices were distributed perfectly even in the drawing area, each cell would contain a constant small number of vertices. Thus, the set of neighbors of a given vertex is thus the vertices in its own cell, and the vertices in the cells adjacent to its own cell (see Fig. 6). Since assigning the set of vertices to cells only requires a single iteration over \( V \) and largely reduces the number of repulsing vertex pairs, this construction reduces the overall number of computations.
Figure 6: The algorithm divides the drawing area is partitioned into rectangular cells and the neighbors of a vertex is those vertices that lie in the same cell or in an adjacent cell. In this example, the orange vertices are considered neighbors to the red vertex. Note that not all vertices of the graph are shown in this figure.
4.3.2 Minimizer

The second part of MINGRAD is the local minimizer. We provide pseudocode and comments for the essential routines of this algorithm below, listed in ascending order by the number of times called during execution.

```
procedure LinearMinimization(
    structure graph g,
    array grad,
    functionPointer f(structure graph)
) do
    [p1,p2,p3] = call BracketMinimum(g,f)
    [fmin,xmin] = call IsolateMinimum(g,f,p1,p2,p3)
    call MoveVertices(g, grad, xmin)
    return fmin
end procedure

procedure RunLocalMinimizer(String fileName) do
    structure graph g = InitializeGraph(fileName)
    for N iterations do
        fmin = call LinearMinimization(g,grad,CalculateEnergy);
        if (call CloseToTarget(fmin,e)) do
            end for
        end if
        e = call CalculateEnergy(g)
        grad = call CalculateGradient(g)
    end for
    return call GetVertexPositions(g)
end procedure
```

1. RunLocalMinimizer at line 13 is the main procedure. It computes a set of coordinates representing a minimized graph layout given a textual string representing the path to the file containing graph data.

2. InitializeGraph at line 14, parses the file containing graph data and loads internal data-structures.

3. GetVertexPositions at line 23 returns a sequence of numbers, representing the final coordinates of the vertices in two dimensional space.

4. LinearMinimization at line 2 performs linear minimization on a procedure representing the energy function, given a graph and a gradient of the
energy function.

5. **CloseToTarget** at line 17 is true if the new smallest value of the energy function is close to the last value.

6. **BracketMinimum** at line 7 performs a golden section search, returning a triplet points that bracket the minimum of the provided function.

7. **IsolateMinimum** at line 10, uses Brent’s method to compute and return both the minimum and the abscissa of the minimum of the provided function.

8. **MoveVertices** at line 11 updates the positions of the vertices of the graph, moving them according to the gradient and the distance to the minimum.

### 4.4 Global Minimizer

The global minimizer is inspired by simulated annealing and Metropolis-Hastings algorithm [41], both methods known for their usefulness in large search spaces and large dimensions. The global minimizer is the top level part of the minimizer and uses the local minimizer in search for an optimal graph layout. More specifically, the global minimizer searches the function “landscape” by performing a series of moves and minimizations, deciding after each iteration whether to keep or discard the new minimized layout (see Fig. 7). If the algorithm is entirely greedy, that is only accepting new layouts that have a lower energy than the last layout, it might be hard to break out and discover new local minima. We want to allow the algorithm to explore the surroundings of the current minimal value, which implies sometimes accepting an inferior layout to the current.

We introduce the concept of temperature – a manually adjustable measure for how likely it is that the global minimizer will accept an inferior layout. More specifically, the algorithm accepts a new layout if the following expression evaluates to true.

\[
(c_{e1} < c_{e0}) \lor ((c_{e1} \geq c_{e0}) \land (r < e^{(c_{e1} - c_{e0})/t}))
\]

\(c_{e0}\) is the energy value of the previous layout, \(c_{e1}\) is the energy value of the new layout, \(e\) is the natural number and \(r\) is a random number between 0 and
1. A move performed by the global minimizer implies changes to the positions of a chosen set of vertices. Only random distance movement is supported, more informed operations are listed as a future improvement in section 7.

Figure 7: A flow chart of the global minimizer. Starting with some initial layout, this layout is repetitively mutated and minimized for a predetermined number of iterations. When the end condition is met, the algorithm returns what is considered the best layout.

4.5 Complexity

4.5.1 Local Minimizer

With $|V|$ as $n$ we refer to the time taken by CalculateEnergy as $t_f(n) = t_{f1}(n) + t_{f2}(n) + t_{f3}(n) + t_{f4}(n)$ where $t_{f_k}$ is the time taken by the $k$'th part of $f$. $f_1$ requires a simple iteration over the vertices of the graph, hence $t_{f1}(n) \in O(n)$. The implementation of $f_2$ requires the computation of two separate procedures. The first procedure iterates over the bonds of the graph, while the second iterates over groups of neighboring vertices. Remembering that $|V| \approx |B|$, iterating over the bonds of the graph requires $O(n)$ computations. Assuming that the vertices are fairly evenly spread in the drawing area (which indeed is the case
for the majority of intermediate layouts handled by the algorithm) the second procedure scales linearly with $n$ as well. Hence, $t_{f2}(n) \in O(n + n) \in O(n)$.

$f_3$ and $f_4$ are more complex to compute with our implementation. Both the set of bond pairs that forms minimum angles and the set of overlapping bond pairs can change during the minimization process and needs thus be recalculated repetitively. The number of calculations for computing these sets are bound by $O(2(n(n - 1)/2)) \in O(n^2)$. Since this computation is costly, we should avoid it if possible. Thus, we should only recompute these sets when vertices move a significant distance. In fact, our experiments indicate that we only need recomputing of these sets the first few iterations of the minimization algorithm. Even if this is a pleasant observation, we cannot disregard the overall complexity of the terms $t_{f3}$ and $t_{f4}$ is bound by $O(n^2)$. The total time complexity for CalculateEnergy is thus $O(3n + n^2) \in O(n^2)$ and so is the complexity of CalculateGradient.

With $f$ roughly approximated as a quadratic form $f(x) = \frac{1}{2}x^TAx + x^Tb + c$ (section 2.8), the number of unknown parameters in $f$ is equal to the number of free parameters in $A$ and $b$ which is of order $n^2$. Changing any of these parameters could move the location of the minimum. We should therefore not expect to find the minimum of $f$ before having collected information content corresponding to numbers of order $n^2$. A possibility would be to perform $n^2$ separate line minimizations to achieve the desired result, but the conjugate gradient method is more efficient. By wisely using the $n$ new components of information yielded by each evaluation of the gradient, our implementation lets us perform only $n$ line minimizations calling CalculateEnergy and CalculateGradient. Thus, the overall time complexity of the local minimizer $t_{lm}(n) \in O(nm^2) \in O(n^3)$.

4.5.2 Global Minimizer

The global minimizer performs local minimization a predefined number of times $k$. Since it moves the vertices small distances, the altered layouts are likely close to minimized should only require small number of iterations to re-minimize, closer to 1 rather than $n$, the common case for the first local minimization of a layout. Nevertheless, the average time complexity of the global minimizer is
$O(kn^3)$, and $O(kn^4)$ in the worst case. The algorithm has a space complexity of $O(n^2)$, the type allocations are either linear with the number of vertices or cubic in the cases of bond pairs.

5 Experiments and Results

5.1 Overview

This section contains a series of examples demonstrating features of MINGRAD and the layouts it produces. Even if we would like to associate a produced layout to a measure of success, finding objectives of this kind is difficult. While various layouts of the same graph may have different intermediate and final energy-values, it is not certain that an inferior layout always have a higher final energy than a superior layout. Because energy values may change depending on different definitions and configurations of the algorithm, using the measure of energy to compare the quality of different layouts should be done carefully. In this section, we only compare the energy-value of layouts if the layouts were produced using the same settings and definitions. Also, since the minimum possible energy value is zero, absolute zero is a relevant value of reference. The graphs presented in this section and Appendix A (except the ones in Fig. 8, Fig. 9 and Fig. 11) are taken from DrugBank [11], and represents molecules clustered by similarity.

5.2 Sample Visualizations

Fig. 8 displays four intermediate steps of the placement algorithm. Here, MINGRAD finds a planar drawing of the graph, although no priori knowledge of its planarity. This graph is more dense than the typical graphs of our experiments, but it demonstrates some properties of force directed drawing. As seen, the final layout is of good quality. It is also highly symmetric, thus this result should be taken as evidence that our choices of drawing aesthetics are of relevance to symmetric drawings.

One trait not illustrated by Fig. 8 but common to force directed al-
Figure 8: Four intermediate placements of a graph with 19 vertices. (a) displays the initial layout of the graph. (b) displays the layout after 6 iterations. (c) displays the layout after 12 iterations. (d) displays the layout after 22 iterations.
gorithms is flexibility. Our algorithm is indeed flexible in the way that we can adjust the weights of the individual terms to obtain different results. An example of a simple graph is shown in Fig. 9. As seen, the layout of the default weight setting in (a) has two overlaps and resembles a three dimensional box. In (b) the value of bond overlap is increased, resulting in a planar drawing. In addition to illustrating the effect of changing the weights of the energy function, this exemplifies the difficulty of fulfilling several aesthetics criteria at the same time. In (a), the vertex spacing and edge lengths are uniform while (b) has minimized bond overlap. While (b) is preferable in this case, configurations biased towards uniform bond length and vertex distributions seems to generally contribute to the best results.

Figure 9: Different layouts of the same graphs with different weight configurations. (a) shows the resulting layout with the default configuration, while (b) shows the resulting layout when the bond overlap term is more dominant.

By setting the weights of angular resolution and bond overlap to zero, we may observe the overall influence of angular resolution and bond overlap. Modest values of these higher order terms mostly produces layouts with a more pleasing layout compared to configurations completely excluding these measures. Fig. 10 illustrates a graph visualized with and without certain elements. (a) displays the visualization of the graph without higher order terms. (c) displays the graph with the inclusion third order term and the fourth order term. (a) has 18 overlaps while (c) has 9 overlaps. In addition, the angular resolution of (c) is 5% higher than the angular resolution of (a).
In Fig. 10, we managed to find a pretty good weight balance. Badly configured weight values seems to confuse the algorithm more, resulting in poor quality visualizations. In particular, angular resolution is a difficult measure to set. The types of small improvements from increase angular resolution is exemplified in Fig. 11. Here, the same graph is displayed in two different visualizations – (a) has a slightly lower influence of angular resolution than (b).

Figure 10: Different layouts of a graph with 52 vertices. (a) shows a minimized layout of the graph where the terms for angular resolution and bond overlap were set to zero. (b) shows a minimized layout of the graph where only the term for bond overlap were set to zero. (c) shows a minimized layout of the graph where no term was set to zero and (d) shows the layout after global minimization.
5.3 Global minimization

Our experiments indicate that the energy most layouts can be improved by the global minimizer. An example is displayed in Fig. 10 (d). The layout is superior to (c) which is the best layout obtained without global minimization. After global minimization, the graph had 7 overlaps instead of 9, a 7% decreased energy and a 2% increased angular resolution. Also, by inspection we conclude this layout to be the better of the four layouts of Fig. 10. With only 10 iterations of the global minimizer, we obtained an average of 15% improvement of the energy value for a set of 19 test graphs of different sizes. With 20 iterations, the average improvement was 19%. In addition to the lowered energy, many layouts had a more natural layout after global minimization. Contradicting our theory regarding greedy and non-greedy minimization in section 4.4 the global minimizer seemed to perform best in a greedy manner with no exploration. We experimented with various types of stochastic heuristics, but the best configuration for a selection and modification was found is as follows: Choose a random vertex subset of size $|V|/5$, generate a set of vectors $(x_i, y_i)$ of magnitude $m$ and move each individual vertex according to the corresponding vector. $m$ is a random number between zero and the padding constant between vertices.
5.4 Bond time complexity

Profiling the program using GNU gprof [23] indicates that computing bond overlap requires a substantial amount of computation. In fact, computing which bonds are overlapping corresponds to 32% of the CPU time required by the program. The second most computation intensive aesthetic to ensure is repulsion between vertices, which sums up to 12% of the CPU time required. The full profiling is presented in Appendix B.

6 Discussion and Conclusions

In contrast to the influential study by D&H, we did not choose simulated annealing as our minimization technique. Even though simulated annealing statistically guarantee the optimal solution, it is a rather slow method [1,10]. We wanted an efficient algorithm and thus attempted the conjugate gradient method. Conjugate gradient methods have been used before, for instance by Hu’s powerful multilevel algorithm [26]. As far as we know though, no study has attempted to use conjugate gradient methods to minimize a layout by defining the energy in continuous functions of sharp angles and bond overlap.

As made evident in section 5, angular resolution did not improve our algorithm convincingly. The boundary between beneficial and disadvantageous values of the angular resolution weight is very delicate. The visual improvements from an increased angular resolution was very slight, and it is highly debatable if this term should be included in the algorithm at all. Without drawing strict conclusions, we theorize on the reasons why the angular resolution term failed to contribute to better layouts.

1. Our assumption regarding the importance of angular resolution could be wrong. As mentioned, most studies on angular resolution treated the measure in a specific environment. In fact, the graphs (both in prior and this study) benefiting from angular overlap seems to be the more symmetrical type, different from the typical target graph of this application.

2. The design and implementation of the third term of the energy function could be of poor quality. Designing the energy from connected bonds as
the sum of squared angles seemed to be the most natural and reason-
able approach and not much resources was devoted into researching other
alternatives.

We suspect our algorithm would have benefited more of a term similar
to the Node-edge distance term of D&H. Considering bond overlap, we have
seen the computational difficulty both in theory and practice. We have also
seen examples of limitations of our design, the function having an extremum
and thus a zero-gradient when a bonds overlap is symmetrical. Nevertheless,
we achieved good results from bond overlap reduction. This technique almost
always reduced the amount of bonds overlapping, resulting in a lower energy,
better looking visualization. Through this, it is demonstrated that FDA can
adjust to any kind of force, not only the ones observed in nature.

Implementation wise, the earlier mentioned study by Fruchterman and
Reingold is an important source of inspiration for this work. For instance, we
adopted the idea of a grid based layout where each vertex is assigned to a
neighborhood to increase the speed of calculating repulsive forces. In contrast,
we invented the process of a predetermined initial layout without any external
influence. We have not been able to locate a prior study considering the process
of initial configuration of importance. For example D&H used a random initial
vertex configuration. In our case, the predefined initial configuration greatly
improved the quality of the output compared to a random initial configuration.

We used the programming language C as we wanted a simple, portable
and efficient language for the implementation. C turned out a decent choice, but
as the program grew, there was a difficulty of naturally separating the logic in
an purely imperative manner. As the program essentially represents a group of
of objects interacting, we ultimately had to introduce object-like structures. We
suspect a language with more support of object oriented programming efficient
such as C++ or Java better choices for this kind of application.

In October 2015, the source code of this project was merged into the
code base of Molsoft L.L.C [42], a company that provides software for molecular
visualization, among other products.
7 Future work

7.1 Scalability

Scalability was one of the main goals of this work. As made apparent in previous sections, this goal was not completely fulfilled. The time complexity of the algorithm is $O(n^3)$, $n = |V|$, making it scale rather poorly. On a particular machine, graphs of size 100 requires 3–4 seconds to compute, not the performance efficiency we had in mind. To achieve the goal of a scalability, we suggest two major extensions to the current solution. The first improvement would be to include data structures for parallel computation for CalculateEnergy and CalculateGradient. Since the members of the graph are independent during iterations, we could likely compute separate parts of energies and gradients simultaneously in order to increase throughput.

The second change affects the overall structure of the algorithm. Similar to the multilevel approach proposed by Hu, we would like to extend the algorithm to support a hierarchical sequence of minimizations and placements, affecting groups of data points instead of minimizing the entire set simultaneously. In this manner, a vertex would no longer correspond to a specific data point, but instead one of the data point groups. The algorithm would be capable of automatically clustering the data points and assigning the cluster to an appropriate minimization process. As the proposed change is purely a structural enlargement and not a redesign of the fundamental parts of the algorithm, it should be achievable in reasonable time.

7.2 Global Minimizer

While our stochastic methods for global optimization have yielded some good results, there are many other heuristics that possibly could improve optimization process. We could for instance select vertices according to some kind of fitness, instead of randomly. Perhaps this fitness could depend on the individual energy value or the number of bond overlap from a certain vertex. Genetic algorithms also inspire for new types of moves, for example swapping positions between vertices or generating new positions depending on the combination of
old positions. The global minimizer could also benefit from concurrency. Using multiple starting points for the minimizing procedure is a different approach for global optimization.

8 Acknowledgments

I would like to thank Ruben Abagyan for initiating and supervising the project, and Eugene Rausch for provided help and support when implementing the algorithm. My gratitude to Carl Nettelblad as he played an important role as reviewer.
Appendix

A. Sample Layouts

Figure 12: 37 vertices
Figure 13: 39 vertices
Figure 14: 42 vertices
Figure 15: 44 vertices
Figure 16: 69 vertices
B. Sample Run Profiling Listing

Each sample counts as 0.01 seconds.

<table>
<thead>
<tr>
<th>% cumulative</th>
<th>self seconds</th>
<th>self seconds calls</th>
<th>s/call</th>
<th>s/call name</th>
</tr>
</thead>
<tbody>
<tr>
<td>32.10</td>
<td>3.22</td>
<td>3.22</td>
<td>140208534</td>
<td>0.00 BondPair_intersect</td>
</tr>
<tr>
<td>8.38</td>
<td>4.06</td>
<td>0.84</td>
<td>92847</td>
<td>0.00 second_order_repulsion_energy</td>
</tr>
<tr>
<td>6.98</td>
<td>4.76</td>
<td>0.70</td>
<td>113691157</td>
<td>0.00 VertexPair_repulsion_energy</td>
</tr>
<tr>
<td>6.38</td>
<td>5.40</td>
<td>0.64</td>
<td>280528556</td>
<td>0.00 BondSet_get_bond</td>
</tr>
<tr>
<td>5.58</td>
<td>5.96</td>
<td>0.56</td>
<td>88612</td>
<td>0.00 Graph_detect_overlapping_bonds</td>
</tr>
<tr>
<td>Time</td>
<td>Value1</td>
<td>Value2</td>
<td>Value3</td>
<td>Value4</td>
</tr>
<tr>
<td>------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>4.14</td>
<td>6.38</td>
<td>0.42</td>
<td>128093599</td>
<td>0.00</td>
</tr>
<tr>
<td>3.39</td>
<td>6.72</td>
<td>0.34</td>
<td>258287706</td>
<td>0.00</td>
</tr>
<tr>
<td>2.04</td>
<td>6.92</td>
<td>0.21</td>
<td>6324776</td>
<td>0.00</td>
</tr>
<tr>
<td>1.99</td>
<td>7.12</td>
<td>0.20</td>
<td>107320416</td>
<td>0.00</td>
</tr>
<tr>
<td>1.89</td>
<td>7.31</td>
<td>0.19</td>
<td>88612</td>
<td>0.00</td>
</tr>
<tr>
<td>1.40</td>
<td>7.45</td>
<td>0.14</td>
<td>140271604</td>
<td>0.00</td>
</tr>
<tr>
<td>1.30</td>
<td>7.58</td>
<td>0.13</td>
<td>92847</td>
<td>0.00</td>
</tr>
<tr>
<td>1.10</td>
<td>7.69</td>
<td>0.11</td>
<td>169037</td>
<td>0.00</td>
</tr>
<tr>
<td>0.95</td>
<td>7.79</td>
<td>0.10</td>
<td>18340571</td>
<td>0.00</td>
</tr>
<tr>
<td>0.90</td>
<td>7.88</td>
<td>0.09</td>
<td>169037</td>
<td>0.00</td>
</tr>
<tr>
<td>0.80</td>
<td>7.96</td>
<td>0.08</td>
<td>48769867</td>
<td>0.00</td>
</tr>
<tr>
<td>0.80</td>
<td>8.04</td>
<td>0.08</td>
<td>169037</td>
<td>0.00</td>
</tr>
<tr>
<td>0.70</td>
<td>8.11</td>
<td>0.07</td>
<td>169037</td>
<td>0.00</td>
</tr>
<tr>
<td>0.60</td>
<td>8.17</td>
<td>0.06</td>
<td>118019340</td>
<td>0.00</td>
</tr>
<tr>
<td>0.60</td>
<td>8.23</td>
<td>0.06</td>
<td>18344234</td>
<td>0.00</td>
</tr>
<tr>
<td>0.60</td>
<td>8.29</td>
<td>0.06</td>
<td>169037</td>
<td>0.00</td>
</tr>
<tr>
<td>0.60</td>
<td>8.35</td>
<td>0.06</td>
<td>169037</td>
<td>0.00</td>
</tr>
<tr>
<td>0.60</td>
<td>8.41</td>
<td>0.06</td>
<td>796</td>
<td>0.00</td>
</tr>
<tr>
<td>0.60</td>
<td>8.47</td>
<td>0.06</td>
<td>1</td>
<td>0.06</td>
</tr>
<tr>
<td>0.60</td>
<td>8.53</td>
<td>0.06</td>
<td>22496507</td>
<td>0.00</td>
</tr>
<tr>
<td>0.55</td>
<td>8.58</td>
<td>0.06</td>
<td>22496507</td>
<td>0.00</td>
</tr>
<tr>
<td>0.50</td>
<td>8.63</td>
<td>0.05</td>
<td>724389</td>
<td>0.00</td>
</tr>
<tr>
<td>0.50</td>
<td>8.68</td>
<td>0.05</td>
<td>169037</td>
<td>0.00</td>
</tr>
<tr>
<td>0.45</td>
<td>8.73</td>
<td>0.05</td>
<td>41142407</td>
<td>0.00</td>
</tr>
<tr>
<td>0.40</td>
<td>8.77</td>
<td>0.04</td>
<td>41532012</td>
<td>0.00</td>
</tr>
<tr>
<td>0.40</td>
<td>8.81</td>
<td>0.04</td>
<td>13039002</td>
<td>0.00</td>
</tr>
<tr>
<td>0.40</td>
<td>8.85</td>
<td>0.04</td>
<td>3900439</td>
<td>0.00</td>
</tr>
<tr>
<td>0.40</td>
<td>8.89</td>
<td>0.04</td>
<td>724389</td>
<td>0.00</td>
</tr>
<tr>
<td>0.40</td>
<td>8.93</td>
<td>0.04</td>
<td>169037</td>
<td>0.00</td>
</tr>
<tr>
<td>0.40</td>
<td>8.97</td>
<td>0.04</td>
<td>92847</td>
<td>0.00</td>
</tr>
<tr>
<td>0.35</td>
<td>9.00</td>
<td>0.04</td>
<td>6051493</td>
<td>0.00</td>
</tr>
<tr>
<td>0.35</td>
<td>9.04</td>
<td>0.04</td>
<td>853</td>
<td>0.00</td>
</tr>
<tr>
<td>0.30</td>
<td>9.07</td>
<td>0.03</td>
<td>22674784</td>
<td>0.00</td>
</tr>
<tr>
<td>0.30</td>
<td>9.10</td>
<td>0.03</td>
<td>6324776</td>
<td>0.00</td>
</tr>
<tr>
<td>0.30</td>
<td>9.13</td>
<td>0.03</td>
<td>724389</td>
<td>0.00</td>
</tr>
<tr>
<td>0.30</td>
<td>9.16</td>
<td>0.03</td>
<td>724389</td>
<td>0.00</td>
</tr>
<tr>
<td>0.30</td>
<td>9.19</td>
<td>0.03</td>
<td>92847</td>
<td>0.00</td>
</tr>
<tr>
<td>0.30</td>
<td>9.22</td>
<td>0.03</td>
<td>38842</td>
<td>0.00</td>
</tr>
<tr>
<td>0.30</td>
<td>9.25</td>
<td>0.03</td>
<td>38842</td>
<td>0.00</td>
</tr>
<tr>
<td>0.30</td>
<td>9.28</td>
<td>0.03</td>
<td>34821</td>
<td>0.00</td>
</tr>
<tr>
<td>0.30</td>
<td>9.31</td>
<td>0.03</td>
<td>34821</td>
<td>0.00</td>
</tr>
<tr>
<td>0.30</td>
<td>9.34</td>
<td>0.03</td>
<td>19</td>
<td>0.00</td>
</tr>
<tr>
<td>0.30</td>
<td>9.37</td>
<td>0.03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.20</td>
<td>9.39</td>
<td>0.02</td>
<td>16551419</td>
<td>0.00</td>
</tr>
<tr>
<td>0.20</td>
<td>9.41</td>
<td>0.02</td>
<td>12980994</td>
<td>0.00</td>
</tr>
<tr>
<td>0.20</td>
<td>9.43</td>
<td>0.02</td>
<td>9007632</td>
<td>0.00</td>
</tr>
<tr>
<td>0.20</td>
<td>9.45</td>
<td>0.02</td>
<td>8501064</td>
<td>0.00</td>
</tr>
<tr>
<td>0.20</td>
<td>9.47</td>
<td>0.02</td>
<td>6051493</td>
<td>0.00</td>
</tr>
<tr>
<td>0.20</td>
<td>9.49</td>
<td>0.02</td>
<td>4346334</td>
<td>0.00</td>
</tr>
<tr>
<td>0.20</td>
<td>9.51</td>
<td>0.02</td>
<td>4346334</td>
<td>0.00</td>
</tr>
<tr>
<td>0.20</td>
<td>9.53</td>
<td>0.02</td>
<td>4328183</td>
<td>0.00</td>
</tr>
<tr>
<td>0.20</td>
<td>9.55</td>
<td>0.02</td>
<td>4328183</td>
<td>0.00</td>
</tr>
<tr>
<td>0.20</td>
<td>9.57</td>
<td>0.02</td>
<td>88612</td>
<td>0.00</td>
</tr>
<tr>
<td>0.20</td>
<td>9.59</td>
<td>0.02</td>
<td>88260</td>
<td>0.00</td>
</tr>
<tr>
<td>0.20</td>
<td>9.61</td>
<td>0.02</td>
<td>38842</td>
<td>0.00</td>
</tr>
<tr>
<td>0.20</td>
<td>9.63</td>
<td>0.02</td>
<td>38842</td>
<td>0.00</td>
</tr>
<tr>
<td>0.20</td>
<td>9.65</td>
<td>0.02</td>
<td>38842</td>
<td>0.00</td>
</tr>
<tr>
<td>0.20</td>
<td>9.67</td>
<td>0.02</td>
<td>34821</td>
<td>0.00</td>
</tr>
<tr>
<td>0.20</td>
<td>9.69</td>
<td>0.02</td>
<td>34821</td>
<td>0.00</td>
</tr>
<tr>
<td>0.20</td>
<td>9.71</td>
<td>0.02</td>
<td>3757</td>
<td>0.00</td>
</tr>
<tr>
<td>0.15</td>
<td>9.72</td>
<td>0.02</td>
<td>7028069</td>
<td>0.00</td>
</tr>
<tr>
<td>0.15</td>
<td>9.74</td>
<td>0.02</td>
<td>4680378</td>
<td>0.00</td>
</tr>
<tr>
<td>0.10</td>
<td>9.75</td>
<td>0.01</td>
<td>19064960</td>
<td>0.00</td>
</tr>
<tr>
<td>0.10</td>
<td>9.76</td>
<td>0.01</td>
<td>6568340</td>
<td>0.00</td>
</tr>
<tr>
<td>0.10</td>
<td>9.77</td>
<td>0.01</td>
<td>4548391</td>
<td>0.00</td>
</tr>
<tr>
<td>0.10</td>
<td>9.78</td>
<td>0.01</td>
<td>4346334</td>
<td>0.00</td>
</tr>
<tr>
<td>0.10</td>
<td>9.79</td>
<td>0.01</td>
<td>4240754</td>
<td>0.00</td>
</tr>
<tr>
<td>0.10</td>
<td>9.80</td>
<td>0.01</td>
<td>3900439</td>
<td>0.00</td>
</tr>
<tr>
<td>0.10</td>
<td>9.81</td>
<td>0.01</td>
<td>3900439</td>
<td>0.00</td>
</tr>
<tr>
<td>0.10</td>
<td>9.82</td>
<td>0.01</td>
<td>3899643</td>
<td>0.00</td>
</tr>
<tr>
<td>0.10</td>
<td>9.83</td>
<td>0.01</td>
<td>724389</td>
<td>0.00</td>
</tr>
<tr>
<td>0.10</td>
<td>9.84</td>
<td>0.01</td>
<td>724389</td>
<td>0.00</td>
</tr>
<tr>
<td>Time</td>
<td>Duration</td>
<td>User</td>
<td>Size</td>
<td>CPU</td>
</tr>
<tr>
<td>-------</td>
<td>----------</td>
<td>-------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>10481</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>10481</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>10481</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>7600</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>7326</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>3757</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>3757</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>3757</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>3757</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>3757</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>3757</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>3757</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>3757</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>3757</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>3757</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>3757</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>3663</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>3548</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>2696</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>2478</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>2090</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>1520</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>959</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>959</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>815</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>796</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>469</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>406</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>355</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>209</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>209</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>190</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>190</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>190</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>190</td>
<td>0.00</td>
</tr>
<tr>
<td>0.00</td>
<td>10.03</td>
<td>0.00</td>
<td>19</td>
<td>0.00</td>
</tr>
</tbody>
</table>
% the percentage of the total running time of the
time program used by this function.

cumulative a running sum of the number of seconds accounted
seconds for by this function and those listed above it.

self the number of seconds accounted for by this seconds function
alone. This is the major sort for this listing.

calls the number of times this function was invoked, if
this function is profiled, else blank.

self   the average number of milliseconds spent in this ms/call function per call, if this function is profiled, else blank.

total  the average number of milliseconds spent in this ms/call function and its descendents per call, if this function is profiled, else blank.

name   the name of the function. This is the minor sort for this listing. The index shows the location of the function in the gprof listing. If the index is in parenthesis it shows where it would appear in the gprof listing if it were to be printed. Copyright (C) 2012-2014 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification, are permitted in any medium without royalty provided the copyright notice and this notice are preserved.
List of Figures

Fig. 1 (p. 4): Examples of data visualization. (a) displays a large set of data visualized with Gephi. (b) displays a protein to protein network visualized with Cytoscape. For image sources, see List of Figures. From: http://thepolishedmommy.com/bryan/wp-content/uploads/2012/02/pic.png, http://gmdd.shgmo.org/Computational-Biology/ANAP/ANAP_V1.1/help/anap-userguide/figures/anap.png

Fig. 2 (p. 10): A force directed graph layout. With the right parameters, force directed layouts in two dimensions often resemble the projection a three dimensional layout in the plane. For image source, see List of Figures. From: [35]

Fig. 3 (p. 11): Lombardi drawing of a graph, each bond is a circular arc and every vertex has perfect angular resolution. For image source, see List of Figures. From: [35]

Fig. 4 (p. 22): (a) displays two overlapping bonds (A, C) and (B, D) as well as the point of intersection X. The overlapping distance in this figure is AX+BX, since A and B are the two leftmost vertices. (b) displays the shape of the function mapping the overlapping distance of a particular overlap to an energy value. The x-axis corresponds to the overlapping distance while the y-axis corresponds to the energy value. Here, we assume the combined length of the two bonds is 2.0 units and the maximum energy 0.5 units..

Fig. 5 (p. 24): The initial placement and distribution of a sample set of 23 vertices. Black dots represent vertices and numbers represent the indices of the vertices. In this picture, The vertices are ordered so that the vertex with the highest degree has the lowest index. Thus, vertices with many neighbors are placed close to the center. Bonds are not displayed in this picture..

Fig. 6 (p. 26): The algorithm divides the drawing area is partitioned into rectangular cells and the neighbors of a vertex is those vertices that lie in the same cell or in an adjacent cell. In this example, the orange vertices are considered neighbors to the red vertex. Note that not all vertices of the graph are shown in this figure.
Fig. 7 (p. 29): A flow chart of the global minimizer. Starting with some initial layout, this layout is repetively mutated and minimized for a predetermined number of iterations. When the end condition is met, the algorithm returns what is considered the best layout.

Fig. 8 (p. 32): Four intermediate placements of a graph with 19 vertices. (a) displays the initial layout of the graph. (b) displays the layout after 6 iterations. (c) displays the layout after 12 iterations. (d) displays the layout after 22 iterations.

Fig. 9 (p. 33): Different layouts of the same graphs with different weight configurations. (a) shows the resulting layout with the default configuration, while (b) shows the resulting layout when the bond overlap term is more dominant.

Fig. 10 (p. 34): Different layouts of a graph with 52 vertices. (a) shows a minimized layout of the graph where the terms for angular resolution and bond overlap were set to zero. (b) shows a minimized layout of the graph where only the term for bond overlap were set to zero. (c) shows a minimized layout of the graph where no term was set to zero and (d) shows the layout after global minimization.

Fig. 11 (p. 35): Visualization of a graph in tree form. (a) displays the visualization with a zero weight for angular resolution. (b) displays the visualization with a modest weight for angular resolution. The red circle highlights an area that was improved after including angular resolution.

Fig. 12 (p. 40): 37 vertices.

Fig. 13 (p. 41): 39 vertices.

Fig. 14 (p. 42): 42 vertices.

Fig. 15 (p. 43): 44 vertices.

Fig. 16 (p. 44): 69 vertices.

Fig. 17 (p. 45): 95 vertices.
References


