Karlsruhe Institute of Technology

Schematic Graph-Based Layout of Secondary Protein Structures

Bachelor Thesis of

Eric Sallermann

At the Department of Informatics
Institute of Theoretical Computer Science

Reviewers:	Dr. Martin Nöllenburg
	Prof. Dr. Peter Sanders
Advisors:	Dr. Martin Nöllenburg
	Dipl.-Inform. Benjamin Niedermann

Time Period: 01st July 2014 – 31th October 2014

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, 31st October 2014

Abstract

Proteins are part of all cells of living organisms. Therefore, they are the focus of various research, especially those aiming to get a better understanding of the human body. The spatial structure of a protein is examined using four distinctive structure layers. This bachelor thesis particularly deals with the secondary structure layer, the so-called secondary structure of the protein. We model the protein as a graph and formulate a mixed-integer program with the goal to automatically generate a schematic representation of the secondary structure of a protein. We define hard constraints which the representation has to fulfil and soft constraints, which affect its visual aspects. These constraints are implemented using the interface of the optimiser Gurobi. Then they are evaluated regarding their impact on the representation using two of our given graphs. At last, we show computed schematic representations of various proteins and compare these to already existing, but manually drawn ones. We also discuss why the mixed-integer program fails to generate representations for some proteins and which improvements can be made to speed up the computation.

Deutsche Zusammenfassung

Proteine sind Bestandteil aller Zellen lebender Organismen. Sie sind daher der Schwerpunkt vieler Untersuchungen, besonders derjeniger, in denen es um das bessere Verständnis des menschlichen Körpers geht. Die räumliche Struktur eines Proteins wird auf verschiedenen Ebenen betrachtet. Diese Bachelorarbeit beschäftigt sich insbesondere mit der zweiten Ebene, der sogenannten Sekundärstruktur des Proteins. Wir modellieren das Protein als Graph und stellen anhand dieses Modells ein gemischt-ganzzahliges Programm auf, mit dem Ziel, eine schematische Darstellung der Protein-Sekundärstruktur automatisch zu erzeugen. Wir definieren notwendige Kriterien, die die Darstellung erfüllen muss, und hinreichende, welche sich auf die visuellen Aspekte jener auswirken. Diese Kriterien implementieren wir für die Schnittstelle des Optimierers Gurobi. Danach folgt eine Evaluation ihrer Auswirkung auf die Darstellung mit Hilfe zweier von uns selbst vorgegebenen Graphen. Schließlich zeigen wir die von uns berechneten schematischen Darstellungen einiger Proteine und vergleichen diese mit bereits existierenden, die jedoch manuell gezeichnet sind. Wir diskutieren zudem, warum das gemischt-ganzzahlige Programm die Darstellung einiger Proteine nicht generiert und welche Verbesserungen vorgenommen werden können, um die Berechnung zu beschleunigen.

Contents

1. Introduction

Bioinformatics as interdisciplinary science aims to find solutions to problems in biology with the aid of computers. One field of research is the theoretical analysis and modification of proteins as well as the simulation of the behaviour of these modified proteins. Protein modification is used, for example, "for the production of protein therapeutic products and use of proteins as plastics and adhesives".[Lun14] Biobased polymer, also known as bioplastic, is used in the manufacturing of casings for personal computers and provides less of an environmental burden than other plastics.[Hor05]

To find these modifications it is first necessary to analyse proteins and identify secondary protein structures. The three-dimensional protein model contains the placement of a proteins chemical atoms. Figure 1.1 shows such a model, in which atoms are represented as coloured sticks. It is also possible to combine the atoms into residues and illustrate how these residues form structures in the protein. This abstraction can be seen in Figure 1.2. It is taken from the same perspective as Figure 1.1.

The advantage of the three-dimensional protein model is that there are many existing programs like PyMOL [Sch10b], Jmol [Han14] and BallView [MHLK05], which we used to create these screenshots. Using these programs, we can view and modify the protein and run simulations on it. Another advantage is that the spatial arrangement of the atoms is preserved and can be taken into account in simulations. However, modifications in a protein do not only have local effects. Hence it is usually hard to immediately spot all changes of the protein in this model, especially if they occur behind other structures. The view then needs to be rotated, which is tedious and also leads to disorientation. This can be avoided by using a two-dimensional representation of the protein model. These representations are sought when it is more important to see the order of the secondary structures and their closeness to other ones. A sample is provided in Figure 1.3, which is manually drawn.

The idea of this bachelor thesis therefore is to find a two-dimensional model of a protein and automatically visualise protein secondary structures and their bonds using graph layout techniques. We want our schematic representation to be visually pleasing and calculated in a reasonable amount of time. Since automatic graph layout creation is a broad domain, we first find a suitable approach that supports our desired criteria. Our choice is to use a mixed-integer program. Afterwards, we develop a model for protein secondary structures and informally describe the criteria. We then provide formal constraints that are implemented in Python for the interface of the optimiser Gurobi.[GO14] Then we discuss

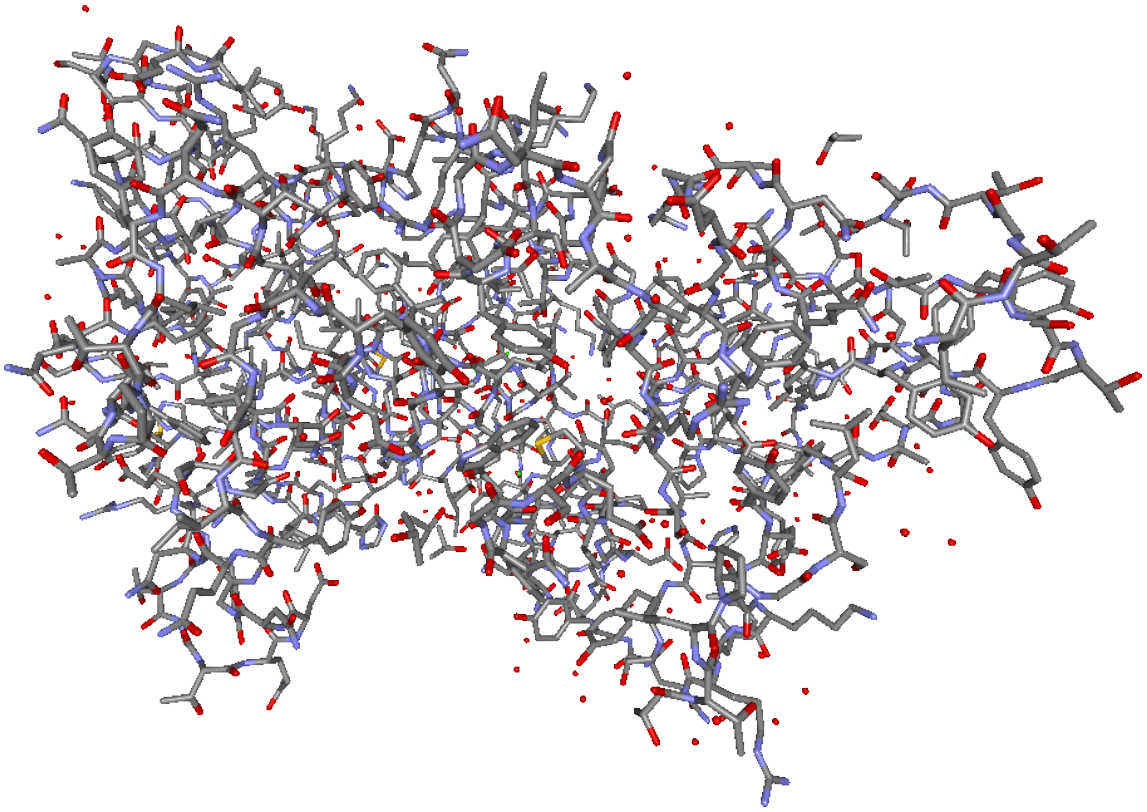


Figure 1.1: A stick model of the protein 8TLI. Atoms are presented as sticks and different atoms use different colours.

how well our constraints work using example graphs, how we can improve the model and also elaborate where shortcomings lie. In the last chapter we conclude and provide points for further research.

Note that this bachelor thesis is developed in cooperation with the group of Prof. Dr. Holger Gohlke at Universität Düsseldorf. All input data for the methods used in this thesis is generated by the VisualCNA [PM14] tool, a graphical user interface for constraint network analysis (CNA). The layout therefore depends on that data.

Also, to the best of our knowledge, no other documented automated graph-based visualisation approach exists for proteins and their secondary structures.

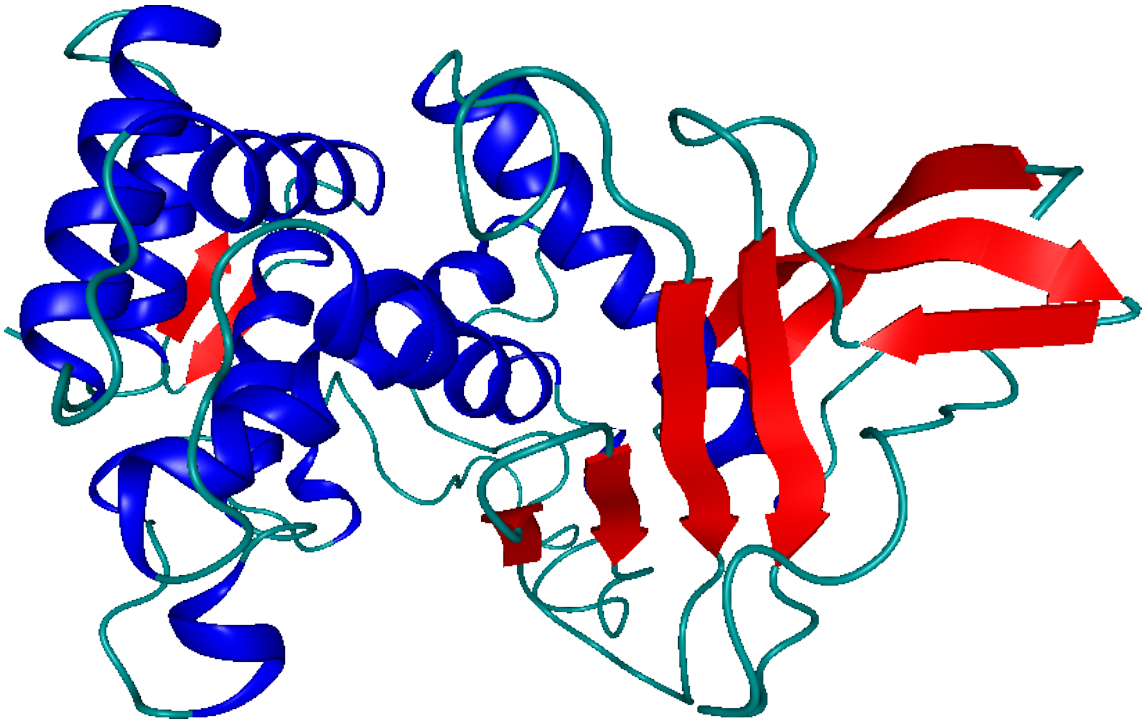


Figure 1.2: A cartoon model of the protein 8TLI. Residues are coloured and shaped, depending on which structure they form.

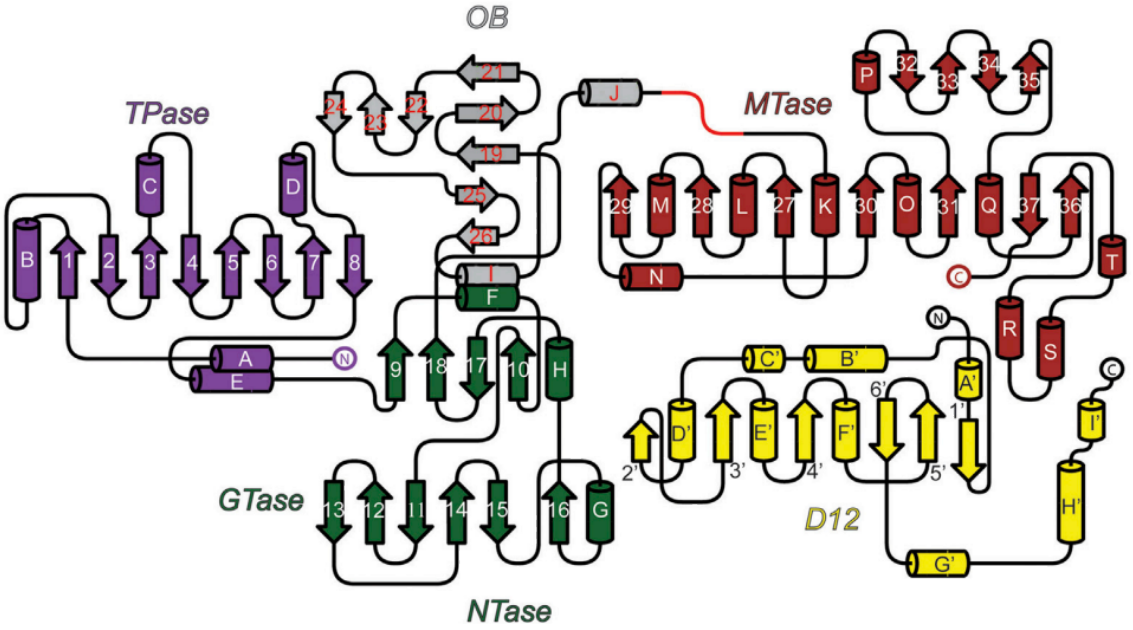


Figure 1.3: A manually drawn schematic for the protein 4CKB, taken from [KCdlP+14].

2. Related Work

In this chapter we describe approaches that are used for similar problems. We also elaborate on how these approaches work in regard to our problem.

In the book of Tamassia, there are several methods described to create a visually pleasing graph layout. A common "highly effective way to draw graphs [...] is to use only edges that are rectilinear, or orthogonal." [Tam13] This means that edges are either horizontal or vertical. An example drawing is shown in Figure 2.1. While there are no crossing edges, the restriction to limit the edges to two axes is too tight. However, we refine the idea to limit edges to axes in general manner.

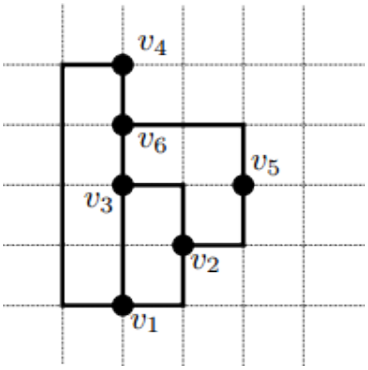


Figure 2.1: An orthogonal graph drawing, taken from [Tam13]. v_1 through v_6 represent vertices, whereas the edges are aligned on a grid.

Since our problem is similar to drawing metro-map based layouts, we take a look at two approaches that are used to create those.

Chivers and Rodgers [CR14] use a multi-levelled, force-directed approach, as seen in Figure 2.2. They define an initial repelling force between vertices of the graph and then, while weakening this force, apply a second force to transform the graph into desired layout. The advantage of this approach is the small amount of forces needed to create a good layout and therefore easy to implement. However, the disadvantage is the need to repeatedly adjust force coefficients until the layout is satisfying. This makes predicting the final layout difficult.

Nöllenburg and Wolff [NW11] use a mixed-integer linear program. They define mandatory constraints, also called hard constraints, which the resulting layout must conform to. Then

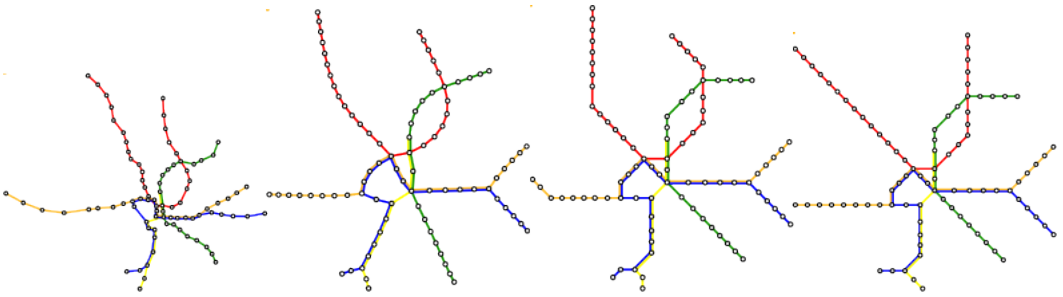


Figure 2.2: A force-directed drawing process, taken from [CR14]. The left picture shows the initial layout, and the following pictures illustrate how it is modified during the application of forces.

they add soft constraints which tweak several aspects, like total edge length, of the resulting layout. This allows them to choose criteria which the layout must fulfil and therefore predict it. Figure 2.3 shows how the initial layout is modified and customised using different weights for the soft constraints. Additionally, the optimiser finds a first layout in less time than the force-directed approach of Chivers and Rodgers.[NW11] However, preliminary work is required and a good optimiser must be found for this approach to work. It is also required to implement the constraints using the interface given by the optimiser.

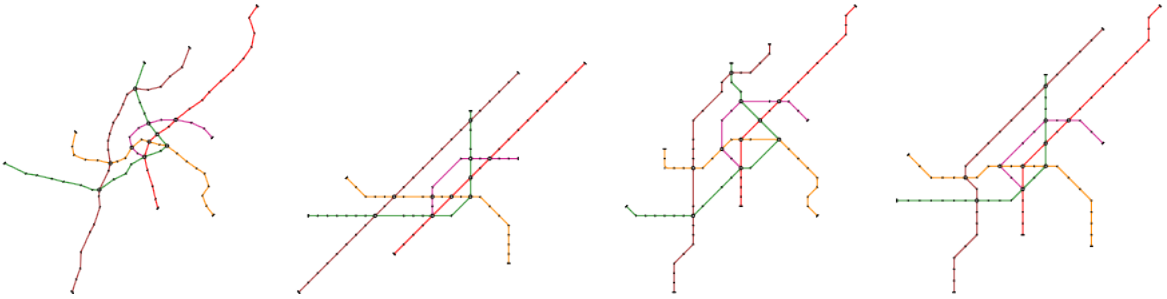


Figure 2.3: A mixed-integer program generated layout, taken from [NW11]. The left picture shows the initial layout, and the following pictures show layouts in which certain aspects are more prevalent than others.

Therefore, we decide to use a mixed-integer program, since it is faster and gives us more control over the representation of the graph. The optimiser we choose is Gurobi. [GO14]

3. Composition of Proteins

In this chapter we explain the basic principles of proteins that are used throughout the rest of our thesis. First we explain what proteins are composed of, since this composition is important for our model.

Biological molecules which are composed of amino acids are called proteins. Amino acids are chemical molecules which possess at least one carboxylic (-COOH) and one amine ($-NH_2$) group. The carboxylic group is referred to as the C-terminus, whereas the amine group is called the N-terminus. The amino acids form a chain in the protein and are interconnected by the respective C- and N-termini of each amino acid. This chain is also called the backbone of the protein. The sequence of the amino acids in the backbone is called the primary structure. Given the direction from which it starts, the sequence is non-ambiguous and can be numbered. When looking at the spatial arrangement of the protein's amino acids, repeating patterns can be identified, as seen in Figure 1.2. These patterns include α-helices, β-sheets and loops. Figure 3.1 shows a pair of such helices. The mapping of amino acids to these patterns, also known as structures, is called the secondary structure. Structures appear due to interactions and bonds between different amino acids. There are multiple bond types, which are categorised as covalent and non-covalent bonds. Covalent bonds are strong bonds between atoms and make up the cohesion of a chemical molecule. Non-covalent bonds, of which the most common type is the hydrogen bond, stabilise a structure. The more non-covalent bonds a structure has, the more stable it is. [Sch10a]

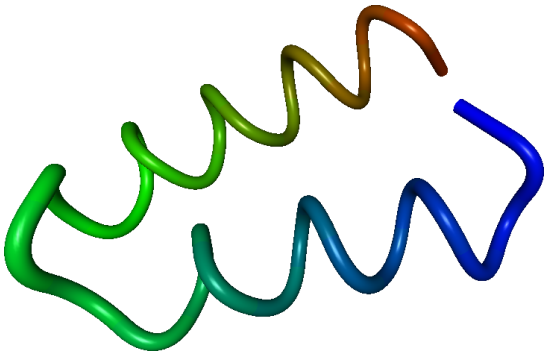


Figure 3.1: Two parallel helices, connected by a loop.

4. Model

In this chapter we create, based on the data provided by VisualCNA, a graph model and informally describe features we want our protein secondary structure to have.

The output data of VisualCNA contains three-dimensional coordinates of all atoms of a given protein. Furthermore the atoms' mapping to their amino acid and structure is provided. It also contains the bonds between all amino acids and their respective bond type. Table 4.1 lists several attributes that are given.

To model our graph based on that data, residues are treated as vertices and bonds between residues as edges of the graph. The residues then can be grouped together to form the secondary structures in the graph. Since the given coordinates are three-dimensional, an appropriate projection to two dimensions must be found for the graph layout. For visual reasons the resulting graph should also be planar while still showing the original arrangement of the structures as good as possible. Unfortunately, in most cases these two characteristics contradict each other.

Before any steps towards an implementation are taken, we first need to define the abstract framework for the data that is provided.

Let $G = (V, E)$ be the directed input graph and let $|V| = n$ and $|E| = m$. For every vertex $v \in V$, a position attribute $\text{pos}(v) = (x(v), y(v), z(v))^T$, modelling the real-world

Section	Identifier	Description
Vertices	atom_id	Unique ID of the node
	atom_code	PDB code for the atom
	atom_name	Atom name, i.e. C for carbon
	residue_id	Residue number in the chain
	residue_name	PDB residue name, i.e. ALA for alanine
	coordinates	Tuple containing the x, y, z coordinates
	backbone	True if atom is in the backbone
	structure	Simplified secondary structure identifier, i.e. H for helix
Edges	nodes	Tuple containing the IDs of the atoms the edge connects
	type	The constraint type, i.e. COV for covalent, HBOND for hydrogen bond

Table 4.1: An excerpt of the data provided by Universität Düsseldorf.

three-dimensional coordinates, as well as a structure information attribute struct(v), with struct(v) $\in$ structureTypes = {HELIX, SHEET, LOOP, NONE}, is given. The latter models the secondary structure an atom belongs to. For every edge $e \in E$ there is a bond type attribute bond(e) $\in$ bondTypes = {COV, HBOND, HPHOB, SBRIDGE, DBRIDGE}.

With the basic model above, we identify structures in the graph. These model residues, which represent the primary structure. However, if vertices are treated as residues and edges as the bonds between them, then the graph structures represent the secondary structure of a protein. The following definition describes what a structure in a graph is.

Definition 4.1. *A subgraph of G is a* structure $S = (V' \subseteq V, E' \subseteq E)$*, if it is a maximal long directed path in V such that*

- *V' contains at least two vertices,*
- *each edge in E' is of COV type,*
- *each vertex in V' has the same structure type.*

Then the first vertex of the directed path is denoted by first(S) and the last vertex by last(S) respectively. The type of the structure is denoted by struct(S). To obtain an unique structure representation of a protein, we define what it means for a set of structures to be ordered.

Definition 4.2. *A set $S^* = \{S_0, S_1, \dots\}$ of structures is* ordered *if and only if for every structure S_i, $i \in 0 \leq i \leq |S^*|-1$, the last vertex* $\mathrm{last(S_i)}$ *is adjacent to first vertex* $\mathrm{first}(S_{i+1})$ *of the next structure in the given ordering.*

If S^* is ordered, we will use round brackets $S^* = (S_0, S_1, \dots)$.

Definition 4.3. *An ordered set of structures $S^* = (S_0, S_1, \dots)$ is* valid *if for every i, $i \in 0 \leq i \leq |S*| - 1$, the current structure S_i is a loop structure and the next one S_{i+1} is either a helix or a sheet, or vice versa.*

In a valid, ordered set of structures, covalent edges between structures are assigned to their corresponding loop structures. The goal is to find a layout, as given in the following definition, for such a set of structures.

Definition 4.4. *A* layout *of a graph $G = (V, E)$ is a mapping of the vertices of V to fixed two-dimensional coordinates (x, y).*

We will use a mixed-integer programming approach to achieve said goal. A mixed-integer program consists of a target function which should be minimized/maximized under a given set of restrictions for variables. These restrictions are also called constraints. We distinguish these by formulating constraints which must be met under all circumstances, so-called *hard constraints*, and constraints with weights that directly affect the target function. We refer to these as *soft constraints*.

4.1 Hard Constraints

We want the layout of our graph to fulfil certain requirements. For that, we first informally describe the constraints and then summarise them in a formal way.

Starting with the placement of the graph's edges, our goal is too reduce the amount of options to place an edge around a vertex. Therefore the edges of the graph should follow given axes. The axes are numbered from zero to $k - 1$, which totals to k axes and $2k$ directions. The following definition shows what it means for an edge to be aligned on an axis.

Definition 4.5. *Given a directed graph $G = (V, E)$ and a layout Γ of G, an edge in the layout $\Gamma(e)$ is* k-linear *if its direction follows one of the axises which divide a half-plane of $\mathbb{R}^2$ into k equal parts.*

Another goal is to make the layout as planar as possible, which means that the amount of crossing edges should be minimal. In [NW11], Nöllenburg and Wolff describe that this characteristic can be achieved by forcing pairwise non-incident edges to be placed such that an area around an edge does not contain another edge. We apply this criterion to all covalent edges in the graph to improve its readability, but we will still allow overlapping where it is necessary. To further improve the visual aspect of the layout, each helix and sheet should be represented as a straight path without bends. We extend this aspect to also include the incident edges of these structures. Furthermore, the length of each covalent edge of the graph should exceed a given minimum value to further spread out the vertices. To control the length of loop structures, we add another constraint which requires these to have a given minimum length.

The aforementioned constraints are summed up as follows.

(H1) For each covalent edge $e \in E$, $\Gamma(e)$ must be k-linear, $k \in \mathbb{N}$, $k \geq 2$.
(H2) Additionally, every covalent edge must be a least $d_{min} > 0$ units away from every other non-incident covalent edge.
(H3) All covalent edges in a non-loop structure must face the same direction in the layout Γ.
(H4) The sum of the lengths of all covalent edges in a loop structure must be at least $l_{min}^{LOOP} > 0$ units.
(H5) Every covalent edge must have a length of at least $l_{min} > 0$ units.

4.2 Soft Constraints

Soft constraints help making certain features, e.g. the total size of the graph layout, more controllable. By giving each a coefficient and summing the constraints up, we can decide which feature should gain a greater weight in the layout.

One of these constraints is the amount of overlaps we allow in the layout. For each overlap the penalty in the target function increases linearly. By increasing the cost of these penalties, edges are more likely to be placed with a gap between each other, as seen in Figure 4.1. In the hard constraints section we introduced several requirements to ensure that the layout is not cramped. Since these are just lower bounds for distances between edges, we also need to introduce an upper bound. This is realized by minimizing the total edge length of covalent edges of the graph. The same is introduced for hydrogen bonds. It ensures that structures having many hydrogen bonds to other structures are more tightly placed together. Finally, we want the paths of loop structures which connect helices and/or sheets to have as few bends as possible. Thus the structure path is easier to follow in the layout.

The soft constraints are summed up as follows.

(S1) The total edge length of the layout Γ should be small.
(S2) The number of overlapping structures should be minimized.
(S3) The length of all hydrogen edges should be minimized.
(S4) The number of line bends of loop structures should be small.

Since all constraints are now formally drafted, the next step is to find linear expressions which can be used by a mixed-integer program.

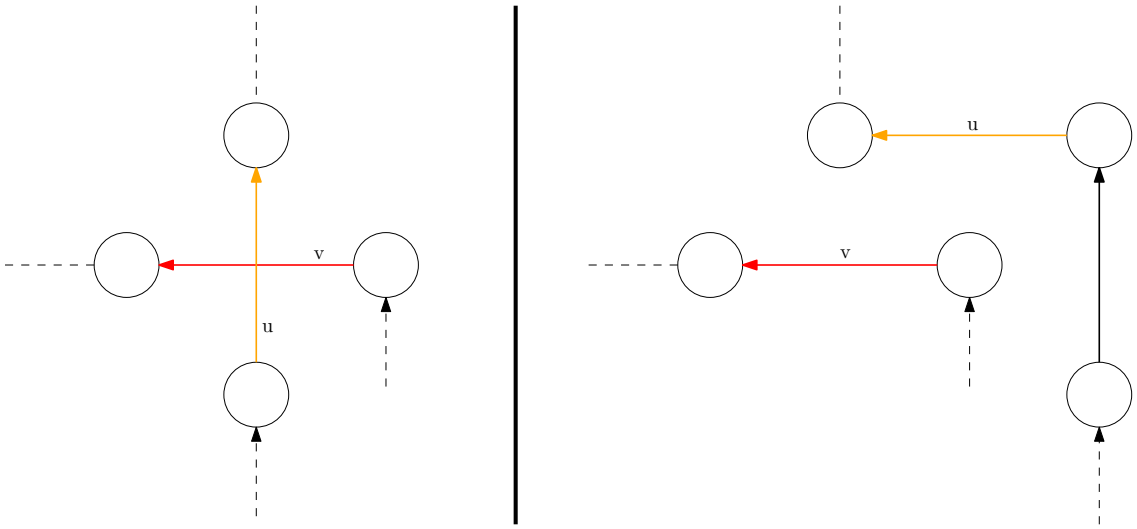


Figure 4.1: Left side: edges u and v overlapping, right side: by giving overlapping a higher cost, u or v will be placed in a non-intersecting way

5. Mixed-Integer Program

In this chapter we formalise the constraints mentioned in Chapter 4 so that they can be implemented for an optimiser.

5.1 Coordinate System

The given atom coordinates are given in a Cartesian coordinate system, which therefore is applied to all of our constraints as well. To allow some generalisation however, we use k axes instead of just two. Henceforth for our k-linear system we refer to our axes as z_i, $0 \leq i < k$. Given a vertex $v \in V$, each z_i is defined as $z_i(v) := x(v) \cdot \cos(\theta) + y(v) \cdot \sin(\theta)$, $\theta = \frac{\pi i}{k}$. These axes are numbered counter-clockwise, starting with the positive x-axis. For example, using $k = 4$, the usual x-axis would be z_0, whereas the y-axis would be z_2. Figure 5.1 illustrates this.

To measure distances in the k-linear system, we use the L^∞-metric, which is defined as $\max\{|x(u) - x(v)|, |y(u) - y(v)|\}$ for two vertices $u, v \in V$.

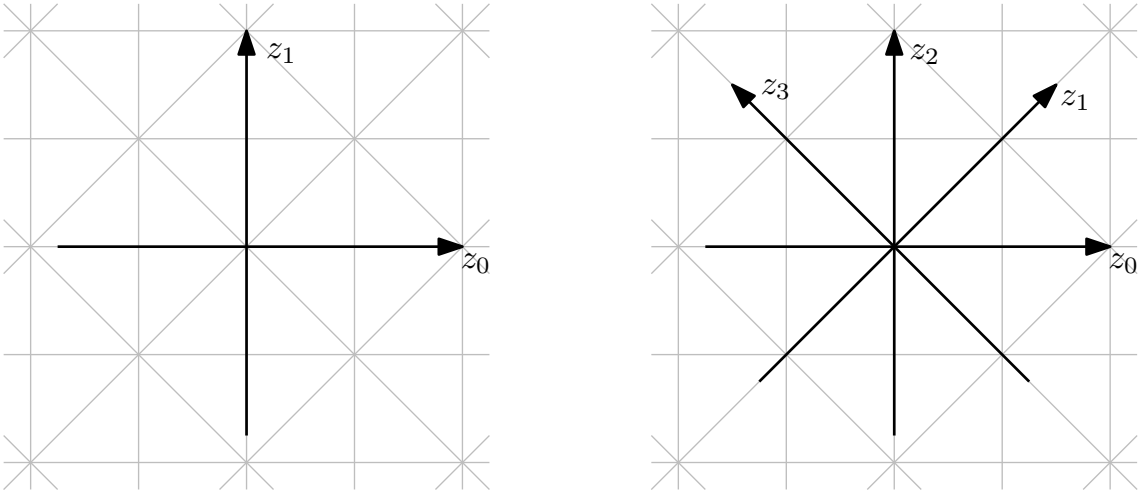


Figure 5.1: Left side: A 2-linear coordinate system. Right Side: A 4-linear one.

5.2 k-linearity for Covalent Edges and Edge Lengths

Nöllenburg and Wolff [NW11] describe an octalinear system, which is a special case of our approach when using $k = 4$. Therefore as a more general version for each vertex u we define a partition of the plane into $2k$ sectors. Each sector is a $\frac{180°}{k}$-wedge with apex u. The wedges are centred around rays that emanate from u and follow the k-linear directions. The sectors are numbered from 0 to $2k$ counterclockwise starting with the positive x-direction. For each pair of covalent edges (u, v) and (v, u), we introduce variables $dir(u, v)$ and $dir(v, u)$ to denote the k-linear directions of (u, v) and (v, u) in layout Γ. We identify each k-linear direction with its corresponding sector, as seen in Figure 5.2.

With the given sectors, first of all, we limit the amount of directions an edge can have to one. Equation 5.1 models that the binary variable α_i is 1 for the sector i that an adjacent vertex v lies in when centred on vertex u.

$$\sum_{i \in \{0,\dots,2k-1\}} \alpha_i(u, v) = 1 \tag{5.1}$$

Since the sum equals one, only one sector is active. Then the correct sector index is assigned to each edge and for each of the $2k$ directions, as well as its opposite direction, as seen in equation 5.2. Because the graph is a directed tree, setting the opposite direction to the reverse edge is impossible. Therefore, dummy reverse edges are added.

$$\begin{array}{rcl} dir(u, v) & = & \sum_{i \in \{0,\dots,2k-1\}} i \cdot \alpha_i(u, v) \\ dir(v, u) & = & \sum_{i \in \{0,\dots,2k-1\}} ((i + k) \mod 2k) \cdot \alpha_i(u, v) \end{array} \tag{5.2}$$

Finally, depending on which direction is chosen for an edge, we require its length to be at least l_{min} in said direction. The value of l_{min} is set by the user.

$$\begin{array}{rcl} z_{\mathrm{orth}(i)}(u) - z_{\mathrm{orth}(i)}(v) & \leq & M(1 - \alpha_i(u, v)) \\ -z_{\mathrm{orth}(i)}(u) + z_{\mathrm{orth}(i)}(v) & \leq & M(1 - \alpha_i(u, v)) \\ -z_i(u) + z_i(v) & \geq & -M(1 - \alpha_i(u, v)) + l_{min}(u, v) \end{array} \tag{5.3}$$

$$\begin{array}{c} \forall i \in \{0, \dots, 2k - 1\} \\ \mathrm{orth}(i) := (i + \frac{k}{2}) \mod k \end{array}$$

These equations work, because for a chosen direction i, the edge must follow the corresponding z_i axis and extend at least l_{min} units in said direction. The length of the edge in the orthogonal direction equals zero, which ensures that the edge is indeed on the correct axis. The constraints for all other directions are easily satisfied using the large constant M.

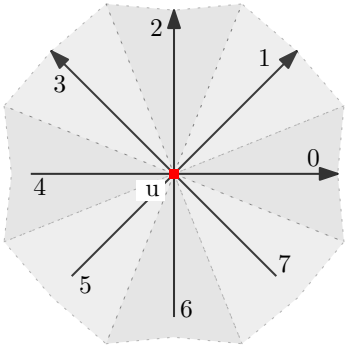


Figure 5.2: Sectoring for a 4-linear coordinate system, centred on a vertex u.

5.3 Covalent Edge Spacing

To make the layout as planar as possible, we will use the same method as Nöllenburg and Wolff.[NW11] That is, for each pair of non-incident covalent edges $(e_1, e_2) = ((u_1, v_1), (u_2, v_2))$ we require

$$\sum_{i \in \{0,\dots,k-1\}} \beta_i(e_1, e_2) \geq 1 \tag{5.4}$$

where β_i is a binary variable and denotes the z_i direction. The next four inequalities (5.5) make sure that each two edges are at least d_{min} units apart for each other. Since we do not require our layout to be strictly planar, overlapping, denoted by a binary variable $\omega(e_1, e_2)$, is also possible. Overlapping occurs when the associated cost of $\omega(e_1, e_2)$ in the target function is less expensive than adding enough edge spacing between all necessary edges and thus increasing the total edge length.

$$\begin{array}{rcl} z_i(u_2) - z_i(u_1) & \leq & M(1 - \beta_i(e_1, e_2)) + M \cdot \omega(e_1, e_2) - d_{min} \\ z_i(u_2) - z_i(v_1) & \leq & M(1 - \beta_i(e_1, e_2)) + M \cdot \omega(e_1, e_2) - d_{min} \\ z_i(v_2) - z_i(u_1) & \leq & M(1 - \beta_i(e_1, e_2)) + M \cdot \omega(e_1, e_2) - d_{min} \\ z_i(v_2) - z_i(v_1) & \leq & M(1 - \beta_i(e_1, e_2)) + M \cdot \omega(e_1, e_2) - d_{min} \end{array} \tag{5.5}$$

$$\forall i \in \{0, \dots, k-1\}$$

5.4 Helix/Sheet Size and Direction

As mentioned in the hard constraints explanation 4.1, the structure direction serves as a purely cosmetic constraint to reduce the time needed to spot helices and sheets in the layout. Therefore, for our ordered set of all structures of the graph, we require that all covalent edges of a helix or sheet structure $S = (V', E')$ use the same direction. Additionally, to make the layout more visually pleasing, we include the incident edges of first(S) and last(S) in G in this constraint. This is achieved by simply equalising all covalent edge directions within the structure.

$$dir(u, v) = dir(v, w) \tag{5.6}$$

$$\forall u, w \in V, \forall v \in V'$$

However, when using only the constraints above, structures are of arbitrary size and do not reflect the size of the structure given in the data. Since we want the size of helices and sheets to be preserved, we also require that each covalent edge in a said structure has the same length. Together with the minimum covalent edge length and the fact that we want to minimize the total edge length of all structures, the size of a structure is proportional to its amount of vertices.

$$\begin{array}{rcl} x(u) - x(v) & = & x(v) - x(w) \\ y(u) - y(v) & = & y(v) - y(w) \end{array} \tag{5.7}$$

$$\forall u, v, w \in V'$$

5.5 Loop Structure Length

We do not want to restrict the size of loops as much as helices and sheets. Therefore, we only require that the length of all covalent edges of each loop structure $S = (V', E')$ sum up to at least l_{min}^{LOOP}. To achieve that, we define a new real-valued, non-negative variable $\lambda(u, v)$ for each covalent edge (u, v) in the graph. $\lambda(u, v)$ serves as an upper bound for the corresponding edge length, which is minimized in one of the following soft constraints.

$$\sum_{(u,v)\in E'} \lambda(u, v) \quad \geq \quad l_{min}^{LOOP} \tag{5.8}$$

5.6 Total Covalent Edge Length

As defined in Section 5.5, the variable $\lambda(u, v)$, that denotes the edge length of the covalent edge (u, v), is measured in the L^∞-metric. The sum of all $\lambda(u, v)$ directly affects the target function and is one goal of the minimisation.

$$cost_{(S2)} \quad = \quad \sum_{(u,v)\in E} \lambda(u, v) \tag{5.9}$$

$$\begin{aligned} x(u) - x(v) &\leq \lambda(u, v) \\ -x(u) + x(v) &\leq \lambda(u, v) \\ y(u) - y(v) &\leq \lambda(u, v) \\ -y(u) + y(v) &\leq \lambda(u, v) \end{aligned} \tag{5.10}$$

5.7 Overlapping

As defined in Section 5.3, each binary variable $\omega(e_1, e_2)$ that is 1 denotes that overlapping is being used for a pair of edges. We simply use the sum of these binary variables.

$$cost_{(S1)} \quad = \quad \sum_{e_1,e_2\in E} \omega(e_1, e_2) \tag{5.11}$$

5.8 Total Hydrogen Edge Length

We define a new real-valued, non-negative variable $\gamma(u, v)$ for each hydrogen edge (u, v) with the same characteristic as $\lambda(u, v)$. This time however the L^∞-metric would be too rough to measure the length of a hydrogen edge. But since we cannot use the L^2-metric in linear programs, we will use a projection of $x(u)$ and $y(u)$ onto the axis z_i. Fortunately, the very definition of z_i is a projection, which allows us to directly use that trait.

$$cost_{(S3)} \quad = \quad \sum_{(u,v)\in E} \gamma(u, v) \tag{5.12}$$

$$\begin{aligned} z_i(u) - z_i(v) &\leq \gamma(u, v) \\ -z_i(u) + z_i(v) &\leq \gamma(u, v) \end{aligned} \tag{5.13}$$

$$\forall i \in \{0, \ldots, k-1\}$$

5.9 Loop Line Bends

Nöllenburg and Wolff also describe how to incorporate line bends in the target function. Thus, we want to reduce the line bends of all loop structure edges in our layout using the same method. First, each two incident edges (u, v) and (v, w) are given a line bend value $\mathrm{bd}(u, v, w)$. The sum of these values act as the cost in the target function.

$$cost_{(S4)} \quad = \quad \sum_{(u,v),(v,w)\in E'} \mathrm{bd}(u, v, w) \tag{5.14}$$

For two adjacent edges (u, v) and (v, w) in a loop structure we define $\Delta dir_{u,v,w} := dir(u, v) - dir(v, w)$. Then $\mathrm{bd}(u, v, w)$ is defined as

$$\mathrm{bd}(u, v, w) \quad = \quad \min\{|\Delta dir_{u,v,w}|, 2k - |\Delta dir_{u,v,w}|\} \tag{5.15}$$

In term of linear constraints, this is

$$\begin{array}{rcl} -\mathrm{bd}(u, v, w) & \leq & \Delta dir_{u,v,w} - 2k\delta_1(u, v, w) + 2k\delta_2(u, v, w) \\ \mathrm{bd}(u, v, w) & \geq & \Delta dir_{u,v,w} - 2k\delta_1(u, v, w) + 2k\delta_2(u, v, w) \end{array} \tag{5.16}$$

where δ_1 and δ_2 are binary variables.

5.10 Complexity

With the constraints formalised, we investigate the impact of these constraints on the mixed-integer program. A summary of these are given in the following table. By m we denote the amount of covalent edges and m' is the amount of hydrogen bond edges. $|S^*|$ is the number of structures in the graph.

Constraint	# MIP variables	# MIP constraints
(H1) + (H5)	$2km + 2m$	$3m + 6km$
(H2)	$\leq 2k(m^2 - m)$	$\leq (4k + 1)(m^2 - m)$
(H3)	0	$\leq 3m$
(H4)	m	$\leq \lvert S^* \rvert$
(S1)	0	0
(S2)	0	$4m$
(S3)	m'	$2m'$
(S4)	$\leq 3m$	$\leq 2m$
total	$\leq 2km^2 + 6m + m'$	$\leq 4km^2 + 2km + m^2 + 13m + \lvert S^* \rvert$

We see that (H2) causes the mixed-integer program to grow quadratically. This disturbs the otherwise linear constraints and needs to be addressed. We therefore use a feature of Gurobi called lazy constraints. Instead of adding constraints right at the beginning, Gurobi polls for constraints after finding a solution. For each pair of non-incident edges we calculate whether these edges overlap and then add a constraint to resolve the crossing. However, this can make a previously found solution infeasible.

6. Case Studies

In this chapter we first study the effects of the constraints on two small example graphs. We then take a look at the protein 8TLI and discuss how well the resulting layout matches our desired layout. From the observation of these results, we deduce more constraints that speed up the computation time. At last, we discuss why the mixed-integer program fails to generate layouts for certain proteins.

The system the computations run on is composed of an Intel Xeon E3-1230V3 with four cores which are running at 3.3 GHz each. Hyperthreading is enabled, simulating a total of eight cores on the machine. Additionally, there is 16 GB of RAM available. Our implementation of the mixed-integer programs is written in Python and solved using Gurobi 5.6. The solution contains the coordinates for each vertex and the layout is created from these. Additionally, each graph layout is computed multiple times with varying parameters. These parameters include the k-linearity and the coefficients for the soft constraints (S1) through (S4). We refer to the coefficients as c_1 for constraint (S1), c_2 for (S2), and so on. We also specify a time limit t_{max} in seconds for the computation. If the time limit is reached, we state the percentaged gap between the optimal solution and the feasible solution that is found. Each figure is annotated with the parameters that are used to get that layout. Note that in the previous chapter, we mention that the minimum covalent edge length and the minimum loop structure length are user-defined. For all of our computed layouts, both are set to 1.

All screenshots of the layouts are taken from our own programs. In the first program, the example graphs are shown in a minimalistic environment, where vertices are represented as numbered circles and directed edges as lines with arrow tips. The colour of the circle determines the structure the vertex belongs to. A black circle denotes a loop structure, a green circle a sheet and a blue circle a helix. Edges are also colour coded, whereas black full lines denote covalent bonds and red dashed lines hydrogen bonds. See Figure 6.1 for the keys used in the example graphs.

The second program shows the layouts in a more visually pleasing way by calculating minimum rectangle bounding boxes for helices and sheets. Loop vertices are not shown, which results in loop structures to look like a set of connected lines. Figure 6.2 shows a small section of such a representation. Ultimately, we want our helices to look like cylinders, sheets like arrows and our loops less rectangular, as seen in the manually drawn layout for the protein 1KN0 by Kagawa et al. in Figure 6.3. [KKI+02]

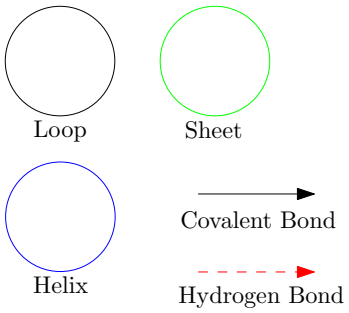


Figure 6.1: The keys used for the example graphs.

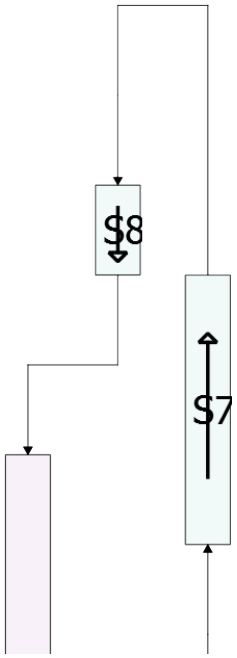


Figure 6.2: A small section of our automatically generated representation of the protein 8TLI. Blue boxes represent sheets, pink boxes represent helices and lines are loops. This is done using our second program.

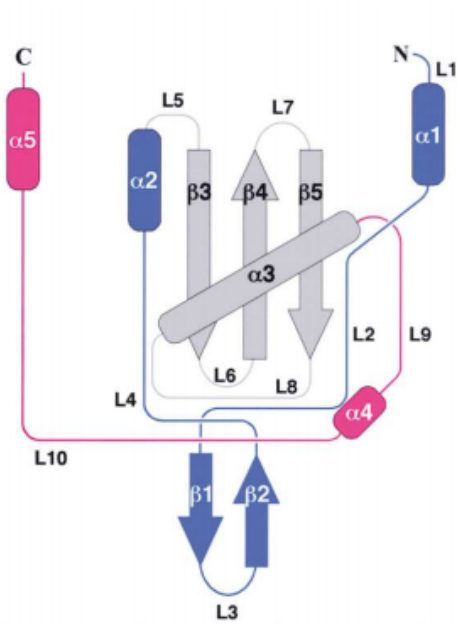


Figure 6.3: A manually drawn graph representation of the protein 1KN0 by Kagawa et al. [KKI+02]

6.1 Example Graph 1

Our first small example graph contains ten vertices in a directed path. The vertices are numbered from 0 to 9, whereas the first vertex in the path is assigned the number 0, and each following vertex the next natural number respectively. The graph consists of three secondary structures: two sheets, connected by a loop. There are three hydrogen bonds between the sheet structure vertices, with the objective to align both sheets in parallel. The features of the graph are:

# Vertices: 10		# Edges: 12	
Vertices	Secondary Structure	Edge	Bond Type
0-2	Sheet	$(0,1),(1,2),\ldots,(8,9)$	Covalent
3-6	Loop	$(7,2)$	Hydrogen Bond
7-9	Sheet	$(8,1)$	Hydrogen Bond
		$(9,0)$	Hydrogen Bond

6.1.1 Four Directions

The following layouts are computed using $k = 2$, meaning that we use only two axises on which the edges are aligned on. This results in faster computations, as the amount of constraints linearly depends on k, as calculated in Section 5.10. For the first layout, all coefficients are set to 1. Since none of them receives a greater weight than the other ones, we expect that all soft constraints have an equal impact on the layout. We refer to this layout as the standard one.

This indeed is the positioning of the vertices we have in mind. Both sheets are parallel, not overlapping, and all edge lengths are as short as possible. Moreover, it only takes 0.11 seconds to compute. For the following layouts, we individually set each coefficient to 10 and analyze the results. By increasing the total covalent edge length cost, we do not expect any changes compared to the standard layout of Figure 6.4.

However, the layout of Figure 6.5 deviates from Figure 6.4. The layout is rotated and mirrored. We cannot explain this behaviour from our constraints and therefore think that

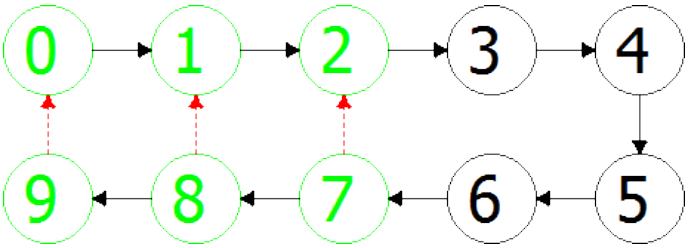


Figure 6.4: The first example graph using $k = 2, (c_1, c_2, c_3, c_4) = (1, 1, 1, 1)$. The optimal solution is found in 0.11 seconds.

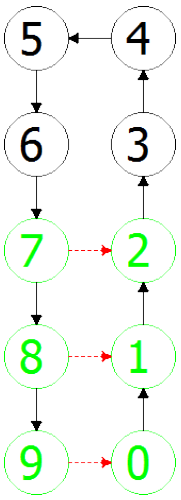


Figure 6.5: The first example graph using $k = 2, (c_1, c_2, c_3, c_4) = (10, 1, 1, 1)$. The optimal solution is found in 0.16 seconds.

Gurobi reaches the given optimal solution in other ways than the previous one. While this is not satisfying, both layouts are completely acceptable. Increasing the cost of overlaps does not result in changes from Figure 6.4, as there are no overlaps in the first place. Further penalising the length of hydrogen bonds results in the same layout as increasing the total covalent edge length cost. The final deviation occurs when setting the cost of loop line bends to 10, as seen in Figure 6.6. The layout of the graph is a straight path with no bends whatsoever.

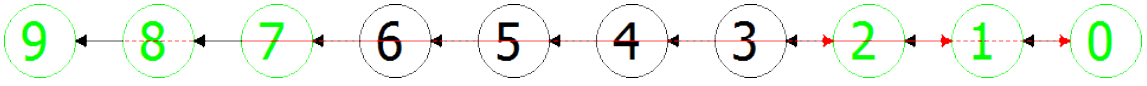


Figure 6.6: The first example graph using $k = 2, (c_1, c_2, c_3, c_4) = (1, 1, 1, 10)$. The optimal solution is found in 0.11 seconds.

All in all, the constraints deliver the desired effects. In the first example graph with 10 vertices and 12 edges, all computation times to find the optimal solution are in the range between 0.1 seconds and 0.4 seconds.

6.1.2 Eight Directions

In this subsection, we increase the number of possible directions by increasing the parameter k to 4. We expect an increase in the time needed to find an optimal solution, but no serious changes regarding the same tests with four directions when modifying the coefficients c_1 through c_4.

As above, we present the layout in which all constraints are equally weighted. See Figure 6.7 for the result. The noteable difference between said figure and Figure 6.4 is the alignment of all edges on other axises, causing a rotation of the layout. Also, the bend $(3, 4, 5)$ is a left turn, whereas the bend in Figure 6.4 is a right turn. This layout is more similiar to the one given in Figure 6.5. However, since both bends are equal in costs, this is not regarded any further. The computation time increased to 2.03 seconds, whereas the four-directional layout only needed 0.11 seconds. This is eighteen times longer. We discuss methods to decrease the time in a later section.

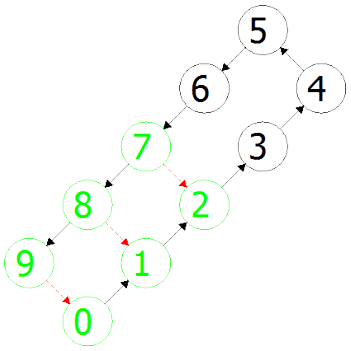


Figure 6.7: The first example graph using $k = 4, (c_1, c_2, c_3, c_4) = (1, 1, 1, 1)$. The optimal solution is found in 2.03 seconds.

Increasing the coefficient for total covalent edge lengths or total hydrogen bond edge lengths results in the same layout as the standard one of Figure 6.7. Unfortunately, the same is not true when increasing the cost of overlaps, as seen in Figure 6.8. We can observe the same phenomenon as in the subsection above, seeing the layout being rotated and mirrored.

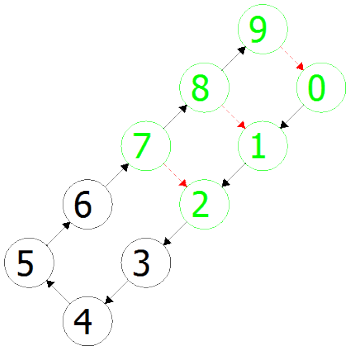


Figure 6.8: The first example graph using $k = 4, (c_1, c_2, c_3, c_4) = (1, 10, 1, 1)$. The optimal solution is found in 1.81 seconds.

The layout that results from increasing the line bend costs is a straight path, as seen in Figure 6.9. The time needed by Gurobi to find the optimal solution with these parameters is 0.53 seconds, which is the lowest of all other times needed by the other layouts using eight directions. We therefore assume that the increase of the cost of loop line bends is stronger than the increased costs of the other coeffecients, which allows Gurobi to discard worse solutions faster.

To conclude this section, we see that the soft constraints work as intended. There are slight deviations regarding the orientation of the layout, which however do not decrease the overall quality of the layout. The computation time increases when using more directions, but it is also affected by the coefficients. The results are summarized in Table 6.1.

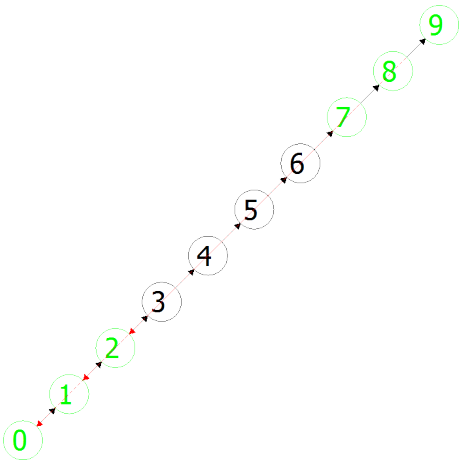


Figure 6.9: The first example graph using $k = 4, (c_1, c_2, c_3, c_4) = (1, 1, 1, 10)$. The optimal solution is found in 0.53 seconds.

k-linearity	Coefficients	Time needed (in seconds)	Gap to Optimal Solution
2	(1,1,1,1)	0.11	0%
	(10,1,1,1)	0.16	0%
	(1,10,1,1)	0.36	0%
	(1,1,10,1)	0.25	0%
	(1,1,1,10)	0.11	0%
4	(1,1,1,1)	2.03	0%
	(10,1,1,1)	4.35	0%
	(1,10,1,1)	1.81	0%
	(1,1,10,1)	3.24	0%
	(1,1,1,10)	0.53	0%

Table 6.1: The table shows the computation time needed for each parameter setting for the first example graph.

6.2 Example Graph 2

Our second example graph contains 30 vertices in a directed path. The vertices are numbered from 0 to 29, whereas the first vertex in the path is assigned the number 0, and each following vertex the next natural number respectively. In contrast to the first example graph, this one contains seven secondary structures. It also starts and ends with loops, with two sheets and one helix inbetween. There are also more hydrogen bonds to make the latter two structures parallel. The features are:

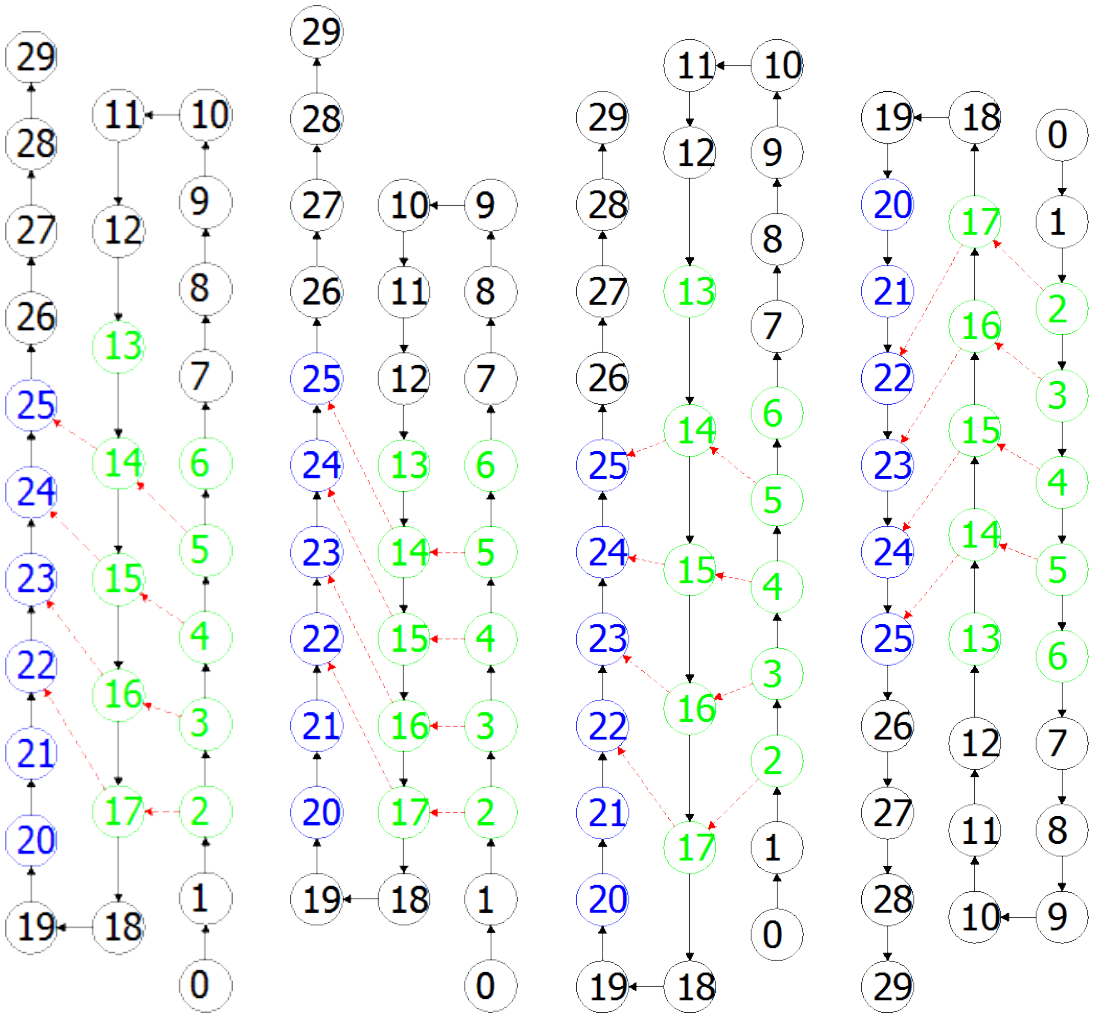
# Vertices: 30		# Edges: 37	
Vertices	Secondary Structure	Edge	Bond Type
0-1	Loop	$(0,1),(1,2),\ldots,(28,29)$	Covalent
2-6	Sheet	$(2,17)$	Hydrogen Bond
7-12	Loop	$(3,16)$	Hydrogen Bond
13-17	Sheet	$(4,15)$	Hydrogen Bond
18-19	Loop	$(5,14)$	Hydrogen Bond
20-25	Helix	$(14,25)$	Hydrogen Bond
26-29	Loop	$(15,24)$	Hydrogen Bond
		$(16,23)$	Hydrogen Bond
		$(17,22)$	Hydrogen Bond

Using this graph, we study the computation times in more detail. With the increased number of vertices and edges and thus the increased number of constraints, we expect Gurobi to take much longer to find the optimal solution.

The standard layout using $k = 2$ is shown in 6.10a. We use a time limit of $t_{max} = 60$ and let Gurobi compute the optimal solution. The first feasible solution is found within the first two seconds. We reach the time limit with a gap of 38.94% to the optimal solution. In another computation with t_{max} changed to 1800, the optimality is proven after 1045 seconds, but with no new solution being found. Therefore, we assume that Gurobi quickly finds feasible solutions and spends most of the computation time proving the optimality. The aforementioned data can be found in Table 6.2.

k-linearity	Coefficients	Time needed / Maximum Time	Gap to Optimal Solution
2	(1,1,1,1)	60s / 60s	38.94%
	(1,1,1,1)	1045s / 1800s	0%
	(10,1,1,1)	60s / 60s	36.80%
	(1,10,1,1)	60s / 60s	31.20%
	(1,1,10,1)	60s / 60s	12.10%
	(1,1,10,1)	1200s / 1200s	10.98%
	(1,1,1,10)	60s / 60s	47.63%
	(10,10,10,1)	60s / 60s	10.80%
4	(1,1,1,1)	60s / 60s	32.97%
	(1,1,1,1)	1800s / 1800s	24.07%
	(10,1,1,1)	60s / 60s	20.07%
	(1,10,1,1)	60s / 60s	36.16%
	(1,1,10,1)	60s / 60s	45.76%
	(1,1,1,10)	60s / 60s	71.16%
	(10,10,10,1)	60s / 60s	70.60%

Table 6.2: The table shows the computation time needed for each parameter setting for the second example graph.

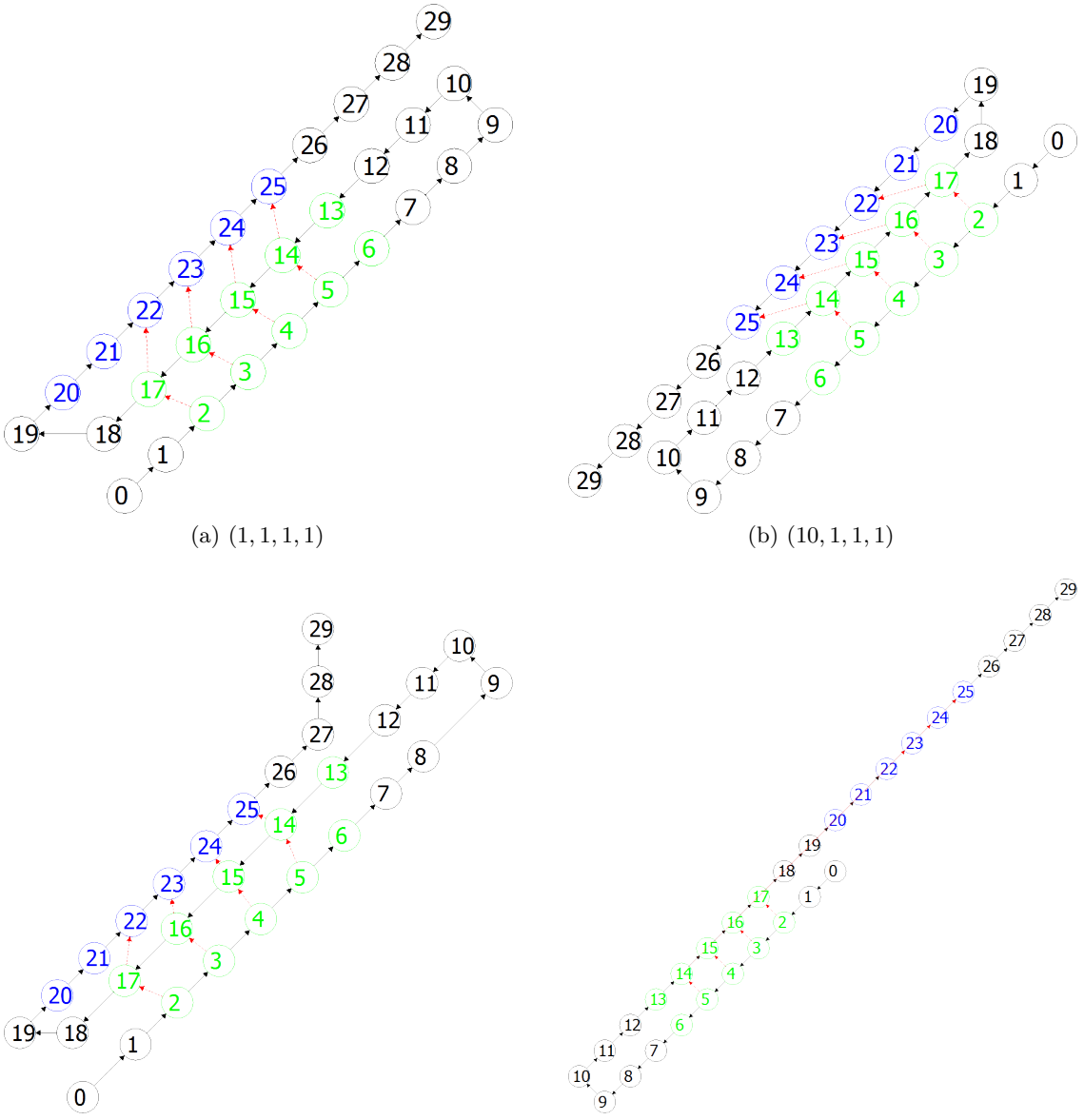


(a) $(1, 1, 1, 1)$ (b) $(10, 1, 1, 1)$ (c) $(1, 1, 10, 1)$ (d) $(1, 1, 1, 10)$

Figure 6.10: The second example graph using $k = 2$. The values for the coefficients (c_1, c_2, c_3, c_4) are stated below their respective layouts. Note that the parameter set $(1, 10, 1, 1)$ is missing because the resulting layout is identical to the standard one.

When we increase the coefficients one by one again, we notice that for all but the loop line bend constraint the gap to the optimal solution is lower at the 60 seconds time limit than the one for the standard layout. However, this is not consistent when also taking the same settings for $k = 4$ in mind. In that case, only the total edge length coefficient causes the gap to be smaller, whereas all other ones increase it. This is related to the fact that with higher values for k, the varying of the coefficients has higher impacts on the resulting layout. For $k = 2$, all layouts have loop line bends at the same set of vertices, namely $(8, 9, 10), (9, 10, 11), (17, 18, 19)$ and $(18, 19, 20)$, with the same bend value, which is defined in Section 5.9. For $k = 4$ however, there are either additional bends like in Figure 6.11c or less ones like in Figure 6.11d. Note that the bend in Figure 6.11c at $(26, 27, 28)$ persists even when the optimality of the solution is proven.

Therefore, with more directions the layout of the graph becomes more sensitive to changes in the weights of our soft constraints. We also conclude that the time needed to find and prove the optimal solution drastically increases together with the size of the graph, which makes our method not scalable in its current state.



(a) $(1, 1, 1, 1)$

(b) $(10, 1, 1, 1)$

(c) $(1, 1, 10, 1)$

(d) $(1, 1, 1, 10)$

Figure 6.11: The second example graph using $k = 4$. The values for the coefficients (c_1, c_2, c_3, c_4) are stated below their respective layouts. Note that the parameter set $(1, 10, 1, 1)$ is missing because the resulting layout is identical to the standard one.

6.3 The Protein 8TLI

8TLI is a medium-sized protein with 316 residues. Figure 6.12 shows the three-dimensional representation of the secondary structure of the protein. We can see that the protein is composed of many sheets and helices, interconnected by loops that have up to 20 residues. Judging from our previous tests, we can tell that the computation to find the optimal solution will take a long time. We will use our second program as described in the starting section of Chapter 6 to visualise the layout. The optimal layout we have in mind is shown in Figure 6.13.

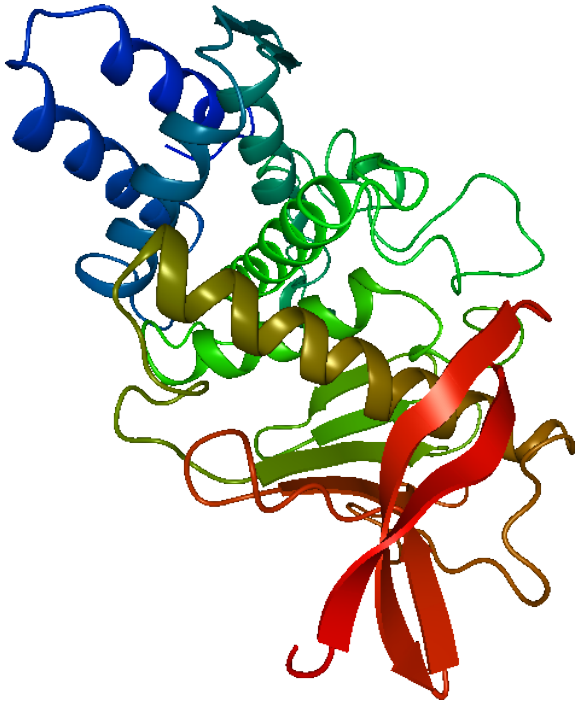


Figure 6.12: The secondary structure of the protein 8TLI. This screenshot is taken with BallView. [MHLK05]

As parameter settings, we use $k = 4$ and $(c_1, c_2, c_3, c_4) = (1, 1, 1, 1)$ for the coefficients. We also apply the parallelism method that is described in Section 6.4 to speed up the computation, since Gurobi crashes during the root relaxation without it. The root relaxation is one step for Gurobi to find a first, but not necessarily feasible minimum bound. We do not know why the crashes occur, since no crash logs are created. Figure 6.14 shows the first feasible solution found with these parameter settings. We can see that most structures are aligned in parallel. However there are many overlaps, like in the top right at the N-terminal, which makes it hard to follow the covalent edge path. There are also many unnecessary loop line bends, i.e. between sheet 9 (S9) and sheet 10 (S10), which makes the layout less visually pleasing. The layout is found after 1114 seconds with an optimality gap of 97.5%. We continue computing to find a better feasible solution for 36 hours, but none is found. We do not even reach the time limit since Gurobi crashes after 23 hours. Therefore, we try to find constraints that help reducing the time needed to find feasible solutions.

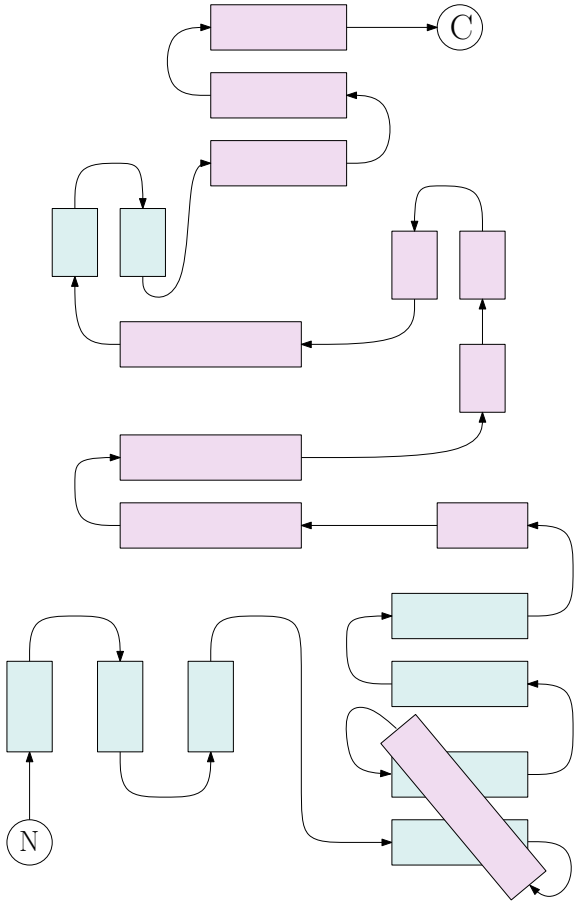


Figure 6.13: The schematic representation of 8TLI as we have it in mind. This layout is manually created and does not include small sheets.

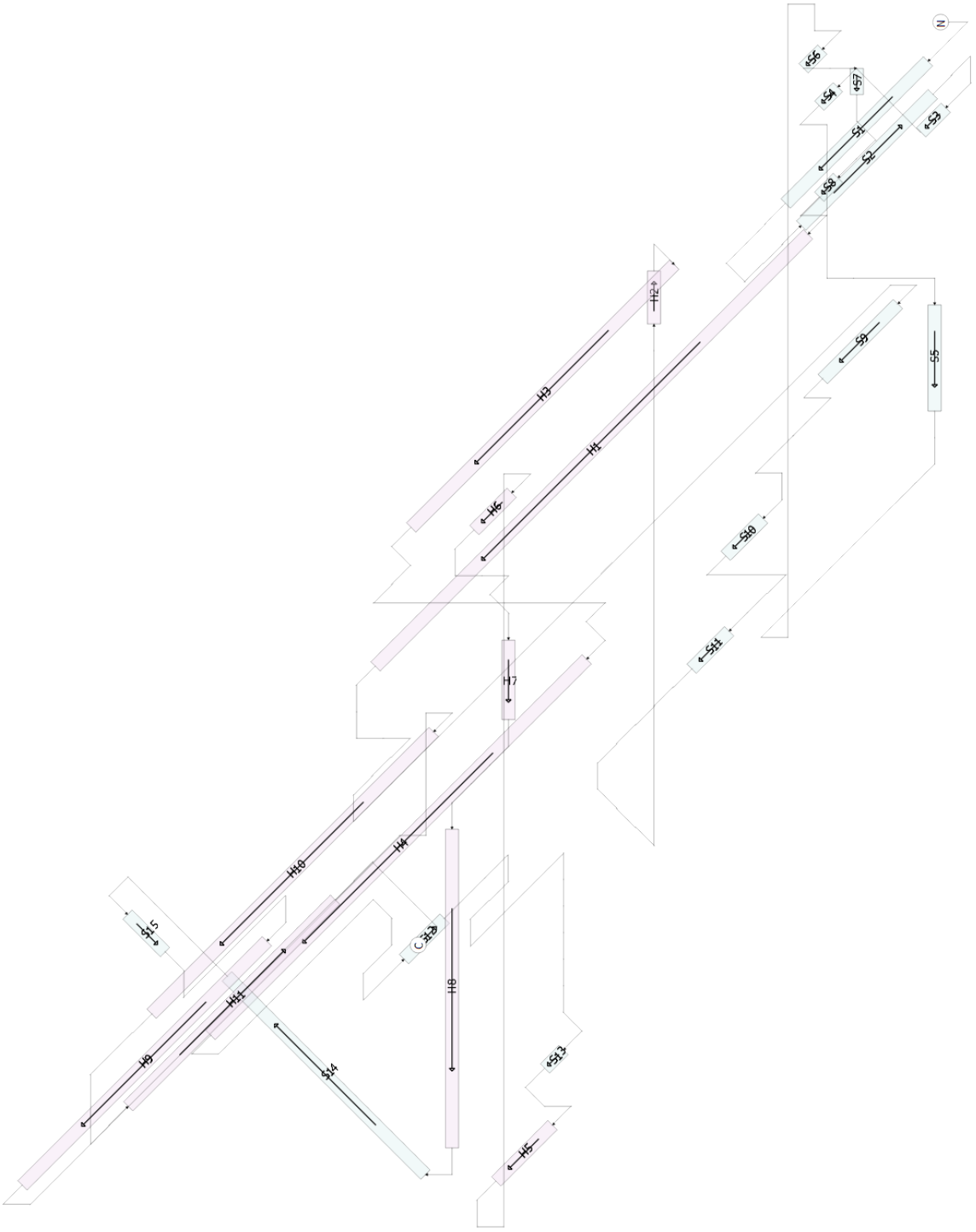


Figure 6.14: The automatically generated schematic representation of 8TLI. The parameter set $k = 4$, $(c_1, c_2, c_3, c_4) = (1, 1, 1, 1)$ is used. This is the first feasible solution which is found after 1114 seconds with an optimality gap of 97.5%. Note that the layout has been rotated by 90° counter-clockwise to better fit on this page.

6.4 Improving the Model

In this section we describe two constraints and one modification in the input data that speed up the computation time needed to find the optimal solution.

6.4.1 Parallelism

The first one bases on the observation that we want sheet structures to be parallel in the final layout. That is, if each pair of structures is connected by a sufficient amount of hydrogen bonds. We therefore apply this constraint to sheets of which at least $f = 67\%$ of vertices in that structure possess a hydrogen bond to another specific structure. Then, depending on order of the hydrogen bonds, we set these two structures to be parallel or anti-parallel by fixing the directions of their first covalent edges. It is sufficient to only fix the first edge, since we set all other structure edges accordingly using the constraints described in Section 5.4.

For anti-parallel structures S_1 and S_2, we use

$$\begin{aligned} dir(u,v) &= dir(s,t) + k - 2k \cdot \mu(s,t) \\ (u,v) &= \text{first}(S_1) \\ (s,t) &= \text{first}(S_2) \end{aligned} \tag{6.1}$$

where $\mu(s,t)$ is a binary variable to describe the modulo operation

$$dir(u,v) = (dir(s,t) + k) \mod 2k$$

within a linear constraint. For parallel structures S_1 and S_2, the constraint simply is

$$\begin{aligned} dir(u,v) &= dir(s,t) \\ (u,v) &= \text{first}(S_1) \\ (s,t) &= \text{first}(S_2) \end{aligned} \tag{6.2}$$

Note that the parameter f is determined experimentally. A higher percentage reduces the number of structures that are aligned in parallel, whereas a lower percentage includes structures with almost no hydrogen bonds at all.

6.4.2 First Edge Direction

The second observation is that Gurobi usually aligns the whole graph in a certain direction during the computation. By fixing the direction of the first covalent edge in the directed path of the graph, Gurobi tries to align all other edges according to this one. In terms of linear constraints, this simply is

$$dir(u,v) = c \tag{6.3}$$

where (u,v) is the first edge in the graph and c is constant, $0 \leq c \leq 2k$.

6.4.3 Loop Structure Size

The last observation bases on the fact that we do not need more than k loop vertices to form a smooth 180° bend, as seen in the first example graph 6.4. Therefore, before adding constraints to the model, we remove vertices and their incident edges from a loop structure until only k vertices are left. We add new edges between the remaining vertices so they form a path again. After a layout is calculated for the reduced model, we then linearly place the removed vertices between the existing loop vertices to reintegrate them into the model. To make this process easier, we do not remove the first and last vertex of loop structures.

This modification reduces the size of models with many loop vertices and speeds up the computation, since there are less edges to regard for the quadratic overlapping constraint in Section 5.3.

To summarise, there are additional constraints that can speed up the computation time needed to find the optimal solution for a graph layout. It remains to be tested how well these work.

6.5 Crashes and Infeasibility

We mention in Section 6.3 that Gurobi crashes when certain constraints are omitted. However, even though the computation of 8TLI works with these, other proteins like 1J7X, 1N6E and 4GHN, which are similiar to 8TLI in size, still crash during the root relaxation. Gurobi is a blackbox in these cases and we can not find out what is causing the crashes. We also attempt to generate layouts for big proteins like 1KN0, but the optimiser is not able to find a feasible starting solution within an hour, even after choosing different approaches Gurobi uses to find solutions.

7. Conclusion

We have described a general two-dimensional graph model of secondary protein structure, using residues as vertices and covalent bonds between these as edges. Based on this model and according to prior agreement with domain experts from Universität Düsseldorf, we formulated the desired characteristics for our schematic representation of the protein's secondary structure. We translated the characteristics into hard and soft constraints and used a mixed-integer program to automatically calculate such a representation. In our effort to make it as visually pleasing as possible by enforcing planarity and minimising the length of bonds between structures, we used constraints that have quadratic impact on the size of the program. We have not been able to completly eliminate these constraints, but by adding them in a lazy way during the computation of the representation we achieved a reduction in the size of the mixed-integer program.

Nevertheless, our method is only feasible for small to medium sized graphs which are sparsely connected, as seen in our provided example graphs. We showed that the formulated constraints work in the intended way and that the program produces a good layout in less than one minute for our example graphs. Unforunately, for our proteins in our case study, we learned that we either find an initial layout in less than one hour or none at all. The one we get is still far from visually pleasing, however. Therefore, further investigation is required to produce feasible solutions and to reduce the time needed to find one. A possible way for the latter two is to divide the protein into segments, then calculating the layout for each of these segments and finally bringing them back together to form the layout of the whole protein.

It also remains to be clarified how other non-covalent bond types like salt bridges can be integrated into the program and what effects they should have on the schematic representation.

7.1 Acknowledgement

I would like to thank the whole Gohlke group at Universität Düsseldorf for letting me visit them in Düsseldorf and giving me insight into their work. I especially appreciate the mentoring Prof. Dr. Holger Gohlke, Christopher Pfleger and Daniel Mulnaes have given me during my stay and their continous assistance throughout the whole creation of this Bachelor thesis. I am also deeply grateful for my advisors Dipl.-Inform. Benjamin Niedermann and Dr. Martin Nöllenburg, who not only enabled me to write this thesis, but

also provided me with feedback in meetings and via e-mail correspondence from the very beginning until the very end.

Bibliography

[CR14] Daniel Chivers and Peter Rodgers. Octilinear force-directed layout with mental map preservation for schematic diagrams. In *Diagrammatic Representation and Inference*, pages 1–8. Springer, 2014.

[GO14] Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2014.

[Han14] Bob Hanson. Jmol: an open-source java viewer for chemical structures in 3d. `http://jmol.sourceforge.net/`, October 2014.

[Hor05] V Koichi Kimura V Yuzo Horikoshi. Bio-based polymers. *Fujitsu Sci. Tech. J*, 41(2):173–180, 2005.

[KCdlP+14] Otto J.P. Kyrieleis, Jonathan Chang, Marcos de la Peña, Stewart Shuman, and Stephen Cusack. Crystal structure of vaccinia virus mrna capping enzyme provides insights into the mechanism and evolution of the capping apparatus. *Structure*, 22(3):452 – 465, 2014.

[KKI+02] Wataru Kagawa, Hitoshi Kurumizaka, Ryuichiro Ishitani, Shuya Fukai, Osamu Nureki, Takehiko Shibata, and Shigeyuki Yokoyama. Crystal structure of the homologous-pairing domain from the human rad52 recombinase in the undecameric form. *Molecular cell*, 10(2):359–371, 2002.

[Lun14] R.L. Lundblad. *Chemical Reagents for Protein Modification, Fourth Edition*. Taylor & Francis, 2014.

[MHLK05] Andreas Moll, Andreas Hildebrandt, Hans-Peter Lenhof, and Oliver Kohlbacher. Ballview: an object-oriented molecular visualization and modeling framework. *J Comput Aided Mol*, 19(11):791–800, 2005.

[NW11] Martin Nöllenburg and Alexander Wolff. Drawing and labeling high-quality metro maps by mixed-integer programming. *Visualization and Computer Graphics, IEEE Transactions on*, 17(5):626–641, 2011.

[PM14] Christopher Pfleger and Daniel Mulnaes. A GUI for interactive constraint network analysis and protein engineering for improving thermostability. `http://cpclab.uni-duesseldorf.de/software`, 2014.

[Sch10a] Tamar Schlick. *Molecular Modeling and Simulation: An Interdisciplinary Guide*, volume 21. Springer, 2010.

[Sch10b] Schrödinger, LLC. The PyMOL molecular graphics system, version 1.3r1, August 2010.

[Tam13] Roberto Tamassia. *Handbook of Graph Drawing and Visualization*. Chapman& Hall/CRC, 2013.