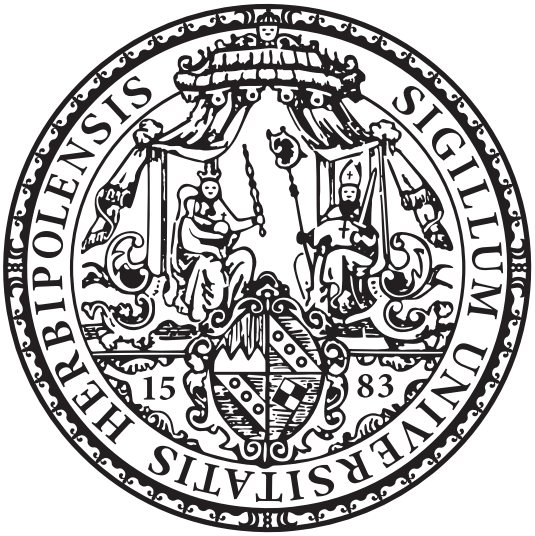
Bachelor Thesis

Crossing Reduction in Circular Layouts under Grouping Constraints

Arash Torabi Goodarzi

Date of Submission: 15. September 2022
Advisors: Prof. Dr. Alexander Wolff
Tim Hegemann, M. Sc.



Julius-Maximilians-Universität Würzburg
Lehrstuhl für Informatik I
Algorithmen und Komplexität

Abstract

Crossing reduction in graph drawing aims to produce readable drawings of graphs by reducing the number of crossings in them. This is an important aspect for graph drawings since it impacts the visual clutter directly. A high visual clutter reduces the readability of a drawing which increases the time and cost when it comes to maintaining a network using its graph visualization. Crossing reduction in circular layouts is the problem of creating a graph drawing in which the vertices are placed along the perimeter of a circle and the number of produced crossings is minimized. This thesis takes on a variation of this problem where certain vertices of a graph are required to be placed next to each other. We suggest a fast heuristic algorithm for this NP-complete problem derived from the work of Brandes and Baur in 2005 and show that our heuristic produces only 4.39% more crossings than an exact solution through a series of experimental evaluations. We also present an experimental analysis that shows a case in which the heuristic performs particularly well.

Zusammenfassung

Kreuzungsminimierung der Graphenzeichnen setzt sich das Ziel, lesbare Visualisierungen für Graphen zu produzieren, indem die Anzahl der Kantenkruzungen minimal gehalten werden. Dies ist ein wichtiger Aspekt der Graphenzeichnen, da die visuelle Durchschaubarkeit der Zeichnung direkt von der Anzahl der Kreuzungen beeinflusst wird. Niedrige Durchschaubarkeit reduziert die Lesbarkeit. Somit steigen die Zeit und Kosten der Aufrechterhaltung von Netzwerken durch Nutzung ihrer Graph-Visualisierung. Kreuzungsreduktion in zirkulären Layouts ist das Problem der Zusammenssetzung eines Graphenzeichnens, wobei die Knoten des Graphen auf der Umfang eines Kreises platziert werden und die Anzahl der Kreuzungen möglichst minimiert werden sollen. Diese Arbeit beschäftigt sich mit einer Variation dieses Problems, wo bestimmte Knotengruppen nebeneinander platziert werden müssen. Wir schlagen einen schnellen Heuristik-Algorithmus für dieses NP-völlständiges Problem vor, die sich auf die Arbeit von Brandes und Baur in 2005 basiert. Wir Zeigen dann durch eine Reihe von Experimenten, dass unsere Heuristik nur $4,39\%$ mehr Kreuzungen als ein exakter Algorithmus produziert und präsentieren ebenfalls eine experimentelle Analyse, welche zeigt, dass die Heuristik in einem bestimmten Fall besonders gut funktionieren kann.

Contents

1 Introduction

Graph drawing is one of the most researched fields of graph theory in computer science. With the expansion of the internet the need for drawing aesthetically pleasing and readable graphs is growing. This is an important tool for administrators and enables them to maintain their networks with lower cost, more ease and more efficiency. There are different criteria that make a graph drawing aesthetically pleasing and more readable such as total length of edges and grouping of clusters. The most important criteria however, is the number of crossings in a drawing, since this has a direct impact on visual clutter and plays an important role in the readability of the graph.

There are many ways to form a graph drawing to create a so called layout. A popular way is to form a circle. A circular graph layout is a formation for a graph drawing, in which the vertices are placed along the perimeter of a circle. In such layouts, if we assume that none of the edges may be drawn outside of the circle, the only structure needed to specify the drawing is a list establishing the ordering of the vertices. It is an important objective to order these vertices in a way that the number of occurring edge crossings in the layout is minimized in order to produce aesthetically pleasing circular drawings. This is the problem of *circular crossing minimization*, which is well-known to be NP-complete [MKNF87].

A variation of this problem is to add further grouping constraints to it, meaning to require certain subsets of vertices to be placed next to each other. This variation is particularly useful for industrial network visualizations, where certain nodes often belong to a sub-network and should therefore be placed close to one another. A real-world example of this is a company whose network is sub-divided into four sub-nets for each of the four geographical directions and requires to have a circular drawing of its network, where it is still possible to recognize each sub-network.

The goal of this thesis is to present and examine a heuristic method drived from the work of Brandes and Baur [BB05] as well as to evaluate its performance against an optimal solution with exponential time complexity.

1.1 Previous Work

Several methods have been presented to overcome the NP-completeness of the circular crossing minimization problem, since circular drawings are widely used in the field of graph drawing. A $O(\log^2 n)$-approximation algorithm for the circular layout problem has been presented by F. Shahrokhi et al. [SSSV95] to overcome this problem.

E. Mäkinen [Mä88] aims to overcome this by presenting a greedy heuristic method to minimize the circular dilation in the circular layout. Circular dilation is defined as

the total edge length for a given circular drawing. This heuristic however, does not minimize the number of edge crossings and also rather increases this by placing vertices with higher degrees next to each other to strive for shorter edges.

U. Dogrusoz et al. [DMM96] provide an explanation of the algorithm used by the Circular Library of the Graph Layout Toolkit, which is a family of graph layout libraries that are designed to be integrated into GUI programs, as well as some suggestions to improve their performance. The algorithm gives layouts that are clustered in circular groups. The time complexity of the algorithm is $O(n^2 + e)$, where e is the number of edges in a graph cluster and n the number of vertices.

J. Six et al. [ST99] give a two-phase heuristic algorithm with a time complexity of $O(m^2)$ to minimize crossings in a circular layout, where m is the number of edges in the graph. The first phase of their heuristic is inspired by the recognition algorithm for outerplanar graphs [Mit79] and gives a crossing free circular layout for them. The complexity of this first phase is $O(nm)$. However, if the input graph is outerplanar the algorithm finishes in $O(n)$ time. The second phase of the algorithm is a crossing reduction technique used to reduce the crossing number into a local minimum.

A similar strategy to the strategy of J. Six has been presented by U. Brandes and M. Baur [BB05]. In that paper another two-phase heuristic has been presented and tested against the heuristic of J. Six. These tests showed that that the method of Brandes and Baur performs significantly better than that of J. Six. This method counts as the current state of the art in circular crossing reduction and is the basis of our work. A detailed explanation of the algorithm is given in Chapter 4.

1.2 Contribution

We propose a heuristic derived from the state of the art heuristic of Brandes and Baur for circular crossing reduction that respects grouping constraints. Our contribution in this thesis is as follows.

- We define the problem of crossing reduction in grouped circular layouts and we give an integer linear program formulation to solve both this problem and the problem case of not having groupings.
- We describe the algorithm of Brandes and Baur for crossing reduction in circular layouts alongside the acceleration methods for its implementation and modify it to respect grouping constraints.
- We examine the performance of the heuristic against the ILP optimal solution in an experimental analysis for the both cases of having groupings and not having grouping constraints.

1.3 Organization

In Chapter 2, we start by laying out some ground work definitions and notations used in the document. We also define our problem at hand formally and discuss its NP-completeness. In Chapter 3 then, we define the two problems defined in Chapter 2 as a 0-1-linear integer program. Chapter 4 reviews the algorithm of Brandes and Baur for crossing reduction in circular layouts. We then modify this algorithm to support grouping constraints in Chapter 5. In Chapter 6, we perform the experimental evaluation of the heuristic against the optimal ILP solution.

2 Preliminaries

In this Section, we define some notations and review some backgrounds, that will be used throughout this document.

throughout this paper, we let $G = (V, E)$ be a simple undirected graph with $n = |V|$ vertices and $m = |E|$ edges. The notation $N(v) = \{u \in V : \{u, v\} \in E\}$ denotes the set of all neighbours of vertex $v \in V$. A *circular layout* of G is given by a bijection $\pi : V \to \{0, \ldots, n-1\}$, which is to be interpreted as a clockwise sequence for the circular ordering.

2.1 Crossings in a Circular Drawing

Given a circular drawing π and a reference vertex $s \in V$ we obtain a linear ordering of vertices denoted by $\prec_s^\pi$ and defining it as

$$u \prec_s^\pi v \Leftrightarrow ((\pi(u) - (s)) \mod n) < ((\pi(v) - \pi(s)) \mod n)$$

for all $u, v \in V$. This denotes that if we traverse the vertices of the graph along the circular layout π starting from vertex s in a clockwise manner, we encounter vertex u before vertex v. Given a such ordering, two edges $e_1 = \{u_1, v_1\}$ and $e_2 = \{u_2, v_2\}$ cross in the circular drawing, if $X_\pi(\{u_1, v_1\}, \{u_2, v_2\}) = 1$, where the function X_π is defined as

$$X_\pi(\{u_1, v_1\}, \{u_2, v_2\}) = \begin{cases} 1 & \text{if } u_1 \prec_s^\pi u_2 \prec_s^\pi v_1 \prec_s^\pi v_2 \\ 0 & \text{otherwise} \end{cases}$$

assuming that $\pi(u_i) < \pi(v_i)$ for $i \in \{1, 2\}$. Formulating this into words of natural language, the end vertices of the two edges e_1 and e_2 must appear alternately in a circular traverse for the edges to cross. Note that if the assumption $\pi(u_i) < \pi(v_i)$ is not given, there are then 8 cases to be checked for every pair of edges that result in a crossing. Since s is an arbitrary reference vertex, X_π detects a crossing if one of the following conditions is given. For this listing, let the edges to be checked for crossing be $e_1 = \{a, b\}$ and $e_2 = \{c, d\}$.

1. $\pi(a) < \pi(c) < \pi(b) < \pi(d)$
2. $\pi(c) < \pi(a) < \pi(d) < \pi(b)$
3. $\pi(a) < \pi(d) < \pi(b) < \pi(c)$
4. $\pi(d) < \pi(a) < \pi(c) < \pi(b)$
5. $\pi(b) < \pi(c) < \pi(a) < \pi(d)$
6. $\pi(c) < \pi(b) < \pi(d) < \pi(a)$
7. $\pi(b) < \pi(d) < \pi(a) < \pi(c)$
8. $\pi(d) < \pi(b) < \pi(c) < \pi(a)$

These conditions can be verified in an implementation to check whether a pair of edges cross in a circular layout implemented in a linked list. Given this, we can now define the *crossing number* for a given circular layout as follows. For a circular layout π the crossing number $\chi(\pi)$ is the number of crossings in layout π.

$$\chi(\pi) = \sum_{e_1,e_2 \in E} X_\pi(e_1, e_2)$$

Calculating the crossing number in a brute force manner takes $O(m^2)$ time. An algorithm has however been presented by Tollis and Six [ST06] that calculates the crossing number of a circular layout in $O(m + x)$ time, where x is a variable depending on the layout. Also this algorithm has a worst case running time of $O(m^2)$.

Two vertices $u, v \in V$ are *consecutive*, denoted as $u \curvearrowright_\pi v$, if $\pi(v) - \pi(u) \equiv 1 \mod n$.

2.2 Problem Definition

The problem of discussion in this paper is the problem of crossing reduction in circular layouts under grouping constraints which is a derivation of the problem of simple crossing reduction in circular layouts. These problems are formally defined as follows.

Crossing Reduction in Circular Layouts (CR)
Input: A graph $G = (V, E)$
Question: Can we give a circular layout π such that $\chi(\pi)$ is minimized?

Crossing Reduction in Grouped Circular Layouts (CRG)
Input: A graph $G = (V, E)$ and a list $S_1, \ldots, S_g \subseteq V$ of groups such that $S_i \cap S_j = \emptyset$ for $i \neq j$ and $\bigcup_{i=1}^{g} S_i = V$.
Question: Can we give a circular layout π such that $\chi(\pi)$ is minimized and for every $u \in S_i$ and $v \in S_j$ with $i < j$ it holds that $\pi(u) < \pi(v)$?

The NP-completeness of CR has been shown by Masuda et al. in 1987 [MKNF87]. From this, it is easy to see that CRG is also NP-complete.

Theorem 2.1. *Crossing Reduction in Grouped Circular Layouts is NP-complete.*

Proof. CR is a special case of CRG where number of groups g is set to one. The containment in NP can be shown by a brute force algorithm to test all of the at most $n!$ orderings of vertices in a list. From a such ordering it is possible to construct a circular layout in polynomial time. □

Note that a graph has a planar circular drawing if and only if it is outerplanar. A recognition algorithm for outerplanar graphs [Mit79] can be extended onto a drawing algorithm to give a crossing-free circular layout [ST99].

3 ILP Formulation of the Problem

Integer linear programs (ILP) are widely used for optimization problems. In this chapter we present a 0-1 ILP for the problem of crossing reduction both in circular layouts and grouped circular layouts as defined in Section 2.2 as the basis for our experimental evaluation in Chapter 6. To learn more about integer linear programming, see [GG11].

3.1 ILP without Groupings

At first, let us focus on the simpler crossing reduction problem without groupings. We will then modify it to respect grouping constraints afterwards. Let a graph $G = (V, E)$ be given.

3.1.1 First Type of Variables

The model we use for our ILP uses an ordering similar to the ordering defined in Section 2.1. We build an ordered listing of vertices to represent our circular drawings. It is clear that there are then n such listings that represent the same drawing, but that does not change our results. So we can treat the listing the same way as π in Chapter 2. For every pair of vertices $u, v \in \binom{V}{2}$ we define a variable X_{uv} as

$$X_{uv} = \begin{cases} 1 & \text{if } \pi(u) < \pi(v) \\ 0 & \text{otherwise} \end{cases}$$

that denotes whether vertex u comes before v in order. We also define a symmetrical variable X_{vu} for the same pair that is the negation of X_{uv}. That is X_{vu} is equal to 1 if and only if v comes before u in order and 0 otherwise. We call these variables the X-variables.

Given a valid assignment of these variables that satisfy the constraints explained in Section 3.1.2, an ordering with a minimum and a maximum is created from which it is easily possible to obtain a circular drawing.

3.1.2 Constraints for Transitivity

We add two types of constraints that are needed to ensure that the X-variables retain an actual ordering.

The first type of constraints were mentioned while defining the X-variables. Since we have both X_{uv} and X_{vu} for every pair of vertices u and v, we must first ensure that they correspond as the negation of each other. This is easily done by adding a constraint for

every pair of vertices u and v to make $X_{uv} = 1 - X_{vu}$. We therefore add the following constraint.

$$X_{uv} + X_{vu} = 1$$

The second type of variables needed for the ordering to be retained is that for every 3 vertices, the respective X-variables must be correspondent to that of an ordering. This is best explained with and example. Let u, v, and r be vertices. The second type of variables is to ensure that if $X_{uv} = 1$ and $X_{vr} = 1$, then X_{ur} must be 1. Because if v comes after u and r comes after v in order then necessarily r comes after u. This is the transitivity of order. And reversely if $X_{uv} = 0$ and $X_{vr} = 0$ then $X_{ur} = 0$ is required. The exact cases are shown in the following table.

X_{uv}	X_{vr}	X_{ur}
0	0	0
0	1	0 or 1
1	0	0 or 1
1	1	1

These requirement is held if $0 \leq X_{uv} + X_{vr} - X_{ur} \leq 1$ is satisfied for every $u, v, r \in \binom{V}{3}$. We therefore add the following two requirements for every triple of vertices u, v, r.

$$1 \geq X_{uv} + X_{vr} - X_{ur}$$

$$0 \leq X_{uv} + X_{vr} - X_{ur}$$

3.1.3 Objective Function

Our objective is obviously to reduce the number of crossings. Therefore an objective function that correctly counts the crossings in the circular layout represented by the X-variables is necessary. For that, we introduce 8 new variables for every pair of vertices corresponding to the eight cases of crossing discussed in Section 2.1 that are equal to 1 if the case of crossing has appeared. The name of the variables have the schema Y_{acbd} for edges $\{a, b\}$ and $\{c, d\}$, where the ordering of the index of the variable is correspondent to the ordering appeared in the specific case for crossing. We call these variables the Y-variables. To understand how the constraints for the Y-variables are constructed, let us take the case of crossing 1 in Section 2.1 for the pair of edges $\{a, b\}$ and $\{c, d\}$. The case is constructed of three requirements. These are $\pi(a) < \pi(c)$, $\pi(c) < \pi(b)$ and $\pi(b) < \pi(d)$. Each of these requirements is fulfilled if their correspondent X-variable is equal to 1. We therefore create the following constraints for this crossing case.

$$Y_{acbd} \geq 0$$

$$Y_{acbd} \geq X_{ac} + X_{cb} + X_{bd} - 2$$

Given these two constraints the Y-variable will definitely become 1 if the crossing case is given. That is if the corresponding X-variables are all equal to 1. Such variables

are created with the mentioned two constraints for every of the eight crossing cases for every pair of edges. In total that makes $8 \cdot \binom{m}{2}$ Y-variables and $16 \cdot \binom{m}{2}$ constraints. The objective function is the sum of all Y-variables. Given that this objective function is to be minimized, each Y-variable will be equal to 0 whenever this is possible. With this, a correct ILP formulation for our ungrouped problem is given.

3.2 ILP with Groupings

The same ILP described in Section 3.1 can be easily modified to support groupings. Let us assume that a grouping $S_1, \dots S_g$ with The constraints described in the definition of CRG in Section 2.2 is given. A grouping requires for every $u \in S_i$ and $v \in S_j$ with $i < j$ to hold that $\pi(u) < \pi(v)$. We can translate this easily using the X-variables defined in the last section. For every $u \in S_i$ and $v \in S_j$ with $i < j$ we add the following constraint.

$$X_{uv} = 1$$

A correct ILP formulation of the grouped problem is thusly given.

The described ILPs in this chapter can be written in a desirable ILP solver such as CPLEX or Gurobi [Cpl09] [Gur22]. These solvers will then return a vector that gives a valid assignment of X and Y-variables for a given input graph such that the objective function is minimized and all of the set constraints are fulfilled. The assignments given for the X-variables provide a full ordering for the vertices of the graph. A sorting algorithm that sorts the vertices according to this ordering then brings us the listing of vertices that represents the desired circular drawing. The number of crossings can also be calculated easily by adding up all of the Y-variables. We used an implementation of these two ILPs to obtain the circular layouts in Figure 3.1

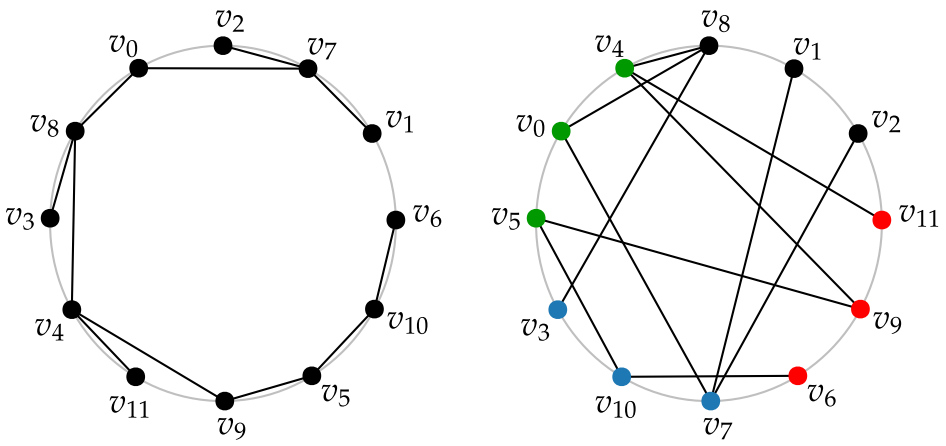


Fig. 3.1: A graph with 12 vertices and 11 edges has been circularly drawn in a circle using the ILP with no groups (left). The vertices of the same graph have been grouped randomly into 4 groups and drawn in a circle using the ILP with groupes (right). Groups have been visualized with colors.

4 Algorithm of Brandes and Baur

Ulrik Brandes and Michael Baur proposed a heuristic algorithm for crossing reduction in circular layouts [BB05]. This algorithm is the basis of the algorithm we propose in Chapter 5 to support grouping constraints. In this chapter, we therefore give a brief explanation of how the algorithm of Brandes and Baur works and discuss its time complexity.

The heuristic proposed by Brandes and Baur works in two stages. First, it creates an inital drawing using a greedy appending paradigm. The second stage is then an adaption of the sifting procedure for layered layouts, which is a method for local optimization.

4.1 Initial Layout

The basic idea for creating the initial layout in the Brandes and Baur heuristic is the following.

Algorithm 1: General Greedy-Append Heuristic

```
1 place start vertex s ∈ V arbitrarily;
2 remove s from V;
3 while V ≠ ∅ do
4 |   greedily choose v ∈ V;
5 |   append v at either end of the layout;
6 |   remove v from V
```

At line 1, a starting vertex should be chosen. In the implementation used for the experiments done in Chapter 6, we decided to choose the starting vertex randomly. Two other degrees of freedom are yet to be discussed. First, how exactly the next vertex to append should be chosen and second, to which end should the chosen vertex be appended. Several paradigms for each degree of freedom have been proposed and tested by Brandes and Baur. We will however only discuss the paradigms that performed best in their experimental evaluations.

During the procedure of greedy-append some vertices are placed and others are not. An *open* edge is an edge that has exactly one placed end. An edge is called *closed* if both its endings are placed.

The following paradigm is used for determining the insertion sequence for line 4 in Algorithm 1.

Connectivity. At each step a vertex with the least number of unplaced neighbors is

selected. If a tie appears it is broken in favor of the vertex with higher number of placed neighbors.

The paradigm used for the second degree of freedom is the following.
Crossings. Each chosen vertex is appended to the end that yields fewer crossings of the edges being closed and the open edges.

Experiments done in the original paper suggest a significant gain in performance compared to other tested paradigms, when using the connectivity and the crossings paradigms in combination. Both the mentioned paradigms can be implemented with their respective acceleration strategies. These are explained in Section 4.3. Using these acceleration methods, the initialization algorithm can be implemented in $O((n+m)\log n)$ time.

4.2 Circular Sifting

Circular sifting is a local round-based optimization process for crossing reduction in circular layouts. The general idea of the process is the following.

Each vertex v is moved along the layout, and the crossing number is calculated in each position. v is then placed in the position with the lowest calculated crossing number. One round of circular sifting does this process for each vertex, changing the layout in every step.

In every step of one round of circular sifting, a vertex is swapped with its consecutive vertex. The change of crossing number then only depends on the edges that are incident to the two vertices that are being swapped, because the edges not incident to the two swapping vertices do not change their crossing status when two consecutive vertices are being swapped. We therefore define the crossing number

$$c_{uv}(\pi) = \sum_{x \in N(u)} \sum_{y \in N(v)} X_\pi(\{u,x\},\{v,y\})$$

for every pair of consecutive vertices $u \curvearrowright_\pi v \in V$, to count the number of edge crossings that are between the edges incident to u and the edges incident to v. The following lemma then gives an exchange property to track the number of crossings.

Lemma 4.1. *Let $u \curvearrowright_\pi v \in V$ to be consecutive vertices in the circular drawing π. Let π' be the drawing obtained when u and v are swapped. Then it holds*

$$\begin{aligned}\chi(\pi') &= \chi(\pi) - c_{uv}(\pi) + c_{uv}(\pi') \\ &= \chi(\pi) - \sum_{x \in N(u)} |\{y \in N(v) : y \prec_x^\pi u\}| + \sum_{y \in N(v)} |\{x \in N(u) : x \prec_y^\pi v\}|.\end{aligned}$$

Proof. Figure 4.1 shows the second equality. See [BB05] for a more detailed proof. □

These two sums can be calculated easily in an implementation. Algorithm 2 in [BB05] gives an accelerated suggestion to implement the circular sifting in $O(nm)$. The sifting can be repeated and in each repetition the number of crossings can only be reduced. It

is possible to repeat the sifting process so many times, that the number of crossings does no longer improve. In this case, the complexity is $O(nm \cdot w)$, where w is the number of repetitions done.

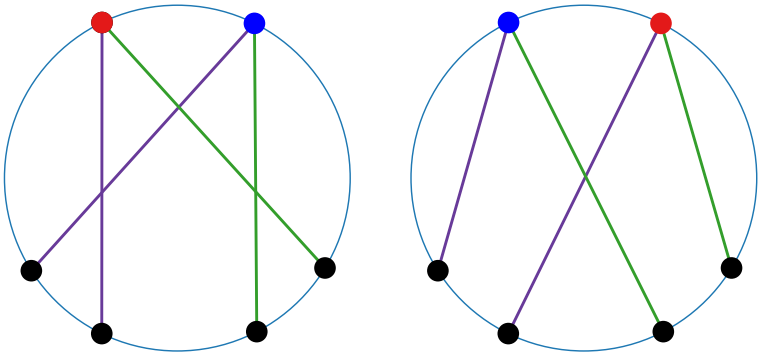


Fig. 4.1: After swapping two consecutive vertices (right) exactly those incident edges cross that did not cross before (left).

4.3 Accelerating the Initialization

In this section, we explain the acceleration methods mentioned in theorem 2 of [BB05].

The connectivity insertion order presented in Section 4.1 can be realized using a two-dimensional priority queue for unplaced vertices, where the first key stores the number of unplaced neighbors and the second key the number of placed neighbors. This priority queue can be sorted increasingly by the first key and then decreasingly by the second key to break ties in $O(n \log n)$. Each extraction and update operation requires $O(\log n)$ time. On the other hand, each vertex is extracted once and each edge causes exactly one update operation. The total time needed is therefore $O((n + m) \log n)$ for greedy choosing.

The crossing paradigm needs the crossings between open and closed edges to be calculated. When creating the initial layout, we implement the circular layout in a linked list. The decision to be made is whether we should insert the chosen vertex at the beginning of the linked list or at its end. The prefix sum at a vertex v is the number of open edges that are before v in the list. The suffix sum is the number of open edges that are after v in the list. Both numbers are calculated with exclusion of the open edges of v itself. Using prefix and suffix sum at each vertex, it is possible to calculate the number of crossings between open edges and edges being closed when appending a vertex at either end. These sums can be efficiently calculated and maintained for every vertex using a balanced binary tree as follows. The tree will store in its leaves the number of open edges for each placed vertex. The inner nodes store the sum of the values of their two children. The prefix sum at a vertex is then the sum of the values of the left children of the nodes that are on the path from the root to that vertex's correspondent leaf, excluding the nodes on the path themselves. The suffix sum can be calculated symmetrically. With this structure the calculation of the prefix and suffix sum to decide the insertion takes

$O(\log n)$ time and $O(d(v)\log n)$ is needed to update the tree after each insertion. Since exactly n insertions are done, the total time complexity is $O((n+m)\log n)$.

As an example, in Figure 4.2 we are adding vertex a to the layout which connects with the vertices b, d and f and we are trying to decide which end of the layout it should be added to. To decide this, the number of edge crossings with open edges that would be created for either end should be calculated. This can be calculated using the prefix and suffix sums at the connecting nodes to a, obtained from the balanced binary tree in Figure 4.3, as follows. For addition to the right suffix sum is used and for the left, we use the prefix sum. Let us consider the case of adding to the right. The suffix sum of f is zero, so an addition to the right side of f does not produce any crossings with open edges. At d, the suffix sum is 2, but one of the open edges after d is connected to f and is being closed by a. So the number of crossings produced by the connection to d is 1. Lastly, the suffix sum of b is 6 and two of the edges after b are being closed by a, this means that the number of produced crossings with open edges by the connection of a and b is 4. This adds up to 5 crossings with open edges for an addition to the right. This number can be calculated for the addition to the left symmetrically using the prefix sum and based on that, the decision can be made.

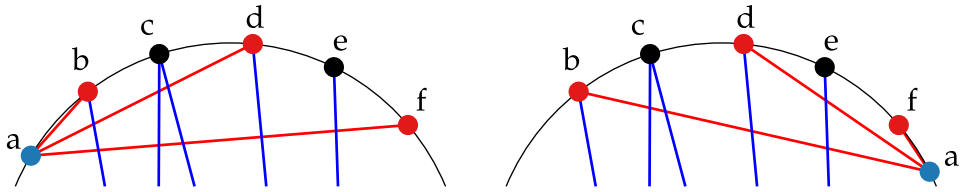


Fig. 4.2: Vertex a should be appended at the right because this makes 5 crossings with open edges while adding to left would make 8 crossings with open edges.

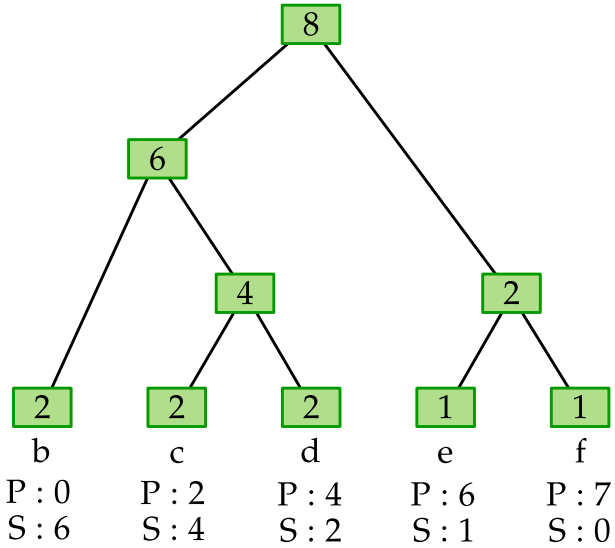


Fig. 4.3: "P" denotes the prefix sum at each node and "S" denotes the suffix sum.

5 Supporting Groupings

In this chapter, we present a modification of the Brandes and Baur heuristic explained in Chapter 4 to support grouping constraints as explained in Section 2.2. To do so, we modify the initialization and the sifting stages each. The main idea for these modifications is to do the same as in ungrouped layouting separately for each group at every stage, while calculating the crossing numbers in a way that cross-group crossings are as well considered. We therefore represent our grouped circular layouts in a two-dimensional linked list in contrast to the one-dimensional lists for the ungrouped normal circular layout. A circular layout as defined in Chapter 2 can be obtained from a 2D-list by appending every inner list to one linked list for our circular layout, and honoring the ordering of the groups while doing so.

The precise modifications necessary to achieve grouping constraints in the heuristic are explained in the following.

5.1 Grouped Initialization

The initialization used by Brandes and Baur is a greedy-append heuristic. Our goal is to do the same heuristic for each group separately, while keeping the layout together for the calculation of crossing numbers.

Let $G = (V, E)$ be an instance of CRG with the g groups $S_1, \dots S_g \subseteq V$ as defined in Section 2.2. Algorithm 1 begins by choosing one vertex randomly to start with. We modify this by starting with g random vertices each from one of the given groups. We use these vertices to create the first initialization of our inner lists according to the ordering of S_i. These inner lists are stored in a list that represents our two-dimensional grouped layout. We then modify the greedy choosing by applying it g times for each group and every time using the connectivity paradigm explained in Section 4.1. Having chosen the vertices to append we then decide which end of each group to append them to using the crossings paradigm explained in Section 4.1. These modifications bring us our grouped initialization heuristic. The precise definition of the modified algorithm is shown in Algorithm 2.

The time complexity of the greedy-append heuristic does not change with the modifications done. Choosing the beginning vertices takes $O(g)$ time and it is sensible to assume $g \leq n$. This part therefore has a loose time complexity of $O(n)$. The choose and append part of the algorithm takes place exactly n times and the accelerations explained in Section 4.3 can still be used in the grouping case. We therefore do not exceed the time complexity of $O((n + m) \log n)$.

Algorithm 2: General Grouped Greedy-Append Heuristic

```
1  Let list L = [S'_1, ..., S'_g], where S'_i is a list of vertices;
2  foreach S'_i ∈ L do
3      place start vertex s ∈ S_i in S'_i chosen arbitrarily;
4      remove s from S_i;
5  while S_1, ..., S_g ≠ ∅ do
6      foreach S'_i ∈ L do
7          if S_i ≠ ∅ then
8              greedily choose v ∈ S_i;
9              append v at either end of S'_i;
10             remove v from S_i;
```

5.2 Grouped Circular Sifting

Circular sifting in the Brandes and Baur heuristic is essentially the process of dragging each vertex around the circular layout one-by-one and putting it in its locally optimal position. We modify this heuristic to drag each vertex inside of its group. The precise way to achieve this is the following. For each vertex, we first move it forward through the other vertices one-by-one and then after arriving at the end of the group, we move it backward towards the first index of its group. During this sifting process, we keep track of the crossing number as explained in Section 4.2 and put each vertex in its locally optimal position. This can be done because in this manner, we only swap consecutive vertices and therefore the change of the crossing number depends only on the edges that are incident to these two consecutive vertices. This means that Lemma 4.1 holds for this manner of grouped circular sifting.

The worst case of grouped circular sifting is the case of having one group only, which is equivalent to the ungrouped case discussed in Section 4.2. Using the grouped sifting method mentioned, we would do twice as many sifts as the ungrouped algorithm. Hence, the time complexity for one round of grouped circular sifting does not succeed the $O(nm)$ margin with an efficient implementation that uses adjacency lists that are sorted according to the circular layout. This process can again be repeated until the crossing number does not improve by one round of sifting and the time complexity would then be $O(nm \cdot w)$, where w is the number of repetitions.

6 Experimental Results

In this chapter, we are going to review the extensive experiments performed to test the performance of the heuristics presented in Chapters 4 and 5. As a measure for the performance, we used an implementation of the integer linear program defined in Chapter 3 with the Gurobi solver [Gur22].

All the tested algorithms have been implemented by myself in Java using the iPraline Library for Java from University of Würzburg [WZ21] as the graph data structure. The Oracle Standard-JDK 17.0.2 was the development kit of choice. The instances of the ILP were also created using the Java-API of Gurobi. The experiments were performed on a laptop with the following specifications.

- Manufacturer/Model: Dell Alienware m15 R1
- CPU: Intel Core i7-8750H @4.1 Ghz – 6 Cores and 12 Threads
- RAM: 32 GB of DDR4
- Operating System: Windows 11 21H2 64 bit

The charts were drawn using *Matplotlib* on Python 3.10, where each data point is the average the the results for each size and the result for each graph is the average of the results of 10 runs with a different random initialization process.

The following set of graphs were used for the experimental evaluations. These are available for public to download at http://www.graphdrawing.org/data.html.

- *Rome graphs.* A set of 11534 graphs with a size of 10 to 107 vertices.

6.1 Evaluation of Algorithm without Groupings

The ungrouped heuristic of Chapter 4 has been tested by Brandes and Baur extensively in [BB05] in comparison to other heuristics for circular crossing reduction. We have however, tested it against the ILP described in Chapter 3. Because of the long running time required by the ILP to find an optimal solution, we have only done the comparison for smaller graphs of the data sets.

The following chart in Figure 6.1 shows the result of the implementation tested on the Rome Graphs with a size of 10 to 80 vertices. Each data point represents the average of the crossings made for the graphs of each size.

The charts in Figure 6.2 show the performance of the heuristic in comparison to the ILP for the Rome Graphs that have less than or equal to 26 vertices. The ILP took

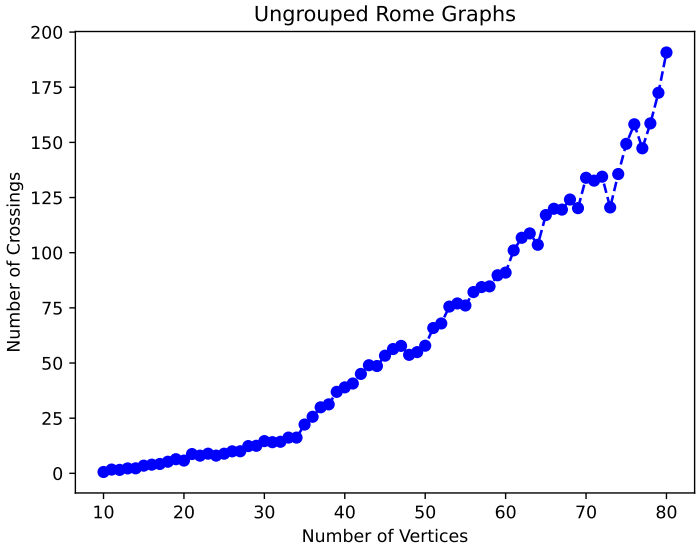


Fig. 6.1: Result of heuristic - A total of 9409 Rome Graphs with the size of 10 to 80 vertices were plotted.

longer than the allowed 3 minutes per graph for bigger graphs to find an optimal solution and the test could therefore only be done for smaller graphs. In average, 23.29% more crossings were produced by the heuristic than the ILP amongst the tested Rome Graphs.

The following Figure 6.3 is the result drawing produced by the heuristic in comparison to that of the ILP on a Rome Graph with 15 vertices and 16 edges to give a feeling of the results.

The tendency is that with growing number of vertices, the heuristic will have a harder time finding an optimal solution. This was also confirmed in our single tests with graphs that had higher number of vertices.

6.2 Evaluation of Algorithm with Groupings

In this section, we will test the heuristic presented in Chapter 5 for solving the problem of crossing reduction in circular layouts under grouping constraints as defined in Section 2.2 and evaluate its performance against the ILP formulation for the problem presented in Chapter 3. For this, we have used the Rome Graphs and grouped their vertices randomly in 4 groups such that the number of vertices per group is distributed as evenly as possible. Since the number of possible orderings for the circular layout is significantly lower because of the grouping constraints, the ILP was able to find the optimal solution for all of the Rome Graphs in less than 3 minutes in our experiments. This enabled us to get a better vision of the performance of the heuristic. The results are plotted in Figures 6.4 and 6.5. This is also one of the reasons that the grouped heuristic is expected to perform better.

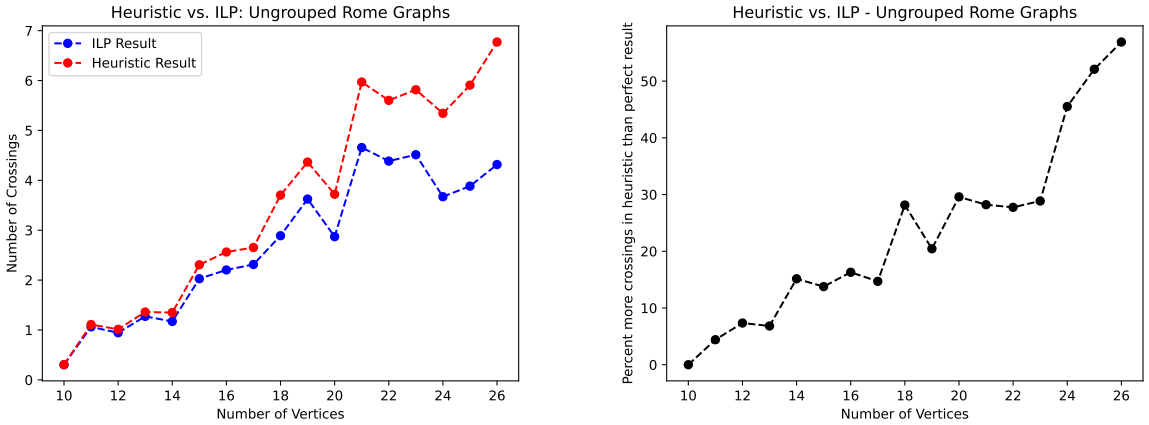


Fig. 6.2: A total of 2000 Rome Graphs with the size of 10 to 26 vertices were plotted.

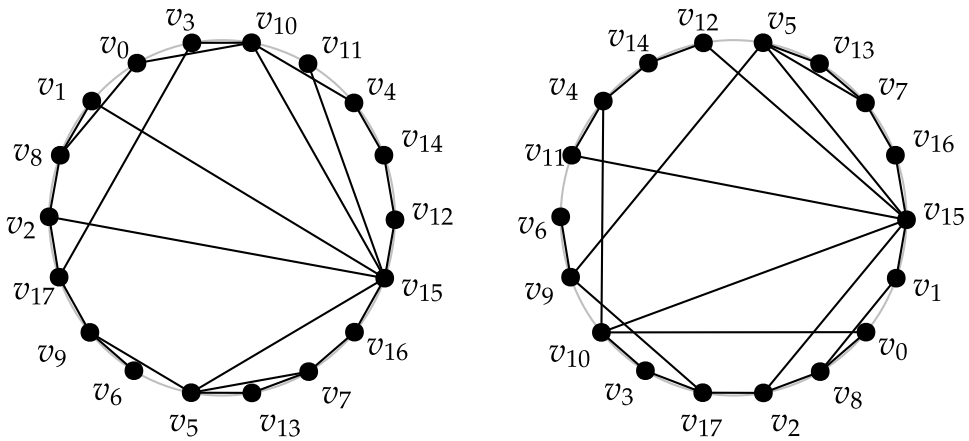


Fig. 6.3: The ILP (left) produced 5 crossing while the heuristic (right) produced 9 crossings.

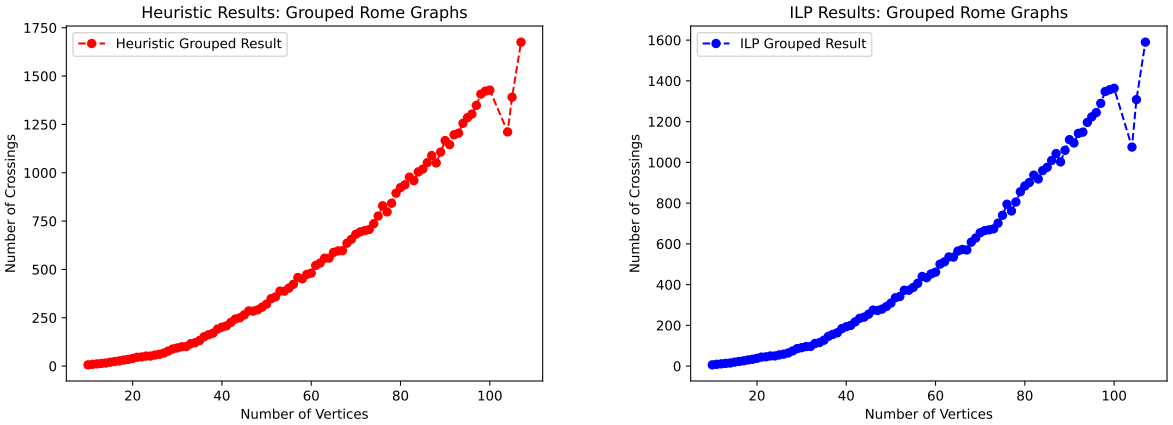


Fig. 6.4: Results of the heuristic (left), results of ILP (right)

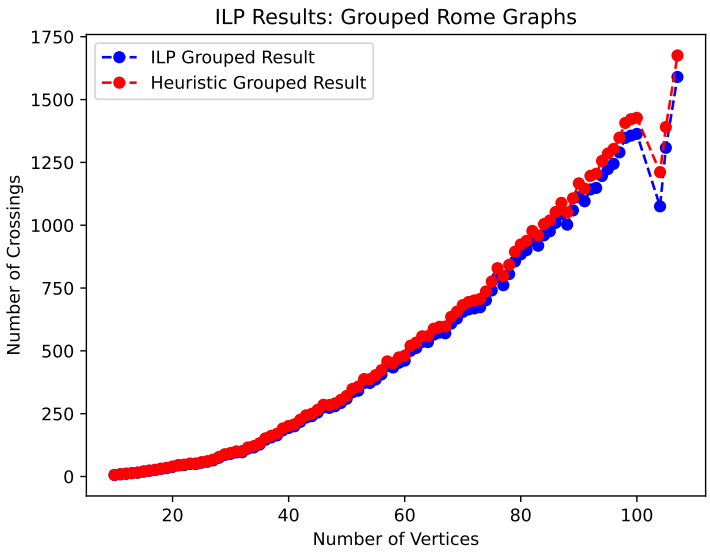


Fig. 6.5: Result of heuristic vs. ILP - A total of 11534 Rome Graphs with the size of 10 to 107 vertices were plotted.

In average, 4.39% more crossings were produced by the heuristic among all of the tested graphs. This percentage of difference had an almost consistent behavior across all sizes of graphs and was in the 2 to 6 percent range. See Figure 6.6. For the Rome Graphs with less than or equal to 26 vertices, 3.83% more crossings were produced by the heuristic, which is about 20% less crossings in average than ungrouped case and 16.5% better performance. The grouped graphs with 40 to 107 vertices have a number of vertices per group that is close to the number of vertices in our ungrouped testing and there are therefore a comparable number of orderings possible in both. The performance however, is significantly better in the grouped testing.

Given the charts in Figures 6.5 and 6.6, we can suspect that the performance of the heuristic improves when there are more crossings in an optimal solution. To evaluate this suggestion we plotted the percentage of the accuracy of the heuristic dependant on the number of crossings per vertex in Figure 6.7. This plot supports the hypothesis. This observation explains why the ungrouped evaluation does a worse performance than the grouped evaluation because there are fewer crossings in a layout without groupings.

The tendency however, was again that the heuristic had a harder time finding the optimal solution with growing number of vertices. This is visualized in Figure 6.6.

The running time of the heuristic is better than the ILP by a great margin. See Figure 6.8 for an experimental comparison.

The following Figure 6.9 is the result of the heuristic against that of the ILP on a graph with twelve vertices to give a feeling of the performance. In this case, an optimal solution has been produced by the heuristic.

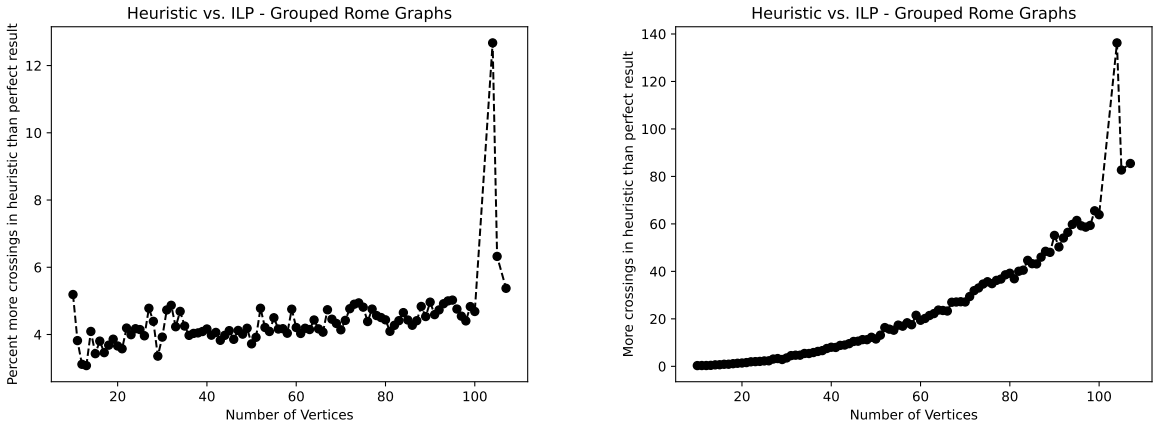


Fig. 6.6: Heuristic vs. ILP: Performance in percentage of crossings (left), performance by crossing number (right)

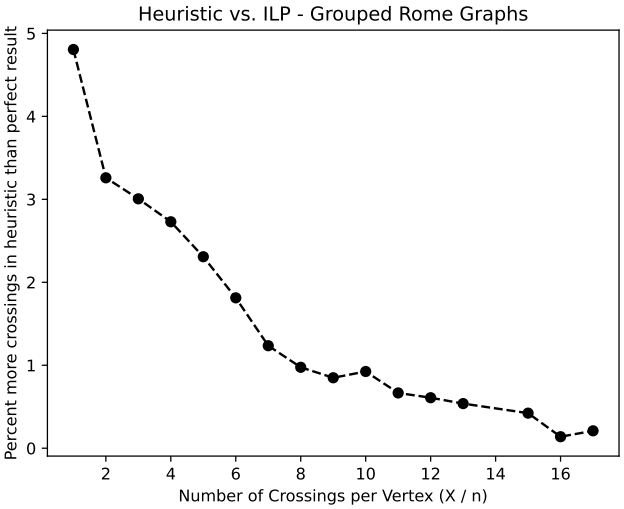


Fig. 6.7: Result of heuristic vs. ILP - The same data was used as in Figures 6.5 and 6.6. Each data point is the average of all data in a period of 1 unit

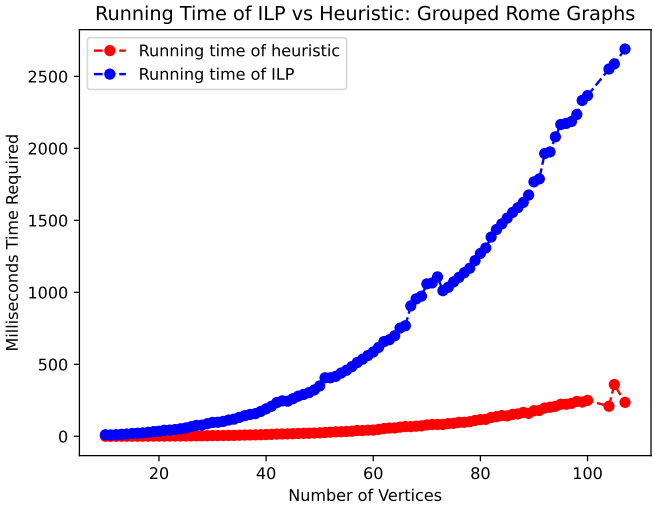


Fig. 6.8: Average time per each graph size needed by the ILP and the heuristic

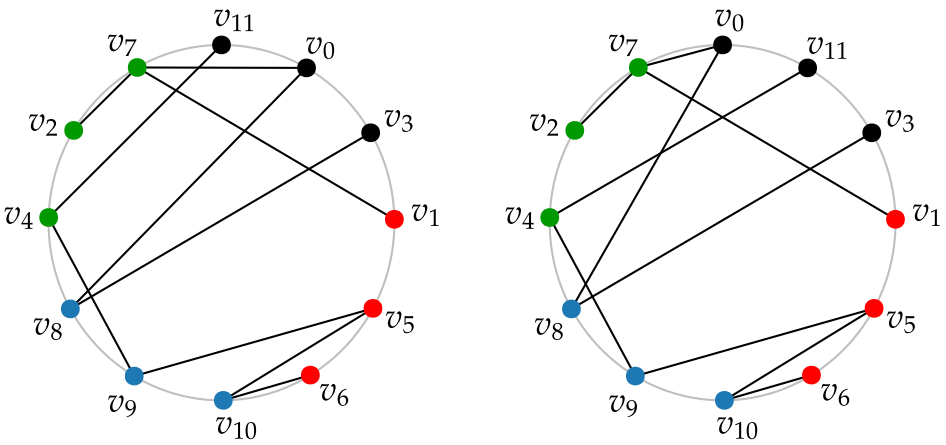


Fig. 6.9: The ILP (left) produced 6 crossings and the heuristic (right) produced an almost equivalent optimal solution.

7 Conclusion And Future Work

In conclusion, we presented an ILP-formulation for the circular layout problem with and without grouping constraints where the number of crossings in the layout is to be minimized. We reviewed the heuristic of Brandes and Baur [BB05] for the case of not having groupings and modified this heuristic to support grouping constraints. Our modifications did not exceed the original time complexity of $O((n+m)\log n)$ for the initialization phase and $O(nm \cdot w)$ for the optimization phase. We then tested the performance of these heuristics against the ILP in a series of experimental evaluations and saw that the performance of the heuristic is improved when supporting groupings on the graphs of same size. We then explained this observation using the number of crossings in an optimal solution. The chart in Figure 6.7 showed us that the performance of the heuristic improves when there are more crossings in an optimal solution.

With this, we have presented a heuristic for the problem of crossing reduction in circular layouts under grouping constraints that performs quite well and in most tested cases is on pair with the optimal solution. Our work can however be further examined. First and for most, one can examine the performance of the heuristic while taking the best result out of n runs for a graph with n vertices. The best result should be taken out of n runs because each vertex is then expected to be taken as the first vertex of the initialization once. Other than that, we have also noticed that the initialization process often creates a crossing-free drawing when the input graph is outerplanar. One could examine certain graph classes that will always result in a crossing-free circular drawing by the initialization. J. Six has presented an initialization algorithm that always makes a crossing-free drawing for outerplanar graphs [ST99]. One could modify this algorithm to support grouping constraints to use it as the initialization process and combine this with the grouped circular sifting of Section 5.2, and then evaluate to see if the performance improves.

Our work provides a fast method for creating circular drawings with or without groups with a performance fairly close to that of an exact solution. This helps network administrators to draw their networks more efficiently and enables them to focus more on the network rather than spending time to read the network graph.

Bibliography

[BB05] Michael Baur and Ulrik Brandes: Crossing reduction in circular layouts. In Juraj Hromkovič, Manfred Nagl, and Bernhard Westfechtel (editors): *Graph-Theoretic Concepts in Computer Science*, pages 332–343. Springer Berlin Heidelberg, 2005, 10.1007/978-3-540-30559-$0_2$8.

[Cpl09] IBM ILOG Cplex: V12. 1: User's manual for cplex, 2009.

[DMM96] Ugur Dogrusoz, Brendan Madden, and Patrick Madden: Circular layout in the graph layout toolkit. pages 92–100, 1996, 10.1007/3-540-62495-$3_4$0.

[GG11] Krasimira Genova and Vassil Guliashki: Linear integer programming methods and approaches–a survey. *Cybernetics and Information Technologies*, 11, 2011.

[Gur22] Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual, 2022. `https://www.gurobi.com`.

[Mit79] Sandra L. Mitchell: Linear algorithms to recognize outerplanar and maximal outerplanar graphs. *Information Processing Letters*, 9(5):229–232, 1979, 10.1016/0020-0190(79)90075-9, ISSN 0020-0190.

[MKNF87] Sumio Masuda, Toshinobu Kashiwabara, Kazuo Nakajima, and Toshio Fujisawa: On the np-completeness of a computer network layout problem. *Proceedings: IEEE International Symposium on Circuits and Systems*, 1987.

[Mä88] Erkki Mäkinen: On circular layouts. *International Journal of Computer Mathematics*, 24(1):29–37, 1988, 10.1080/00207168808803629.

[SSSV95] Farhad Shahrokhi, Ondrej Sýkora, László A. Székely, and Imrich Vrt'o: Book embeddings and crossing numbers. In Ernst W. Mayr, Gunther Schmidt, and Gottfried Tinhofer (editors): *Graph-Theoretic Concepts in Computer Science*. Springer Berlin Heidelberg, 1995, 10.1007/3-540-59071-$4_5$3.

[ST99] Janet M. Six and Ioannis G. Tollis: *Circular Drawings of Biconnected Graphs*, pages 57–73. Springer Berlin Heidelberg, 1999, 10.5555/646678.702166.

[ST06] Janet M. Six and Ioannis G. Tollis: A framework and algorithms for circular drawings of graphs. *Journal of Discrete Algorithms*, 4(1):25–50, 2006, 10.1016/j.jda.2005.01.009, ISSN 1570-8667.

[WZ21] Alexander Wolff and Johannes Zink: ipraline: Interactive problem analysis and solving in complex industrial networks, 2021. `go.uniwue.de/ipraline`.