Minimizing length in linked bar charts

Bjorn Fidder ✉
Department of Informatics and Computing Science, Utrecht University, the Netherlands

Abstract

In this report, we consider linked bar charts, where each bar is a stack of blocks, and linked with blocks in other bars by orthogonal lines. We define heuristics for minimizing the summed horizontal and vertical length of these links, first optimizing the bar order for horizontal link length, then given this fixed bar order, stacking the bars in an order that attempts to optimize for vertical link lengths. The performance of these algorithms is experimentally tested and compared on randomly generated instances, as well as instances constructed from real network data.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Theory and algorithms for application domains; Theory of computation → Graph algorithms analysis; Theory of computation → Random search heuristics

Keywords and phrases Computational Geometry, Heuristic algorithms, Visualisation

1 Introduction

A common topic in information visualisation is to represent data that has been grouped into sets. Visualising these sets becomes more intricate once there are relations between the sets. Assume that we have a collection of N sets $S_1, \ldots, S_N$. Then, a possible relation might be that we know that some subset of the data points is either present in set S_1, or in set S_2. This induces a *pairwise uncertainty* between sets S_1 and S_2. One might think of a group of floating voters that can not make up their mind between two political parties.

In [9], a method is introduced to visualise these pairwise uncertainties among sets, in the form of *linked bar charts*. The pairwise uncertainty is presented by a block that appears with equal size in two bars, and a link connecting these blocks between the bars. An example is shown in Figure 1, where only the uncertain parts of each set are shown. The readability or visual appeal of such a chart can be quantified in various ways, including measuring the horizontal and vertical link lengths, but also the link spans, the cut-width or the number of link crossings [10].

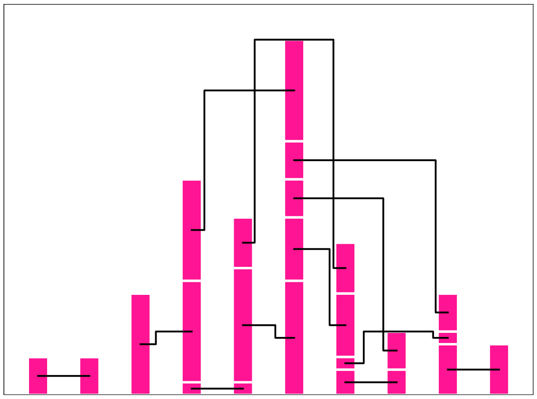


Figure 1 An example of a linked bar chart. Each block occurs in two bars with equal size, and there is an orthogonal line connecting the blocks.

Related work. Optimizing for horizontal link length is known to be NP-complete in general [6], but for the particular case that the graph representing the bars and their links is outerplanar, polynomial-time algorithms exist [5]. The general case is equivalent to the *minimum linear arrangement problem*, which is experimentally studied in [8]. In [10], the problem of minimizing vertical length given a fixed bar order is studied, providing polynomial algorithms for some restricted cases concerning the types of links.

Contributions and organization. We study the minimization of the summed horizontal and vertical link length by reordering bars and their blocks. Several heuristic algorithms to minimize the link lengths are defined, and experimentally compared on both randomly generated instances and instances based on real network data. Section 2 defines the problem in a graph-theoretical setting, while the methodology of our experiments are defined in Section 3. The problem is attacked in a two-step approach: first optimizing the bar order for horizontal link length, then the block order for vertical link length. The algorithms and results of these two steps are given in Sections 4 and 5 respectively, with further discussion in Section 6.

2 Notation and observations

As in [10], we will model the input of our problem as an edge-weighted graph $G = (V, E, w)$. In the context of set visualisation, the N vertices in V correspond to the collection of sets $S_1, \ldots, S_N$, while the edges in E model the pairwise uncertainties between the sets, with the weight $w : E \to \mathbb{R}^+$ indicating the size of the uncertainty. These uncertainties are visualised in the form of a linked bar chart. For an example of such a linked bar chart, we refer back to Figure 1.

Bars. To each vertex $v \in V$ we associate a bar, containing a block of height $w(e)$ for each edge $e \in E$ of weight $w(e)$ that is incident to v. The height of a vertex (or its associated bar) $h : V \to \mathbb{R}^+$ is defined as the sum of the weights of the edges incident to it. The position of the bars is given by the *bar order*, which can be modelled as a permutation $\phi : \{1, \ldots, N\} \to \{1, \ldots, N\}$.

Blocks and links. For each edge e, from vertex v to w, there is a *link* (v, w) that connects the blocks in the bars corresponding to vertices v and w. In the notation (v, w) for a link, we will assume that $\phi(v) < \phi(w)$. This link is drawn as an *orthogonal line*, that is, a line consisting of vertical and horizontal line segments. Given a *block order*, the link will connect the center point of the blocks it connects. Links are not allowed to pass through other bars. Therefore, if there is an intermediate bar, with position between $\phi(v)$ and $\phi(w)$, that has height larger than the centers of the start and end blocks, the link will have to cross over this bar. While in actual visualisation applications, one might want to add some padding here, we will for simplicity let links pass over exactly at the height of such an intermediate bar. Note that this means that if multiple links have to pass over the same bar, they cannot be properly distinguished.

The horizontal link lengths are determined by the bar order. Once the bar order is fixed, the vertical link lengths are determined by the block order.

Link crossings. Given a bar order, it is possible that two links have to cross. Note that links (v_1, w_1) and (v_2, w_2) cross if $\phi(v_1) < \phi(v_2) < \phi(w_1) < \phi(w_2)$, or $\phi(v_2) < \phi(v_1) < \phi(w_2) <$

$\phi(w_1)$. Since we will optimize the bar order for horizontal link lengths, we do not consider these link crossings directly. However, we do expect that the number of crossings is smaller if the horizontal link lengths are smaller.

For the block order, we explicitly require that it may not induce further crossings of links. For a vertex $v \in V$, we can split the links into two sets L_v and R_v. The set L_v consists of all links (w, v), while, the set R_v consist of all links (v, w). To prevent further crossings, we have to require that the blocks corresponding to the links $(w, v) \in L_v$ are placed in order of decreasing $\phi(w)$, while the blocks corresponding to the links $(v, w) \in R_v$ are placed in order of increasing $\phi(v)$. Therefore, the only degree of freedom in the block order that remains, is the order in which the sets L_v and R_v are interleaved.

3 Methodology

3.1 Generated graph instances

We use several graph instances for our experiments, which we list below.

Random graphs. We consider the Erdös-Renyi random graph $G(N, p)$ consisting of N vertices, where each pair of vertices is connected by an edge independently with probability p. Additionally, the edges are weighted uniformly in the range $\{1, \ldots, 10\}$.

Random geometric graphs. To induce some local structure in the graph instances, we also test the algorithms on the *Random Geometric Graph* (RGG), which places N vertices on the $[0, 1]^2$ square, and then connects two vertices if their Euclidean distance is less than some *neighborhood radius* r. We will use the notation $RGG(N, r)$ for these graphs. As for the random graphs, edges are weighted uniformly in the range $\{1, \ldots, 10\}$.

3.2 Real network instances

For more diversity in the types of connections, we have also tested algorithms on real network data. These networks were mainly chosen based on diversity and availability, and do not represent networks that are the object of applications, i.e. do not represent pairwise uncertainties between sets. The network sets were selected through the Colorado Index of Complex Networks [1]. We define the edge density p, in accordance with connection probability for random graphs, by $\frac{|E|}{\binom{N}{2}}$ where $|E|$ is the number of edges and $\binom{N}{2}$ is the binomial coefficient equal to $\frac{1}{2}N(N-1)$.

Teenage friends This network consist of snapshots of friendships among $N = 50$ girls from the Teenage Friends and Lifestile Study [7], across 3 consecutive years. Edges indicate friendships among students and are originally unweighted. They are additionally weighted for this report uniformly in $\{1, \ldots 10\}$.

Roll-call votes This is a dataset of 220 networks ranging from 66 to 440 nodes based on yes/no answers by legislators from [11]. The networks are complete graphs, but edges are weighted according to the proportion of votes on which two legislators agreed. These networks are used for our experiments by defining a threshold weight θ, and including the edges with weight at least θ. This threshold is then selected to obtain a desired edge density p. The edges are then reweighted linearly from the range $[\theta, 10]$ to the range $[1, 10]$ and then rounded. Only the first network of 66 nodes is used for experiments, the other networks being so large that computation time was infeasible for this report.

Movie actors Social graphs for over 700 movies from [3], generated automatically from movie scripts. Each node represents a character in a movie and each edge is a same-scene appearance between two characters in that movie. The weight gives the number of same-scene appearances. Similar as for the roll-call votes, edges are reweighted by mapping linearly from the range $\{1, \ldots, w_max\}$ to the range $\{0, 1, \ldots, 10\}$ with rounding. The dataset consists of networks of a wide range of node numbers, averaging at 35 nodes. For our experiments, we select a limited number of networks in a desired range of $N = 40$ to $N = 50$ nodes.

3.3 Implementation

The algorithms defined in what follows, are implemented in Python, version 3.10.6. The source code is available at [4]. In general, we will test an algorithm on 20 independently generated graph instances, and use graphs of size $N = 50$, which was chosen to be sufficiently large that solving the problem is not trivial, yet not too large to make computation infeasible. Note also that for graphs larger than 50, the resulting linked bar chart would become quite extensive and impractical. For both bar order and block order, we will in the following sections also define baseline algorithms, with which we can compare the performance of the heuristics.

4 Bar order

4.1 Algorithms

In this section, we will define several heuristics for finding a bar order that minimizes the horizontal link length. They are listed in expected order of increasing computation time, and increasing performance.

Baseline. As a baseline algorithm, we simply do not permute the bars at all, i.e. the bar order is given by the identity permutation. Hence, it is a completely random order of the bars.

Greedy incremental. This algorithm belongs to the family of Succesive Augmentation heuristics, see [8]. It works by incrementally expanding the layout of the bars in a greedy fashion. The first vertex is placed at position 1. Then, each subsequent vertex is placed either completely left, or completely right of all placed vertices, determined by which placement minimizes the horizontal link length. Afterwards, the positions are shifted such that they take values in $\{1, \ldots, N\}$.

Adjacent 2-OPT. This is a local search approach, which we have named similar to a common local search heuristic used for the travelling salesman problem [2]. Starting from an initial bar order, which may simply be the baseline ordering, it checks all pairs of *adjacent* bars, i.e. two vertices v, w with $|\phi(v) - \phi(w)| = 1$, for whether swapping their positions decreases the total horizontal link length. If so, the positions are swapped. This is continued until there are no adjacent swaps that lead to further improvement, checking each time all adjacent vertices in fixed order. Note that the decrease in horizontal link length induced by

Table 1 The computation times in seconds (s) of the heuristic algorithms averaged over 20 random graph $G(50, 0.01)$ instances.

	Computation time (s)
Baseline	0.0009 ± 0.0009
Greedy incremental	0.003 ± 0.005
Adjacent 2-OPT	0.9 ± 0.4
Complete 2-OPT	2.4 ± 1.0

a swap is computed based only on the relevant links, i.e. the links adjacent to vertices v and w.

Complete 2-OPT. Similar to Adjacent 2-OPT, but now all pairs of bars are checked, no matter their positions, until no further improvement is possible by swapping the positions of two bars. Note that the local search 2-OPT heuristics will not necessarily find the optimal solution, since it is possible to get stuck in a local optimum.

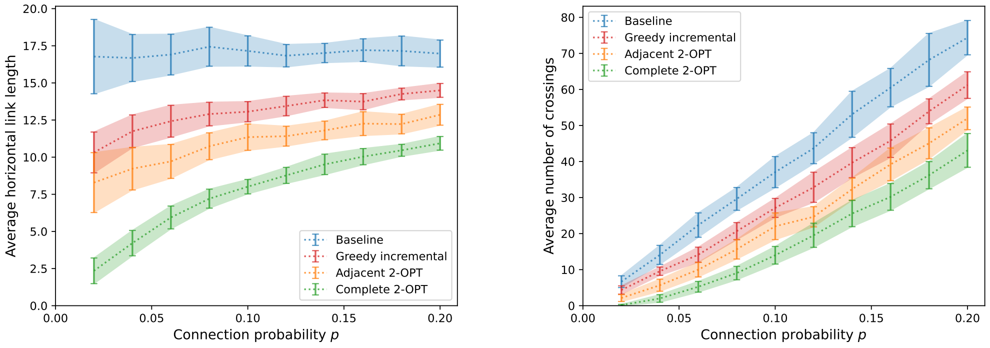
4.2 Results

In this section, we will test and compare the algorithms for bar order defined above.

Random graphs. First of all, we will consider random graph instances $G(N, p)$ with $N = 50$. Figure 2a shows that, in accordance with what was expected, the greedy incremental, adjacent 2-OPT and complete 2-OPT algorithms increasingly improve upon the baseline. Note that the baseline link lengths measure on average approximately 17, which is in accordance with the expected average distance between random links of $16\frac{2}{3}$. The improvement over the baseline is especially significant for lower connection probability p, i.e. a lower number of edges, where one would indeed expect there to still be efficient orderings of the bars. For larger p, there are so many edges, that it gets difficult to improve over a random order. Meanwhile, in Figure 2b we confirm that although we do not directly minimize the number of crossings, minimizing the horizontal link length does lead to less crossings.

In Table 1 we see that the algorithms were indeed listed in increasing order of runtimes, where the baseline and greedy incremental have neglibile computation times compared to the local search heuristics. Note that the algorithms were implemented in Python, and without focus on efficiency, so these computation times are only suitable for superficial comparison. Perhaps surprisingly, although the complete 2-OPT heuristic considers a quadratic number of vertex pairs, compared to a linear number for the adjacent 2-OPT heuristic, the computation time is less than 3 times as long. This suggest that the numbe of iterations to reach a local optimum using Complete 2-OPT is not much larger than the number needed for Adjacent 2-OPT. From testing, it was clear that run-times increased with both the number of nodes N and the connection probability p.

Random geometric graphs. Figure 3 shows that for random geometric graphs perform increasingly well in the same order. Variances are notably high for small neighborhood radius, however, the random geometric graph has less edges for r at the same value of p in the random graph in the used parameter range. Notably, the random geometric graph allows for smaller horizontal link lengths, which can be explained by the local structure present in the random geometric graph instances.



(a) **(b)**

Figure 2 The horizontal link length in **(a)** and number of crossings in **(b)**, averaged over all links, when applying heuristic algorithms on random graph instances $G(N, p)$ with $N = 50$ and varying connection probability p. The mean and standard deviation are shown for each value of p, based on 20 generated instances.

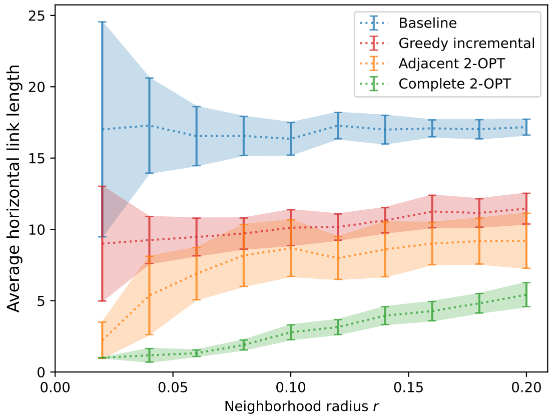


Figure 3 The horizontal link length averaged over all links, as in Figure 2a, but now applied to random geometric graph instances $RGG(N, r)$ with $N = 50$ and varying neighborhood radius r.

5 Block order

5.1 Algorithms

In this section, we will define several heuristics for finding a block order that minimizes the vertical link length. As observed in Section 2, the restriction on link crossing means that the order of the blocks with links going left (L_v), and the blocks with links going right (R_v) are fixed, and only the order in which these blocks are interleaved can be changed.

Baseline The baseline algorithm interleaves for each bar v the blocks from L_v and R_v randomly, in their respective orders, proportional to the number of remaining elements in L_v and R_v. That is, a block from L_v is selected with probability $\frac{|L_v|}{|L_v|+|R_v|}$, and is popped from the front of L_v, otherwise a block from R_v is placed and the link popped from R_v.

While one could also select from L_v and R_v with probability $\frac{1}{2}$, taking into account the set sizes as above leads to a more even distribution of left and right links in the case of disbalance in the set sizes.

2-OPT Since for each vertex v, the order of blocks from L_v are fixed, and so are the blocks from R_v, it is only possible to swap two adjacent blocks, of which one is from L_v, and one is from R_v. The 2-OPT algorithm checks for each vertex whether there are two such adjacent blocks whose swap decreases the total vertical link length. This effect is computed based on only the vertical link lengths of the two links involved. If the total link length is indeed lowered, the blocks are swapped and all vertices are checked again for possible swaps, in fixed order of the vertices. This is repeated until there are no beneficial swaps remaining.

Iterative DP The optimal order of the blocks in one bar can be solved to optimality using dynamic programming, if the blocks in all other bars are fixed. Indeed, one can construct a table $T(L, R)$ storing the optimal vertical link length when stacking the first L blocks from L_v and the first R blocks from R_v. Let $L_v = \{l_1, l_2, \dots, l_{|L_v|}\}$ and $R_v = \{r_1, r_2, \dots, r_{|R_v|}\}$ in their respective predefined orders, and write $VLL(l)$ for the vertical link length of link l if the block is placed on top of the current stack of blocks. Then, the table can be recursively expanded according to

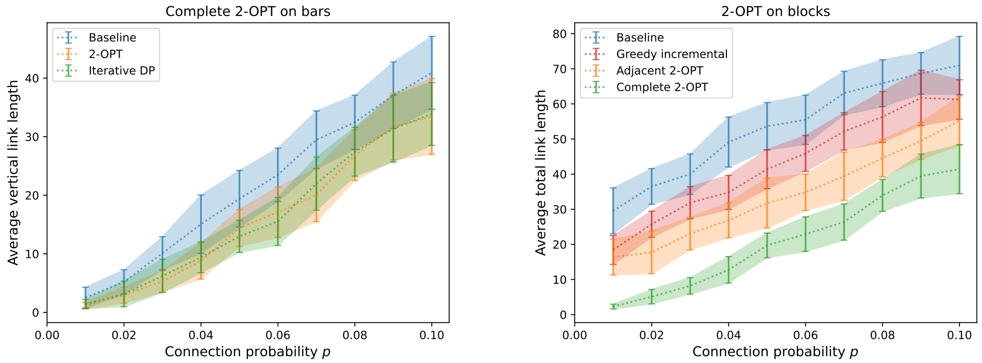
$$T(L+1, R+1) = \min\{T(L, R+1) + VLL(l_{L+1}),\, T(L+1, R) + VLL(r_{R+1})\},$$

where the base cases $T(L, 0)$ and $T(0, R)$ are defined separately. Then, the optimal vertical link length can be found from $T(|L_v|, |R_v|)$. To obtain the actual block order, an additional table $T'(L, R)$ is used and concurrently obtained that stores the blocks in the optimal order of insertion.

The iterative DP algorithm picks in each iteration a vertex v uniformly at random and optimally arranges its blocks using this dynamic programming approach, the blocks in the other bars remaining fixed. This is repeated for a number of $5|V|$ iterations, which was experimentally confirmed to be approximately be the number of iterations required to reach an equilibrium.

5.2 Results

Random graphs. Figure 4a shows that the 2-OPT and Iterative DP algorithms both lead to lower vertical link lengths than the random, baseline approach. However, there is not a significant difference seen in performance between 2-OPT and Iterative DP. From Figure 4b



(a) **(b)**

Figure 4 In **(a)**, the vertical link length averaged over all links, when applying first the Complete 2-OPT algorithm on the bar order, then one of the heuristic block algorithms, on random graph instances $G(N, p)$ with $N = 50$ and varying connection probability p. The mean and standard deviation are shown for each value of p, based on 20 generated instances. In **(b)**, the bar algorithms are varied, while the block algorithm is fixed to 2-OPT, and the resulting average total link length is shown.

Table 2 Computation time in seconds of each combination of heuristic algorithms, averaged over 20 random graph $G(50, 0.01)$ instances.

Block algorithm	Bar algorithm			
	Baseline	Greedy incremental	Adjacent 2-OPT	Complete 2-OPT
Baseline	0.01 ± 0.01	0.01 ± 0.01	0.55 ± 0.30	1.42 ± 0.64
2-OPT	0.16 ± 0.18	0.17 ± 0.18	0.84 ± 0.56	1.58 ± 0.90
Iterative DP	0.19 ± 0.10	0.21 ± 0.10	0.81 ± 0.42	1.65 ± 0.63

it is clear that using a bar algorithm that led to a lower horizontal link length, also leads to a lower total link length. In fact, experiments showed that also the vertical link lengths after the 2-OPT block algorithm lower in order of better bar performance. As can be seen in Figure 5, in the chosen horizontal and vertical units, the vertical link length is the main contribution to the total link length, especially as the number of links increases. In Figure 6, we can see that these algorithms indeed lead to less cluttered linked bar charts, for a still manageable number of $N = 20$ bars. Note that the algorithms are tested for $N = 50$ bars, for which the linked bar charts get rather massive.

Table 2 shows that the run-time generally increases with performance of the algorithms. Note that the Iterative DP algorithm tends to run slightly longer than the 2-OPT algorithm, while performance was seen to be similar.

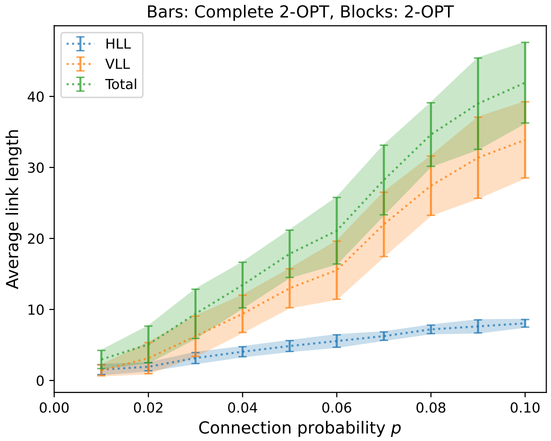
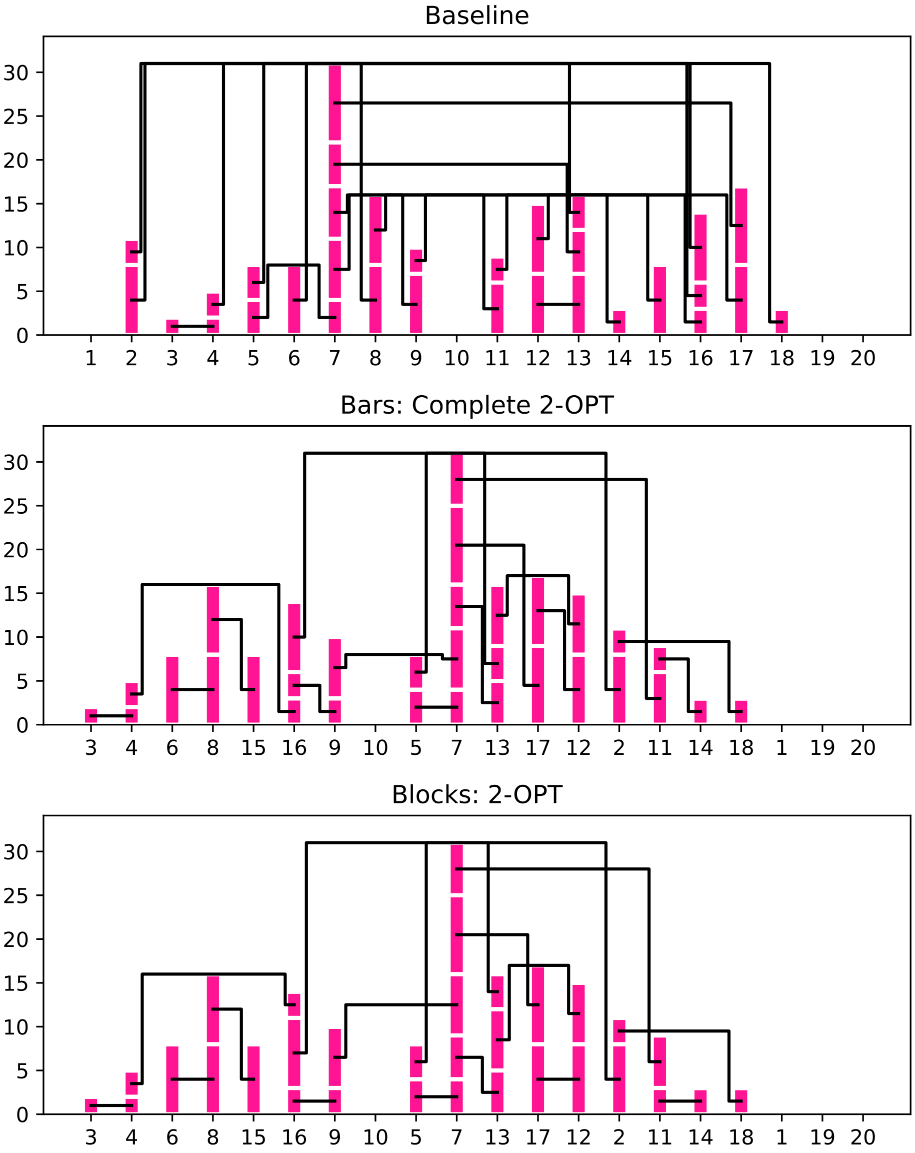


Figure 5 The horizontal link length (HLL), vertical link length (VLL) and total link length averaged ove all links, when applying Complete-OPT on bars, then 2-OPT on blocks, on random graph instances $G(N, p)$ with $N = 50$ and varying connection probability p. The mean and standard deviation are shown for each value of p, based on 20 generated instances.

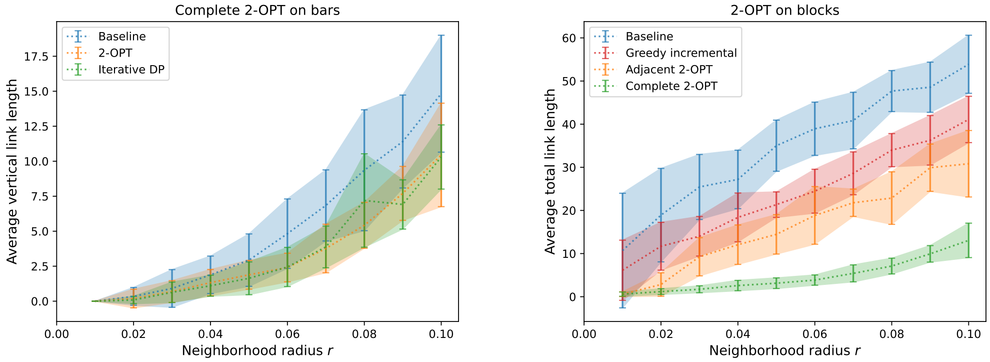

Baseline

Bars: Complete 2-OPT

Blocks: 2-OPT

Figure 6 The linked bar chart shown in three steps: first when for both bar and block order the baseline algorithm is applied, with the 20 bars labeled in order. Then, Complete 2-OPT is used for the bar order, and blocks again using the baseline algorithm, which leads to the second chart. Finally, in the last chart, 2-OPT is used for the block order, with the previous chart as the starting point. The total link length is in this way decreased from 399 to 179. The network instance was a random graph $G(N, p)$ of $N = 20$ nodes and connection probability $p = 0.1$. Note that for such instances, the average total link length using the baseline algorithm is 548 ± 205, hence in this case the baseline performed relatively well.

Random geometric graphs. In Figure 7 we can see that also for random geometric graphs, combining the Complete 2-OPT with the 2-OPT or Iterative DP algorithm works best. Note that the total link length using the Complete 2-OPT algorithm is significantly lower than using the other bar algorithms, with a more significant improvement than for random graphs.



(a) **(b)**

Figure 7 In **(a)**, the vertical link length averaged over all links, when applying first the Complete 2-OPT algorithm on the bar order, then one of the heuristic block algorithms, on random geometric graph instances $RGG(N, r)$ with $N = 50$ and varying neighborhood radius r. The mean and standard deviation are shown for each value of r, based on 20 generated instances. In **(b)**, the bar algorithms are varied, while the block algorithm is fixed to 2-OPT, and the resulting average total link length is shown.

Real networks Figure 8 suggests that applying the algorithms to instances based on real network data leads to similar relative performance. Note that the total link length is in the same order as it was for the random graphs. Meanwhile, Figure 8b shows no clear relation between the edge density of the network and the resulting link lengths acquired by the algorithms, in contrast to the randomly generated networks. This might be explained by the degree distribution that can vary among the networks, and can be vastly different from the degree distribution seen in random graphs.

Figure 9 shows that for the roll call vote network, the algorithms do not acquire significant improvement in the total link length over the baseline algorithm. This can be explained by the very skewed degree distribution in the networks: for $p = 0.1$, all but 4 nodes have very small degree (at most 4), and the other nodes have degrees 35, 58, 61, and 65.

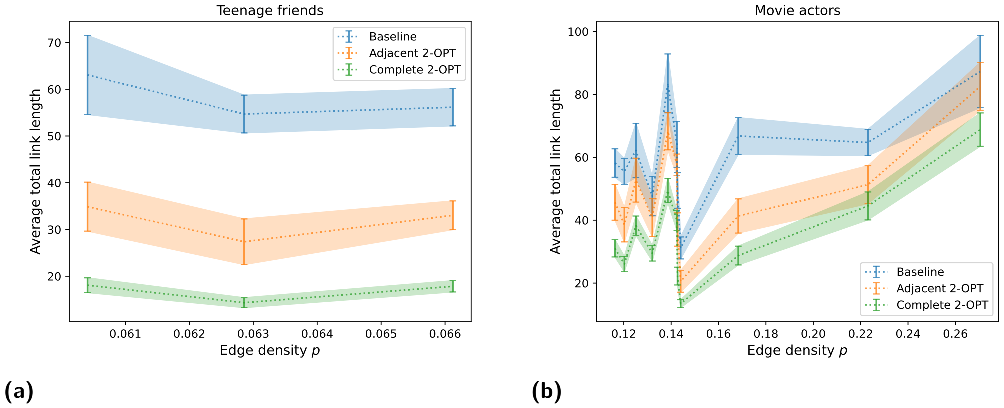


Figure 8 The total link length averaged over all links, when applying first one of the bar order algorithms, and then the 2-OPT algorithm for block order, in **(a)** on instances generated from friendships among 50 teenagers in thee consecutive years, and in **(b)** on instances generated from same-scene occurences in movies with 40 to 50 actors, with respect to the edge density p in these networks. The mean and standard deviation are shown over 20 repetitions using the same network, but a randomly permuted initial permutation of the bars.

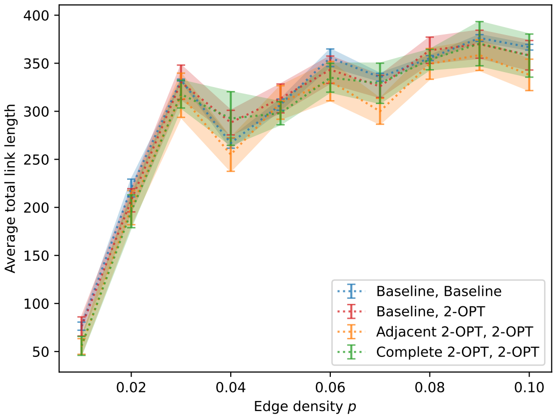


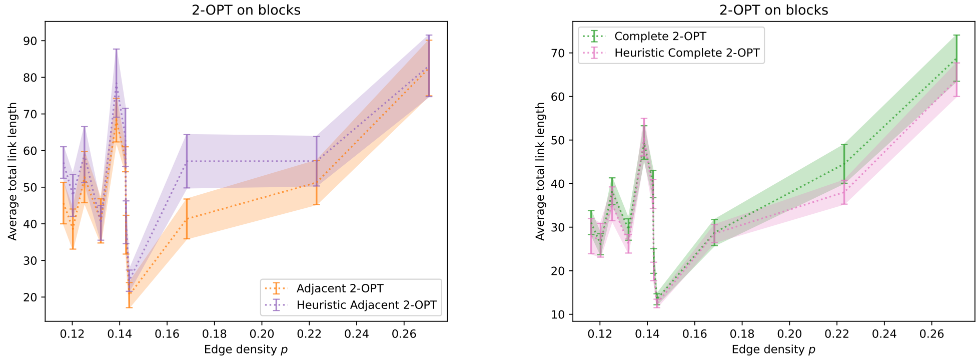
Figure 9 The total link length averaged over all links, when applying first one of the bar order algorithms, and then the 2-OPT algorithm for block order, applied on instances based on roll-call votes of 66 legislators, the edge density forced to a particular value by setting a threshold on the agreement between legislators. The mean and standard deviation are shown over 20 repetitions using the same network, but a randomly permuted initial permutation of the bars.

6 Discussion

As can be seen in Figure 6, the Complete 2-OPT algorithm can leave empty bars splitting the bars into two sets, unnecessarily leading to larger link lengths between these two sets, and would be better placed on the side. This is a limitation of only considering swaps of two bars, and not considering moving more than two bars at a time.

Note furthermore that in the two-step approach of first optimizing the bar order for horizontal link length, and only then considering block heights in the block order, the height of bars and the consequent vertical link lengths is not taken into account in the bar order. Still, this approach was shown to have good performance. The question remains if this performance could be improved upon by taking into account vertical link length already in the bar order. Some experiments were performed to answer this question. First of all, a 2-OPT approach was attempted for the bar order, applying before and after the swap the baseline block algorithm to measure vertical link length. However, in some first experiments on random graphs, this approach did not seem to improve performance.

An alternative approach was to estimate the vertical link lengths by for each bar counting the number of links that cross over it, and multiplying this by the fixed height of the bar. This led to two additional bar order algorithms, the *Heuristic Adjacent 2-OPT* and the *Heuristic Complete 2-OPT* algorithm. For the first, the impact of swapping two adjacent bars can be still quite simply computed by considering the neighbors of both bars to recompute the number of links going over the bars. For the Heuristic Complete 2-OPT algorithm however, updating these counts is not trivial. To deal with this, the impact of swapping two arbitrary bars u, v was computed by taking a sequence of adjacent bar swaps that are together equivalent to swapping u, v. These Heuristic algorithms were also superficially tested on random instances, but did not seem to lead to improved performance. However, on some of the real network instances, there was some improvement in performance, as can be seen in Figure 10.



(a) **(b)**

Figure 10 The total link length averaged over all links, when applying first one of the (heuristic) 2-OPT algorithms for bar order, and then the 2-OPT algorithm for block order, on instances generated from same-scene occurences in movies with 40 to 50 actors, with respect to the edge density p in these networks. The mean and standard deviation are shown over 20 repetitions using the same network, but a randomly permuted initial permutation of the bars.

7 Conclusion

Of the algorithms considered, using the Complete 2-OPT for bar order, and the 2-OPT algorithm for block order is most effective, not only leading to best performance in randomly generated instances, but also in instances generated from real social networks. If the network has very skewed degree distribution, the algorithms considered did not improve significantly over the baseline in the total link lengths. For real networks, taking into account vertical link lengths already for bar order seemed beneficial, but more experiments are required to confirm this. This could be a possible alleviation in case the degree distribution is very skewed.

Relevant future work further includes considering other algorithms for the bar order, for example those additionally considered in [8], considering unified approaches that simultaneously optimize bar and block order, and considering other quality measures than link length. In particular, one could pay more attention to other aspects that lead to visually appealing charts, for example also minimizing the number of links that pass over one bar, or letting links pass over a bar at different heights.

References

1 Ellen Tucker Aaron Clauset and Matthias Sainz. The colorado index of complex networks, 2016. URL: https://icon.colorado.edu.

2 G. A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812, 1958. URL: http://www.jstor.org/stable/167074.

3 Harvard Dataverse. Moviegalaxies - social networks in movies, 2018. doi:10.7910/DVN/T4HBA3.

4 Bjorn Fidder. Experiments on linked bar charts, 2025. URL: https://github.com/BjornFidder/LinkedBarCharts.

5 G.N. Frederickson and S.E. Hambrusch. Planar linear arrangements of outerplanar graphs. *IEEE Transactions on Circuits and Systems*, 35(3):323–333, 1988. doi:10.1109/31.1745.

6 Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1990.

7 Lynn Michell and Amanda Amos. Girls, pecking order and smoking. *Social Science & Medicine*, 44(12):1861–1869, 1997. URL: https://www.sciencedirect.com/science/article/pii/S027795369600295X, doi:10.1016/S0277-9536(96)00295-X.

8 Jordi Petit. Experiments on the minimum linear arrangement problem. *ACM J. Exp. Algorithmics*, 8, December 2004. URL: https://doi-org.utrechtuniversity.idm.oclc.org/10.1145/996546.996554, doi:10.1145/996546.996554.

9 Nathan van Beusekom, Steven Chaplick, Amy Griffin, Marc van Kreveld, Martin Krzywinski, Wouter Meulemans, and Hsiang-Yun Wu. Visualizing Set Sizes with Dependent and Independent Uncertainties. In *IEEE PacificVis 2024*, 2024.

10 Steven van den Broek, Marc van Kreveld, Wouter Meulemans, and Arjen Simons. Minimizing Vertical Length in Linked Bar Charts. In *EuroCG 2025 Proceedings*, pages 343 – 349, 2025.

11 Andrew Scott Waugh, Liuyi Pei, James H. Fowler, Peter J. Mucha, and Mason A. Porter. Party polarization in congress: A network science approach, 2011. URL: https://arxiv.org/abs/0907.3509, arXiv:0907.3509.