Journal of Mathematics in Industry
a SpringerOpen Journal

RESEARCH Open Access

Data-driven graph drawing techniques with applications for conveyor systems

Simone Göttlich[1*], Sven Spieckermann[2], Stephan Stauber[2] and Andrea Storck[1]

*Correspondence:
goettlich@uni-mannheim.de
[1]Department of Mathematics, University of Mannheim, 68131, Mannheim, Germany
Full list of author information is available at the end of the article

Abstract

The visualization of conveyor systems in the sense of a connected graph is a challenging problem. Starting from communication data provided by the IT system, graph drawing techniques are applied to generate an appealing layout of the conveyor system. From a mathematical point of view, the key idea is to use the concept of stress majorization to minimize a stress function over the positions of the nodes in the graph. Different to the already existing literature, we have to take care of special features inspired by the real-world problems.

MSC: 68R10; 90C35

Keywords: Graph drawing; Numerical methods; Simulation study

1 Introduction

Currently, graphs are widely used in various domains to visualize important information clearly. Subway timetables, road maps, mobile networks, family trees and social networks can be easily represented by graphs. In this paper, we discuss how well a graph can represent the exact physical structure of a real-world conveyor system by using only its communication data as input.

Graph drawing dates back to the works of Fáry [13] on theorems about planarity, the spring algorithm by Tutte [28] or the visualization of flowcharts by Knuth [23]. In the preceding three centuries, graph drawing has been extensively researched. Good reviews of various approaches and methods are provided in [7, 10, 19, 21]. Many of these methods do not consider distances between nodes. However, doing so is one of crucial criteria in drawing the physical structure of a conveyor system. Additionally, most algorithms are designed to draw aesthetically pleasing graphs. More specifically, e.g., the nodes should be spread well over the drawing, and edge crossing should be avoided. In this paper, we focus not on these aesthetic criteria but on visualizing the structure of a conveyor system as accurately as possible. Drawing the physical structure of a network as a graph is an approach mainly used in the area of sensor networks [18, 25] and molecular structures [1]; however, to the best of our knowledge, it has never been applied to conveyor systems. Eades and Wormald proposed in [12] that to draw a graph with edges being straight lines that do not cross each other and with the same as well as different distances between all nodes is an NP-hard problem. One of the most common force-directed algorithms that

consider the distances between nodes is that proposed by Kamada and Kawai [20]. It iteratively minimizes an energy or stress function that consists of differences of real or desired distances for all pairs of nodes and their Euclidean distances. Gansner et al. [16] improved this algorithm by applying the majorization method for the area of multidimensional scaling to minimize the energy or stress function. Further algorithms for large graphs have also been proposed. Two of them are COAST [14] and the maxent-stress method [15, 24]. Both methods reformulate the stress function by splitting the summation in two parts. One summation consists of the set of all directly connected nodes and the other one of indirectly connected nodes. In [30] Zheng et al. use a modified stochastic gradient descent to minimize the stress function instead of majoriztaion. Although this stochastic gradient descent seems to be beneficial regarding performance issues, we focus on the stress majorization proposed by Gansner, Koren and North [16] as our major concern is the generation of an appealing layout of the real conveyor system. Subsequently, we will discuss a second method for graph drawing, namely, the classic multidimensional scaling [4–6, 27] that computes a good initial layout for stress majorization.

The motivation of applying graph drawing approaches to conveyor systems is to ease the application of process mining on logistic material flow. Process Mining in highly automated logistics and production systems essentially focuses on the analysis and visualization of process data to improve business processes in the operational area [17]. Due to the high manual effort, there is often no direct projection of the process graph onto a realistic representation of the system components. This, however, is expected by involved systems engineers as they are used to such visual representations for example from conducting simulation studies of logistic systems with discrete event simulation software packages [3]. For the application of both methodologies, process mining and discrete event simulation, an efficient and generic creation of the visual representation of the conveyor system is important and very helpful. In the field of discrete event simulation, existing concepts for automatic model generation always focus on the transformation of CAD data or the transfer of data from other software systems like process and structure describing modeling environments [2]. Accordingly, these approaches are highly tool-dependent or rather abstract in their representation, and thus can only be generalized to a limited extent. The idea to automatically generate layout structures of the conveyor system only from the communication data between subordinate and superordinate IT systems is a completely different independent approach and served as motivation for the research presented in this article.

2 Model description

In conveyor systems, scanners installed at various locations register all passing goods. Given this data, we want to draw a graph that shows the physical structure of the conveyor system as effectively as possible. The locations of scanners will be represented by the nodes of the graph, and the conveyor will be represented by the corresponding edges. Using the communication data of scanners, we can identify the exact path or the distance traveled by the goods and the required time. Let the velocity of the system be known. Then, we can compute the length of every single distance. This information is required for the graph drawing algorithms presented in this paper.

Before we introduce the methods used to draw graphs, we explain how we modify the communication data and the assumptions we make. Given that many goods are transported over the same path or at least the same distances, the scanners register these paths

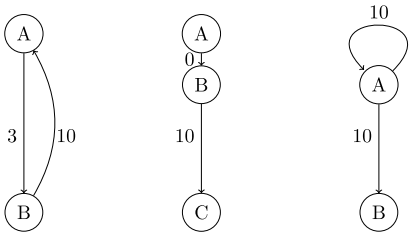


Figure 1 Three examples of our modification of communication data

more than once with likely different time durations. However, to compute the distance of the path in the graph, we have to pick one of these different time durations. Therefore, we assume that the fastest good transported along the path represents the length of this distance. Another important assumption we make is that only the time duration of distances that are directly connected is known. The recorded time duration for every distance is positive and valid. Furthermore, we assume that the names of scanners are random, and hence, we do not use such naming information to draw the graph. In addition, all goods are the same and have the same shape and properties. We neglect the capacity of conveyor systems. It is easier to draw graphs with straight lines than those with curves. Therefore, we assume that all edges in a graph are straight rather than curvy or polygonal, even though in real-world conveyor systems, there are numerous curved conveyors. Although the goods can only be transported in the same direction, we assume the graph to be undirected. The algorithm of stress majorization that we use to draw the graph can visualize only connected graphs or, if the graph is not connected, only one part of the graph. Hence, we ensure that the graph is connected and delete unconnected nodes.

We modify communication data so that if there are inverse edges with different distance lengths, the shorter distance will be used to draw the graph, as shown in the first graph in Fig. 1. The distance over which the goods are transported from node A to node B in this example is 3, but the distance over which the goods are transported from B to A is 10. We cannot draw nodes A and B at a distance of 3 and at the same time at a distance of 10. Therefore, we take the smaller distance and draw the nodes with a distance of 3. In the second graph in Fig. 1, nodes A and B are connected with a distance of 0. In real-world systems, the scanners may happen to be close to each other, and therefore, the time duration of transporting the goods from A to B will be practically 0. However, the algorithm for drawing graphs cannot cope with this distance because a distance of 0 means that there is no direct connection between nodes A and B. To avoid these problems, we remove all connections with distances of 0 from the data, and the nodes of scanners will be edited so that the graph stays connected. In our example in Fig. 1, node B will be deleted from the data, and we will add a new connection from node A to node C with a distance of 10. If node B would be directly connected to nodes other than node C, then it would be replaced by node A. Furthermore, we delete all loops from the data, as shown in the last example in Fig. 1. The loop from node A to node A will be removed from the data because the algorithm for drawing graphs cannot handle loops.

2.1 Stress majorization

In this section, we present stress majorization as described in [11, 16, 29]. Let an undirected graph $G = (V, E)$ with vertex or node set $V = \{1, \ldots, n\}$ and edge set E be given.

The placement of nodes will be defined by matrix $X \in \mathbb{R}^{n\times a}$, where a represents the dimension of the visualization. $X_i \in \mathbb{R}^a$ denotes the coordinates of node i. To obtain a good visualization of the graph, we minimize the following stress or cost function:

$$f(X) = \sum_{i<j} \omega_{ij}\big(\|X_i - X_j\| - d_{ij}\big)^2, \tag{2.1}$$

where $\omega_{ij} \geq 0$ is the weight for the measurement of vertex pair (i,j), d_{ij} is the graph-theoretical distance of nodes i and j, and $\|\cdot\|$ is the Euclidean norm. For weight $\omega_{ij} = d_{ij}^q$, we take $q = -2$ in this paper. Normally, just the distances of the adjacent nodes are known. To calculate the shortest distances of all non-adjacent vertices, we use the Dijkstra's algorithm. Hence, d and w are dense and represent matrices of all distances and of all weights, respectively.

For minimizing cost function (2.1), Kamada and Kawai proposed the Newton–Raphson method [20]. The majorization of this cost function was proposed by de Leeuw [9] and was first used in multidimensional scaling. It guarantees a monotonic decrease of cost and is therefore more robust and less time-consuming. Expanding cost function (2.1), we obtain the following equation:

$$f(X) = \sum_{i<j} \omega_{ij} d_{ij}^2 + \sum_{i<j} \omega_{ij}\|X_i - X_j\|^2 - 2\sum_{i<j} \omega_{ij} d_{ij}\|X_i - X_j\|. \tag{2.2}$$

The first term of the expanded cost function (2.2) is a constant and thus independent of the placement of nodes. The second term is a quadratic sum and can be written with a so-called weighted Laplacian [16] as follows:

$$\sum_{i<j} \omega_{ij}\|X_i - X_j\|^2 = \sum_{k=1}^{a} \big(X^{(k)}\big)^T L^\omega X^{(k)} = \operatorname{tr}\big(X^T L^\omega X\big),$$

where the elements of the Laplacian are defined as

$$L_{ij}^\omega = \begin{cases} -\omega_{ij}, & i \neq j, \\ \sum_{l=1, l\neq i}^{n} \omega_{il}, & i = j. \end{cases} \tag{2.3}$$

By definition, this matrix is symmetric and weakly diagonal-dominant. Hence, the Laplacian is positive semidefinite.

Using the Cauchy–Schwarz inequality and matrix $Z \in \mathbb{R}^{n\times a}$, we can bound the third term from below as follows:

$$\sum_{i<j} \omega_{ij} d_{ij}\|X_i - X_j\| \geq \sum_{i<j} \omega_{ij} \cdot d_{ij} \cdot \operatorname{inv}\big(\|Z_i - Z_j\|\big)(X_i - X_j)^T (Z_i - Z_j), \tag{2.4}$$

where the inverse is defined as $\operatorname{inv}(x) = \frac{1}{x}$ if $x \neq 0$ and 0 otherwise. We can simplify inequality (2.4) by using another weighted Laplacian L^Z,

$$\sum_{i<j} \omega_{ij} d_{ij}\|X_i - X_j\| \geq \sum_{k=1}^{a} \big(X^{(k)}\big)^T L^Z Z^{(k)} = \operatorname{tr}\big(X^T L^Z Z\big)$$

with the elements of the Laplacian defined as

$$L_{ij}^{Z} = \begin{cases} -\omega_{ij} \cdot d_{ij} \cdot \operatorname{inv}(\|Z_i - Z_j\|), & i \neq j, \\ -\sum_{l=1, l\neq i}^{n} L_{il}^{Z}, & i = j. \end{cases}$$

Combining all three terms, we obtain the following function:

$$\begin{aligned} g^{Z}(X) &= \sum_{i<j} \omega_{ij} d_{ij}^2 + \sum_{k=1}^{a} \left(\left(X^{(k)}\right)^T L^{\omega} X^{(k)} - 2\left(X^{(k)}\right)^T L^{Z} Z^{(k)} \right) \\ &= \sum_{i<j} \omega_{ij} d_{ij}^2 + \operatorname{tr}\left(X^T L^{\omega} X\right) - 2\operatorname{tr}\left(X^T L^{Z} Z\right). \end{aligned}$$

Now, we can bound stress function $f(X)$ from above:

$$f(X) \leq g^{Z}(X). \tag{2.5}$$

Equality holds for $Z = X$ and $Z \in \mathbb{R}^{n\times a}$. Differentiating $g^{Z}(X)$ with respect to X, we can find the minimum by solving the following equation:

$$L^{\omega} X = L^{Z} Z. \tag{2.6}$$

Equivalently, we can solve this equation for every dimension k:

$$L^{\omega} X^{(k)} = L^{Z} Z^{(k)}, \quad k = 1, \ldots, a. \tag{2.7}$$

This linear system has no unique solution. As mentioned before, Laplacian (2.3) is positive semidefinite. Because the row sum $\sum_{j=1}^{n} l_{ij}^{\omega} = 0, \forall i \in \{1, \ldots, n\}$ of L^{ω} one of the eigenvalues has to be $\lambda_k = 0$ with multiplicity 1. Thus, the rank of the Laplacian is $\operatorname{rank}(L^{\omega}) = n - 1$. Conversely, $g^{Z}(X)$ is a quadratic function with a symmetric positive semidefinite matrix $L^{\omega} \in \mathbb{R}^{n\times n}$. Hence, every local minimum of $g^{Z}(X)$ has to be a global minimum. To obtain a solution of equation (2.7), if a solution exists, in a classic multidimensional scaling approach, the Moore–Penrose inverse is used to solve $X = (L^{\omega})^{\dagger} L^{Z} Z$. However, it is too time-consuming to compute the solution in this way. Therefore, we apply another method proposed by Gasner et al. [16]. By setting any $X_i = 0$, we can remove the ith row and column of Laplacian L^{ω}. Consequently, we obtain an $(n-1) \times (n-1)$ matrix that is strictly diagonal-dominant because removing a row and column makes the row sum unequal to 0. Furthermore, the ith row of $L^{Z} Z$ can be removed. Since Laplacian L^{ω} is positive definite, the global minimum is unique, and we can either use direct methods such as the Cholesky decomposition or iterative methods like conjugate gradients to solve the linear system (2.7). Note that in [30] Zheng et al. the various solution techniques are compared in a numerical study. Throughout this paper we only apply the Cholesky decomposition.

Now, we can formulate the iterative optimization process. Let $X(t)$ be the coordinates of nodes. We want to compute the new coordinates $X(t+1)$ so that $f(X(t+1)) < f(X(t))$, and we know that inequality (2.5) leads to $g^{X(t)}(X(t)) = f(X(t))$ for $X(t) = Z(t)$. To find the

new coordinates $X(t+1)$ with a lower cost, we have to solve the following linear system for every dimension k:

$$L^{\omega}X(t+1)^{(k)} = L^{X(t)}X(t)^{(k)}, \quad k = 1,\dots,a. \tag{2.8}$$

Using the computed solution $X(t+1)$, we can iteratively find the optimal coordinates by solving system (2.8) with the new solution. In every step, matrix L^{ω} remains the same, so just one Cholesky decomposition is necessary. Because of the abovementioned properties, we have a strictly monotonically decreasing cost function:

$$f\big(X(t+1)\big) \le g^{X(t)}\big(X(t+1)\big) < g^{X(t)}\big(X(t)\big) = f\big(X(t)\big), \quad \text{for } X(t+1) \neq X(t). \tag{2.9}$$

Theoretically, we stop the iteration if $X(t+1) = X(t)$. However, in practice, we stop if the difference between costs of new and old placements relative to the cost of the old placement is less than $\varepsilon > 0$, i.e.,

$$\frac{f(X(t)) - f(X(t+1))}{f(X(t))} < \varepsilon. \tag{2.10}$$

In general, it is difficult to predict the number of iterations, the rate of convergence is very low and the computation of the shortest path via Dijkstra's algorithm is $\mathcal{O}(n^3)$, so stress majorization is impractical for large graphs. Thus, we apply the successive over-relaxation (SOR) method proposed by Wang and Wang [29]. This method combines two solutions—the current and new values of the iterative process—to compute the new corrected coordinates X^*. The current and new solutions approximate the true solution, but the new solution $X(t+1)$ is better than the current $X(t)$ due to the property $f(X(t+1)) < f(X(t))$ for $X(t+1) \neq X(t)$. The intuition behind this method is that the new and better solution $X(t+1)$ has been strengthened through the relaxation factor $\tau \ge 0$, while the worse solution $X(t)$ has been suppressed. If the cost of the combined solution $X^* = (1+\tau)X(t+1) - \tau X(t)$ is less than that of the computed solution $X(t+1)$ of the linear system, then we accept X^* and reject $X(t+1)$. Otherwise, we reject X^* and continue with the iteration by solving linear system (2.8). The relaxation factor τ determines the degrees of weighting of the two computed solutions. At $\tau = 0$, we obtain the original optimization process without SOR. Algorithm 1 presents stress majorization with successive overrelaxation.

There are several other methods to make the stress majorization scalable to large graphs, e.g. the sparse stress model [26] or MARS [22]. As already mentioned, Zheng et al. [30] proposed another method to minimize the stress function. They use a modified stochastic gradient descent with random reshuffling and step size annealing. Since the major concern of this paper is the reproduction of the physical structure of conveyor systems and not enhanced performance we stick to stress majorization with SOR.

2.2 Classical multidimensional scaling

In this section, classic multidimensional scaling (CMDS) will be presented. In general, this method produces a good initial layout $X(0)$ for the optimization process. This section is based on [4–6, 27].

Multidimensional scaling was designed to show dissimilarities of different objects as graphs. Classic multidimensional scaling interprets dissimilarities as Euclidean distances

Algorithm 1 Stress majorization [29, p. 681]

Input: Matrix $d \in \mathbb{R}^{n\times n}$ with the respective distances, relaxation factor $\tau \geq 0$

Output: Matrix $X \in \mathbb{R}^{n\times a}$ with the coordinates of all nodes

1: Initialize the placement $X(0)$ and set $t = 0$
2: **while** $\frac{f(X(t))-f(X(t+1))}{f(X(t))} \geq \varepsilon$ **do**
3: Solve the system of linear equations $L^{\omega}X(t+1)^{(k)} = L^{X(t)}X(t)^{(k)}$, $k = 1,\dots,a$ for all dimensions k
4: Set $X^* = (1+\tau)X(t+1) - \tau X(t)$
5: **if** $f(X^*) \leq f(X(t+1))$ **then**
6: Set $X(t+1) = X^*$
7: **end if**
8: Set $t = t+1$
9: **end while**
10: **return** $X(t)$.

between particular objects and tries to show the global structure of the data. To represent the global structure, CMDS focuses on long distances in contrast to stress majorization that focuses on shorter distances due to weight $\omega = d^{-2}$.

As in the setting for stress majorization, we have the object or node set $V = \{1,\dots,n\}$ and the matrix $d \in \mathbb{R}^{n\times n}$ with the dissimilarities or distances d_{ij} for all nodes $i,j \in V$. For any distance, we have $d_{ij} = d_{ji} \geq 0, i \neq j, i,j \in V$ and $d_{ii} = 0, \forall i \in V$. Hence, matrix d is symmetric with entries on the main diagonal $d_{ii} = 0$ and nonnegative entries outside. As above, $X \in \mathbb{R}^{n\times a}$ represents the coordinates of all nodes. In this paper, we only consider graphs in the two-dimensional space. Therefore, we only discuss CMDS for $a = 2$. Nevertheless, the algorithm also applies to higher-dimensional spaces. The goal of classic multidimensional scaling is to find the coordinates of nodes in 2D such that

$$d_{ij}^2 = \|x_i - x_j\|^2 = \langle x_i - x_j, x_i - x_j\rangle = \langle x_i, x_i\rangle - 2\langle x_i, x_j\rangle + \langle x_j, x_j\rangle. \tag{2.11}$$

The distances in a Euclidean space are invariant under translation, so we assume that coordinates X are centered with respect to the origin so that $\sum x_i = 0$ holds. Solving equation (2.11) for $\langle x_i, x_j\rangle$, we obtain

$$\langle x_i, x_j\rangle = -\frac{1}{2}\left(d_{ij}^2 - \langle x_i, x_i\rangle - \langle x_j, x_j\rangle\right). \tag{2.12}$$

Due to centering with respect to the origin, we obtain the following equations for the row average, the column average and the total average:

$$\frac{1}{n}\sum_{i=1}^{n} d_{ij}^2 = \frac{1}{n}\sum_{i=1}^{n}\langle x_i, x_i\rangle + \langle x_j, x_j\rangle,$$

$$\frac{1}{n}\sum_{j=1}^{n} d_{ij}^2 = \langle x_i, x_i\rangle + \frac{1}{n}\sum_{j=1}^{n}\langle x_j, x_j\rangle,$$

$$\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}^2 = \frac{2}{n}\sum_{i=1}^{n}\langle x_i, x_i\rangle.$$

Algorithm 2 Classic multidimensional scaling [4, p. 105f.]

Input: Matrix $d \in \mathbb{R}^{n\times n}$ with the respective distances

Output: Matrix $X \in \mathbb{R}^{n\times 2}$ with the coordinates of all nodes

1: Compute d^2 by squaring every entry of d
2: Convert the squared distances to scalar products through double centering of d^2: $B = -\frac{1}{2} Z_n d^2 Z_n$ with $Z_n = I_n - \frac{1}{n}\mathbb{1}_n$, where I_n is the $n \times n$ identity matrix, and $\mathbb{1}_n$ is the $n \times n$ unit matrix
3: Compute the eigendecomposition $B = U\Lambda U^T$
4: Take the two largest positive eigenvalues λ_1 and λ_2 and the respective eigenvectors u_1 and u_2, and set $X = [\sqrt{\lambda_1}u_1, \sqrt{\lambda_2}u_2]$
5: **return** X.

Using these averages in equation (2.12) allows for the entries of matrix $B = (b_{ij}) \in \mathbb{R}^{n\times n}$ to be written as

$$b_{ij} = \langle x_i, x_j \rangle = -\frac{1}{2}\left(d_{ij}^2 - \frac{1}{n}\sum_{i=1}^{n} d_{ij}^2 - \frac{1}{n}\sum_{j=1}^{n} d_{ij}^2 + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}^2 \right)$$

or as matrix

$$B = XX^T = -\frac{1}{2} Z_n d^2 Z_n$$

with matrix d^2 computed by squaring every entry and centering matrix $Z_n = I_n - \frac{1}{n}\mathbb{1}_n$, where I_n is the $n \times n$ identity matrix, and $\mathbb{1}_n$ is the unit matrix. To compute coordinates $X \in \mathbb{R}^{n\times 2}$, we can use the eigendecomposition. Due to the symmetry of B and the real-valued entries, all eigenvalues are real, and we can find orthonormal eigenvectors. Let λ_1 and λ_2 be the two biggest positive eigenvalues and u_1 and u_2 be the corresponding eigenvectors. To obtain coordinates X, we set

$$X = [\sqrt{\lambda_1}u_1, \sqrt{\lambda_2}u_2].$$

The eigenvalues and the corresponding eigenvectors can be easily computed by power iteration. The classic multidimensional scaling minimizes the following cost function:

$$f(X) = \left\| B - XX^T \right\|^2 = \sum_{i,j} \left(b_{ij} - x_i^T x_j \right)^2.$$

Algorithm 2 summarizes this method.

The complexity of computing matrix d with all short distances with the Dijkstra's algorithm is $\mathcal{O}(n^3)$; it takes $\mathcal{O}(n^2)$ to compute matrix B and $\mathcal{O}(n^2)$ per iteration. Therefore, the total complexity of the classic multidimensional scaling is $\mathcal{O}(n^3)$.

3 Applications

In this section, we will discuss the results of our tests. First, we test the implementation on graphs from the extant literature. Then, we apply Algorithms 1 and 2 to real-world data of conveyor systems and discuss the problems that occur. All datasets of graphs from the

Table 1 Datasets from the literature*

Name	$\|V\|$	$\|E\|$	Average degree	Description
1138_bus	1138	1458	2.5624	Power system network
plat1919	1919	15,240	15.8833	Symmetric finite difference three-ocean model
USpowerGrid	4941	6594	2.6691	US power grid

* Source: https://sparse.tamu.edu/, retrieved on: March 6, 2020.

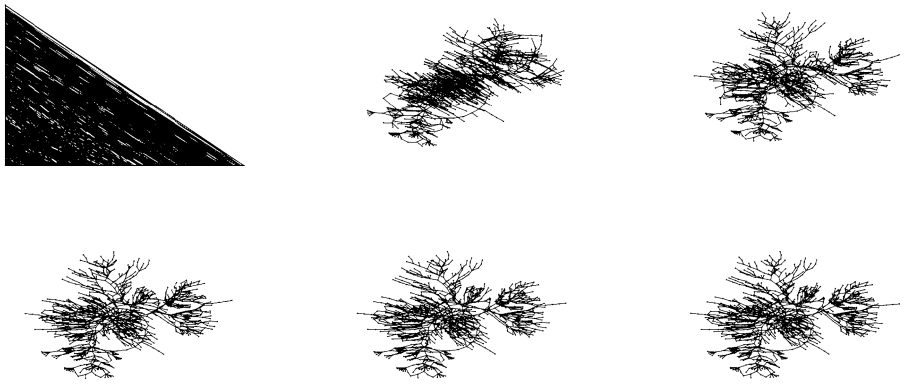


Figure 2 Stress majorization applied to graph 1138_bus after 10, 20, 30, 40 and 47 iterations, respectively

literature that are used in this section are from [8]. The implemented algorithms are run on a computer with Windows 10 version 1809, 8.00 GB of RAM and Intel Core i7-7500U CPU operating at 2.70 GHz.

3.1 Academic examples

The three examples commonly used in the literature are described in Table 1.

All of these graphs are modified so that the distance of every edge is $d_{ij} = 1$ m. The error bound is $\varepsilon = 0.0001$. Algorithm 1 of stress majorization stops if

$$\frac{f(X(t)) - f(X(t+1))}{f(X(t))} < \varepsilon$$

or

$$\left\| X(t) - X(t+1) \right\|_\infty < 0.01,$$

where $\|\cdot\|_\infty$ defines the row-sum norm. In the literature, $\omega = d^{-2}$ has proven to be a good choice of weight, so we use this value as a standard.

Figure 2 shows stress majorization applied to graph 1138_bus. The first image in the top left shows the initial layout. Here, we do not use CMDS to compute the initial layout. Instead, we compute a simple initial layout by placing different nodes alternately at the x- and y-axes with distance $d_{ij} = 1$ m and adding the edges afterwards. Subsequently, stress majorization is applied to the initial layout to optimize the graph. The relaxation factor is $\tau = 2$. We will discuss the optimal choice of this relaxation factor further on. The other images show the progress of the graph after 10, 20, 30, 40 and 47 iterations, respectively. We observe that after 20 iterations the main structure of the graph is visible, and after-

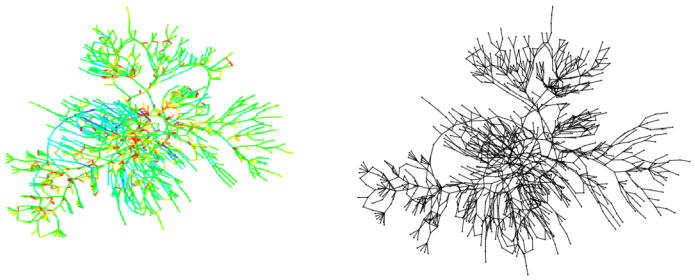


Figure 3 1138_bus, left: result from [14], right: our result

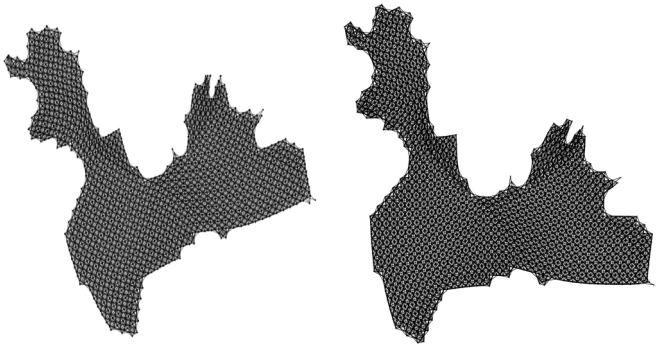


Figure 4 plat1919, left: result from [26], right: our result

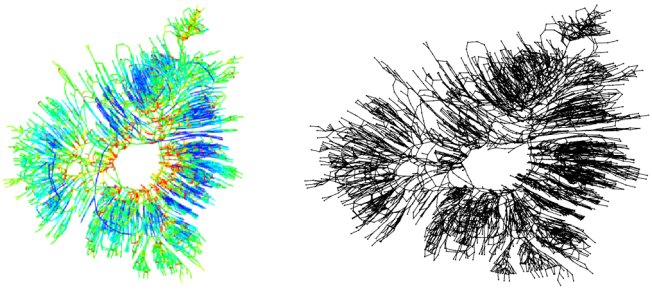


Figure 5 USpowerGrid, left: result from [14], right: our result

wards, only small changes occur. After 47 iterations with 35 accepted SOR values, one of the stop criteria is fulfilled, and the algorithm terminates.

There is no true solution, and the graph drawn depends on the configuration. Therefore, we verify Algorithm 1 for stress majorization by comparing our results with those in the literature. The configurations are based on those in the literature as far as they are known.

Figures 3, 4 and 5 show the comparisons for the three respective examples. We observe that our computed graphs are similar to those in the literature.

3.1.1 SOR

In this section, we will discuss the optimal choice of relaxation factor τ for successive overrelaxation. Using the optimal factor is very important since a reasonable choice of τ

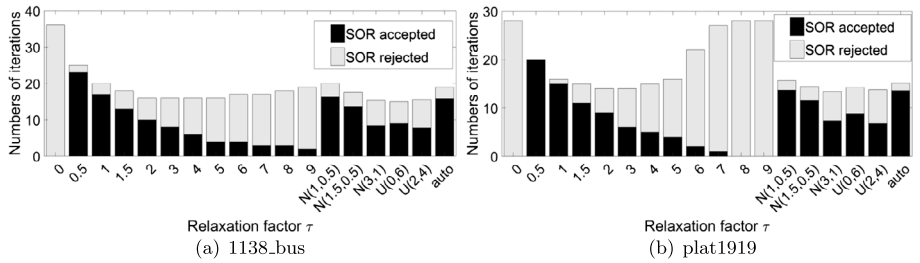


Figure 6 Comparison of the numbers of iterations for various values of relaxation factor τ with classic multidimensional scaling as the initial layout

will greatly enhance the algorithm's computation speed. Thus, we would not need as many iterations of stress majorization to draw the optimal graph.

As explained above, the SOR method combines two solutions and computes a new candidate X^*. The relaxation factor determines how heavily solution $X(t+1)$ will be weighted. At $\tau = 0$, we perform stress majorization without the SOR strategy. Due to the relaxation factor, we can accelerate the convergence of the cost function.

Figure 6 shows the total number of iterations for various choices of relaxation factor τ used with datasets 1138_bus and plat1919 with CMDS as the initial layout. The gray bar shows the number of relaxations rejected because the cost of the computed solution X^* is higher than that of the solution computed by solving linear system (2.8). In contrast, the black bar shows the accepted relaxations. We try strategies of different types for the relaxation factor to determine the impact. As proposed in [29], we use fixed and probabilistic strategies. In the fixed strategy, we choose the value of the relaxation factor before the iteration starts and never change it during the iterations. We consider the following values of the factor: $\tau \in \{0, 0.5, 1, 1.5, 2, 3, 4, 5, 6, 7, 8, 9\}$. In the probabilistic strategy, we choose probability distribution 'auto' proposed in [29], which is defined as $P(\tau = 0.5) = P(\tau = 1) = 0.3, P(\tau = 1.5) = P(\tau = 2) = 0.2$. In addition, we test uniform distributions $U(0, 6)$ and $U(2, 4)$ as well as normal distributions $N(1, 0.5)$, $N(1.5, 0.5)$ and $N(3, 1)$. The results for probability distributions are based on the arithmetic average of 20 runs. Figure 5 shows that in most cases, it is beneficial to use the SOR method, and $\tau = 0$, i.e., $X^* = X(t+1)$, is a bad choice. Furthermore, we observe that for factors $\tau > 1$ the number of accepted relaxations becomes progressively lower, yet the total number of iterations also decreases. For large factor values, the number of iterations increases due to relaxations being accepted rarely. The probability distributions with expectation $\mu < 2$, i.e., $N(1, 0.5)$ with expectation $\mu = 1$, $N(1.5, 0.5)$ with $\mu = 1.5$ and probability distribution 'auto' proposed in [29] with expectation $\mu = 1.15$, are even worse choices than the others.

Figure 7 shows the speedup of the SOR method for various values of relaxation factor if the method is applied to the USpowerGrid dataset. The running time of Algorithm 1 without SOR is 133.3 s. However, if the SOR method is used with probability distribution $\tau \sim U(0, 6)$, stress majorization requires only 65.05 s to reach the same cost level. Even though Algorithm 1 needs more time in every step to compute the coordinates of nodes $X^* = (1 + \tau)X(t + 1) - \tau X(t)$ and the cost $f(X^*)$, the speedup of the SOR method is undeniably significant even for large relaxation factors. Because using probability distribution $U(0, 6)$ results in a good graph being drawn after few iterations for the analyzed

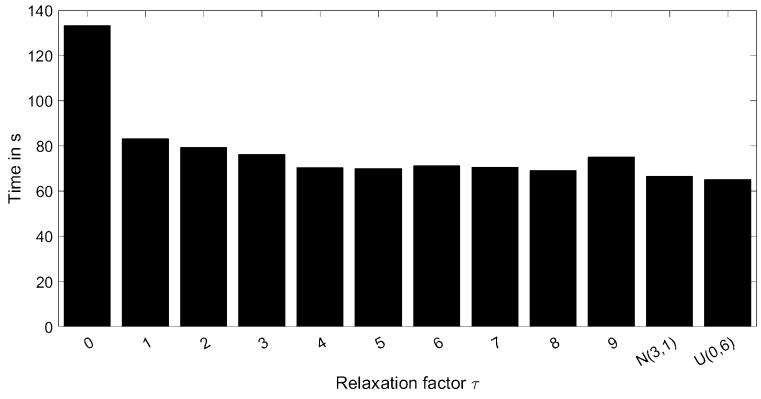


Figure 7 Comparison of running times for various values of relaxation factor τ in the SOR method applied to the USpowerGrid dataset

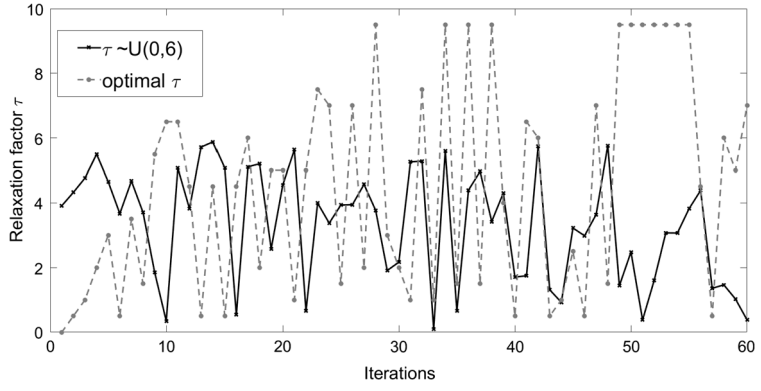


Figure 8 Comparison of various strategies for relaxation factor τ applied to the USpowerGrid dataset

datasets (cf. Fig. 6) and therefore speeds up the optimization process, in what follows, we will henceforth consider relaxation factor $\tau \sim U(0,6)$.

In [29], the researchers propose an enumerating strategy that in every iteration, seeks the best relaxation factor by using an exhaustive search method. For simplicity, the search for the optimal factor uses the discrete candidate set $\tau \in \{0, 0.5, 1, 1.5, \ldots, 8.5, 9, 9.5\}$. We compare this strategy with our proposed probabilistic strategy $\tau \sim U(0,6)$. Figure 8 shows the results. The black line represents our strategy, and the gray dashed line corresponds to the optimal factor found in the given set in every iteration. Although a uniformly distributed τ differs greatly from the optimal factor, it leads to good graphs being obtained after few iterations within a short time. It would be too time-consuming to consider all candidate relaxation factors in each iteration, and the time needed for this computation cannot be canceled out by the time saved by the reduction of the number of iterations.

3.1.2 Initial layout

Brandes et al. proposed and confirmed in [6] the hypothesis that classic multidimensional scaling provided a good initial layout for stress majorization. The simple initial layout, which places the nodes on the axes (cf. Fig. 2, top left), has no structural similarities with the optimal graph. Furthermore, many iterations are necessary to minimize the cost. As

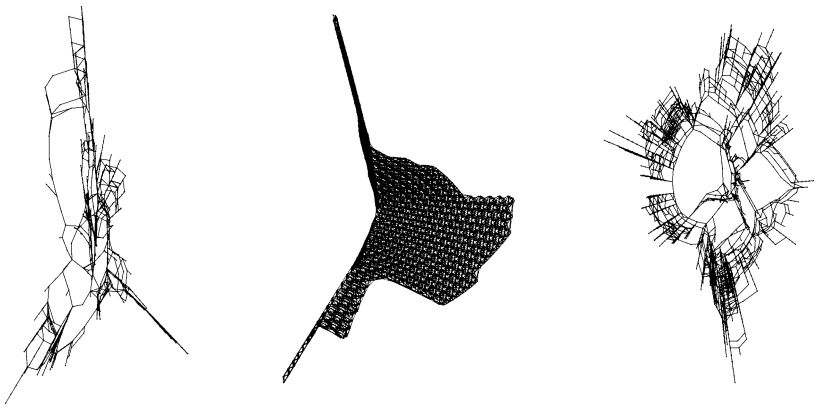


Figure 9 CMDS applied to 1138_bus, plat1919 and USpowerGrid datasets

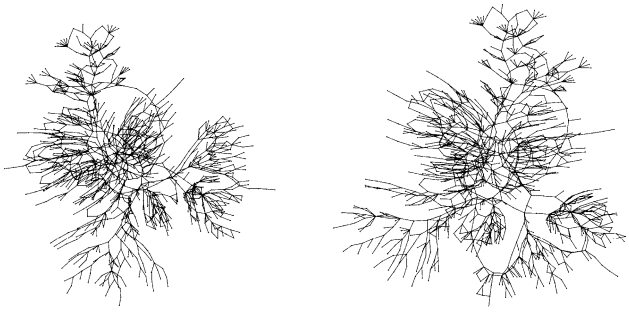


Figure 10 Comparison of stress majorization with a simple initial layout (left) and CMDS (right) applied to the 1138_bus dataset

explained in the previous section, the classic multidimensional scaling is better at representing the global structure with the focus on long distances. In this section, we discuss the main advantages of CMDS. Figure 9 shows the initial layouts for 1138_bus, plat1919 and USpowerGrid datasets. If we compare the graphs drawn by CMDS with Figs. 3–5, we observe that the global structure of the graphs is presented well, but the fine details are not. CMDS is a good method for computing the initial layout because the respective costs are less than those of a simple or random initialization. Hence, Algorithm 1 of stress majorization needs fewer iterations to find the optimum, and the total costs are also lower.

Now, we compare the results of stress majorization with various initial layouts. Figure 10 (left) shows the optimal graph that started with a simple initial layout, and the right part shows the optimal graph with the initial layout computed by CMDS. Both methods lead to good graphs. The reason we should prefer the classic multidimensional scaling for computing the initial layout is shown in Fig. 11.

The figure shows a time-cost plot for the 1138_bus dataset. The gray dashed line represents the cost level of the optimization process with a simple initial layout, and the black line represents the cost level of the initial layout computed by CMDS and optimized with stress majorization. As long as in the beginning of the optimization process the cost level does not change, the initial layout, Laplacian L^{ω} and the cost of the initial layout can be computed. The running time of computing the Laplacian and the costs are the same in both methods. Nevertheless, the classic multidimensional scaling needs more time to start

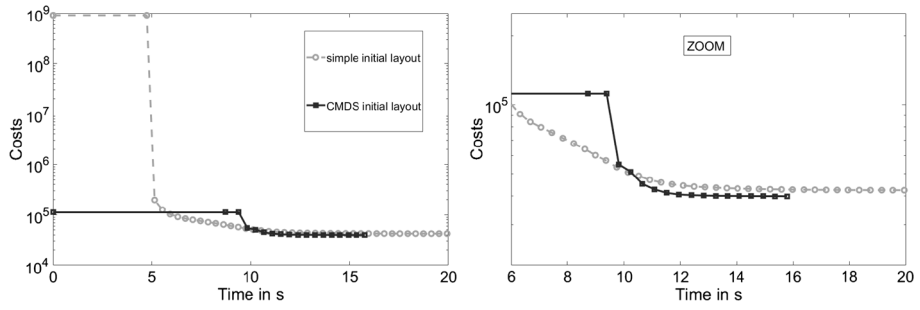


Figure 11 Time-cost plot for various initial layouts for the 1138_bus dataset

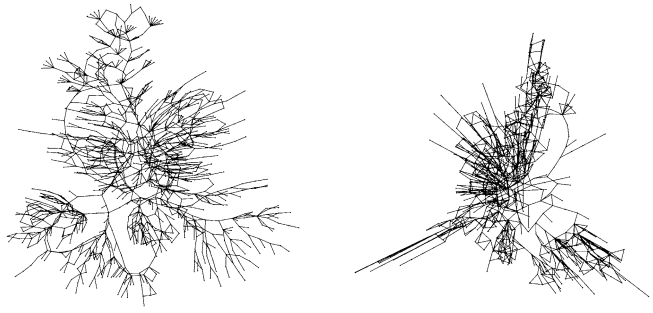


Figure 12 CMDS and stress majorization with full matrix d (left), Pivot MDS and stress majorization with sparse matrix d (right) applied to the 1138_bus dataset

the optimization process (cf. Fig. 11). The simple initial layout is computed in less than a second because the nodes are merely placed on the axes, and there is no need for the Dijkstra's algorithm. To compute all the shortest distances via the Dijkstra's algorithm, approximately 5 s are needed, and both methods have to compute these distances. However, Fig. 11 shows that the optimal graph is found faster by using CMDS to compute the initial layout, even though the initial process requires more time. The speedup of stress majorization with the initial layout computed by CMDS is due to the lower cost level of the initial layout. Therefore, fewer iterations are needed, namely, only 15 iterations on average. It is difficult to see in the figure, but additionally, the total costs at the end are less with CMDS (approximately 39,930 on average) than with the simple initial layout (approximately 42,406 on average).

In [5], Brandes et al. present a fast method to approximate the results of CMDS, the so-called pivot MDS. This method computes the eigendecomposition based on a $k \times n$ matrix with $k \ll n$ pivot elements. However, experiments have shown that the stress majorization applied to initial layouts computed by CMDS lead to a better physical structure compared to initial layouts computed by pivot MDS as seen in Fig. 12. On the left, we computed the initial layout with CMDS and applied the stress majorization to the full matrix d, whereas the right figure shows the result of the stress majorization applied to the pivot MDS with 100 pivot elements as initial layout. Apparently, the graph of the CMDS as initial layout is qualitatively better. This is due to the fact that in pivot MDS the eigendecomposition of the $k \times n$ matrix is computed with the distances of these k pivot elements to all other

Table 2 Table of real-world data

Name	$\|V\|$	$\|E\|$	Average degree
Circuit of an overhead conveyor	141	175	2.482
Conveyor system for electric and electromechanical components	201	244	2.428

nodes while other distances are neglected. Therefore, the stress majorization is applied to the sparse matrix d with just the distances of the k pivot elements to all other nodes.

3.2 Real-world problems

In the previous section, we discussed a reasonable setting for Algorithms 1 and 2 applied to datasets from the literature. However, can the algorithms with the suggested setting also draw graphs that present the physical structure of a conveyor system well? In this section, we will discuss the results of algorithms applied to real-world data of conveyor systems provided by SimPlan.

Table 2 summarizes the data of conveyor systems used in this section. In contrast to the datasets of the literature from the previous section, these data are unmodified. The communication data represent the respective real-world distances of the system; i.e., the algorithms of stress majorization and CMDS have to cope with edges of different lengths.

The real-world datasets are measured data. Therefore, there is uncertainty as to the communication data and distances of the graph. These problems did not occur in the case of the datasets from the literature. How we cope with the uncertainty and the consequences will be discussed in the following sections. The setting for the optimization process is the same as in the previous sections.

3.2.1 Predetermined distances

The distances of two nodes, computed using the communication data, do not always correspond to their Euclidean distances in the conveyor system. For example, spiral conveyors transport the goods in circles to the next stage. In the beginning and at the end of this spiral conveyor, scanners are installed that measure the required transportation time. However, such measured time does not correspond to the physical distance between the locations of scanners. As the goods are transported in circles, they need more time to reach the end of the spiral conveyor. Therefore, the distances computed according to time and velocity overestimate the Euclidean distances. Yet, the CMDS Algorithm 2 and stress majorization 1 try to draw these distances in the graphs as accurately as possible. This causes problems in some cases and leads to inadequate graphs. The conveyor system of a producer of electric and electromechanical components is a good example for a discussion of this problem.

Figure 13 shows the original and initial layouts of the system. The classic multidimensional scaling computes an insufficient initial layout for the graph because of the given communication data. The initial layout is not even similar to the original layout, and therefore, the optimization process is time-consuming.

In this conveyor system, there is no spiral conveyor, but there is a buffer, where the goods are transported and buffered until they can be processed again. During buffering, the goods are not registered by any scanner. Therefore, the computed distance of this buffer is very long. Figure 14 outlines a part of the original layout as a graph. The buffer is marked by the dashed line between the large scanners MFR811 and MFR813. Figure 15 shows the original layout of the conveyor system with marked reference nodes MFR460 and

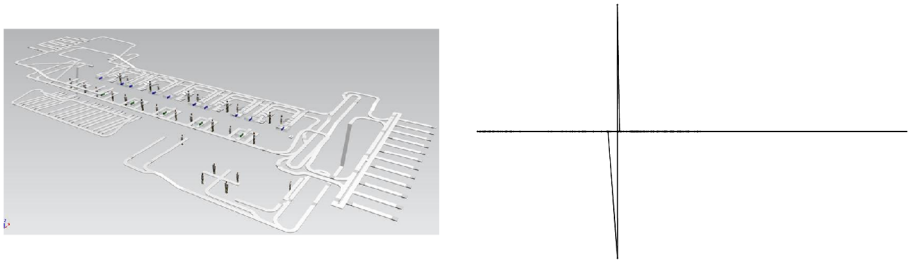
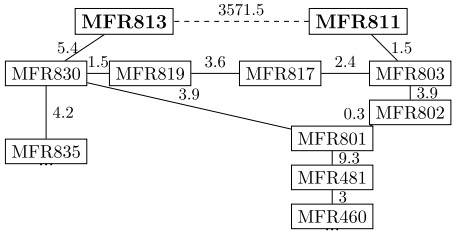


Figure 13 Original and initial layouts computed by CMDS for the conveyor system of electric and electromechanical components

Figure 14 Graph of the buffer



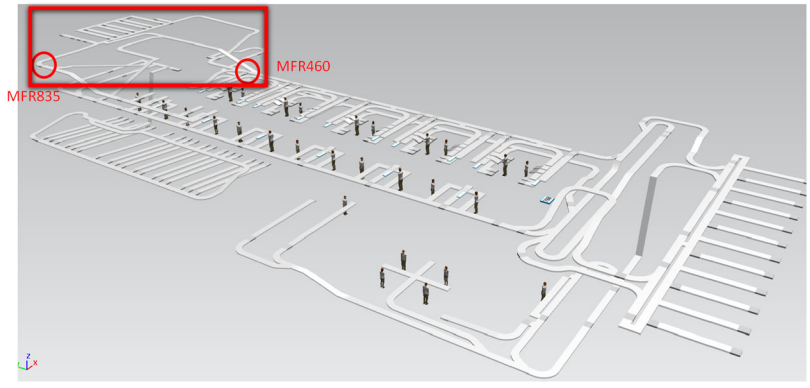


Figure 15 Original layout of the conveyor system with a marked buffer

MFR835; the buffer is located in that area. The distance between the two nodes MFR811 and MFR813 is $d_{MFR811,MFR813} = 3571.5$ m and is thus very long compared to the other distances in the graph. As explained in Sect. 2.2, the classic multidimensional scaling draws the global structure of a graph with the focus on longer distances. Hence, CMDS computes an initial layout that approximates the distance $d_{MFR811,MFR813} = 3571.5$ m well but neglects the shorter distances. This leads to an insufficient initial layout shown in Fig. 13.

Using stress majorization to optimize the initial layout, we obtain the graph shown in Fig. 16. Even though the initial layout is insufficient and not at all similar to the original layout, stress majorization leads to a reasonable graph being drawn. Because of weight $\omega = d^{-2}$, the optimization process focuses on shorter distances, and therefore, the long distance of the buffer has a relatively weak influence on the cost. However, optimization is time-consuming due to the insufficient initial layout. To improve the graph drawn by CMDS and hence the required running time of optimization, we try to fix the uncertainties of

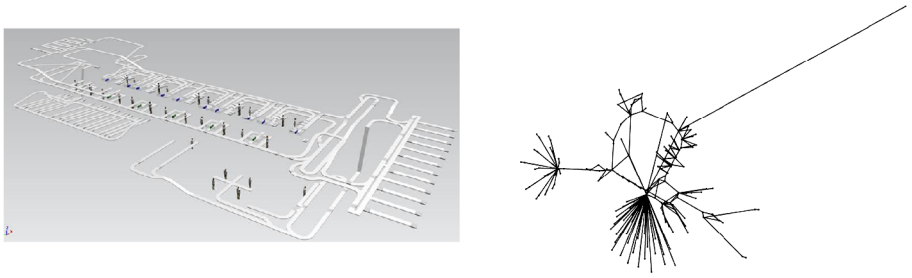


Figure 16 Stress majorization applied to the initial layout computed by CMDS

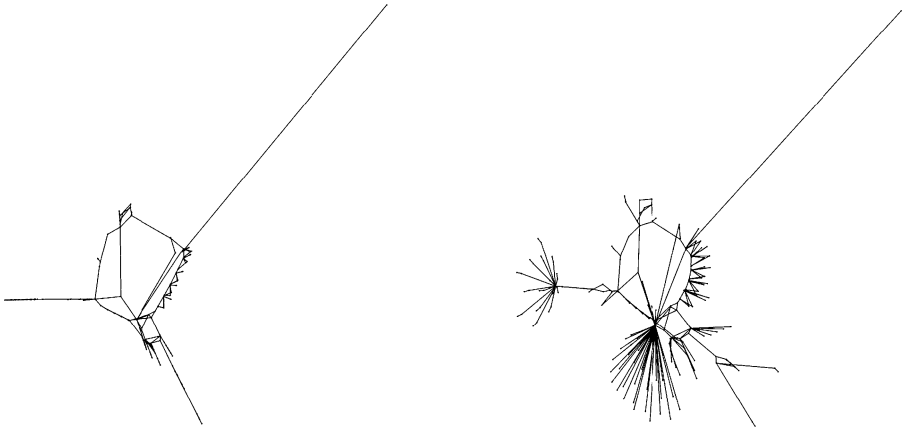


Figure 17 Initial layout and stress majorization with a predetermined distance $d^{\text{new}}_{MFR811,MFR813} = 3.5$

distances by predetermining more realistic values for the problematic distances of buffers or spiral conveyors. Figure 17 shows the results of choosing the distance of the buffer.

We preset the distance of the buffer to the value of $d^{\text{new}}_{MFR811,MFR813} = 3.5$ m. Next, we use classic multidimensional scaling to compute the initial layout with the new distance (cf. Fig. 17 left). The initial graph with the predetermined distance is more similar to the original layout than the initial layout with the distance given by communication data. This occurs because we change the value of the buffer's distance $d^{\text{new}}_{MFR811,MFR813} = 3.5$ m. Now, CMDS does not focus on approximating this long distance exactly anymore but also considers the other distances. If we apply stress majorization to the new computed initial layout, then we obtain an optimal graph similar to that before. However, the optimal graph is computed faster because of the better initial layout; additionally, the costs at the end of the optimization process are less than in the graph shown in Fig. 16.

The distances of buffers or spiral conveyors have to be predetermined manually to compute the optimal graph. Algorithms 1 and 2 cannot themselves recognize these uncertainties.

3.2.2 Reference nodes

Most conveyor systems consist not only of straight conveyors but also of curves and junctions meant to transport goods to different stations. While Algorithm 1 can draw some of the physical structures well, it fails at drawing curvy parts and junctions. This problem is caused not by the uncertainty of communication data but by the use of Dijkstra's

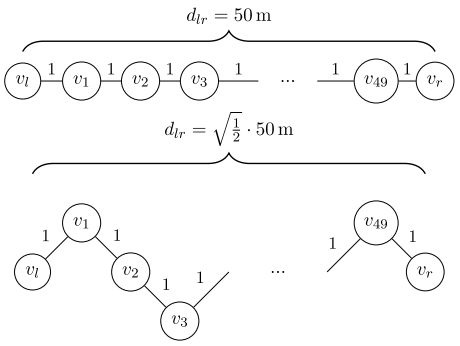


Figure 18 Two possible transportation routes or v_l-v_r paths

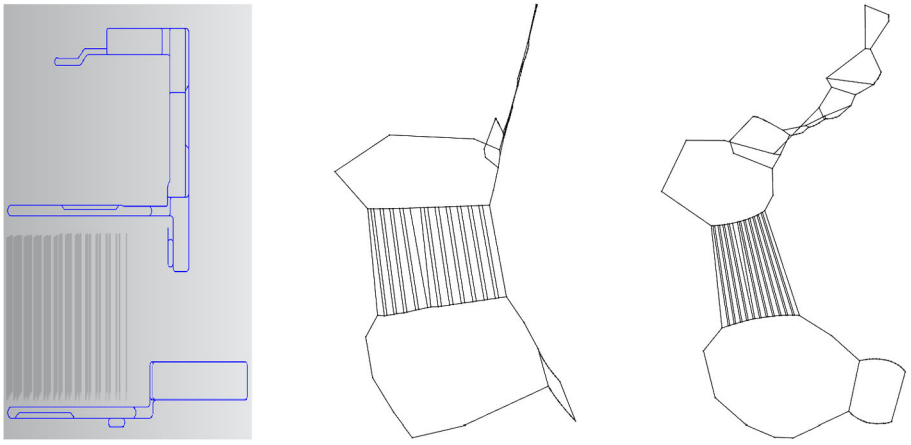


Figure 19 Original, initial and optimal layouts of OCS without reference nodes

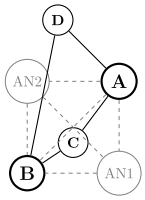
algorithm. The latter computes the shortest distances of all nodes if the distances are unknown. However, the physical distance in the original layout of two nodes does not have to match with the transportation route.

Figure 18 shows two possible transportation routes or v_l-v_r paths. Given the communication data, we cannot decide if path v_l-v_r runs straight ahead or zigzags. By using the Dijkstra's algorithm, we assume that the path is straight. However, in real-world conveyor systems this is not always true: e.g., in Fig. 18, if we compute the distances of both paths by using the Dijkstra's algorithm, then we obtain the result $d_{lr} = 50$ m for both paths. However, the real-world physical distance of the two nodes v_l and v_r in the second example is $d_{lr} = \sqrt{\frac{1}{2}} \cdot 50$ m and therefore much shorter than the computed distance. Hence, the transportation route does not match the physical distance of two nodes.

To improve the visualization of the physical structure, we implement the option to specify some reference nodes. How we use the specified reference nodes within the optimization process will be explained by the circuit of the overhead conveyor.

However, we first examine the initial layout and the optimized graph of the overhead conveyor system (OCS) shown in Fig. 19. The global structure of twelve parallel high-bay warehouses in the middle of the graph is represented well. However, Algorithm 1 struggles to visualize the rectangular structure of the upper part of the circuit. The main reason for this problem is the use of the Dijkstra's algorithm. As explained above, by using the latter we assume that the circuit is straight. Yet, the real-world distances of opposite nodes in

Figure 20 Example of a graph with two reference nodes



the rectangular structure are shorter than those computed by the Dijkstra's algorithm. Neither CMDS nor stress majorization can visualize the upper part of OCS reasonably well (cf. Fig. 19). To improve the graph, we use reference nodes. Any desired number of nodes can be used as reference nodes by specifying the exact coordinates of these nodes in the conveyor system. The idea is to predetermine a frame of reference nodes and to force the other nodes into that frame. Using the new information of the exact positions of reference nodes in optimization, we cannot merely remove each row and column of these nodes in Laplacian L^{ω}; otherwise, we would lose important information about the distances of direct connections between reference nodes and the other nodes, and the visualization of the graph would be worsened.

With the help of Fig. 20, we explain the use of reference nodes in the optimization process. The figure shows a graph with four nodes, namely, $V = \{A, B, C, D\}$, and the given edge set $E = \{e_{AC}, e_{AD}, e_{BC}, e_{BD}\}$, drawn in black. Nodes A and B are our reference nodes, and node A is the main reference node with coordinates $(0, 0)$. Node B should be forced to position $(-3, -3)$, and the other nodes should be drawn according to their computed distances to nodes A and B. The improved algorithm proceeds as follows: first, auxiliary nodes (in Fig. 20, gray nodes AN1 and AN2) will be added to our graph, but will not be visualized in the graph and will be used only for the computation of the optimal graph. The coordinates of auxiliary nodes can be computed with the help of coordinates of reference nodes. The reference nodes and the auxiliary nodes have to form a rectangle, as shown in Fig. 20. Next, we add auxiliary edges (the dashed edges in Fig. 20). Each auxiliary node will be connected with each reference node. Additionally, each auxiliary node and each reference node will be connected if the respective connection does not already exist. Using the coordinates of the reference and auxiliary nodes, we can compute the exact distances of these nodes or overwrite the distances if the direct connections of these nodes already exist with different distances. Then, the graph will be optimized with the help of auxiliary edges. The latter will be weighted very heavily, namely, $\omega_{ij} = 10{,}000$, to guarantee that the distances of these auxiliary nodes will be drawn exactly by using stress majorization. Hence, the reference nodes are forced to their intended positions.

If more than two reference nodes are specified, then the algorithm will follow the same procedure: each reference node will form a new rectangular shape with the main reference node with coordinates $(0, 0)$. Then, each reference node will be connected with each reference and auxiliary node, and each auxiliary node will be connected with each reference and auxiliary node. Finally, the exact distances will be computed, and the initial layout will be optimized using stress majorization.

Using the method of reference nodes improves the graph. Figures 21 and 22 show the initial layouts and the results of stress majorization of the circuit of the overhead conveyor with different numbers of reference nodes. With three reference nodes, Algorithms 1 and 2 can already draw the rectangular structure of the circuit reasonably well. However, the

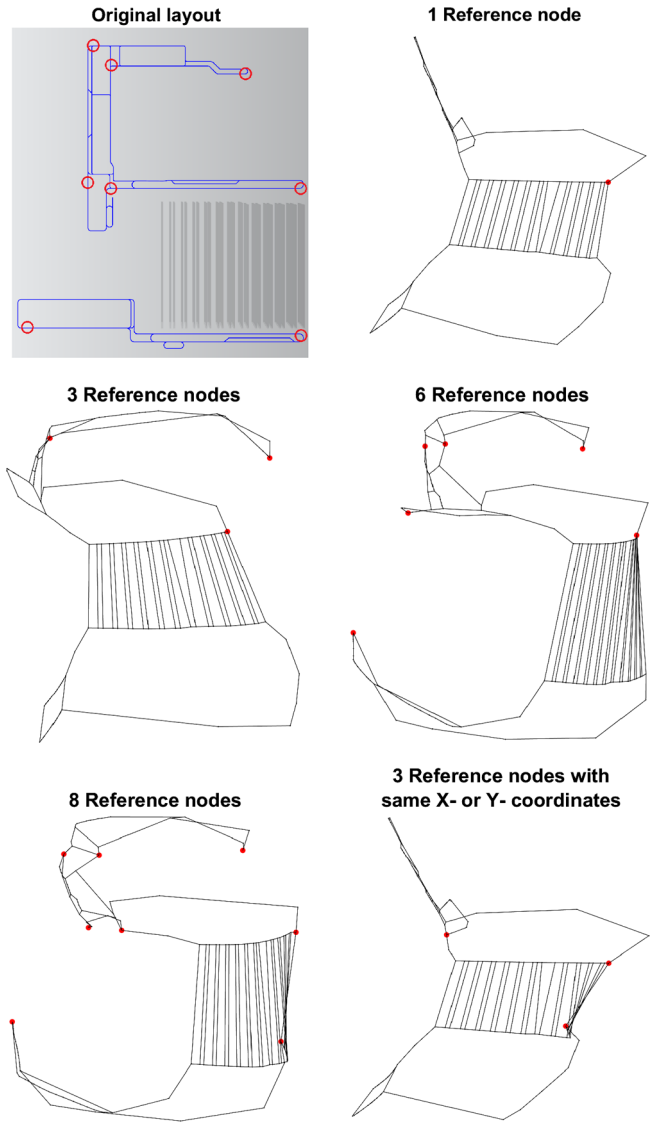


Figure 21 Comparison of initial layouts with different numbers of reference nodes

reference nodes have to be located in the rectangular part. To improve the graph, more reference nodes can be specified. If the optimization process has to consider numerous reference nodes, then the frame of the graph will become increasingly more restricted, and parts of the graph can deteriorate. A comparison of the optimal graphs with three and eight reference nodes shows that the lower part of OCS is visualized more accurately with three than with eight reference nodes. The optimal graph with eight reference nodes visualizes these nodes at the determined positions, but the entire graph seems to be confusing because of its significant limitations. Nevertheless, Algorithm 1 of stress majorization visualizes the physical structure of the original layout well with the help of reference nodes.

To verify our method and the coordinates of our reference nodes we use the Procrustes analysis as proposed in [5]. This analysis compares two matrices $X \in \mathbb{R}^{n\times a}$ and $Y \in \mathbb{R}^{n\times a}$ by determining linear transformations as translation, scaling, reflection and orthogonal

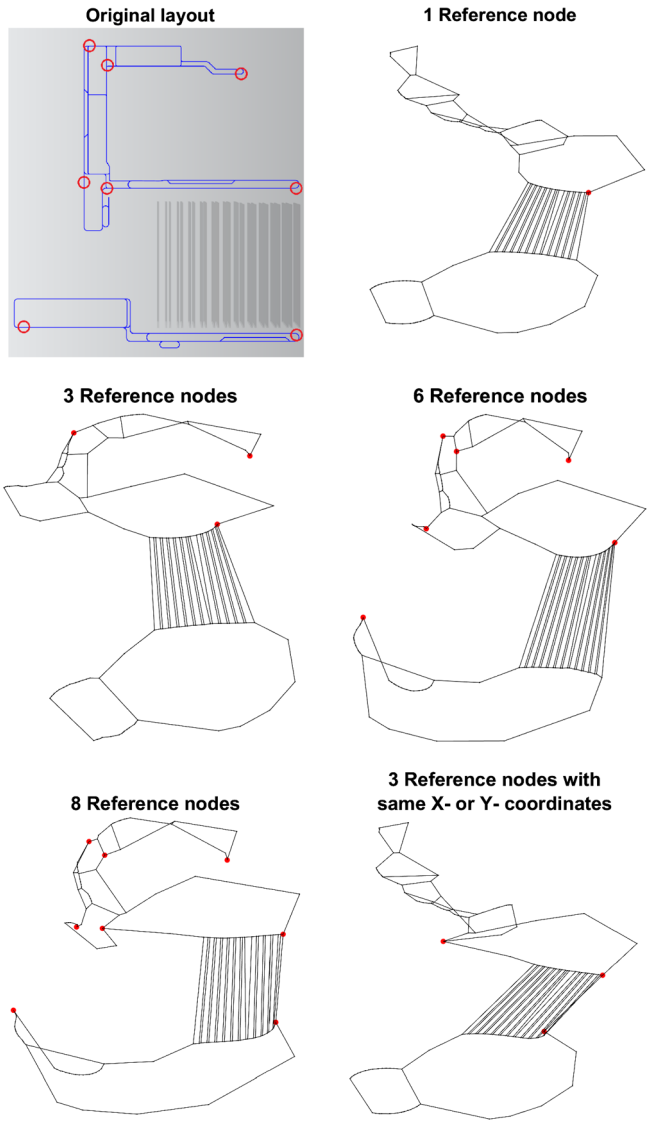


Figure 22 Comparison of the optimized graphs with different numbers of reference nodes

Table 3 Table of Procrustes analysis for the reference nodes

Number of reference nodes n	Procrustes statistic R^2
3	0.0632
6	$3.9746e^{-14}$
8	$4.3965e^{-14}$
3 (same X– or Y– coordinates)	$8.3711e^{-14}$

rotation of coordinates of matrix Y to best fit the coordinates of the matrix X. The aim of the Procrustes analysis is to minimize the sum of the squared errors of the matrix X and the transformed matrix Y^*.

In our case, we compare the coordinates of the reference nodes of the original OCS layout to our result, see Table 3. The Procrustes statistic R^2 is in all four cases very small.

This results verifies that our method is able to draw the structure of the reference nodes as given in the original layout.

4 Conclusions

In this paper, we applied classic multidimensional scaling and stress majorization to communication data of real-world conveyor systems to visualize the physical structure of such systems. Furthermore, we introduced two methods to improve the drawings of systems. We showed that in particular, the specification of reference nodes could help in computing good graphs similar to the original layout.

Future work might include the adaption of the stochastic gradient descent (SGD) method from [30] to the real-world problems we have presented in this work.

Acknowledgements
The publication of this article was funded by the Ministry of Science, Research and the Arts Baden-Württemberg and the University of Mannheim.

Funding
This work was financially supported by the DFG grant No. GO 1920/7-1.

Abbreviations
IT, information technology; NP, complexity class used to classify decision problems; COAST, convex optimization approach to stress-based embedding; CAD, computer-aided design; CMDS, classic multidimensional scaling; SOR, successive over-relaxation; OCS, overhead conveyor system

Availability of data and materials
Relevant data is cited if possible.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
SG and AS worked on the theoretical contribution and numerical simulation while SS and SS provided the problem description and test instances. All authors read and approved the final manuscript.

Author details
[1]Department of Mathematics, University of Mannheim, 68131, Mannheim, Germany. [2]SimPlan AG, 63452, Hanau, Germany.

Publisher's Note

Received: 10 March 2020 Accepted: 29 September 2020 Published online: 02 October 2020

References

1. Asarnow D, Singh R. The impact of structural diversity and parameterization on maps of the protein universe. BMC Proc. 2013;7:1–10.
2. Bergmann S, Strassburger S. Challenges for the automatic generation of simulation models for production systems. In: Proceedings of the 2010 summer computer simulation conference, SCSC'10. San Diego: Society for Computer Simulation International; 2010. p. 545–9.
3. Bleifuss R, Spieckermann S, Stauber S. A case study on simulation and emulation of a new case picking system for a us based wholesaler. In: Proceedings of the 2012 winter simulation conference (WSC). 2012. p. 1–12.
4. Borg I, Groenen PJ, Mair P. Applied multidimensional scaling and unfolding. 2nd ed. Cham: Springer; 2018.
5. Brandes U, Pich C. Eigensolver methods for progressive multidimensional scaling of large data. In: Kaufmann M, Wagner D, editors. Graph drawing. Berlin: Springer; 2007. p. 42–53.
6. Brandes U, Pich C. An experimental study on distance-based graph drawing. In: Tollis IG, Patrignani M, editors. Graph drawing. Berlin: Springer; 2009. p. 218–29.
7. Chen C. Information visualization: beyond the horizon. 2nd ed. London: Springer; 2006.
8. Davis TA, Hu Y. The university of Florida sparse matrix collection. ACM Trans Math Softw. 2011;38:1:1–1:25.
9. De Leeuw J. Convergence of the majorization method for multidimensional scaling. J Classif. 1988;5:163–80.
10. Di G. Battista, graph drawing: algorithms for the visualization of graphs, an Alan R. Apt book. Upper Saddle River: Prentice Hall; 1999.
11. Dwyer T, Koren Y, Marriott K. Stress majorization with orthogonal ordering constraints. In: Healy P, Nikolov NS, editors. Graph drawing. Berlin: Springer; 2005. p. 141–52.
12. Eades P, Wormald N. Fixed edge-length graph drawing is NP-hard. Discrete Appl Math. 1990;28:111–34.

13. Fáry I. On straight line representation of planar graphs. Acta Univ Szeged, Sect Sci Math. 1948;11:229–33.
14. Gansner ER, Hu Y, Krishnan S. COAST: A convex optimization approach to stress-based embedding. CoRR 2013. arXiv:1308.5218.
15. Gansner ER, Hu Y, North S. A maxent-stress model for graph layout. IEEE Trans Vis Comput Graph. 2013;19:927–40.
16. Gansner ER, Koren Y, North S. Graph drawing by stress majorization. In: Pach J, editor. Graph drawing. Berlin: Springer; 2004. p. 239–50.
17. Gellrich A, Wagner T, Vasyutynskyy V, Kabitzsch K. Modeling of transport times in partly observable factory logistic systems based on event logs. In: ETFA2011. 2011. p. 1–7.
18. Gotsman C, Koren Y. Distributed graph layout for sensor networks. In: Pach J, editor. Graph drawing. Berlin: Springer; 2005. p. 273–84.
19. Hu Y, Shi L. Visualizing large graphs. Wiley Interdiscip Rev: Comput Stat. 2015;7:115–36.
20. Kamada T, Kawai S. An algorithm for drawing general undirected graphs. Inf Process Lett. 1989;31:7–15.
21. Kaufmann M, Wagner D. Drawing graphs methods and models. Berlin: Springer; 2001.
22. Khoury M, Hu Y, Krishnan S, Scheidegger C. Drawing large graphs by low-rank stress majorization. vol. 31. Malden: The Eurographics Association and Blackwell Publishing Ltd.; 2012.
23. Knuth DE. Computer-drawn flowcharts. Commun ACM. 1963;6:555–63.
24. Meyerhenke H, Nöllenburg M, Schulz C. Drawing large graphs by multilevel maxent-stress optimization. In: Di Giacomo E, Lubiw A, editors. Graph drawing and network visualization. Cham: Springer; 2015. p. 30–43.
25. Nawaz S, Jha S. A graph drawing approach to sensor network localization. In: 2007 IEEE international conference on mobile adhoc and sensor systems. 2007. p. 1–12.
26. Ortmann M, Klimenta M, Brandes U. A sparse stress model. CoRR 2016. arXiv:1608.08909.
27. Pich C. Applications of multidimensional scaling to graph drawing. PhD thesis, Universität Konstanz, Konstanz, 2009. Available at http://nbn-resolving.de/urn:nbn:de:bsz:352-opus-83992, last checked 06.03.2020.
28. Tutte WT. How to draw a graph. Proc Lond Math Soc (3). 1963;13:743–67.
29. Wang Y, Wang Z. A fast successive over-relaxation algorithm for force-directed network graph drawing. Sci China Inf Sci. 2012;55:677–88.
30. Zheng JX, Pawar S, Goodman DFM. Graph drawing by stochastic gradient descent. CoRR. 2018. arXiv:1710.04626.