Using Network Representation as an Alternative to Multidimensional Scaling

Master's Thesis by Björn Menke

Date of Submission: 14th May 2014

Für meine Familie

Table of Contents

List of Figures

List of Tables

List of Abbreviations

CPU	Central Processing Unit
GA	Genetic Algorithm
MDS	Multidimensional Scaling
SMACOF	Scaling by Majorizing a Complicated Function, initially stood for "Scaling by Maximizing a Convex Function" (Borg and Groenen, 2005, p. 187)

1. Introduction

How does one visualise data that contains information about the similarity or dissimilarity of certain objects? A well established method to visualise such data – like correlations of product characteristics, frequencies of joint properties or ratings on the perceived similarity of political candidates – is multidimensional scaling (MDS). MDS tries to represent the data in geometric space. Typically, the 2-dimensional space is used (Borg et al., 2013, p. 6). The distance between objects in the geometric MDS representation shall express their respective similarity or dissimilarity. Objects which are relatively similar should be arranged closer to each other than less similar objects in a graphical representation.

MDS can be utilised for a variety of different purposes such as using it as explanatory technique for revealing clusters inside the data. MDS is applied in various areas such as economics, political science and psychology.

However, a representation of complex data in low dimensional geometric space through MDS usually leads to information loss. Büchel and Mastrangeli developed a novel approach to represent similarity or dissimilarity data[1] through distances between nodes in network space instead of geometric space (Büchel and Mastrangeli, 2013). The general idea of a network representation of (dis)similarity data in contrast to an MDS representation of the same data is illustrated in figure 1.1.

[1] Henceforth, similarity data and dissimilarity data are combined into one term and written as (dis)similarity data.

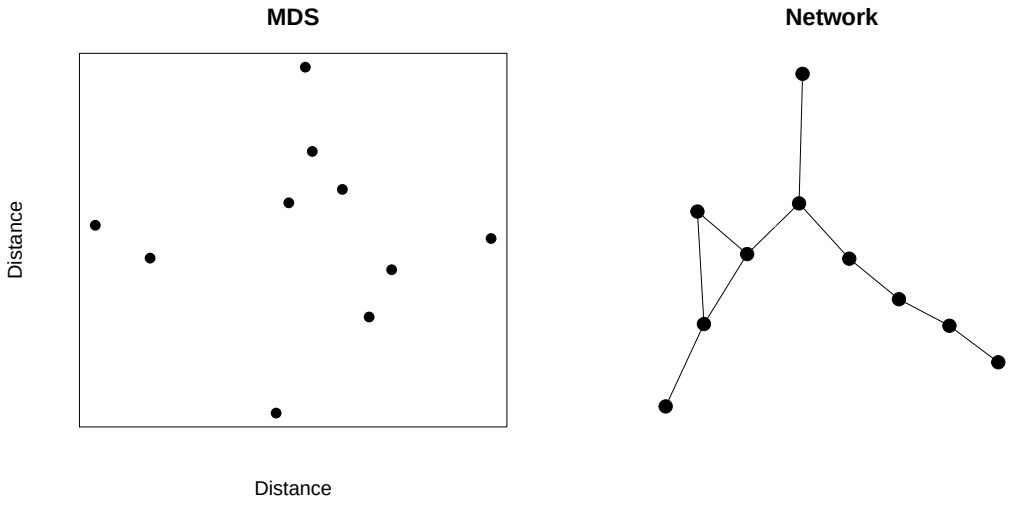

Figure 1.1.: *A visualisation of an MDS representation (left hand side) and a network representation (right hand side) of (dis)similarity data.*

As there has, to the knowledge of the author of this thesis, not been any publication of a closely similar idea in the academic research, this thesis aims to answer the following research questions:

1. How can the best possible network representation be found?
2. How can the quality of both MDS and network representations be measured and compared with each other?
3. When are network representations an alternative to MDS?

In order to answer these questions, a network generating algorithm was completely rewritten in R and significantly expanded for this thesis.[2] This extended algorithm is capable of finding the optimal network representation for given (dis)similarity data. The complete commented R source code can be found in section A.1 in the appendix.

Additionally, this thesis introduces and evaluates new measures to assess the information loss of both representation forms. These measures are also practically applied through the network generating algorithm on a large number

[2] The algorithm was originally developed by Büchel and Mastrangeli using MATLAB.

of different (dis)similarity data sets to answer the research questions. The focus of this thesis hereby lies on ordinal scaled data.

The thesis is structured as follows: Chapter 2 explains MDS. After a general introduction to MDS in section 2.1, the assessment of the quality of an MDS representation and its optimisation is discussed. Therefore, section 2.2 explains the fitness measure *Stress*. This is followed by section 2.3 in which the aforementioned information loss is discussed. The chapter concludes with an overview of important modifications of MDS as well as relevant alternatives in section 2.4.

Chapter 3 is the main part of this thesis. It shows how network representations, like the one on the right hand side of figure 1.1, are generated and how their information loss can be compared with MDS. The chapter starts by briefly introducing the idea of using network representation as an alternative to MDS by Büchel and Mastrangeli. In section 3.2, important basic concepts and terms of network analysis as well as specific requirements on the (dis)similarity data are explained. Thereafter, the network generating algorithm which is used to generate the best possible network representation is explained in detail and discussed in section 3.3. In order to generate such a network representation and to compare it with MDS, measures to assess the respective information loss in both representation forms are required. Fitness functions that aim to fulfil such requirements are discussed in section 3.4. Section 3.5 specifies the parameters for the network generating algorithm which are used for comparing both representation forms. The framework of the application and the results of the comparison are described and discussed in detail in section 3.6. Chapter 3 ends with critical remarks with respect to the concept of network representation as an alternative to MDS and to the chosen approach to compare both representations.

Chapter 4 summarises the main results as well as the achievements of this thesis and gives an outlook for future research.

2. Multidimensional Scaling

2.1. Basics of Multidimensional Scaling

Multidimensional scaling (MDS) is a collection of different methods with the common goal of representing similarity or dissimilarity data. In MDS, objects from the (dis)similarity data are realised as points within geometric space. (Dis)similarity data is any data that contains information about the similarity or dissimilarity of objects. Data from a questionnaire on the similarity of car brands is an example for (dis)similarity data. (Borg and Groenen, 2005, pp. 4)

MDS tries to arrange the objects from the (dis)similarity data in such a way that their similarity in the data is expressed by their distance in geometric space. A pair of two objects which are relatively similar should be placed closer to each other by MDS than any other two objects which are relatively less similar. With respect to the example of the similarity of car brands, the car brands Ferrari and Lamborghini might be represented closer to each other than the brands Ferrari and Fiat. (Borg and Staufenbiel, 2007, pp. 153; Cox and Cox, 2008, pp. 316)

MDS transforms pairwise information on objects' (dis)similarity into geometric distances. This is typically done by a computer.[3] In order to illustrate what MDS does, a 2-dimensional ordinal[4] MDS representation of the data on geographic distances of ten European cities from table 2.1 is illustrated in figure 2.1. This example emphasises that an MDS representation can be

[3] There are various MDS computer programmes available. See for instance chapter 9 in Borg et al. (2013) for an overview.

[4] The difference between metric and ordinal scaled data is explained later in this section.

very helpful for visualising information, especially in contrast to a matrix of (dis)similarity data as shown in table 2.1.

Figure 2.1.: *Ordinal MDS representation of the geographic distances of ten European cities.*

The larger the geographic distance between any two objects, the larger – ideally – the geometric distance in the MDS representation. In this example of city distances, the result is an almost correct, albeit rotated, map.[5]

Instead of using data on geographic distances many further areas for the application of MDS are possible. Any form of (dis)similarity data could be represented by MDS. Possible examples for such (dis)similarity data are similarity ratings for pairs of people, data on trade between nations or

[5] It is just "almost correct" as the ordinal MDS representation is not able to perfectly represent the real distances. This is partly due to use of ordinal and not metric MDS on that data. However, Borg and Groenen (2005) compare the metric and ordinal MDS representation of the same data that is used here with each other. Both of their representations "are very similar" (Borg and Groenen, 2005, p. 29).
Additionally, the distances were taken from a flat map of an atlas and thus do not account for the curved surface of a globe. Compare section 2.4 in Borg and Groenen (2005) for further information.

	Lo.	St.	Li.	Ma.	Pa.	Am.	Be.	Pr.	Ro.	Du.
London	0	569	667	530	141	140	357	396	569	190
Stockholm	569	0	1,212	1,043	617	446	325	423	787	648
Lisbon	667	1,212	0	201	596	768	923	882	714	714
Madrid	530	1,043	201	0	431	608	740	690	516	622
Paris	141	617	596	431	0	177	340	337	436	320
Amsterdam	140	446	768	608	177	0	218	272	519	302
Berlin	357	325	923	740	340	218	0	114	472	514
Prague	396	423	882	690	337	272	114	0	364	573
Rome	569	787	714	516	436	519	472	364	0	755
Dublin	190	648	714	622	320	302	514	573	755	0

Table 2.1.: *Geographic distances in kilometres of ten European cities in a symmetric matrix.*
Data taken from section 2.1 in Borg and Groenen (2005).

correlations of product characteristics. The aim of MDS would still be the same: MDS tries to represent relatively similar pairs of objects closer to each other in geometric space than pairs with relatively dissimilar objects.

MDS originates from the area of psychology, but nowadays it is common in various areas of science (Borg et al., 2013, pp. 18). It is for example used – among other areas – in the fields of archaeology, biochemistry, business administration, economics, geodesy, genetics, political science and psychology.[6]

The desired number of dimensions of the geometric representation can be defined in many MDS computer programmes. Any (dis)similarity data on n objects can be transferred to geometric space with $n - 1$ or more dimensions by MDS without information loss (Borg and Groenen, 2005, pp. 64). As the visualisation of information is one of the main purposes for applying MDS, it is often restricted to 1-, 2- or 3-dimensional space (Borg and Groenen, 2005, pp. 65; Borg et al., 2013, pp. 7 and 18). Typically, the 2-dimensional space – like in figure 2.1 – is used (Borg et al., 2013, p. 6).[7]

[6] See chapter 1 in de Leeuw (2001) or pp. 444 in Buja et al. (2008) for further details.
[7] The visualisation of high-dimensional (dis)similarity data through low dimensional MDS is called dimension reduction. See p. 448 in Buja et al. (2008) for further information.

There are different underlying methods for applying MDS on metric and ordinal scaled (dis)similarity data. Ordinal scaled data contains only information on the rank order of its objects. One might know from ordinal (dis)similarity data that that a is more similar to b than a to c. However, one cannot quantify by how much the pair a and b is more similar than the pair a and c. Metric scaled (dis)similarity data also contains information on the rank order like ordinal data. The difference between both is that metric scaled data contains additional information on the quantity of differences (Borg and Staufenbiel, 2007, p. 4; Wooldridge, 2013, p. 848).

The following example illustrates ordinal scaled data: Imagine the results sheet from an event like a marathon. If there are only the names of the runners and the ranking in which they finished the marathon on that list, one only has information about the rank order of the runners. From this list one would know who is the fastest runner, the second fastest and so on. However, one would not know by how much time the winning runner was ahead of the second one.

Metric scaled data can be described with the following scenario: If the list is expanded by adding the time of each runner, one would be able to quantify the differences between the times of all runners.

MDS can be applied to both ordinal and metric scaled (dis)similarity data. However, ordinal MDS is more popular than metric MDS in research publications (Borg et al., 2013, p. 37). One possible explanation for this is that there simply might be more ordinal data sets in areas where MDS is frequently applied. Additionally, ordinal MDS can also be applied to metric data to limit possible biases due to single extreme values. The focus of this thesis lies on the more popular representation of ordinal (dis)similarity data.

There are various different MDS methods and algorithms to find the best representation of (dis)similarity data in geometric space. Besides the difference between metric and ordinal of the (dis)similarity input data, one distinguishes between Kruskal-Shepard distance scaling, also called distance MDS, and Torgerson–Gower inner-product scaling, also called classical MDS. Distance MDS is an iterative process that tries to minimise a certain error measure. In

contrast, classical MDS leads to an analytical solution in which no iterations are needed.[8] The most common one of these two is distance MDS, especially for ordinal (dis)similarity data (Borg and Groenen, 2005, pp. 37 and pp. 261; Buja et al., 2008, p. 447 and pp. 455).

As already explained, this thesis focuses on ordinal (dis)similarity data. This chapter on MDS therefore focuses on the most common MDS variant: ordinal distance MDS. There are different methods for measuring the fitness or quality of a distance MDS representation. The most common measure of the fitness of an ordinal distance MDS representation is the Stress loss function which is explained in the next section (Borg and Groenen, 2005, p. 37).[9]

2.2. Stress and its Optimisation

Stress measures the differences between the original (dis)similarity data and its geometric MDS representation. As stated above, this thesis focuses on the ordinal distance MDS case. The mathematical framework for that case was developed by Kruskal (1964) (de Leeuw, 2001, p. 13516).

The central aim of Kruskal (1964) is to obtain "a monotone relationship between dissimilarity and distance". That means that any order structure of ordinal data from the (dis)similarity data has to be represented by the distances in MDS as good as possible. Kruskal introduces a quantitative measure for violations of the monotone relationship. This measure is called Stress. (Kruskal, 1964, pp. 2)

Kruskal (1964) suggests the optimisation of Stress with steepest descent algorithms. Since Kruskal's article was published in 1964, newer methods for finding the best representation in geometric space have been developed. Most notably, the Stress majorisation by the *Scaling by Majorizing a Complicated*

[8] There are also some classical scaling methods that use iterations, for example in combination with weighted (dis)similarities (Buja et al., 2008, p. 455).

[9] Stress can also be used to assess the fitness of a metric distance MDS representation. However, as metric MDS is not the subject of this thesis, metric MDS is not explained further.

Function (SMACOF) algorithm which initially goes back to de Leeuw (1977), is nowadays widely used for the MDS generation and is often implemented in modern statistical software (Borg and Groenen, 2005, p. 187).[10] However, Stress is still the most common measure for the quality of a distance MDS representation, and it is still used by modern algorithms like SMACOF (Borg and Groenen, 2005, pp. 39 and 84).

As already clarified, Stress measures the fitness between the original (dis)similarity data and their MDS representation. In the MDS literature, the data on the pairwise similarity of the objects from the (dis)similarity data is simply called dissimilarities.[11] The dissimilarity between the pair of two objects i and j is expressed by δ_{ij}. In most cases, the MDS literature speaks of dissimilarities instead of similarities. That means that a higher value for a pair of two objects in the (dis)similarity data indicates a higher dissimilarity. This is the same as in the example of the European cities in table 2.1 on p. 6. A high number in that table represents a higher geographic distance. To meet this requirement of using dissimilarities instead of similarities, the similarities can be transformed into dissimilarities. In the ordinal case, any similarity can be transformed into a dissimilarity and vice versa without losing any information. (Borg et al., 2013, pp. 21; de Leeuw, 2001, p. 13513)

The geometric distance in an MDS representation between the pair of the two objects i and j is expressed by distance d_{ij}. The distance depends on the space that is used. In Euclidean space for example, d_{ij} can represent the Euclidean distance between the objects i and j.

The dissimilarities contain information on all pairs of any two objects from the original (dis)similarity data. If there was information on n different objects

[10] De Leeuw is for example one of the developers of a SMACOF package for the software R. Compare de Leeuw and Mair (2009) or chapter 7 in Borg et al. (2013) for further information.

[11] The term proximities is also common for the (dis)similarities from the original data in the MDS literature. Borg and Groenen (2005), for example, use the term proximities instead of dissimilarities (Borg and Groenen, 2005, p. 37). In order to prevent confusion and to be in line with Kruskal and de Leeuw, the term dissimilarities will be consistently used in this thesis. Further information on the term proximities can also be found in section 3.1 in Cox and Cox (2008).

in the (dis)similarity data, the pairwise dissimilarities could be represented by a symmetric $n \times n$ matrix – like the one for the city distances in table 2.1. Symmetry ($\delta_{ij} = \delta_{ji}$) is always required for MDS (Borg and Groenen, 2005, pp. 33). The case of self-similarity ($\delta_{ii} \neq 0$) is not covered in this thesis.[12] It is therefore assumed that ($\delta_{ii} = 0$).

Due to the assumption that there is no self-similarity, the $n \times n$ matrix of the (dis)similarity data would contain information on $n\,(n-1)$ pairs of objects – plus a main diagonal of zeros. As symmetry is also required for MDS, the total number of pairs with relevant dissimilarity information is $\frac{n}{2}\,(n-1)$.

The MDS representation of n objects would contain distance information on exactly the same number of relevant pairs, as all objects from the (dis)similarity data are represented in geometric space by MDS. The distances in MDS can therefore also be displayed in a symmetric $n \times n$ matrix with a main diagonal full of zeros.

In order to obtain the Stress of an MDS representation, a monotone least-squares regression, also called isotonic regression, of the distances from the MDS representation on the dissimilarities from the original (dis)similarity data is applied. Before the monotone regression starts, all pairs must be sorted and ranked according to their dissimilarity (δ_{ij}).[13] All distance pairs (d_{ij}) are also ordered according to their rank within the ordered dissimilarities. (Cox and Cox, 2008, pp. 323; Leeuw et al., 2009, pp. 2)

The result of the ranking can be visualised in a graph with dissimilarities on the horizontal and distances on the vertical axis. Such a graph is called a Shepard graph (Borg and Groenen, 2005, pp. 42; Buja et al., 2008, pp. 463). The pair ij of the two objects i and j is represented in a Shepard graph at the geometric point $(d_{ij}; \delta_{ij})$. The monotone regression generates a monotonically

[12] In some cases, self-similarity can be represented by MDS. Compare p. 34 in Borg and Groenen (2005).

[13] This can be in ascending or descending form. In this thesis, the ranking and sorting will uniformly be done in ascending form. That means that all dissimilarities pairs (δ_{ij}) are ordered in an ascending way, starting with the pair of two objects with the smallest dissimilarity – or highest similarity.

increasing[14] fit through these points. This step function aims to minimise the squared residuals. (Cox and Cox, 2008, pp. 323; Leeuw et al., 2009, pp. 2)

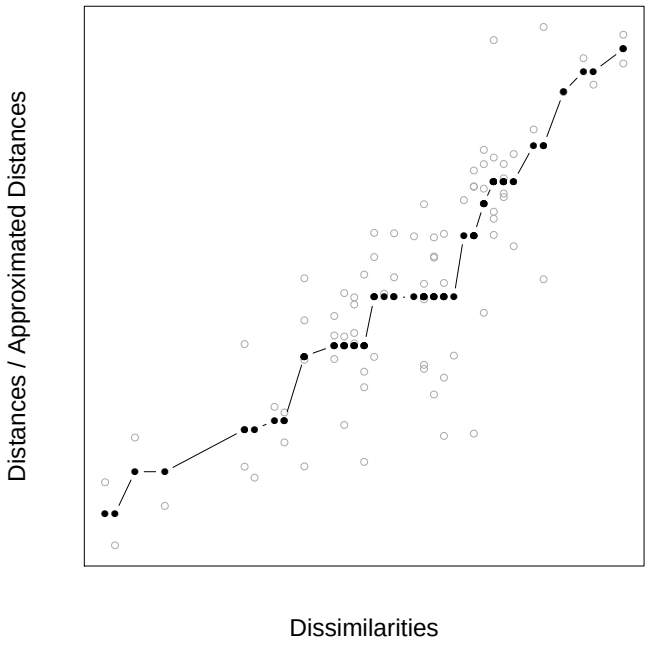

Figure 2.2.: *Shepard graph of an ordinal distance MDS representation of data on 13 work values from West German workers.*
The underlying data is taken from table 7.2 in Borg et al. (2013) and was originally assessed by ALLBUS (1991).

Figure 2.2 shows an example of a Shepard graph of an MDS representation. The distances from the MDS representation, expressed by the open circles, are plotted against the respective dissimilarities. That means that the open circles represent the mentioned points $(d_{ij}; \delta_{ij})$ for all pairs. The black line represents the monotone regression of the distances from the MDS representation on the dissimilarities. This monotonically increasing line shows the approximated distances $\left(\hat{d}_{ij}\right)$ from the monotone regression with respect to the dissimilarities. The filled circles on that line therefore represent the points $\left(\hat{d}_{ij}; \delta_{ij}\right)$.

If the MDS representation was able to present the ordinal (dis)similarity data perfectly, all open circles in the Shepard graph would be exactly on the

[14] In the case of a sorting by descending rank order, the monotone regression would be monotonically descending.

monotonically increasing regression line. Any open circle above the ascending regression line in the Shepard graph represents a point whose distance in the MDS representation is – with respect to its ordinal rank – larger than its dissimilarity in the original (dis)similarity data. The opposite is true for points below the regression line.

In order to quantify the fitness of an MDS representation, the vertical[15] distance between the distance (d_{ij}) from the MDS representation and the approximated distance $\left(\hat{d}_{ij}\right)$ from the monotone regression can be computed. This is the residual. The squared residual of the pair ij is therefore $\left(d_{ij} - \hat{d}_{ij}\right)^2$. (Borg and Groenen, 2005, pp. 43)

The sum of all squared residuals, which the monotone regressions aim to minimise, is shown in the following equation 2.1. This is known as raw Stress (σ_r) (Kruskal, 1964, p. 8).

$$\text{Raw Stress} = \sigma_r = \sum_{i<j} \left(d_{ij} - \hat{d}_{ij}\right)^2 \tag{2.1}$$

The raw Stress is scale-dependent. Any change in the scale of the MDS representation leads to a change in the raw Stress value. To get rid of this undesired effect, the raw Stress can be normalised. This can be done by dividing the raw Stress by the sum of all squared distances from the MDS representation. Kruskal (1964) also suggests taking the square root of that expression. This normalised version of the raw Stress is called Stress-1 and is shown in equation 2.2. (Borg and Groenen, 2005, pp. 43).

$$\text{Stress-1} = \sigma_1 = \sqrt{\frac{\sigma_r}{\sum_{i<j} \left(d_{ij}\right)^2}} = \sqrt{\frac{\sum_{i<j} \left(d_{ij} - \hat{d}_{ij}\right)^2}{\sum_{i<j} \left(d_{ij}\right)^2}} \tag{2.2}$$

In general, a smaller Stress-1 value indicates a better fit.[16] A perfect fit of the

[15] "Vertical" in the case of the described Shepard graph with dissimilarities on the horizontal and distances on the vertical axis.

[16] There are cases in MDS where a smaller Stress-1 value does not indicate a better fit.

ordinal dissimilarities by the distance MDS representation would lead to a raw Stress value of zero and hence to a Stress-1 of zero. The highest possible Stress-1 value is one. (Kruskal, 1964, p. 9)

Stress-1 is used in many MDS algorithms to judge the quality of an MDS representation (Borg and Groenen, 2005, p. 37). Iterative MDS algorithms change the position of the objects in the geometric space – and therefore the respective distances of the respective objects in the MDS representation – to minimise and thereby optimise the Stress-1.

With respect to both Stress of MDS representations as well as fitness measures of network representations, the handling of multiple pairs of objects with the same distance is important. These are called or ties.

Let there be two pairs of objects with the same distance in the (dis)similarity data: $\delta_{ij} = \delta_{kl}$. There are two approaches to handle such ties in MDS. The so-called secondary approach requires that pairs with identical dissimilarities must also have an identical distance in the MDS representation. That means that $\delta_{ij} = \delta_{kl}$ requires $d_{ij} = d_{kl}$. The primary approach does not impose this requirement. In that case, the tied objects can be placed without this restriction on the distance of the tied pairs in order to minimise the Stress-1. If the (dis)similarity data contains ties, Stress-1 is often lower when using the primary instead of the secondary approach (Borg and Groenen, 2005, pp. 40, 49 and 211).

In addition, results from MDS and the underlying Stress calculation might vary with respect to the chosen requirements on the strength of the relationship between dissimilarities and their MDS representation. A weak monotone function requires $d_{ij} \geq d_{kl}$ if $\delta_{ij} > \delta_{kl}$. In contrast, a strong monotone function requires $d_{ij} > d_{kl}$ if $\delta_{ij} > \delta_{kl}$. As a strong monotone function imposes higher requirements on the MDS representation, its application typically results in a higher Stress-1 value, like the secondary approach.

For further information, see section 13.1 in Borg and Groenen (2005) about degenerate solutions in ordinal MDS.

In most cases, a weak monotone function in combination with the primary approach is used with ordinal MDS (Borg and Groenen, 2005, p. 40). Therefore, this thesis also focuses on that case – unless stated otherwise.

2.3. Information Loss in Multidimensional Scaling

In general, the Stress-1 value is non-zero. In other words: The MDS representation is unable to visualise the given (dis)similarity data without generating errors in the ordinal ranking order in geometric space. There are two main reasons for this.

First, there might already be contradictions within the (dis)similarity data. Imagine for example the case of three objects A, B and C in the (dis)similarity data. Let the following dissimilarity relations exist within this (dis)similarity data: $\delta_{A,B} > \delta_{A,C}$, $\delta_{A,B} < \delta_{B,C}$ and $\delta_{A,C} > \delta_{B,C}$. This would lead to the order $\delta_{B,C} > \delta_{A,B} > \delta_{A,C} > \delta_{B,C}$, which clearly shows a contradiction. An MDS representation of such (dis)similarity data would also not be able to solve this contradiction. Any such contradiction in the (dis)similarity data would therefore raise the Stress-1 value. However, it is debatable whether this case should – beside its role in increasing the Stress-1 value – be counted as information loss due to MDS.

Such contradiction within the (dis)similarity data might for example exist due to measurement errors. Closely related to this issue is the large number of possible order violations which is exponentially increasing with the number of objects in the (dis)similarity data.[17]

The other main source of possible errors in the MDS representation is information loss through MDS itself. As stated in section 2.1, any (dis)similarity data on n objects can be perfectly represented with $n - 1$ or more dimensions in geometric space (Borg and Groenen, 2005, pp. 64). However, typically the

[17] The number of possible order violations is also an important topic in section 3.4.2. Equation 3.6 shows the formula to compute the number of possible order violations with respect to the number of objects.

2-dimensional space is used (Borg and Groenen, 2005, pp. 65; Borg et al., 2013, pp. 6 and 18).

Such low dimensional geometric space can be the source of information loss through the application of an MDS representation. Imagine the case of four objects that all have the same dissimilarity to each other within the (dis)similarity data. It is not possible to arrange more than three objects with the same Euclidean distance to each other in a 2-dimensional geometric space.[18] Therefore in cases of a high probability of objects with the same dissimilarities to each other, the application of MDS might theoretically lead to information loss. This theory is practically tested in section 3.6 in chapter 3.

2.4. Modifications of Multidimensional Scaling and Alternatives

There are various modifications of MDS and possible alternatives for visualising (dis)similarity data. The aim of this section is to give a brief overview about them.

The previous sections describe Stress and distance MDS and the related possible information loss in Euclidean space. Instead of using Euclidean space in MDS, the use of an alternative distance metric is also possible. For example, Minkowski distances and the related city-block[19] distances can be used alternatively. However, the interpretation of an MDS representation within Minkowski space might be not as intuitive as the interpretation of an MDS representation of Euclidean distances within geometric space. As the visualisation of information is one of the main purposes of MDS, this modification of MDS is therefore not further discussed in this thesis.[20]

[18] The same holds for more than two objects in 1-dimensional and more than four objects in 3-dimensional geometric space with Euclidean distances.

[19] Also known as Manhattan distances (Groenen et al., 1995, p. 5).

[20] Detailed information on using MDS within Minkowski space can be found in Arabie (1991), section 17.2 in Borg and Groenen (2005), Groenen et al. (1995) and Groenen et al. (1999).

An alternative for the representation of (dis)similarity data might be hierarchical clustering. Hierarchical clustering helps to detect and visualise clusters within data – a purpose MDS is also being used for (Borg and Groenen, 2005, pp. 5). The graphical representation of hierarchical clusters is typically done in a dendrogram. A dendrogram is a tree-like structure, which looks similar to a tree graph in social network analysis (Jackson, 2010, pp. 27).

With each additional lower level, a dendrogram shows a more detailed clustering than the previous one. That means that many of the levels may include groups of objects and not just single objects. Dendrograms are therefore very different from the network representation that is presented in this thesis.[21]

Dendrograms are used in various areas. The graphical representation of a phylogenetic tree, which is for example used in various fields of biology to visualise evolutionary relationships, is usually done in a dendrogram. (McLachlan et al., 2004, pp. 64; Fitch and Margoliash, 1967, pp. 279)

Network graphs are also used in the academic literature to visualise (dis)similarity data. Associative networks were previously primarily used in psychology (Henderson et al., 1998, p. 307). Henderson et al. (1998) as well as Teichert and Schöntag (2010) use associative networks in the field of marketing. Associative networks can for example visualise data on brands as a network graph. However, the underlying generation process and thus also the resulting network is completely different to the network approach presented in this thesis.

Büchel and Mastrangeli (2013) developed a novel approach to represent (dis)similarity data within a network graph as an alternative to MDS. Their idea and their approach are explained and expanded in the following chapter.

[21] Further information on hierarchical clustering and the underlying mathematical processes can be found in Breiger et al. (1975) and Shepard (1980, pp. 390).

3. Network Representation as an Alternative to Multidimensional Scaling

3.1. Büchel and Mastrangeli's Network Representation

A novel approach to use network representation as an alternative to visualise (dis)similarity data has been developed by Büchel and Mastrangeli since 2009 (Büchel and Mastrangeli, 2013).[22]

With regard to section 2.3, MDS can lead to information loss. The idea of Büchel and Mastrangeli is to transfer the (dis)similarity data into the best possible undirected and unweighted network representation.[23] The dissimilarity of any two objects is then represented by the length of the shortest path between both objects within the network.

In order to illustrate that idea, figure 3.1 shows the best network representation of the data on the geographic distances of ten European cities. The (dis)similarity data that is represented in that figure is shown in table 2.1 on p. 6. According to this network graph, one could assume that the distance between Dublin and Amsterdam is larger than the distance between Dublin and London or that Stockholm and Lisbon seem to be relatively far away from

[22] Besides serving as an introduction to the topic, this section enables the reader of this thesis to distinguish between Büchel and Mastrangeli's work and the contribution of this thesis to this field of research.

[23] The terms undirected and unweighted are explained in section 3.2.

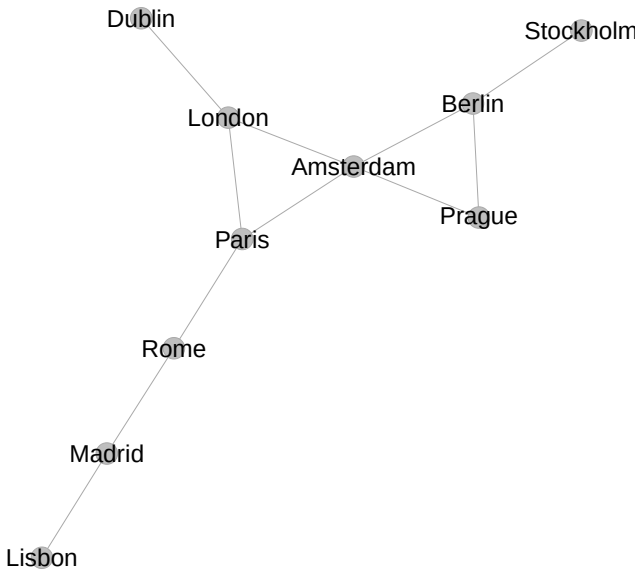

Figure 3.1.: *Network representation of the geographic distances of ten European cities.*

each other with respect to the other cities. However, this network graph contains no information on the quality or fitness of this representation. At this point, it is unclear, whether such a network graph is a good representation of these geographic distances and whether one should prefer the MDS representation of the same data in figure 2.1 on p. 5. An intensive practical comparison of MDS and network representations is conducted later in section 3.6.

Büchel and Mastrangeli wrote an algorithm for the generation of the optimal network representation of given dissimilarity[24] data in the computer programme MATLAB. Their algorithm is able to iteratively find the optimal network representation by optimising one of two different fitness measures. Their algorithm is henceforth called Büchel and Mastrangeli's algorithm, while the

[24] Their algorithm relies on dissimilarity data. Any similarity data has to be transformed into dissimilarity data before their algorithm can be applied.

algorithm, which is mainly presented in this thesis, is called network generating algorithm.[25]

The first of these two fitness measures is to count the number of violations in the order relations between the dissimilarities from the (dis)similarity data and their network representation. That means that if the order relation between any two pairs of two objects is different between the (dis)similarity data and their network representation, an order violation occurs. This fitness measure is also used in the network generating algorithm presented in this thesis and therefore explained in detail in section 3.4.2.

The second fitness measure uses a scaling of the resulting network distances to compare them with the dissimilarities from the (dis)similarity data. This measure has some similarities to the Stress-1 calculation within MDS. However, as the scaling value has to be set manually, this measure seems to be inappropriate to judge the quality of a network representation, especially in the case of ordinal (dis)similarity data.

To generate the fitness optimising network representation, Büchel and Mastrangeli apply an iterative genetic algorithm (GA). Their GA optimises one of the two described fitness measures or functions to find the best network within a population of several candidates.[26] The starting population features special network types, such as a complete, empty, line, and tree network. The rest of the population is filled with randomly generated networks. Any two objects within these random networks can form a link between each other with a probability of 0.5. After each iteration the best networks are kept and some of their characteristics are passed on to other networks in the following iteration. In addition, a mutation that deletes or adds a single link can occur in some networks with a certain probability. After a given amount of time, the algorithm stops and shows the adjacency matrix of the best network representation and returns the fitness value of that network.

[25] The R source code of the network generating algorithm is shown in section A.1 in the appendix.

[26] The choice of fitness functions to be used by the GA must be made by the user.

Büchel and Mastrangeli also implemented the possibility of adding dummy nodes in the network representation in order to improve the fitness. Dummy nodes are explained in section 3.3.4 on p. 31.

3.2. Basics of Networks and Requirements on (Dis)similarity Data

A network graph consists of a finite set of n nodes.[27] With respect to the application on (dis)similarity data in this thesis, each node represents one object from the (dis)similarity data.

Let g be an $n \times n$ adjacency matrix of the set of nodes. Within the adjacency matrix, g_{ij} describes the relationship of the nodes i and j. In a weighted network, the strength of the relationship can be expressed by a set of non-negative numbers. In an unweighted network, a relationship between two nodes is either present or not ($g_{ij} \in \{0, 1\}$). The network of the city distances in figure 3.1 on p. 18 is for example an unweighted network as there nodes are either connected or not. (Jackson, 2011, p. 522)

Any (dis)similarity data could trivially be represented without information loss in a weighted network graph. However, such a weighted network graph would not enable the user to quickly see the inner structure of a given (dis)similarity data set, as almost all nodes would be directly connected. An unweighted network representation is more useful than a weighted network in this case. Therefore, the network representation discussed in this thesis focuses on the representation through unweighted networks.

In keeping with the requirements on the (dis)similarity data in MDS, symmetry is also required for the network to ensure a fair comparison of both forms of representation. That means $g_{ij} = g_{ji}$ for any node i and j. Hence, the adjacency matrix is symmetric as well. Such a network is called an undirected network (Jackson, 2011, p. 522). Additionally, in keeping with the requirements on MDS,

[27] Nodes are also called vertices (Jackson, 2011, p. 522).

there should be no self-similarity. That implies that the main diagonal of the adjacency matrix consists of zeros.

A certain relationship between any two nodes within an undirected and unweighted network is visualised in a network graph by a link that connects both nodes.[28] The distance within a network representation can be expressed by the length of the shortest path between any two nodes. This length is the number of links on a shortest path between any two objects i and j. The distance on the shortest path is also called a geodesic distance. A matrix that contains the geodesic distances for all pairs of two nodes in a network is called a distance matrix. (Jackson, 2011, pp. 522)

Depending on the fitness function, one possible problem can emerge when there is no connection between nodes.[29] The geodesic distance for this case is subject to definition. Such a distance could for example be treated as infinite or as another number that is larger than or equal to the largest of all the other geodesic distances in that network. (Jackson, 2006, pp. 4 and Jackson, 2010, pp. 32)

3.3. Description of the Network Generating Algorithm

The following sections explain the network generating algorithm which is used in this thesis in order to find the best network to represent given (dis)similarity data.

Section 3.3.1 introduces the abilities of the network generating algorithm and describes the differences between the network generating algorithm and Büchel and Mastrangeli's algorithm. To understand the network generating algorithm in detail, section 3.3.2 explains the application of a GA with respect to the generation of networks. A GA is used within the network generating algorithm in order to find the optimal network representation. The procedure of how the

[28] Links are also called edges in the network literature (Jackson, 2011, p. 522).

[29] A network with nodes that are neither directly nor indirectly connected is called an unconnected network.

network generating algorithm finds the best possible network representation is described in detail section 3.3.3. Finally, section 3.3.4 explains the concept of dummy nodes.

3.3.1. General Properties of the Network Generating Algorithm

The network generating algorithm is able to find the best network representation of given (dis)similarity data within a given time. Additionally, it generates the MDS solution for the same input data and graphically visualises both representations. The quality of both representation forms in accordance with certain fitness measures is also compared with each other. As stated earlier, the complete and commented source code can be found in the appendix A.1 on p. 56.

The algorithm offers options for both metric and ordinal scaled (dis)similarity data. However, as the focus of this thesis lies on the representation of ordinal scaled data, the approach to find the best network representation of metric scaled data is not discussed in detail in this thesis.[30]

The network generating algorithm is an extension of the algorithm by Büchel and Mastrangeli. It adds many new features, such as the calculation of Stress-1 fitness values for both representation forms, the ability to generate an MDS representation for given (dis)similarity data, the graphical visualisation of both representations without the need for additional programmes as well as the ability to also use similarity data – and not only dissimilarity data.

It also modifies and expands many existing functions of Büchel and Mastrangeli's algorithm. The generation of the initial population for example is far more advanced in the network generating algorithm as it improves the general quality of the initial population by the incorporation of heuristic network proposals and improved random networks. The quality of the mutation during

[30] The algorithm optimises a different fitness function for metric scaled (dis)similarity data to find the best network representation. The function can be seen in the source code in the appendix. This might be the subject of further research on this topic.

the iteration process of the GA is also improved through a modified mutation function.

All listed functions are explained and described in the following sections. Furthermore, the network generating algorithm was completely rewritten in R, using only fragments from the MATLAB code of Büchel and Mastrangeli's algorithm.[31] This enables anyone to use the source code in the appendix to reproduce the results of this thesis and to apply the network generating algorithm on any (dis)similarity data.[32]

3.3.2. Using a Genetic Algorithm with Respect to Networks

The algorithm developed by Büchel and Mastrangeli as well as the network generating algorithm use a GA to find the global optimum of a specified fitness function and thus the best network representation of given (dis)similarity data. A GA is used due to its capability at finding the global optimum among local optima, especially in combination with non-smooth fitness functions (Scrucca, 2013, p. 35). It is available in R through an add-on by Scrucca (2013).[33]

GAs belong to the family of evolutionary algorithms (Weise, 2009, pp. 100). A GA generates a population of possible solution candidates. In the network case, each individual in the population of the GA is a network representation. Each network representation is represented in the population by a vector with the length of the number of unique pairs of two nodes. Each element in such a vector represents the relationship between a unique pair of two nodes. An element of such a vector equals either one if there is a link between this unique

[31] R is a free and open software system (Venables and Ripley, 2002, p. 1). R is a modular software. It can be expanded by installing modules or add-on packages to offer additional functions and features (Hornik, 2014, p. 3 and 21). The source code of the network generating algorithm also uses add-ons to generate the best network representation of given (dis)similarity data. A list of the add-ons that are used can be seen in the source code in the appendix. The source code also contains information on the purpose of the respective add-on for the algorithm.

[32] Please contact the author of this thesis for a digital copy of the source code of the R script.

[33] The name of this add-on is *GA*.

pair of two nodes in the network, or zero otherwise.[34] The population size, which is the number of networks in the population, can be set.

A GA searches iteratively for the best network representation in the population. At each iteration, a GA calculates the fitness of each of the networks in the population according to a specified fitness function. In order to simplify the description of the network generating algorithm and the GA, fitness functions are discussed separately in section 3.4.

A GA tries to mimic natural selection. The better networks in the population dominate weaker ones. Only the fittest networks survive an iteration and persist. New offspring or in this case new networks are generated by mixing the characteristics of two networks from the previous iteration. This is called crossover. (Scrucca, 2013, p. 3)

Similar to processes in nature there is also a chance for mutation. Mutation changes characteristics of randomly chosen networks from the population (Scrucca, 2013, p. 3). This means that mutation randomly adds or deletes a link in randomly chosen networks from the population.

By default, the mutation function in the R add-on by Scrucca (2013) only changes one link. As the number of possible links increases exponentially with the number of objects in a network, the mutation of just one link might not be appropriate to find the global optimum in the presence of a relatively high number of objects in a network within a given time.[35]

In order to increase the power of the mutation within the network generating algorithm, one should consider using a modified mutation function within the GA. This modified mutation allows a variation in the number of characteristics that will be mutated network. Instead of changing the relationship of one pair of two nodes, more links can be deleted or added in the same network through the mutation. The exact number of links added or deleted is set by a random function. The number of links that might be mutated ranges from one to five.

[34] Therefore, a GA of binary type can be used in this case (Scrucca, 2013, p. 5).

[35] The calculation of the number of possible links is shown later in equation 3.6 in section 3.4.2.

Additionally, a certain share of all possible links, for example a share 0.01 or 0.02, might be mutated. However, in most cases just one link is added or deleted – like in the standard mutation function. Using this modified mutation increases the variation in the population. Hence, it might help to find the global optimum instead of local ones, if the network generating algorithm is run adequately long enough.

The GA also offers a function to ensure that the best networks of an iteration survive untouched. This is called elitism. (Scrucca, 2013, p. 3)

The GA repeats the procedure of keeping the best solutions while updating the rest of the population by crossovers and mutations for a set number of iterations. If the maximum number of iterations is reached, the solution with the highest fitness will be returned.

3.3.3. Procedure of the Network Generating Algorithm

The network generating algorithm offers important options and switches. It has to be defined, whether the input data consists of similarities or dissimilarities. This is done by the switch *higher.value.means.higher.similarity*. Additionally, it has to be set whether the input data is metric or ordinal scaled data via the option *scale.of.input.data*. The fitness function varies for metric and ordinal data. As stated earlier, this thesis focuses on the case of ordinal (dis)similarity data.

Compared to the algorithm by Büchel and Mastrangeli the network generating algorithm also offers an option to generate dummy nodes in order to improve the fitness of the network representation. The number of dummy nodes that shall be included can be set by the option *dummy.nodes*. Dummy nodes are explained in section 3.3.4.

In order to reduce the computation time that is required to find the best network representation, the switch *ga.use.parallel* allows the GA to use multiple threads

or cores of the central processing unit (CPU). This feature works only on supported operating systems and hardware.[36]

Furthermore, the network generating algorithm offers options for the GA it uses. The maximum number of iterations that the GA within the network generating algorithm computes can be set by the option *ga.maxiter*. This can be overridden if there has not been any improvement in the fitness value of the best solution for a set number of iterations. This additional iteration limit can be set by the option *ga.run*.

As described in the previous section, the size of the GA's population can be defined by the option *ga.minimum.popSize*. The variable is called "minimum", as the population size is either *ga.minimum.popSize* or the number of dimensions of the symmetric dissimilarity input matrix plus the number of included dummy nodes, if the latter is larger.

The initial population of the GA can either consist of randomly generated networks or feature a combination of random networks, heuristic network proposals as well as special network types, like the empty or complete network. This can be set by using the *ga.use.suggestions.matrix*.

As described in the previous section, the occurrence of crossovers and mutations can be set by the options *ga.pcrossover* and *ga.pmutation*. The former option sets the probability for a crossover of two networks, whereas the latter sets the probability for a mutation in a network. (Scrucca, 2013, p. 6)

As also described in the previous section, the network generating algorithm offers the option to use a modified mutation function. By default, only one possible link is mutated. Enabling *ga.use.modified.mutation* allows a mutation of more than just one possible link in a network. Such modified mutation increases the effect of the mutation and thus increases the variation, which might help to find the global optimum network representation quicker within a given time.

[36] Further information can be found in the manual of the respective R add-on packages. If the user of the network generating algorithm is unsure whether his computer system supports this feature, the user should enable the *ga.use.parallel* switch, unless errors are encountered. This is recommended because the script will only load the required add-ons, if multiple CPU threads or cores are detected properly.

The amount of best fitness networks of the population that survive at each iteration can be set as a share of the number of total networks within the population by *ga.elitism.popsize.share*. The minimum number of surviving networks is set to three in the network generating algorithm.

After the appropriate settings for these important options and switches are chosen, the (dis)similarity input data can be imported. This can be done in the section *Defining the inputmatrix, which shall be visualised as network* in the script. The input data that is visualised as a network by the algorithm has to be defined as *inputmatrix* there.[37]

The input data has to be a symmetric matrix containing (dis)similarities. If the data contains similarity data, it is transformed into dissimilarity data by the script.[38] As stated earlier, no information is lost by this transformation, as this thesis focuses on ordinal scaled data.

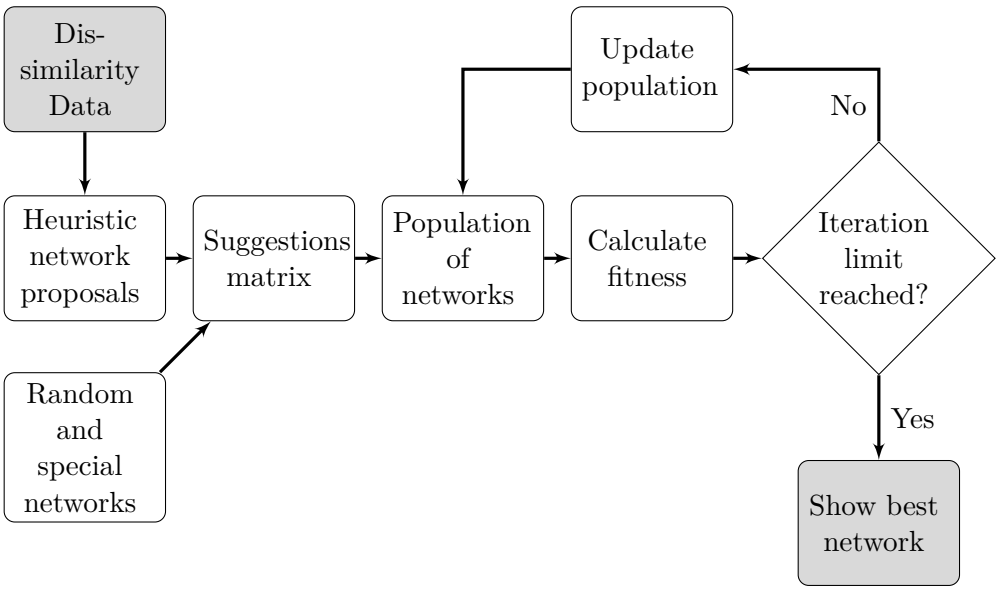

Figure 3.2.: *Flowchart of the procedure of the network generating algorithm.*

[37] The code to include the data from a matrix named *sample.data* in the R script would for example be: `inputmatrix <- sample.data`
As long as the options under *Cleaning out the workspace* in the R script are accounted for, (dis)similarity data can also be imported otherwise.

[38] If the proper options are set as explained above, especially *higher.value.means.higher.similarity*.

The flowchart in figure 3.2. visualises the procedure of the network generating algorithm after the (dis)similarity data has been imported and transformed into dissimilarities. The grey objects in the flowchart visualise the beginning and the end of the procedure of the network generating algorithm.

The network generating algorithm starts by generating heuristic network proposals from the transformed (dis)similarity data. These heuristic network proposals are networks that can serve as first guesses for the initial population of the GA. They try to represent the dissimilarities from the (dis)similarity data in a network by following a simple rule: In these heuristic network proposals, a link between two objects is present in its network representation if the dissimilarity from the (dis)similarity data between these two objects is above a certain threshold.

To generate these heuristic network proposals, each object in the (dis)similarity data is divided by the largest dissimilarity. Therefore, the largest of these then scaled dissimilarities takes the value of one, whereas the other scaled dissimilarities range from zero to less than one. The heuristic network proposals are then generated with respect to certain thresholds. The threshold 0.4 for example, generates a network with links between all pairs of two objects whose scaled dissimilarity value is less than 0.4.

Nine such heuristic network proposals are generated. One for each 0.1 step from 0.1 to 0.9. The incorporation of these nine different thresholds, instead of just one threshold, makes these heuristic network proposals more robust to different kinds of distributions of the (dis)similarity data.

As visualised in the flowchart in figure 3.2, these heuristic network proposals are part of the suggestions matrix that can serve as an initial population for the GA within the network generating algorithm.[39] The idea behind the use of the suggestions matrix is the better the initial population, the quicker the search for the best possible network representation.[40]

[39] The suggestions matrix "can serve" as an initial population, as its use can be manually enabled or disabled in the network generating algorithm, as described earlier.

[40] The initial configuration is also important in MDS. For more information, see section 7.5 in Borg et al. (2013).

In addition to the heuristic network proposals, random networks as well as special network types can also be part of the suggestions matrix and thus serve as first guesses. The special networks that enter the suggestions matrix are a complete and an empty network. The implementation of further special network types, like a line, star or tree networks was rejected because those networks offer too many different possible variations to incorporate an adequate share of these networks in the suggestions matrix.[41]

In order to improve their use in the initial population, the randomly generated networks for the suggestions matrix should neither have too many nor too few links between their nodes.[42] To balance these two requirements, the threshold function for the connectedness of Poisson random networks by Erdős and Rényi is used. Erdős and Rényi (1964) state that, if the probability to form a link between any two nodes is larger than the threshold function $t(n)$, then the probability to generate a connected network tends to one (Jackson, 2010, pp. 92). Such a threshold function $t(n)$ that depends on the number of n nodes in the network is shown in equation 3.1. This threshold function is used as the probability to form a link in the generation of random networks for the suggestions matrix to meet above requirements.[43]

$$t\left(n\right) = \frac{\log\left(n\right)}{n} \tag{3.1}$$

In addition to the Erdős and Rényi threshold, a few random networks, whose pairs of two nodes have a fixed probability of 0.3 to form a link, are also included in the suggestions matrix.[44]

The suggestions matrix therefore consists of nine heuristic network proposals, two special networks, and fills up the rest of this matrix with random networks, until the defined maximum population size is reached. This suggestions matrix serves as the initial population for the GA. However, as described earlier, it

[41] The number of possible variations of these network types is exponentially increasing with the number of nodes in a network.
[42] Otherwise, the networks might be over- or under-connected.
[43] The expected degree of such a network is $\log n$ (Jackson, 2010, p. 93).
[44] The probability of 0.3 to form a link can be manually set in the R script.

is also possible to only use a randomly generated initial population instead of using the described suggestions matrix.

Technically, each network in the whole network generating algorithm is realised as a vector that contains information on all possible pairs of two objects, as already mentioned in section 3.3.2 with respect to the population of the GA. An element of such a vector is either one, if there is a link between the two objects of the respective pair, or zero otherwise. In contrast to the realisation of a network as an adjacency matrix, this approach reduces the required computation time to find the best possible network representation. This is the case as the networks are required to be undirected in this thesis. An adjacency matrix would therefore contain unnecessary information – in contrast to the described vector.[45]

The GA within the network generating algorithm then begins its work and computes the fitness for each of its population's networks according to the chosen fitness functions. Fitness functions are explained and discussed separately in section 3.4.

If the maximum number of iterations has not been reached yet, the networks inside the population of the GA are updated through crossovers and mutations, as described in section 3.3.2. The best networks survive untouched. Therefore the population always contains the best networks it has yet generated at each iteration. After the update, the fitness of each of the updated networks in the population is calculated again. This procedure continues until the iteration limit is reached.

The network generating algorithm then graphically presents the best network representation.[46] The best network solution is also presented as an adjacency and distance matrix. In addition, the R script also offers the functionality to compare the network representation with the MDS representation with respect to certain fitness functions. These fitness functions to measure the fitness of both

[45] As it is assumed that there is no self-symmetry, the main diagonal is also omitted in the vector realisation.

[46] The script notifies the user, if there are more than one best network.

representation forms for ordinal scaled (dis)similarity data are introduced in section 3.4, after section 3.3.4 explains the already mentioned dummy nodes.

3.3.4. Dummy Nodes

In order to improve the fitness of a network representation of given (dis)similarity data, Büchel and Mastrangeli implemented the possibility to add dummy nodes to the network representation in their MATLAB code. This option is also implemented in the network generating algorithm described in this thesis.

Dummy nodes are nodes inside a network representation which are used by the network generating algorithm to better represent the order structure of given (dis)similarity data. The network generating algorithm tries to find the optimal position within the network representation for each dummy node – as it does for each node that represents an object from the (dis)similarity data.

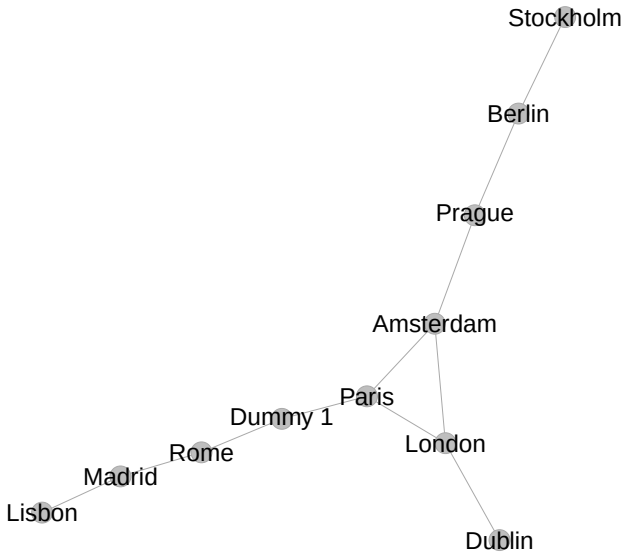

Figure 3.3.: *Network representation of the geographic distances of ten European cities and one dummy node.*

As an illustration, figure 3.3 shows a dummy node inside the best network representation of the geographic city distances, which is also shown in the figures 2.1 and 3.1. The dummy node in this example is used by the network generating algorithm to increase the geodesic distance between Paris and Rome. This has also an effect on many other pairs of nodes such as the increased distance between London and Madrid.

The number of dummy nodes is important for the quality of the output and the required computation time in order to find the best network representation. A high number of dummy nodes might lead to a better fitness value of the network representation compared to the case of fewer or zero dummy nodes. However, the resulting network graph might be less intuitive to interpret, the more dummy nodes it contains. Additionally, the required computation time is exponentially increased with each additional object in the (dis)similarity data, as the number of possible pairs between any two objects exponentially increases.[47] This also holds for each additional dummy node.

One of the main goals of this thesis is to derive which representation form is best for which kind of (dis)similarity data. This is done in the practical comparison section of both representation forms in section 3.6. The incorporation of dummy nodes into the comparison of MDS and network representation, is beyond the scope of this thesis. It should therefore be addressed in detail in a separate work along with the question of the ideal number of dummy nodes for given (dis)similarity data.

3.4. Ordinal Fitness Functions for Both Multidimensional Scaling and Network Representations

After the description of the network generating algorithm and its procedure in the last sections, the following sections present and discuss appropriate fitness

[47] This is shown as the maximum number of possible order violations in equation 3.6 in section 3.4.2.

functions to compare the fitness of an MDS representation with a network representation of the same ordinal (dis)similarity data.

There are certain requirements on such a fitness function. First and foremost, it should be able to measure the fitness for both MDS and networks representations and thus make their quality comparable with each other. Second, the fitness function that measures the quality of a representation form should ideally also be the function that is optimised during the iterative generation process of that representation form.

With these two requirements in mind, the next two sections introduce two different possible fitness functions.

3.4.1. Stress-1

The Stress loss function is the most common measure to assess the fitness of an ordinal distance MDS representation (Borg and Groenen, 2005, p. 37). It should therefore be examined whether it could also be used to measure the fitness of a network representation and serve as fitness function in the GA to generate the best possible network representation for given (dis)similarity data.

According to the description in section 2.2, the Stress-1 value depends on the pairwise differences between the distances (d_{ij}) from the MDS representation and the approximated distances $\left(\hat{d}_{ij}\right)$. The approximated distances are obtained by performing a monotone regression of the distances from the MDS representation on the dissimilarities (δ_{ij}) from the (dis)similarity data.

In order to transfer this approach to network representations let the geodesic distances between any nodes i and j within a network representation be d_{ij}^N. A monotone regression of such distances from a network representation on the dissimilarities from the (dis)similarity data can be applied analogously to the MDS case.[48]

[48] The latter case is described in detail in section 2.2.

The estimations from such a monotone regression can be expressed in a graph. Such a graph could, like a Shepard graph, show the dissimilarities from the (dis)similarity data, the distances from the network representation and the approximated distances from the monotone regression. That means that the same pairwise comparison of the distances from the representation and the approximated distances, as in MDS, can be done for a network representation.

Therefore, a Stress-1 calculation, as shown in equation 2.2 on p. 12, can be realised for a network representation. The following equation 3.2 is equivalent to the former equation, besides the fact that the distances are taken from a network and not from an MDS representation here.

$$\text{Stress-1 (Network case)} = \sigma_1^N = \frac{\sum\limits_{i<j} \left(d_{ij}^N - \hat{d}_{ij}\right)^2}{\sum\limits_{i<j} \left(d_{ij}^N\right)^2} \tag{3.2}$$

The fitness of an MDS and a network representation could therefore generally be compared with each other by their respective Stress-1 value. The question that arises now is whether Stress-1 meets the two requirements on fitness functions that are formulated in section 3.4. In that section it is stated that a fitness function should be able to measure the fitness of both representation forms and thus make their quality comparable with each other. Additionally, a fitness function should ideally also be the function that is optimised during the generation process of the respective representation form.

With respect to MDS representations, both requirements are met by Stress-1, as Stress-1 is the most common measure for the quality of an MDS representation. Stress-1 can also serve as the function that is optimised during the generation of an MDS representation.

In order to check these requirements with respect to network representation, imagine the case of a complete network as a network representation for given (dis)similarity data. All nodes in a complete network are directly connected to all other nodes. The geodesic distance from each node to any other node is therefore always one.

As described above, a monotone regression of the distances from the network representation on the dissimilarities from the (dis)similarity data has to be performed in order to calculate the Stress-1 value for network representations. Such a monotone regression generates approximated distances for each pair of two objects. In the case of a complete network, these approximated distances would all take a value of one.

In order to visualise the reasoning behind this, imagine a graph like the Shepard graph in figure 2.2 on p. 11, with dissimilarities on the horizontal axis and the distances from the network representation on the vertical axis. In the case of a complete network, such a Shepard graph would show all pairs of two objects – which are expressed as open circles in figure 2.2 – in a line parallel to the horizontal axis at the distance value of one. A monotone regression of these points, which show the dissimilarities and distances for all possible pairs of two objects, would graphically result in a line that passes through all these points. All points would therefore lie on the monotone regression line.

Recall the procedure of the Stress-1 calculation as described in section 2.2 to understand the reasoning behind this. First, all pairs of two objects are sorted and ranked according to the size of their dissimilarity (δ_{ij}) in the (dis)similarity data. The distance pairs from the network representation $\left(d_{ij}^N\right)$ are then also sorted according to the rank of their respective pair of two objects (ij) within the ordered dissimilarities. The monotone regression of the distances from the network representation on the dissimilarities is then applied, and generates a monotonically increasing regression line. However, in the case of a complete network, all distances from network representation take the value of one. A regression line through these points $\left(d_{ij}^N; \delta_{ij}\right)$ would lead to a flat line parallel to the horizontal axis. As all points lie on that regression line, the Stress-1 value would be zero – which is the best possible value.

Interestingly, this result is, as shown above, independent of the structure of the (dis)similarity data. The Stress-1 value for a complete network would therefore always be zero.

This result is also independent of the choice between the primary or secondary

approach to handle ties in MDS as all distances in the network representation have an equal value. However, the choice between weak and strong monotonicity in the monotone regression has a tiny influence in this result. The descriptions and conditions of the respective approaches and requirements are shown in section 2.2.

In the case of weak monotonicity, any dissimilarity that is larger than another must have a larger or equal distance in the network representation. The regression line would therefore be a line through all the mentioned points. The Stress-1 value will be zero.[49]

In the case of strong monotonicity, any dissimilarity that is larger than another must also have a larger distance in the network representation to prevent the arise of Stress. In the case of a complete network for given (dis)similarity data, the regression line would therefore not be exactly on the mentioned line of points. Instead, it will have the tiniest possible steps. These steps are so tiny, that the resulting residuals and therefore the Stress-1 value would be almost zero and thus not be of any use.[50]

The described problem is due to a high number of nodes with the same distances in a network. It is therefore also existent in any network where many nodes have the same distances to each other. It is therefore also true for the empty networks as all nodes are unconnected and thus have the same distance to each other.[51]

Additionally, Stress-1 would also overestimate the fitness of networks which are very similar to complete or empty networks. A network where almost all nodes have the same distance and just some nodes do not, would for example also lead to a very good Stress-1 value.

Stress-1 is therefore not able to assess the fitness of complete, almost complete, empty and almost empty networks. Stress-1 should therefore not be used as fitness function in the GA of the network generating algorithm. As it is not

[49] The detailed example from the previous paragraphs describes the case of weak monotonicity.

[50] In line with this, Borg and Groenen describe the low practical use of strong monotonicity for MDS on p. 203 in Borg and Groenen (2005).

[51] The distance of unconnected nodes is a matter of definition, as described on p. 21.

specified here which networks count as almost complete and almost empty, it is at this point not certain, whether Stress-1 might be an appropriate measure for networks that were generated by optimising a different fitness function. This is discussed in the practical comparison of both representation forms in section 3.6.

3.4.2. Error Counting

As the Stress-1 optimisation is not useful for the generation of the best possible network representation, a different fitness measure that is able to assess the quality of both MDS and network representations has to be found.

In their MATLAB script, Büchel and Mastrangeli integrated a fitness function that judges the quality of a representation of (dis)similarity data by counting the number of order violations between the ordinal (dis)similarity data and its network representation. An order violation occurs whenever the relation of two pairs of two objects differs in the network representation $\left(d_{ij}^N\right)$ from their relation in the (dis)similarity data (δ_{ij}).

There are three different possible cases of order violations:

$$d_{ij}^N > d_{kl}^N \text{ while } \delta_{ij} \leq \delta_{kl} \tag{3.3}$$

$$d_{ij}^N < d_{kl}^N \text{ while } \delta_{ij} \geq \delta_{kl} \tag{3.4}$$

$$d_{ij}^N = d_{kl}^N \text{ while } \delta_{ij} \neq \delta_{kl} \tag{3.5}$$

All possible pairs ij and kl are checked with respect to these three possible cases of order violations.[52] Each time a violation occurs, the error count is

[52] As stated earlier, the distance of unconnected nodes is a matter of definition. A special case arises in unconnected network representations, if the distance of unconnected nodes is defined as infinite. In that case, $d_{ij}^N = d_{kl}^N = \infty$ does not lead to an error when $\delta_{ij} = \delta_{kl}$. That means that two pairs of two objects (jk and kl) could be represented as unconnected nodes within a network representation if these two pairs have an equal dissimilarity (δ) in the (dis)similarity data.
The possible options for the case of unconnected networks in R are explained in detail in the manual of the *igraph* package for R by Csárdi and Nepusz (2006) under *shortest.paths*.

increased by one.[53] A perfect representation of given (dis)similarity data would be achieved by an error count of zero.

The number of maximal possible errors is shown in equation 3.6. It depends on the number of objects (n) in the (dis)similarity data (Borg and Groenen, 2005, p. 28).[54]

$$\text{Number of max. errors} = \frac{n \cdot (n-1)}{4} \cdot \left(\frac{n \cdot (n-1)}{2} - 1 \right) \tag{3.6}$$

As the number of possible errors increases with the number of objects in the (dis)similarity data, the error count should be normalised. As shown in equation 3.7, the fitness share measure shows the share of possible errors that the representation prevented from occurring.[55]

$$\text{Fitness share} = 1 - \frac{\text{Error count}}{\text{Number of max. possible errors}} \tag{3.7}$$

In contrast to the described Stress-1 value, this error counting function can be used in the GA as fitness function which is optimised in order to find the best network representation of given (dis)similarity data. The network representation of ten European cities in figure 3.1 on p. 18 for example was generated by optimising the fitness share value.

This fitness measure can analogously to a network representation be applied on an MDS representation. The equations 3.3–3.5 also are applicable for an MDS representation when the geodesic distances within a network $\left(d^N\right)$ are replaced by the MDS distances (d). Therefore, the fitness of both MDS and network representations can be compared with each other by counting

[53] The realisation of this is shown in the R code of the network generating algorithm in the appendix A.1.

[54] This number is not directly affected by the number of dummy nodes as they only affect the geodesic distances. This can also be seen in the R code in the appendix A.1.

[55] This fitness share is later used in section 3.6 to compare the fitness of both representation forms.

the order violations of the respective representation with respect to the given (dis)similarity data.

However, despite the fact that fitness of an MDS representation can be measured by the described error counting function, the MDS representation is generated by optimising Stress – and not by optimising the described error counting function.

The questions is, whether this leads to an unfair advantage of a network representation that was generated by optimising the error counting function when comparing MDS and network representations by the number of order violations. As described in section 2.2, the central goal of Kruskal's non-metric Stress optimisation is to best represent the order structure of the (dis)similarity data – while adding Stress as a measure for violations with respect to the order structure (Kruskal, 1964, p. 3). The described error counting function has the identical central goal. Therefore, a comparison of a Stress-optimising MDS and a network representation that optimises the number of order violations data should be fair.

Furthermore, the error counting does not run into the same issues as the Stress-1 calculations does with regard to complete, almost complete, empty and almost empty networks.

To conclude, the described error counting and its fitness share value can be used within the GA of the network generating algorithm to find the best network representation of given (dis)similarity data. In addition, error counting can be used as a fitness function to compare the fitness of an MDS with a network representation.

3.5. Chosen Parameters for the Network Generating Algorithm

Prior to the comparison of MDS and network representations, the chosen parameter set for the network generating algorithm and therefore also for the GA have to be discussed. The parameters that are discussed in this section are introduced in section 3.3.2 and especially in section 3.3.3 on the network generating algorithm.[56]

The identification of the best values for the parameters of the network generating algorithm relies on both theoretical considerations and practical experiences. The practical experiences were gathered by using different parameters for the representation of the same (dis)similarity data. Each tested parameter set was tested multiple times to limit biases in these results with respect to the important role that chance plays in the network generating algorithm. Chance plays such an important role in the network generating algorithm as parts of the GA's initial population are randomly generated and both the crossover and the mutation function in the GA randomly change the population at each iteration step.

Parameter	Value
ga.pcrossover	0.2
ga.pmutation	0.8
ga.elitism.popsize.share	0.1
ga.use.suggestions.matrix	TRUE
ga.use.modified.mutation	TRUE
dummy.nodes	0

Table 3.1.: *Chosen parameters for the network generating algorithm.*

The chosen parameters for the generation of a network representation are

[56] With respect to MDS the default options of the *smacof* R add-on by de Leeuw and Mair are used. Therefore, the primary approach in combination with weak monotonicity in the monotone regression is used, as explained in section 2.2. Compare Leeuw et al. (2009) for further information on the default options of the mentioned R add-on.

highlighted in table 3.1. These values were used for the generation of the best network representation in the following section.

One of the main challenges for the network generating algorithm is to find the global optimal network representation in-between many locally optimal networks. To allow the network generating algorithm to find that global optimal network representation, a relatively high probability of a mutation (*ga.pmutation*) occurring in the GA's population of networks is chosen. The modified mutation function for the GA within the network generating algorithm, which was introduced in section 3.3.2, is used for the same reason as it allows mutation within a network in more than just one link.

As a result of practical experience, a relatively low probability of a crossover (*ga.pcrossover*) leads to better network representation within a given time. For the same reason the suggestions matrix is used.

The number of the networks with the best fitness value that survive an iteration is set by *ga.elitism.popsize.share*. This number is either three or a 0.1 share of the population size of the GA. These networks with a high fitness value then spread their good characteristics through the crossover function to other members of the population.

A high limit for the maximum number of iteration is set for the application of the network generating algorithm on the respective data sets. This is accompanied with a high number in the *ga.run* option which stops the network generating algorithm if there has not been an improvement in set number of last iterations. The specific values of these options vary for the respective data sets. In most cases the maximum iteration number is set to 20,000 iterations while *ga.run* is set to stop after 2,500 iterations without any improvement in the fitness value of the best network representation. Such high numbers of iteration steps improve the chance to find the global optimal network representation for given (dis)similarity data.

Dummy nodes are not used in the following application for reasons discussed in

section 3.3.4. The influence of dummy nodes within a network representation of (dis)similarity data might be an interesting topic for further research.

The parameters that are presented in this thesis were chosen with respect to the (dis)similarity data in the next sections. Different (dis)similarity data might lead to different recommendable values, especially with respect to the number of objects within that data.

3.6. Application and Comparison of Multidimensional Scaling and Network Representations

With respect to the discussed fitness measures, there is (dis)similarity data that can be better represented by an MDS representation and there is also data that can be better represented by a network representation. Selecting existing (dis)similarity data for a comparison of both methods is therefore always associated with subjectivity. Additionally, one main goal of this thesis is to generally derive in which cases which representation form leads to better fitness values. It is therefore better to choose data that can be generalised easier than specific existing data. Hence, this section focuses on results from the multiple application on different cases of randomly generated (dis)similarity data and not on single non-random data.

One exception from the focus on random (dis)similarity data is made to satisfy the eager reader who is interested in the results of both representation forms of the geographic distance data of the ten European cities that is used as example throughout this thesis: The MDS representation in figure 2.1 achieves an error count of 14 out of 990 possible errors which is a fitness share of 0.9859.[57] Its Stress-1 value is 0.0010.[58] The best network representation which is shown in figure 3.1 has an error count of 251. This translates into a fitness share of 0.7465. The Stress-1 value is 0.1673. With respect to the errors in the ordinal

[57] Recall, a perfect fit has a fitness share value of one.
[58] Recall, a perfect fit has a Stress-1 value of zero.

rank order, the MDS representation is able to better represent the geographic distances of these ten cities.

Including a dummy node in the network representation as shown in figure 3.3 reduces the error count from 251 to 213 and thereby improves the fitness share to 0.7848. Interestingly, instead of improving like the error count, the Stress-1 value of this network representation worsens to a value of 0.1879. This underlines the problems of Stress-1 when used to measure the fitness of networks as discussed in detail in section 3.4.1.

As stated earlier, the focus between the two introduced fitness measures is on the error counting fitness measure and its fitness share value from equation 3.7 on p. 38 and thus not on Stress-1. Nonetheless, Stress-1 is also shown in this section to analyse its use with respect to networks that were generated by optimising the error counting fitness function in the network generating algorithm.

The application and comparison of both representations in this section is conducted for different types of (dis)similarity data. The first distinction between these data sets is the number of objects within them. Both representation forms were applied on data with ten and twenty randomly generated objects. The use of two different data sizes might allow the identification of possible advantages of one representation form with respect to the number of objects in the (dis)similarity data. In keeping with the application of MDS to represent the similarity of for example product brands in business administration or countries in political science, the chosen numbers of objects seems to be appropriate.

As explained in section 2.3, a possible source of information loss within MDS is the inability to represent more than three objects with the same Euclidean distance to each other correctly within 2-dimensional geometric space. In order to test this, two different cases of random data are analysed, in addition to the two different sizes of the (dis)similarity data. One case of random data includes a very high probability to contain many pairs of two objects of the same dissimilarity. The other case leads to data with a very low probability for the occurrence of pairs of two objects with the same dissimilarity.

The random data with the very low probability of identical dissimilarities follows a normal distribution with a mean of 10 and a standard deviation of 1.5.[59] The likelihood of identical dissimilarities between pairs of objects is very low due to the multiple decimal places and the limited number of possible pairs for (dis)similarity data on ten respective twenty objects. This kind of data with a very little probability of same dissimilarities is henceforth called *non-rounded* random data. The reasoning behind this name is explained in the following paragraphs.

Recall, as the random (dis)similarity data is treated as ordinal scaled data in this thesis, only the ordinal rank of the size of a dissimilarity and not the exact size of the dissimilarity itself is important. Transferred to the example of the marathon runners from section 2.1, one is only interested in the ranking in which the runners finished the marathon and not in the exact time of each runner. Therefore, the non-rounded random data consists in almost all cases of no multiple pairs of objects with the same dissimilarities.

In order to generate the case of multiple pairs of objects with the same dissimilarities to each other, the same normal distribution as above is used in the first step of the generation of the data. In a second step, all these dissimilarities are rounded and thus transformed to integers that follow a normal distribution with a mean of 10 and a standard deviation of 1.5. This kind of data is henceforth called *rounded* random data.

In most cases, this data generation results for the rounded data in (dis)similarity data where most pairs of objects have a dissimilarity of ten and many pairs of objects have a dissimilarity of nine or eleven. The other objects have a dissimilarity of below nine or above eleven. As stated above, the exact value of a dissimilarity is not important. The only relevant information is its ordinal rank.

The number of different dissimilarities within this (dis)similarity data varies with the number of objects as a higher number of random objects might increase

[59] Any negative value from this normal distribution is transformed to its absolute value. However, it is very unlikely that a negative value is generated from this distribution.

the amount of different dissimilarities. With respect to the (dis)similarity data that was used for the results in this section, the case of ten objects leads in most cases to rounded random data with six different dissimilarities. The case of twenty objects leads to rounded random data with nine different dissimilarities in most cases.

In order to convey the rationale behind such rounded and non-rounded random data, imagine a questioner would conduct a questionnaire on the similarity of ten respective twenty different car brands. The respondents should pairwise judge the similarity between two car brands on an integer scale from one to ten. In order to analyse the information from the respondents, the questioner could compute the mean of the similarity for all pairs of two car brands. As in the case of non-rounded random data, the likelihood of identical similarities would be very low due to the multiple decimal places of the mean similarity of each pair of two car brands. Imagine the questioner would compute the median to prevent potential biases due to extreme values or measurement errors. The resulting data on the median similarity for all pairs of two car brands would feature up to ten different similarity scales in total. The latter example is therefore very similar to the rounded random data in this section.

The results from the representation of the four different cases with respect to the fitness share from the error counting function are shown in table 3.2.[60] These four different cases are the following: Ten objects with rounded random data, ten objects with non-rounded random data, twenty objects with rounded random data and twenty objects with non-rounded random data.

The row *number of computations* shows the number of randomly generated (dis)similarity data sets that were analysed. The row *better fitness share* displays the share of these computations that is better represented by an MDS or by a network representation respectively. The other rows describe the mean as well as the quartiles of the fitness share. The standard deviation of the fitness shares of all computations is also shown in the row *std. deviation*.

[60] The formula to compute the fitness share is equation 3.7 as explained on p. 38.

		Random data			
		Rounded		Non-Rounded	
Number of objects		10	20	10	20
Number of computations		38	58	35	32
Better fitness share	MDS	0.3421	0.1207	0.9714	1.0000
	Network	0.6579	0.8793	0.0286	0.0000
Fitness share – mean	MDS	0.6314	0.5790	0.7337	0.6819
	Network	0.6709	0.6656	0.6521	0.5914
Fitness share – first quartile	MDS	0.6061	0.5621	0.7162	0.6707
	Network	0.6502	0.6400	0.6374	0.5803
Fitness share – median	MDS	0.6278	0.5844	0.7374	0.6892
	Network	0.6642	0.6656	0.6455	0.6023
Fitness share – third quartile	MDS	0.6535	0.5906	0.7662	0.7002
	Network	0.6960	0.6793	0.6672	0.6150
Fitness share – std. deviation	MDS	0.0318	0.0202	0.0411	0.0232
	Network	0.0244	0.0325	0.0212	0.0352

Table 3.2.: *Results from the comparison of both representations. Measure: Fitness share from the error counting function.*

For twenty non-rounded random objects, an MDS representation generates solutions that have an average fitness share of 0.6819. That means that out of 17,955 possible order violations for the case of twenty objects, 5,711 rank order violations – or 12,244 correct rank orders – were present on average in the MDS representation. The network representations for that case have an average fitness share of 0.5914.[61]

5,711 order violations within the better representation form might sound relatively high. It is of great importance to understand that such high numbers of errors mainly occur due to the pure random origin of the (dis)similarity data in this section. Each of the dissimilarities of any pair of two objects from the (dis)similarity data is completely random. The application on non-random (dis)similarity data should therefore generally lead to less order violations.[62]

[61] The formula to compute the number of possible order violations is given in equation 3.6 in section 3.4.2.

[62] This is also discussed in section 2.3.

When looking at the fitness shares for rounded random (dis)similarity data, it becomes apparent that this data can be better represented by a network graph than by MDS, when using the fitness share as measure. Numerically, 65.79 percent of the data sets containing ten random objects are better represented by a network graph. This increases to 87.93 percent for the case of twenty random objects.[63]

In the case of twenty objects in the rounded random data, the average fitness share of an MDS representation is 0.5790. The network achieves an average fitness share of 0.6656 in that case. The network representation is also better with respect to the quartiles of the fitness share in the case of rounded random data.

In contrast, the non-rounded random (dis)similarity data can clearly be better represented by MDS. Out of 35 data sets of which each contains ten objects, only one data set can be better represented by a network graph. None of the data sets with twenty objects can be better represented by a network.

In keeping with section 2.3 about information loss in MDS, the results in table 3.2 suggest that an MDS representation is more *vulnerable* to the occurrence of multiple pairs of objects with the same dissimilarities. Especially in that case of multiple pairs of objects with the same dissimilarities, the network representation is an adequate alternative to MDS.

Interestingly, for both cases of ten objects, the fitness of the network representation only shows minor changes between rounded and non-rounded random data. The advantage of network representations over MDS representations in the case of rounded random data seems therefore to be more due to MDS's inability to represent multiple pairs of objects with the same dissimilarities than to the network's strength to do so.

Despite the differences in the fitness share between rounded and non-rounded random data, the increase in the number of objects from ten to twenty leads to a reduction in the average fitness share of the network representation of about

[63] As described earlier, a *better fitness share* means that after each complete computation, the fitness share of both representation forms were compared with each other.

0.5 percentage points for the case of rounded random (dis)similarity data. In the case of non-rounded random data, the reduction of the average fitness share is with its six percentage points larger than in the other case. At the same time, the MDS representations lose on average about five percentage points in their fitness share in both cases of the number of objects. The network representation seems to be more robust to an increase in the number of objects in the case of rounded random data. Further research should be conducted whether this effect is also present for the network representation of data sets with even more objects.

As explained in section 3.3.4, the addition of dummy nodes to a network representation should further improve the representation's fitness share. The fitness share of a network representation can only stay equal or increase by adding dummy nodes. Dummy nodes that cannot further improve the fitness share would simply be arranged in a way that they do not affect the fitness share of the rest of the network, like for example as an isolate or a pendant. It is therefore possible that the number of data sets which are better represented by a network graph instead of MDS would further increase with the addition of dummy nodes.

In addition to table 3.2, table 3.3 shows a summary of the Stress-1 values for the same computations.[64] Recall, Stress-1 is shown in section 3.4.1 to be of no use for measuring the fitness of complete, almost complete, empty, and almost empty networks. The Stress-1 values in the table allow answering the question, whether Stress-1 can be used to measure to assess the fitness of network representations in general.

The row *frequency Stress-1 = 0* shows that for the rounded random data, network representations lead to a perfect Stress-1 value of zero in most cases for both ten and twenty objects. However, as the fitness share value found no perfect network representation with a fitness share of one, there are order violations within each of the representations. Therefore, a Stress-1 value of zero

[64] As stated earlier, the better the fitness of the representation, the lower the Stress-1 value in general. A perfect fit would normally result in a Stress-1 value of zero.

		Random data			
		Rounded		Non-Rounded	
Number of objects		10	20	10	20
Number of computations		38	58	35	32
Better Stress-1	MDS	0.1053	0.1207	1.0000	0.9048
	Network	0.8947	0.8793	0.0000	0.0952
Stress-1 – mean	MDS	0.1395	0.2454	0.1943	0.2969
	Network	0.0340	0.0529	0.3433	0.3658
Stress-1 – median	MDS	0.1348	0.2476	0.2038	0.2923
	Network	0.0000	0.0000	0.3559	0.3368
Frequency Stress-1 = 0	MDS	0	0	0	0
	Network	29	34	0	0

Table 3.3.: *Results from the comparison of both representations. Measure: Stress-1.*

for network representations of rounded random data is not due to a perfect quality. Instead, it is due to the described problems Stress-1 has when applied to data sets with multiple pairs of objects with the same distances in their representation.[65]

The Stress-1 values of the network representations of non-rounded random data are also affected by this problem as networks generally have multiple pairs of nodes that have the same geodesic distance. This would lead to possible biases in the Stress-1 value. Therefore, Stress-1 is an inappropriate measure to assess the fitness of network representation and thus not useful for the comparison of MDS and network representations of ordinal scaled data. The error counting function and its fitness share value should currently be the measure of choice to analyse and compare the quality of both MDS and network representations.

In summary, the main results of this section are the following. There is less information loss in network representations than in MDS representations when multiple pairs of objects with the same dissimilarities are present in the ordinal scaled (dis)similarity data. Network representations are therefore better to

[65] The problem is further elaborated in an example in the appendix A.2 on p. 86.

represent this data. This is in line with the expectation in section 2.3 about the information loss. According to the information loss, data with no multiple pairs of objects with the same dissimilarities is in almost all of the tested cases better being represented in MDS representations. Additionally, it is underlined that Stress-1 is not an adequate measure to assess the fitness of network representation.

3.7. Critical Discussion

This thesis introduces two different fitness functions to measure the fitness of both MDS and network representations and thus making them comparable with each other. This thesis strongly favours the error counting fitness function over the Stress-1 value calculation in this context. The error counting fitness function and its fitness share value proved themselves to be robust for measuring the fitness of both representation forms as well as being optimised within the GA of the network generating algorithm. However, further research on this topic might lead to an even better fitness function with respect to this application.[66]

The application and comparison of both representation forms is done by using two different kinds of randomly generated (dis)similarity data. One set of random data features a high likelihood of multiple pairs of objects with the same dissimilarities. That data is called rounded random data. The other set of random data has a very low likelihood for same dissimilarities between pairs of objects. That data is called non-rounded random data. The main motivation of this approach is to determine whether one of these two kinds of data can systematically be better represented by one of the two representation forms.

This likelihood for the occurrence of the same dissimilarities in the rounded data could have been altered. An increase in the standard deviation of the normal distribution used for the data generation should have led to a lesser likelihood of

[66] This is especially true for metric scaled (dis)similarity data, which is not discussed in this thesis. The network generating algorithm, which is available in the appendix A.1, already suggests a fitness function for metric scaled data that could be further analysed.

multiple pairs of objects with the same dissimilarities. The lesser that likelihood, the better the fitness share of the MDS representation. This example of a reduced likelihood of multiple pairs of objects with the same dissimilarities, might – depending on the size of the reduction of the likelihood – have led to results that were between those of the rounded and non-rounded randomly generated (dis)similarity data in section 3.6. Nonetheless, the parameters for the generation of the random data are sufficient to show the advantages of network representations over MDS representation when dealing with multiple objects with the same dissimilarities. They are also realistic as shown in the example of a questionnaire on car brands.

In section 3.6, data sets of two different sizes are analysed. Some data sets consist of ten objects while the others consist of twenty objects. It would be very interesting to see the results of a comparison of both representation forms of larger data sets. However, one has to keep in mind the exponentially increasing computation time that is required to generate the best network representation for every additional object.[67]

Closely related to the number of objects is the question whether the network generating algorithm is able to find the representation that globally optimises the respective fitness function. To find the globally optimal network representation among many possible local optima, the GA is best used in combination with an adequate population size and number of iterations. Each generation of a network representation of rounded random (dis)similarity data either stopped after reaching the 20,000$^{\text{th}}$ iteration or after no further improvement in the fitness of the best representation for more than 2,500 sequential iterations.[68]

[67] The GA of the network generating algorithm took about one hour to compute 2,500 iterations in the case of twenty objects with the settings described in section 3.5. The same number of iterations has been conducted in about half the time for the case of ten objects. The computation time depends of course on the computer one is using. Additionally, the network generating algorithm might be further optimised with respect to computation time, despite the already implemented support for multi-threading CPUs. A comparison of MDS and network representations with respect to computation time would be unfair as MDS has been subject to academic research for a very long time and should therefore be highly optimised. Hence, such a comparison is not shown in this thesis.

[68] As stated in section 2.2, there are cases where MDS does not find the globally optimal

Besides possible other fitness functions, the use of a different algorithm than the GA to find the fitness maximising representation could also be analysed. As described in section 3.3.2, the GA is used in the network generating algorithm for its capability in finding the globally fitness optimising network representation among the locally optimal network representations. However, other algorithms might, for example with respect to the mentioned computation time, have their respective advantages.

After all the quantitative comparisons of both representations so far in this thesis, it is of course also possible that the preference for a geometric MDS representation or a network representation might not solely be determined by the value of fitness measures. One might simply prefer one representation to another for reasons not quantifiable.[69] Hence, the existence of network representations offers a visual alternative to MDS.

Additionally, instead of thinking of network representations as an alternative to MDS, Büchel and Mastrangeli suggest that one could also combine the two in a single graph. Such a graph would offer both visualisation types at the same time. However, that graph might be confusing as it might offer too much information at once. The analysis of such a graph could be subject of further research.

representation. Detailed information can be found in chapter 13 about degenerate solutions in ordinal MDS in Borg and Groenen (2005).

[69] For example, one might favour a network representation over MDS as it might reveal more information about the centrality of objects.

4. Summary and Outlook

This thesis aims to answer the following research questions:

1. How can the best possible network representation be found?
2. How can the quality of both MDS and network representations be measured and compared with each other?
3. When are network representations an alternative to MDS?

In order to answer the first research question and to find the best possible network representation, the network generating algorithm was developed. The network generating algorithm is an extension of an algorithm by Büchel and Mastrangeli. Compared with their algorithm, the novel network generating algorithm adds many new features such as the ability to use both similarity and dissimilarity data, the calculation of Stress-1 fitness values for both representation forms, and the ability to generate an MDS representation for given (dis)similarity data. The network generating algorithm also generates visualisations of both best representation forms. Furthermore, the network generating algorithm modifies and expands many existing functions of Büchel and Mastrangeli's algorithm. For example, the quality of the initial population of the GA is improved due to the implementation of improved random networks and the incorporation of heuristic network proposals. Additionally, the mutation function of the GA is modified with respect to the application for the search of the optimal network representation.

In order to assess and compare the quality of both representation forms, two fitness functions were introduced and analysed: Stress-1 and the error counting fitness function. The Stress-1 calculation is transferred in this thesis from the

MDS literature to the application on network graphs. It is therefore generally applicable on both MDS and network representations. Nonetheless, this thesis proved that the Stress-1 value is unreliable for networks due Stress-1's problems with the occurrence of multiple pairs of nodes with the same geodesic distances. A better alternative to measure and compare the fitness of both representation forms is the error counting function and its fitness share value. This answers the second research question.

The third research question can be answered through the practical application of both representation forms on two different data structures, with two different numbers of objects inside each of them. Certain ordinal scaled (dis)similarity data with multiple pairs of objects with same dissimilarities can in most cases be better represented by a network representation, with respect to the information loss according to the fitness share measure. Data with zero or almost zero occurrences of multiple pairs with the same dissimilarities can generally be better represented by MDS.

This result seems to be originated in the inability of the geometric MDS representation to represent more than three objects with the same Euclidean distance to each other within 2-dimensional geometric space correctly. This can be concluded from the drop in the average fitness shares of MDS representations in the case of multiple pairs of objects with the same dissimilarities. In contrast, the average fitness shares of the network representations are relatively constant for both cases of data. That means that especially in the case of many pairs of objects with the same dissimilarities to each other, network graphs are a good alternative to MDS.

The role of the numbers of objects in the (dis)similarity data is analysed for data sets with ten or twenty objects. From the application in this thesis, it can be concluded that an increase in the number of objects reduces the average fitness of both representation forms. However, in the case of multiple pairs of objects with same dissimilarities, the increase in the number of objects from ten to twenty showed almost no change in the networks' fitness share.

As this thesis features a novel approach to represent (dis)similarity data, there are many additional topics for further research. It is recommended to conduct further research on the effect of the number of objects on the fitness of their representation. Furthermore, an intensive analysis of dummy nodes as well as the search for additional fitness functions for both representation forms offer interesting possibilities to further improve the quality of these new network representations as an alternative to MDS for visualising (dis)similarity data.

This thesis introduced and analysed new methods to measure the information loss in both MDS and network representations, thus making them comparable with each other. Based on a complex network generating algorithm, those measures were applied on a large number of randomly generated ordinal scaled (dis)similarity data and their performances were evaluated. This application proved that (dis)similarity data with many pairs of objects with the same dissimilarities can be better represented by a network representation than by an MDS representation. Hence, a network representation can be an interesting alternative to MDS.

A. Appendix

A.1. R Source Code of the Network Generating Algorithm

The following lines and pages show the R source code of the network generating algorithm. Comments begin with a hash sign (#) and are printed in italics. Please contact the author of this thesis for a digital copy of the source code.

```
1  #Version: 2014-05-02
2
3  #This code made in line with the "rules" by http://google-styleguide.googlecode
       .com/svn/trunk/Rguide.xml
4  #That means variables are all named like "variable.name" (no capital letter,
       but with dots), while functions are named like "FunctionName" (capital
       letters, no dots)
5
6
7  #
8  # General settings ------------------------------------------------------
9  #
10
11 #One has to set the R working directory before starting this script or define
       it here!
12
13 ###Cleaning out the workspace
14
15 #Remove all values/variables. Any input data has to be loaded after this point.
16 rm(list=ls())
17
18 #Reset the plot options (check par() for more information on current settings)
19 par(mfrow=c(1,1))
20 par(mar=c(5.1, 4.1, 4.1, 2.1)) #Standard margins
21
22 #Setting the working directory
```

```
23 #As this script generates many files as output it is recommended to specify a
       working directory. This can be done via ?setwd
24
25
26 #
27 # Important options and switches -----------------------------------------
28 #
29
30 #Optional title that is used in the summary of the output
31 title.of.input.data = "Test" #This string is used in the summary of the output.
32
33 #Determine the scale of the input data
34 scale.of.input.data = "ordinal" #Can be set to "metric" or "ordinal". This
       variable is currently case sensitive and has to be in quotation mark to
       mark it as text.
35
36 ##Is the similarity data you are going to work with coded in a way, that higher
        values mean higher similarity? Then use =1. If higher values means
       higher distance, use =0
37 higher.value.means.higher.similarity = 0 #Very, very important switch! (can be
       0 or 1) #TODO Could be transformed into Boolean
38
39 #Set the number of dummy nodes in the network
40 dummy.nodes = 0 #The number of dummies has to be >=0
41
42
43 ###Options for Genetic Algorithm
44 ga.maxiter = 5000 #Maximum number of iterations
45 ga.run = 1000 #The genetic algorithm will stop if there is no improvement after
        [what is defined here] iterations.
46
47 ga.pcrossover = 0.2
48 ga.pmutation = 0.8
49
50 ga.elitism.popsize.share = 0.1 #The number of best fitness solution that will
       be kept, based on this share of the size of the population in the genetic
        algorithm. popsize.
51
52 ga.use.suggestions.matrix = TRUE #Switch for whether to use to suggestions
       matrix (TRUE) or not (FALSE). It is recommended to use the suggestions
       matrix.
53
54 ga.use.modified.mutation = TRUE #Switch for using the modified mutation
       function "GAMutationNetwork" (TRUE) or the genetic algorithm's default (
       FALSE).
55
56 ga.use.parallel = TRUE #Manual switch for the usage of multiple CPU cores/
       threads. Use TRUE if multiple CPU threads/cores shall be used for the
       genetic algorithm. FALSE if just one thread/core shall be used. The
```

```
      script only uses multiple CPU cores/thread if your computer supports this
      . Therefore using TRUE unless problems occur is the recommended option
      here.
57 ga.minimum.popSize = 40 #The population used in the genetic algorithm has to
      have a population of at least [what is defined here]. "At least" means
      here, that the popSize will either be the here [what is defined here] or
      the dimension of the input matrix (including the dummies). A smaller
      population will increase the computation time per iteration but reduce
      the genetic algorithm's effectiveness as it might need more iterations to
       minimize the the number of errors. Has to be >= 30 to use the
      suggestions matrix.
58
59
60 #
61 # Importing sample (dis)similarity data ----------------------------------
62 #
63
64 ###Random data
65 ##Normal distribution - Rounded
66 #Options (are also used for not rounded numbers)
67 inputmatrix.random.norm.dimension = 10 #Sets the number of objects that shall
      be represented
68 inputmatrix.random.norm.mean = 10
69 inputmatrix.random.norm.sd = 1.5
70
71 #Generation
72 inputmatrix.random.norm.length.of.vector = (inputmatrix.random.norm.dimension^2
       - inputmatrix.random.norm.dimension) / 2
73
74 inputmatrix.random.norm.round.vec <- rnorm(inputmatrix.random.norm.length.of.
      vector, mean = inputmatrix.random.norm.mean, sd = inputmatrix.random.norm
      .sd)
75
76 inputmatrix.random.norm.round.vec <- abs(inputmatrix.random.norm.round.vec)
77
78 inputmatrix.random.norm.round.vec <- round(inputmatrix.random.norm.round.vec)
79
80 #The following package is needed for the vec2sm command:
81 if(require("corpcor")) {require("corpcor")} else {
82   install.packages("corpcor")
83   require("corpcor")
84 }
85
86 inputmatrix.random.norm.round <- vec2sm(inputmatrix.random.norm.round.vec, diag
       = FALSE)
87 diag(inputmatrix.random.norm.round) <- 0 #The diagonal has to be zero.
88
89
```

```
90  inputmatrix.random.norm.names <- colnames(matrix(0,inputmatrix.random.norm.
        dimension, inputmatrix.random.norm.dimension), do.NULL = FALSE, prefix =
        "Point␣")
91
92  colnames(inputmatrix.random.norm.round) <- inputmatrix.random.norm.names
93  rownames(inputmatrix.random.norm.round) <- inputmatrix.random.norm.names
94
95
96  ##Normal distribution - Not rounded
97  #Uses the options from above to generate the random numbers
98
99  #Generation
100 inputmatrix.random.norm.length.of.vector = (inputmatrix.random.norm.dimension^2
        - inputmatrix.random.norm.dimension) / 2
101
102 inputmatrix.random.norm.vec <- rnorm(inputmatrix.random.norm.length.of.vector,
        mean = inputmatrix.random.norm.mean, sd = inputmatrix.random.norm.sd)
103
104 inputmatrix.random.norm.vec <- abs(inputmatrix.random.norm.vec)
105
106 inputmatrix.random.norm <- vec2sm(inputmatrix.random.norm.vec, diag = FALSE)
107 diag(inputmatrix.random.norm) <- 0 #The diagonal has to be zero.
108
109
110 inputmatrix.random.norm.names <- colnames(matrix(0,inputmatrix.random.norm.
        dimension, inputmatrix.random.norm.dimension), do.NULL = FALSE, prefix =
        "Point␣")
111
112 colnames(inputmatrix.random.norm) <- inputmatrix.random.norm.names
113 rownames(inputmatrix.random.norm) <- inputmatrix.random.norm.names
114
115
116 #
117 # Defining the inputmatrix, which shall be visualized as network ----------
118 #
119
120 # The diagonal of this symmetric matrix has to be zero.
121
122 ###Deciding which data will be used
123 #Here it is defined which data is set as "inputmatrix". "inputmatrix" is the
        matrix this code tries to represent in network space.
124
125 ##Samples that requires higher.value.means.higher.similarity == 0
126 inputmatrix <- inputmatrix.random.norm
127 # inputmatrix <- inputmatrix.random.norm.round
128
129 ##Samples that requires higher.value.means.higher.similarity == 1
130 # inputmatrix <- inputmatrix.random.norm
131 # inputmatrix <- inputmatrix.random.norm.round
```

```
132
133 #TODO: Add checks for the inputmatrix like diagonal == 0, no negative values,
        etc.
134
135
136 #
137 # Loading/installing required R packages ---------------------------------
138 #
139
140 #Isotone/monotonistic regression
141 if(require("isotone")) {require("isotone")} else {
142   install.packages("isotone")
143   require("isotone")
144 }
145
146 #Used for symmetric matrix -> vector (and back) calculation
147 if(require("corpcor")) {require("corpcor")} else {
148   install.packages("corpcor")
149   require("corpcor")
150 }
151
152 #Routines for simple graphs and network analysis. igraph can handle large
        graphs very well and provides functions for generating random and regular
         graphs, graph visualization, centrality indices and much more.
153 if(require("igraph")) {require("igraph")} else {
154   install.packages("igraph")
155   require("igraph")
156 }
157
158 #Contains the genetic algorithm
159 if(require("GA")) {require("GA")} else {
160   install.packages("GA")
161   require("GA")
162 }
163
164 #Functions from Venables and Ripley's "Modern Applied Statistics with S".
        Example: Classic MDS. Package is not neccessary.
165 if(require("MASS")) {require("MASS")} else {
166   install.packages("MASS")
167   require("MASS")
168 }
169
170 #SMACOF package. Offers a wide range of practical tools for MDS and can compute
         metric and ordinal MDS.
171 if(require("smacof")) {require("smacof")} else {
172   install.packages("smacof")
173   require("smacof")
174 }
175
```

```
176 #The following packages allow the usage of multiple CPU cores in the genetic
        algorithm (some of the packages will only be installed/loaded if the
        computer running this script has multiple CPU cores)
177 if(require("parallel")) {require("parallel")} else {
178   install.packages("parallel")
179   require("parallel")
180 }
181
182 #The following packages are only loaded/installed on computers with multi-core
        CPU.
183 if(detectCores(all.tests = TRUE, logical = TRUE) > 1) {
184     if(require("doParallel")) {require("doParallel")} else {
185     install.packages("doParallel")
186     require("doParallel")
187   }
188
189   if(require("foreach")) {require("foreach")} else {
190     install.packages("foreach")
191     require("foreach")
192   }
193
194   if(require("iterators")) {require("iterators")} else {
195     install.packages("iterators")
196     require("iterators")
197   }
198 }
199
200 #Needed for text output in PDFs
201 if(require("gplots")) {require("gplots")} else {
202   install.packages("gplots")
203   require("gplots")
204 }
205
206
207 #
208 # Functions ------------------------------------------------------------------
209 #
210
211 ###Functions for the transformation to match the requirements of the genetic
        algorithm
212 ##Transforming a symmetric matrix into a vector of all possible links (based on
         the lower triagonal entries of a symmetric matrix)
213 SymMatrixToVector <- function(input) {
214   #sm2vec takes a symmetric matrix and puts the lower triagonal entries into a
         vector
215   #Function has to be symmetric
216   vectoroflowertriagonal <- sm2vec(input, diag = FALSE) #TOD: Mittels eines
         Packages geloest, evtl. will ich das spaeter aber ja anders machen.
217   return(vectoroflowertriagonal)
```

```
}

##Transforming a vector into a symmetric matrix
VectorToSymMatrix <- function(input) {
  input <- vec2sm(input, diag = FALSE)
  diag(input) <- 0 #The diagonal has to be zero.
  return(input)
}

##Calculating the dimension of the inputmatrix
DimensionAsSingleInteger <- function(input) {
  dim.integer <- length(diag(input)) #This works because the inputmatrix has to
        be symmetric.
  return(dim.integer)
}

##Calculating the number of possible links
NumberOfPossibleLinks <- function(dimension.input) { #Input must be the
      dimension of a symmetric matrix as integer
  number.of.possible.links <- (dimension.input^2 - dimension.input) / 2
  return(number.of.possible.links)
}

##Inverse of metric input data
#Note to self: Never add highmeanssimilarity here as this function is used an
      two different parts in this code.
InverseOfMetricSymMatrix <- function(input) {
  input.max <- max(input)
  input <- (input.max + 1 / input.max) - input #Linear transformation of the
        input. Transforms the highest values into the smallest one. This
        transformation is better than a division of the input through the input
        .max.
    diag(input) <- 0
  return(input)
}

##Inverse of ordinal input data
#Note to self: Never add highmeanssimilarity here as this function is used an
      two different parts in this code.
InverseOfOrdinalSymMatrix <- function(input) {
  input.max <- max(input)
  diag(input) <- Inf #This is necessary to prevent the diagonal from being
        considered for min(input).
  input.min <- min(input)
  input <- input * (-1)
  input <- input + (input.max + input.min)
  diag(input) <- 0 #Reverse the diagonal back to 0.
  return(input)
}
```

```
259
260 ##Scaling of input data
261 ScalingOfSymMatrix <- function(input) {
262   # Scaling of input data to be between 0 and 1. This is only used in the
          generation of the heuristic network proposals / the suggestion matrix.
263   input <- abs(input) # Removing any negative values
264   input <- input / max(input)
265   return(input)
266 }
267
268 ##Rounding of the scaled inputmatrix (all values are rounded according to a
      certain cutoff point to be either 1 or 0). This is mainly used for
      generating the suggestions matrix
269 CutoffRounding <- function (data, cutoff.point) {
270   #Input "data" can be a matrix, a vector or an integer.
271   #Input "cufoff.point" is the threshold for the rounding.
272   if(cutoff.point > 1 || cutoff.point <= 0) {stop("The value for the cutoff.
          point has to be larger than 0 and smaller than 1")}
273   if(max(data) > 1 || min(data) < 0) {stop("The data that shall be rounded by
          this function has to be scaled to be >= 0 and <= 1")}
274
275   data = data - (cutoff.point - 0.5) #This way is simpler than running a loop
          and it returns the same results, as no value in the input data can be
          larger than 1 and no value after this line is smaller than -0.5 (which
          is only the case in the extreme case of cutoff point = 1), which is
          rounded to 0.
276   data = round(data)
277   return(data)
278 }
279
280 ##Defining a matrix as adjacency matrix
281 DefineAsAdjacencyMatrix <- function(input) {
282   input <- graph.adjacency(input, mode="undirected", weighted=NULL, diag=TRUE)
283   return(input)
284 }
285
286 ##Generating a distance matrix from an adjacency matrix
287 GenerateDistanceMatrix <- function(input) {
288   #TODO Add a check that only adjacency matrices can be the input.
289   input <- shortest.paths(input, mode="all", weights=NULL, algorithm="automatic
          ") #Generates a matrix with the distances
290   return(input)
291 }
292
293
294 ###General plot options, realized as functions
295 PlotNetwork <- function(input, show.errors, manual.vertex.label.cex,manual.
        vertex.size) {
296   #Contains the standard options for the plot of network graphs.
```

```
297
298   #Accounting for possible missing input:
299   if(missing(manual.vertex.label.cex)) {manual.vertex.label.cex <- .8}
300   if(missing(manual.vertex.size)) {manual.vertex.size <- 8}
301   if(missing(show.errors)) {show.errors <- TRUE}
302
303   if(show.errors == TRUE) {
304     plot(input, main="Network Algorithm Fit", sub=(paste("Number of Errors:",
            network.output.error.count.best.solution, ", Stress-1:", network.
            stress1.best.solution)), layout=layout.auto, frame=FALSE, vertex.
            shape="circle", vertex.color="grey", vertex.frame.color="darkgrey",
            vertex.size=manual.vertex.size, vertex.label.cex=manual.vertex.label.
            cex, vertex.label.color="black", vertex.label.family="sans", vertex.
            label.dist=0, edge.color="darkgrey")
305   }
306
307   if(show.errors == FALSE) {
308     plot(input, layout=layout.auto, frame=FALSE, vertex.shape="circle", vertex.
            color="grey", vertex.frame.color="darkgrey", vertex.size=manual.
            vertex.size, vertex.label.cex=manual.vertex.label.cex, vertex.label.
            color="black", vertex.label.family="sans", vertex.label.dist=0, edge.
            color="darkgrey")
309   }
310   ##Notes:
311   #See: ?igraph.plotting and ?plot.igraph for further options.
312   #vertex.label.cex controls the size of the labels. Default is 1.
313   #vertex.label.degree can only be used if vertex.label.dist=1.
314   #margin sets margins of the plot. Default is 0.
315   #frame=TRUE would generate a frame.
316 }
317
318
319 PlotMetricMDS <- function(input, show.errors, manual.cex) {
320   #Input "show.errors" offers more information on the plot.
321   #The manual.cex input currently only affects the size of the nodes, not the
          labels. This might be fixed in an upcomming version of the smacof
          package.
322
323   #Accounting for possible missing input:
324   if(missing(manual.cex)) {manual.cex <- .8}
325   if(missing(show.errors)) {show.errors <- TRUE}
326
327   if(show.errors == TRUE) {
328     plot(input, plot.type = "confplot", plot.dim = c(1,2), main = "Metric MDS
            Fit", sub=(paste("Number of Errors:", mds.output.error.count.best.
            solution, ", Stress-1:", mds.stress1.best.solution)), xlab="Distance"
            , ylab="Distance", type="p", cex=manual.cex, label.conf = list(label
            = TRUE, pos = 1, col = 1), sphere = FALSE)
329   }
```

```
330
331   if(show.errors == FALSE) {
332     plot(input, plot.type = "confplot", plot.dim = c(1,2), xlab="Distance",
            ylab="Distance", type="p", cex=manual.cex, label.conf = list(label =
            TRUE, pos = 1, col = 1), sphere = FALSE)
333   }
334   #Note: identify = TRUE is a nice function for huge data sets.
335 }
336
337
338 PlotOrdinalMDS <- function(input, show.errors, manual.cex) {
339   #Input "show.errors" offers more information on the plot.
340   #The manual.cex input currently only affects the size of the nodes, not the
          labels. This might be fixed in an upcomming version of the smacof
          package.
341
342   #Accounting for possible missing input:
343   if(missing(manual.cex)) {manual.cex <- .8}
344   if(missing(show.errors)) {show.errors <- TRUE}
345
346   if(show.errors == TRUE) {
347     plot(input, plot.type = "confplot", plot.dim = c(1,2), main = "Ordinal␣MDS␣
            Fit", sub=(paste("Number␣of␣Errors:", mds.output.error.count.best.
            solution, ",␣Stress-1:", mds.stress1.best.solution)), xlab="Distance"
            , ylab="Distance", type="p", cex=manual.cex, label.conf = list(label
            = TRUE, pos = 1, col = 1), sphere = FALSE)
348   }
349
350   if(show.errors == FALSE) {
351     plot(input, plot.type = "confplot", plot.dim = c(1,2), main = "", xlab="
            Distance", ylab="Distance", type="p", cex=manual.cex, label.conf =
            list(label = TRUE, pos = 1, col = 1), sphere = FALSE)
352   }
353   #Note: identify = TRUE is a nice function for huge data sets.
354 }
355
356 PlotGAIterations <- function(input, manual.cex.points) {
357   #Plots the fitness of the population at each iteration step.
358
359   #Accounting for possible missing input:
360   if(missing(manual.cex.points)) {manual.cex.points <- .8}
361
362   #Input has to be an object of the class ga (which is the standard output of a
          genetic algorithm in the package used in this file)
363   plot(input, col = c("black", "darkgrey"), cex.points = manual.cex.points, lty
          = c(1,1), lwd=3, pch = c(20, 20)) #Shows information on the iterations
          of the genetic algorithm.
364 }
365
```

```
366
367 ###MDS Functions
368
369 MetricMDS <- function(input) {
370   #Only dissimilarity data should be used as input. The input has to be in
          matrix form.
371   smacofSym(input, ndim = 2, weightmat = NULL, init = NULL, metric = TRUE, ties
          = "primary", verbose = TRUE, relax = FALSE, modulus = 1, itmax = 1000,
          eps = 1e-06)
372 }
373
374 OrdinalMDS <- function(input) {
375   #Only dissimilarity data should be used as input. The input has to be in
          matrix form.
376   smacofSym(input, ndim = 2, weightmat = NULL, init = NULL, metric = FALSE,
          ties = "primary", verbose = TRUE, relax = FALSE, modulus = 1, itmax =
          1000, eps = 1e-06)
377 }
378
379
380 #
381 # Modified Genetic Algorithm Mutation Function ----------------------------
382 #
383
384 ###Probalities for the genetic algorithm mutation function.
385 #These are defined here (outside of the GAMutationNetwork function) to reduce
      computation time.
386
387 #The probability of a certain mutation type is given by the following
      information. Important: The sum of all probabilities has of course to be
      1.
388 ga.mutation.prob.1.obj <- 0.30 #Probability that only 1 object will be mutated
389 ga.mutation.prob.2.obj <- 0.25
390 ga.mutation.prob.3.obj <- 0.15
391 ga.mutation.prob.5.obj <- 0.02
392 ga.mutation.prob.1.percent <- 0.15 #Probability that 1 percent of all objects
      will be mutated (the exact number will be rounded)
393 ga.mutation.prob.3.percent <- 0.05
394 ga.mutation.prob.5.percent <- 0.03
395 ga.mutation.prob.10.percent <- 0.02
396 ga.mutation.prob.15.percent <- 0.02
397 ga.mutation.prob.20.percent <- 0.01
398
399 #Cumulative probabilities:
400 ga.mutation.cum.prob.1.obj <- sum(ga.mutation.prob.1.obj)
401 ga.mutation.cum.prob.2.obj <- sum(ga.mutation.cum.prob.1.obj,ga.mutation.prob
      .2.obj)
402 ga.mutation.cum.prob.3.obj <- sum(ga.mutation.cum.prob.2.obj,ga.mutation.prob
      .3.obj)
```

```
403 ga.mutation.cum.prob.5.obj <- sum(ga.mutation.cum.prob.3.obj,ga.mutation.prob
        .5.obj)
404 ga.mutation.cum.prob.1.percent <- sum(ga.mutation.cum.prob.5.obj,ga.mutation.
        prob.1.percent)
405 ga.mutation.cum.prob.3.percent <- sum(ga.mutation.cum.prob.1.percent,ga.
        mutation.prob.3.percent)
406 ga.mutation.cum.prob.5.percent <- sum(ga.mutation.cum.prob.3.percent,ga.
        mutation.prob.5.percent)
407 ga.mutation.cum.prob.10.percent <- sum(ga.mutation.cum.prob.5.percent,ga.
        mutation.prob.10.percent)
408 ga.mutation.cum.prob.15.percent <- sum(ga.mutation.cum.prob.10.percent,ga.
        mutation.prob.15.percent)
409 ga.mutation.cum.prob.20.percent <- sum(ga.mutation.cum.prob.15.percent,ga.
        mutation.prob.20.percent) #This value has to be ==1, otherwise the
        entered ga.mutation.prob values are larger than in total.
410
411
412 #The function:
413 GAMutationNetwork <- function(object, parent, ...) {
414   #TODO Add description.
415   mutate <- parent <- as.vector(object@population[parent, ])
416   n <- length(parent)
417
418   randomizer <- runif(1, min = 0, max = 1)
419
420   ###Allowing a mutation in more than just one link/relation of two objects (
          mutation in just one object is the standard of the ga-package and not
          perfectly fitted for the network case).
421   #This uses the values of ga.mutation.prob... (and ga.mutation.cum.prob).
422
423
424   #Case 1 object:
425   if(randomizer <= ga.mutation.cum.prob.1.obj) {
426     j <- sample(1:n, size = 1) #Only one element of the solution vector is
            mutating.
427
428     mutate[j] <- abs(mutate[j] - 1)
429   }
430
431   #Case 2 objects:
432   if(randomizer > ga.mutation.cum.prob.1.obj & randomizer <= ga.mutation.cum.
          prob.2.obj) {
433     j <- sample(1:n, size = 2) #In this function more (two) than just one
            object of the possible solution vector are mutated.
434
435     for(i in 1:length(j)) {
436       mutate[j] <- abs(mutate[j] - 1)
437     }
438   }
```

```
439
440   #Case 3 objects:
441   if(randomizer > ga.mutation.cum.prob.2.obj & randomizer <= ga.mutation.cum.
          prob.3.obj) {
442     j <- sample(1:n, size = 3)
443
444     for(i in 1:length(j)) {
445       mutate[j] <- abs(mutate[j] - 1)
446     }
447   }
448
449   #Case 5 objects:
450   if(randomizer > ga.mutation.cum.prob.3.obj & randomizer <= ga.mutation.cum.
          prob.5.obj) {
451     j <- sample(1:n, size = 5) #In this function more than just one object of
            the possible solution vector are mutated.
452
453     for(i in 1:length(j)) {
454       mutate[j] <- abs(mutate[j] - 1)
455     }
456   }
457
458
459   #Case 1 percent of all objects:
460   if(randomizer > ga.mutation.cum.prob.5.obj & randomizer <= ga.mutation.cum.
          prob.1.percent) {
461     j <- sample(1:n, size = round(max(1,n*0.01))) #In this function 1% of the
            objects of the possible solution vector are mutated.
462
463     for(i in 1:length(j)) {
464       mutate[j] <- abs(mutate[j] - 1)
465     }
466   }
467
468   #Case 3 percent of all objects:
469   if(randomizer > ga.mutation.cum.prob.1.percent & randomizer <= ga.mutation.
          cum.prob.3.percent) {
470     j <- sample(1:n, size = round(max(1,n*0.03)))
471
472     for(i in 1:length(j)) {
473       mutate[j] <- abs(mutate[j] - 1)
474     }
475   }
476
477   #Case 5 percent of all objects:
478   if(randomizer > ga.mutation.cum.prob.3.percent & randomizer <= ga.mutation.
          cum.prob.5.percent) {
479     j <- sample(1:n, size = round(max(1,n*0.05)))
480
```

```
481     for(i in 1:length(j)) {
482       mutate[j] <- abs(mutate[j] - 1)
483     }
484   }
485
486   #Case 10 percent of all objects:
487   if(randomizer > ga.mutation.cum.prob.5.percent & randomizer <= ga.mutation.
          cum.prob.10.percent) {
488     j <- sample(1:n, size = round(max(1,n*0.1)))
489
490     for(i in 1:length(j)) {
491       mutate[j] <- abs(mutate[j] - 1)
492     }
493   }
494
495   #Case 15 percent of all objects:
496   if(randomizer > ga.mutation.cum.prob.10.percent & randomizer <= ga.mutation.
          cum.prob.15.percent) {
497     j <- sample(1:n, size = round(max(1,n*0.15)))
498
499     for(i in 1:length(j)) {
500       mutate[j] <- abs(mutate[j] - 1)
501     }
502   }
503
504   #Case 20 percent of all objects:
505   if(randomizer > ga.mutation.cum.prob.15.percent & randomizer <= ga.mutation.
          cum.prob.20.percent) {
506     j <- sample(1:n, size = round(max(1,n*0.20)))
507
508     for(i in 1:length(j)) {
509       mutate[j] <- abs(mutate[j] - 1)
510     }
511   }
512
513   return(mutate)
514 }
515
516
517 #
518 # Generating various useful objects ----------------------------------------
519 #
520
521 #Generating variables with the dimension of the input data
522 inputmatrix.dimension <- DimensionAsSingleInteger(inputmatrix)
523 inputmatrix.dimension.incl.dummies <- inputmatrix.dimension + dummy.nodes #w.r.
        t. to the dummy nodes
524
525 #Generating variables with the number of possible links (in the network)
```

```
526 inputmatrix.possible.links <- NumberOfPossibleLinks(inputmatrix.dimension)
527 inputmatrix.possible.links.incl.dummies <- NumberOfPossibleLinks(inputmatrix.
      dimension.incl.dummies) #w.r.t. to the dummy nodes
528
529 #Defining the population size for the genetic algorithm (depends on definied ga
      .minimum.popSize and the dimension of the input matrix plus dummies)
530 ga.popSize = max(ga.minimum.popSize,inputmatrix.dimension.incl.dummies) #The
      popSize be at least as large as the dimension as of original input matrix
       (without any dummies). Including the dimension of input matrix plus
      dummies is necessary due to the size of the suggestions matrix.
531
532 #Share of best solutions that "survives" each iteration/generation of genetic
      algorithm due to the elitism option.
533 ga.elitism <- max(3, round(ga.popSize * ga.elitism.popsize.share)) #The number
      of best fitness individuals to survive at each generation.
534
535 ###Taking care that input data is transformed into dissimilarity data
536 if(higher.value.means.higher.similarity == 0) {inputmatrix.diss.matrix <-
      inputmatrix}
537 if(higher.value.means.higher.similarity == 1 & scale.of.input.data == "metric")
       {inputmatrix.diss.matrix <- InverseOfMetricSymMatrix(inputmatrix)}
538 if(higher.value.means.higher.similarity == 1 & scale.of.input.data == "ordinal"
      ) {inputmatrix.diss.matrix <- InverseOfOrdinalSymMatrix(inputmatrix)}
539
540
541 ###Generating the comparison matrix.
542 inputmatrix.diss.vector <- SymMatrixToVector(inputmatrix.diss.matrix) #
      Transforming this matrix into a vector
543
544 comparison.matrix <- matrix(0,3,inputmatrix.possible.links)
545
546 rownames(comparison.matrix) <- c("Input", "Network", "MDS") #Naming the rows.
547
548 comparison.matrix[1,] <- inputmatrix.diss.vector #Its first row consists of the
       dissimilarity data from the input data.
549 #The second row will be used for the best distance network solution below.
550 #The third row will be used for the best distance MDS solution below.
551
552 #
553 # Suggestions matrix for the genetic algorithm ----------------------------
554 #
555
556
557 ###Generating the suggestions matrix (matrix of solution that is included in
      the initial population of the genetic algorithm)
558
559 ##Some important network types
560 complete.network <- as.vector(matrix(1,1,inputmatrix.possible.links.incl.
      dummies))
```

```
561
562 empty.network <- as.vector(matrix(0,1,inputmatrix.possible.links.incl.dummies))
563
564
565 ###Heuristic network solution (here every value of the scaled inputmatrix is
        converted in a way that values >="Cutoff point" -> 1 and <"Cutoff point"
        ->0)
566 ##Generating a new helper matrix that contains the values and form of the
        original inputmatrix, but its "values"/"entries" are scaled to be between
         1 and 0.
567 heuristic.network.helper <- inputmatrix.diss.matrix
568
569 #Scaling of this matrix (highest value = 1)
570 heuristic.network.helper <- ScalingOfSymMatrix(heuristic.network.helper)
571
572 #Accounting for dummy nodes (by attaching them to this matrix)
573 heuristic.network.dummy.cols <- matrix(0,inputmatrix.dimension,dummy.nodes)
574 heuristic.network.dummy.rows <- matrix(0,dummy.nodes,inputmatrix.dimension.incl
        .dummies)
575
576 heuristic.network.helper <- cbind(heuristic.network.helper, heuristic.network.
        dummy.cols) #Attaching the columns
577 heuristic.network.helper <- rbind(heuristic.network.helper, heuristic.network.
        dummy.rows) #Attaching the rows
578
579 #Transforming this into a vector of all possible links from the scaled (
        symmetric) inputmatrix
580 heuristic.network.helper <- SymMatrixToVector(heuristic.network.helper)
581
582 ##Heuristic network solution with different cutoff points (here every value of
        the scaled inputmatrix is converted in a way that values >=[Cutoff-Point]
         -> 1 (=form a link) and <[Cutoff-Point] ->0)
583 #These vectors could also have put into a single matrix.
584 heuristic.network.0.1 <- CutoffRounding(heuristic.network.helper,0.1)
585 heuristic.network.0.2 <- CutoffRounding(heuristic.network.helper,0.2)
586 heuristic.network.0.3 <- CutoffRounding(heuristic.network.helper,0.3)
587 heuristic.network.0.4 <- CutoffRounding(heuristic.network.helper,0.4)
588 heuristic.network.0.5 <- CutoffRounding(heuristic.network.helper,0.5)
589 heuristic.network.0.6 <- CutoffRounding(heuristic.network.helper,0.6)
590 heuristic.network.0.7 <- CutoffRounding(heuristic.network.helper,0.7)
591 heuristic.network.0.8 <- CutoffRounding(heuristic.network.helper,0.8)
592 heuristic.network.0.9 <- CutoffRounding(heuristic.network.helper,0.9)
593
594 #Deleting helper variables
595 rm(heuristic.network.helper, heuristic.network.dummy.cols, heuristic.network.
        dummy.rows) #The heuristic.network.helper and attached cols and rows are
        no longer needed.
596
597 ###Generating the suggestions matrix
```

```
598 ga.suggestions.matrix <- matrix(runif(inputmatrix.possible.links.incl.dummies*
      ga.popSize, min = 0, max = 1),ga.popSize,inputmatrix.possible.links.incl.
      dummies)
599
600 ##Adding some special networks to the suggestion matrix
601 ga.suggestions.matrix[1:2,] <- complete.network #The suggestions matrix now has
       two times the complete network in it.
602 ga.suggestions.matrix[3:4,] <- empty.network
603 #The element 5 to 10 could be used for other networks
604
605 #Heuristic networks
606 ga.suggestions.matrix[11,] <- heuristic.network.0.1
607 ga.suggestions.matrix[12,] <- heuristic.network.0.2
608 ga.suggestions.matrix[13,] <- heuristic.network.0.3
609 ga.suggestions.matrix[14,] <- heuristic.network.0.4
610 ga.suggestions.matrix[15,] <- heuristic.network.0.5
611 ga.suggestions.matrix[16,] <- heuristic.network.0.6
612 ga.suggestions.matrix[17,] <- heuristic.network.0.7
613 ga.suggestions.matrix[18,] <- heuristic.network.0.8
614 ga.suggestions.matrix[19,] <- heuristic.network.0.9
615
616
617 ##Specifying certain probabilities of forming a link for the randomly generated
       rows in the suggestion matrix
618 #Prob of forming a link according to Erdös-Renyi (log n / n). The "log" command
       computes the natural logarithm in R.
619 prob.forming.link.erdos.renyi <- log(inputmatrix.dimension.incl.dummies) /
      inputmatrix.dimension.incl.dummies #It is also accounted for dummy nodes.
620
621
622 ##Continuing with generating the suggestions matrix
623 ga.suggestions.matrix[20:23,] <- CutoffRounding(ga.suggestions.matrix
      [20:23,],0.3) #Rounding with 0.3 as cutoff-point.
624 ga.suggestions.matrix[24:ga.popSize,] <- CutoffRounding(ga.suggestions.matrix
      [24:ga.popSize,],prob.forming.link.erdos.renyi) #Rounding with respect to
       the Erdös-Renyi prob. of forming a link.
625
626 ga.suggestions.matrix <- CutoffRounding(ga.suggestions.matrix,0.3) #Any row
      that has not been rounded/transformed above are now transformed into ones
       or zeros. This is necessary as the binary form of the genetic algorithm
      is used below.
627
628 ##Using the ga.use.suggestions.matrix switch to turn the use of a suggestions
      matrix off
629 if(ga.use.suggestions.matrix == FALSE) {ga.suggestions.matrix <- as.vector(
      matrix(0,1,inputmatrix.possible.links.incl.dummies))}
630
631
632 #
```

```
633 # Fitness functions ----------------------------------------------------
634 #
635
636 ###Metric input data
637 ##The following fitness function for metric data can be used in the GA fitness
        function and to calculate fitness of MDS results
638 #Note to self: This fitness function is just an idea and has not been tested as
        much as the ordinal one.
639
640 FitnessMetric <- function(comparison.matrix) {
641   ##Measuring the fitness
642   fitness <- 0 #Generating a variable that measures the fitness
643
644   sum.distances.possible.solution <- sum(comparison.matrix[2,])
645   sum.distances.inputmatrix.diss.vector <- sum(comparison.matrix[1,])
646
647   fitness <- sum((comparison.matrix[2,] / sum.distances.possible.solution -
          comparison.matrix[1,] / sum.distances.inputmatrix.diss.vector)^2)
648
649   return(fitness)
650 }
651
652 ##Fitness function that is used by the genetic algorithm only (not by MDS)
653 GAFitness.Metric <- function(input) {
654   #Input has to be a binary solution vector
655
656   ###Transformations
657   ##Transformation of the possible solution "input", which is a vector
          containing information on all possible links
658   input <- VectorToSymMatrix(input) #Transforming the vector into a matrix
659   input <- DefineAsAdjacencyMatrix(input) #Defining this matrix as adjacency
          matrix (required for the igraph package, which generates the distance
          matrix)
660   #Transformation of this adjacency matrix into a distance matrix
661   input <- GenerateDistanceMatrix(input)
662   #TODO Note to self: Currently unconnected nodes lead to a distance of inf for
          that set of nodes. One might add a maximum value for this case here.
663
664   input <- input[1:inputmatrix.dimension,1:inputmatrix.dimension] #Any dummy
          node is deleted from the input as the dummies have no counter-part in
          the original inputmatrix and they also shall not directly affect the
          fitness function.
665
666   possible.solution.vector <- SymMatrixToVector(input) #Transforming this
          distance matrix into a vector containing all distances of the network.
667
668
669   ###Comparison of the network distances with the original input data
```

```
670   ##The comparison.matrix was generated above. Its first row contains the
          dissimilarity values of the original inputmatrix.
671   comparison.matrix[2,] <- possible.solution.vector
672
673
674   ##Calculating the fitness
675   fitness <- FitnessMetric(comparison.matrix) #The first part of the
          calculation of the errors is done in a seperate function. That way the
          that fitness function can be called by MDS and genetic algorithm.
676
677   fitness.invers <- (-1) * fitness #As this package for the genetic algorithm
          tries to maximize the fitness and I use "errors" in the function here,
          the counted errors are transformed to be negative.
678
679   ##Output
680   return(fitness.invers)
681 }
682
683
684 ###Ordinal input data - Minimizing order-relation-errors version
685 ##The following fitness function for ordinal data can be used in the GA fitness
        function and to calculate fitness of MDS results
686
687 FitnessOrdinal <- function(comparison.matrix) {
688   ##Counting the order violations
689   counterrors <- 0 #Generating a variable that counts the errors
690
691   for(i in 1:(inputmatrix.possible.links - 1)) {
692     for(j in (i+1):inputmatrix.possible.links) {
693
694       #Accounting for the problem that Inf - Inf = NaN:
695       #Both of the following conditions are required.
696       if((is.nan(comparison.matrix[2,i] - comparison.matrix[2,j]) & (comparison
              .matrix[1,i] - comparison.matrix[1,j] != 0))) {
697         counterrors <- counterrors + 1
698         next
699       }
700
701       if(is.nan(comparison.matrix[2,i] - comparison.matrix[2,j]) & (comparison.
              matrix[1,i] - comparison.matrix[1,j] == 0)) {next} #This case does
              not generate an error as Inf distance in network is fine for a 0 in
               the dissimilarity matrix.
702
703
704       if((comparison.matrix[2,i] - comparison.matrix[2,j] > 0) & (comparison.
              matrix[1,i] - comparison.matrix[1,j] <= 0)) {counterrors <-
              counterrors + 1}
```

```
705       if((comparison.matrix[2,i] - comparison.matrix[2,j] < 0) & (comparison.
              matrix[1,i] - comparison.matrix[1,j] >= 0)) {counterrors <-
              counterrors + 1}
706       if((comparison.matrix[2,i] - comparison.matrix[2,j] == 0) & (comparison.
              matrix[1,i] - comparison.matrix[1,j] != 0)) {counterrors <-
              counterrors + 1}
707     }
708   }
709
710   return(counterrors)
711 }
712
713 ##Fitness function that is used by the genetic algorithm only (not by MDS)
714 GAFitness.Ordinal <- function(input) {
715   #Input has to be a binary solution vector
716
717   ###Transformations
718   ##Transformation of the possible solution "input", which is a vector
          containing information on all possible links
719   input <- VectorToSymMatrix(input) #Transforming the vector into a matrix
720   input <- DefineAsAdjacencyMatrix(input) #Defining this matrix as adjacency
          matrix (required for the igraph package, which generates the distance
          matrix)
721   #Transformation of this adjacency matrix into a distance matrix
722   input <- GenerateDistanceMatrix(input)
723   #TODO Note to self: Currently unconnected nodes lead to a distance of inf for
           that set of nodes. One might add a maximum value for this case here.
724
725   input <- input[1:inputmatrix.dimension,1:inputmatrix.dimension] #Any dummy
          node is deleted from the input as the dummies have no counter-part in
          the original inputmatrix and they also shall not directly affect the
          fitness function.
726
727   possible.solution.vector <- SymMatrixToVector(input) #Transforming this
          distance matrix into a vector containing all distances of the network.
728
729
730   ###Comparison of the network distances with the original input data
731   ##The comparison.matrix was generated above. Its first row contains the
          dissimilarity values of the original inputmatrix.
732   comparison.matrix[2,] <- possible.solution.vector
733
734
735   ##Calculating the fitness
736   counterrors <- FitnessOrdinal(comparison.matrix) #The first part of the
          calculation of the errors is done in a seperate function. That way the
          that fitness function can be called by MDS and genetic algorithm.
737
```

```
738   counterrors.invers <- (-1) * counterrors #As this package for the genetic
          algorithm tries to maximize the fitness and I use "errors" in the
          function here, the counted errors are transformed to be negative.
739
740   ##Output
741   return(counterrors.invers)
742 }
743
744
745 FitnessOrdinalStress1 <- function(input,type) {
746   #Computes the Stress-1 value for networks and ordinal MDS. Can not be used
          within the genetic algorithm as a Stress-1 optimization might lead to
          an empty or complete network.
747   #input has to be the comparison matrix (or a matrix of similar form).
748   #type has to be either "network" or "mds"
749
750   #Stress-1 for networks:
751   if(type == "network") {
752     #Unconnected network case:
753     if(max(input[2,]) == Inf | max(input[2,]) == 0) { #Case for handling the
            case of isolate. Currently, the Stress-1 value will be defined as
            infinite in an unconnected network.
754       kruskal.stress_1 = Inf
755     }
756
757     #Connected network case:
758     if(max(input[2,]) != Inf & max(input[2,]) != 0) {
759       ##Monotonistic regression (using the isotone R package)
760       mono.regression.network <- gpava(input[1,], input[2,], weights = NULL,
              solver = weighted.mean, ties = "primary") #Monotonistic regression.
               Generalized Pooled-Adjacent-Violators Algorithm. Primary approach
              for ties is standard in MDS literature.
761
762       distances <- mono.regression.network$y #The distances (d_ij) from the
              possible solution provides by the GA. This is the same as input
              [2,].
763
764       predicted.distances <- mono.regression.network$x #The fitted values of
              the monotonistic regression. In the Kruskal framework this would be
               named d-hat_ij.
765
766       kruskal.raw.stress <- sum((distances - predicted.distances)^2) #Kruskal
              (1964)'s raw stress (S^*).
767
768       kruskal.scaling.factor <- sum((distances)^2) #This scaling factor is
              called T^* by Kruskal (1964).
769
770       kruskal.stress_1 <- sqrt(kruskal.raw.stress / kruskal.scaling.factor)
771     }
```

```
772   }
773
774   #Stress-1 for ordinal MDS:
775   if(type == "mds" & scale.of.input.data == "ordinal") {
776     if(max(input[3,]) != Inf & max(input[3,]) != 0) {
777       ##Monotonistic regression (using the isotone R package)
778       mono.regression.network <- gpava(input[1,], input[3,], weights = NULL,
              solver = weighted.mean, ties = "primary") #Monotonistic regression.
               Generalized Pooled-Adjacent-Violators Algorithm. Primary approach
              for ties is standard in MDS literature.
779
780       distances <- mono.regression.network$y #The distances (d_ij) from the
              possible solution provides by the GA. This is the same as input
              [2,].
781
782       predicted.distances <- mono.regression.network$x #The fitted values of
              the monotonistic regression. In the Kruskal framework this would be
               named d-hat_ij.
783
784       kruskal.raw.stress <- sum((distances - predicted.distances)^2) #Kruskal
              (1964)'s raw stress (S^*).
785
786       kruskal.scaling.factor <- sum((distances)^2) #This scaling factor is
              called T^* by Kruskal (1964).
787
788       kruskal.stress_1 <- sqrt(kruskal.raw.stress / kruskal.scaling.factor) #
              Important hint: mds.output$stress.nm reports normalized Stress.
              That normalized Stress is Stress-1 squared.
789     }
790   }
791
792   return(kruskal.stress_1)
793 }
794
795
796 #
797 # Genetic algorithm -------------------------------------------------------
798 #
799
800 #Use of multiple CPU cores of the computer for the genetic algorithm.
801 if((detectCores(all.tests = TRUE, logical = TRUE) > 1) & ga.use.parallel & (
        isTRUE(require("parallel"))) & (isTRUE(require("doParallel"))) & (isTRUE(
        require("foreach"))) & (isTRUE(require("iterators")))) {ga.parallel <-
        TRUE} else {ga.parallel <- FALSE} #Is "TRUE" if there are more than one
        CPU threads, the required packages are installed and
802 ga.parallel #Test, whether parallel is used in GA.
803
804 #Defining which fitness function will be used in the genetic algorithm,
        depending on the scale of the input data
```

```
805 if(scale.of.input.data == "metric") {ga.fitness <- GAFitness.Metric}
806 if(scale.of.input.data == "ordinal") {ga.fitness <- GAFitness.Ordinal}
807
808 #Telling the genetic algorithm to use the modified mutation function ("
        GAMutationNetwork") if the according "switch" is TRUE. Otherwise the
        genetic algorithm's default will be used.
809 if(ga.use.modified.mutation == TRUE) {gaControl("binary" = list(mutation = "
        GAMutationNetwork"))} else{gaControl("binary" = list(mutation = "gabin_
        raMutation"))} #This is saved only for this R session and will be set to
        default ("gabin_raMutation") when starting R and the genetic algorithm
        package the next time. gaControl() lists the functions that are used in
        the genetic algorithm.
810
811 #Start the GA stopwatch
812 time.ga.start <- Sys.time()
813
814 #Test GA mit korrekter Fitness-Funktion
815 ga.output <- ga("binary", fitness = ga.fitness, maxiter = ga.maxiter, run = ga.
        run, elitism = ga.elitism, pcrossover = ga.pcrossover, pmutation = ga.
        pmutation, nBits = inputmatrix.possible.links.incl.dummies, popSize = ga.
        popSize, suggestions = ga.suggestions.matrix, parallel = ga.parallel,
        seed = c(0, 1))
816 #nBits ist extrem wichtig! popSize ist die Anzahl an Zeilen/Versuchen, an denen
         er zeitgleich rumprobiert. Glaube, dass man min = empty.network, max =
        complete.network nicht braucht.
817 #Giuseppe hat noch ein "'MutationFcn',{@mutationgaussian,1.5,0}" also eine
        eigene Mutation-Function im Code. Vielleicht sollte ich das auch machen.
818 #Ueberlegen, ob ich a) elitism reduziere, b) pcrossover reduziere oder c)
        pmutation erhoehe - da dies verhindern/reduzieren sollte, dass sich der
        Algorithmus zu sehr in eine Richtung verrennt.
819
820
821 #Stop the GA stopwatch
822 time.ga.stop <- Sys.time()
823
824
825 ###Benchmarks
826 #Time for the genetic algorithm optimization:
827 time.ga.in.min <- as.numeric(round(difftime(time.ga.stop, time.ga.start, units=
        "mins"),digits=2))
828 time.ga.in.hours <- as.numeric(round(difftime(time.ga.stop, time.ga.start,
        units="hours"),digits=2))
829 time.ga.auto <- round(difftime(time.ga.stop, time.ga.start, units="auto"),
        digits=2)
830
831 #
832 # Preperations for the presentation of the network results ----------------
833 #
834
```

```
835 ###Generating an adjacency and distance matrix of the best solution plus adding
        the geodesic distances into the comparison matrix
836
837 #Converting the result
838 ga.output.solution.best.igraph <- as.vector(ga.output@solution[1,]) #Due to the
        "[1,]", only the first row of the solution is shown in the following
      code. This is necessary as the solution matrix may contain more than just
        one best solution.
839 ga.output.solution.best.igraph <- VectorToSymMatrix(ga.output.solution.best.
      igraph)
840
841 #Naming
842 #Note: Could also be done differently. See: http://igraph.wikidot.com/r-traps
843 inputmatrix.labels.incl.dummies <- dimnames(inputmatrix)[1]
844 inputmatrix.labels.incl.dummies <- unlist(inputmatrix.labels.incl.dummies)
845
846 #Combining the original names from the input matrix with the dummy nodes
847 inputmatrix.labels.incl.dummies.helper <- colnames(matrix(0,dummy.nodes, dummy.
      nodes), do.NULL = FALSE, prefix = "Dummy␣") #Generates names for the
      dummy nodes
848
849 inputmatrix.labels.incl.dummies <- c(inputmatrix.labels.incl.dummies,
      inputmatrix.labels.incl.dummies.helper)
850 colnames(ga.output.solution.best.igraph) <- inputmatrix.labels.incl.dummies
851 rownames(ga.output.solution.best.igraph) <- inputmatrix.labels.incl.dummies
852
853 rm(inputmatrix.labels.incl.dummies.helper) #Removing the helper variables
854
855 ga.output.solution.best.igraph <- DefineAsAdjacencyMatrix(ga.output.solution.
      best.igraph) #Save the best solution as igraph class, which can be
      plotted.
856
857 ga.output.solution.best.adj.matrix <- as.matrix(get.adjacency(ga.output.
      solution.best.igraph)) #Save the best solution as normal R adjacency
      matrix.
858
859 ga.output.solution.best.dist.matrix <- GenerateDistanceMatrix(ga.output.
      solution.best.igraph)
860
861 ga.output.solution.best.dist.vector <- SymMatrixToVector(ga.output.solution.
      best.dist.matrix[1:inputmatrix.dimension,1:inputmatrix.dimension]) #
      Dropping the dummy variables to ensure that they are not stored in the
      final comparison matrix, which is used to calculate the fitness of the
      best representations.
862
863 comparison.matrix[2,] <- ga.output.solution.best.dist.vector #The second row is
        filled with the best distance network solution. The dummy nodes are not
      transfered into the comparison matrix.
864
```

```
865
866 ###Results of network fitness measurements
867
868 ##Error counting / counting the order violations of the best network solution
869
870 #Number of iterations of the genetic algorithm
871 ga.output.number.of.iterations <- ga.output@iter
872
873 #Maximum number of possible errors / order violations (The additional
        dimensions of the dummies do not play a role here)
874 error.count.max.possible.error <- (0.25 * inputmatrix.dimension * (inputmatrix.
        dimension - 1) * (0.5 * inputmatrix.dimension * (inputmatrix.dimension -
        1) - 1 ))
875
876 #Number of errors / order violations for best solution found by the genetic
        algorithm
877 network.output.error.count.best.solution <- FitnessOrdinal(comparison.matrix[c
        (1,2),]) #FitnessOrdinal is used here as this is the error counting
        fitness measure. Very important is the selection of the first and second
        row.
878 #The fitness value of the metric fitness function is not used here.
879
880 network.output.error.share.best.solution <- round(network.output.error.count.
        best.solution / error.count.max.possible.error, digits = 4)
881 network.output.fitness.share.best.solution <- round(1 - network.output.error.
        count.best.solution / error.count.max.possible.error, digits = 4)
882
883 #Taking care of the case of more than just one best solution
884 ga.output.number.of.best.solutions <- dim(ga.output@solution)[1]
885
886 #TODO The following could be moved to the end of this script.
887 if(ga.output.number.of.best.solutions > 1) {
888   print("Two␣or␣more␣different␣best␣network␣solutions␣were␣found␣by␣the␣genetic
          ␣algorithm.")
889 } #Note: It would also be possible to extend the code to plot each of the best
        solutions.
890
891 ##Stress-1 of the best network solution
892 if(scale.of.input.data == "ordinal") {network.stress1.best.solution <-
        FitnessOrdinalStress1(comparison.matrix,"network")}
893 if(scale.of.input.data == "metric") {network.stress1.best.solution <- "Not␣(yet
        )␣implemented"} #TODO This might be added in the future.
894
895 #
896 # Multidimensional scaling ----------------------------------------------
897 #
898
899 #Computing the MDS solution
```

```
900 if(scale.of.input.data == "metric") {mds.output <- MetricMDS(inputmatrix.diss.
        matrix)}
901 if(scale.of.input.data == "ordinal") {mds.output <- OrdinalMDS(inputmatrix.diss
        .matrix)}
902
903
904 #
905 # Preperations for the presentation of the MDS results --------------------
906 #
907
908 ##Computing the fitness of the MDS solution
909 #Transformations
910 mds.output.solution.best.dist.matrix <- mds.output$confdiss #Shows the
        dissimilarity matrix of the MDS (the SMACOF manual calls this: "
        Configuration dissimilarities")
911 mds.output.solution.best.dist.matrix <- as.matrix(mds.output.solution.best.dist
        .matrix) #Transformation into a normal symmetric matrix
912 mds.output.solution.best.dist.vector <- SymMatrixToVector(mds.output.solution.
        best.dist.matrix)
913
914 comparison.matrix[3,] <- mds.output.solution.best.dist.vector #The third row of
         the comparison matrix is used for the best distance MDS solution.
915
916 ###Results of network fitness measurements
917
918 ##Error counting / counting the order violations of the best MDS solution
919 mds.output.error.count.best.solution <- FitnessOrdinal(comparison.matrix[c(1,3)
        ,]) #FitnessOrdinal is used here as this is the error counting fitness
        measure. Very important is the selection of the first and thrid row (and
        not the second row). That way the third row of the comparison matrix will
         be transfered as "second" row to the fitness function, which relies on
        comparing the first with the second row.
920
921 mds.output.error.share.best.solution <- round(mds.output.error.count.best.
        solution / error.count.max.possible.error, digits = 4)
922 mds.output.fitness.share.best.solution <- 1 - mds.output.error.share.best.
        solution
923
924
925 ##Stress-1 of the best MDS solution
926 if(scale.of.input.data == "ordinal") {mds.stress1.best.solution <-
        FitnessOrdinalStress1(comparison.matrix,"mds")} #Which is the same as
        sqrt(mds.output$stress.nm)
927 if(scale.of.input.data == "metric") {mds.stress1.best.solution <- sqrt(mds.
        output$stress.m)}
928
929
930 #
931 # Export results ----------------------------------------------------------
```

```
#

###Data
helper.date.output <- format(Sys.time(), "%Y-%m-%d␣-␣%H-%M-%S")
helper.date.output.path <- paste("output/",helper.date.output, sep = "")
helper.date.output.path.data <- paste("output/",helper.date.output,"/data", sep
      = "")
helper.date.output.path.plots <- paste("output/",helper.date.output,"/plots",
     sep = "")

#Generate folders
if(file.exists("output")) {"Folder␣output␣already␣exists."} else {dir.create("
     output")} #Output folder in the working directory
dir.create(path = helper.date.output.path)
dir.create(path = helper.date.output.path.data)
dir.create(path = helper.date.output.path.plots)

##Write files
#Tables
write.table(inputmatrix, file = paste(helper.date.output.path.data,"/
     inputmatrix.txt", sep = ""), fileEncoding = "UTF-8")
write.table(ga.output.solution.best.adj.matrix, file = paste(helper.date.output
     .path.data,"/ga.output.solution.best.adj.matrix.txt", sep = ""),
     fileEncoding = "UTF-8")
write.table(ga.output.solution.best.dist.matrix, file = paste(helper.date.
     output.path.data,"/ga.output.solution.best.dist.matrix.txt", sep = ""),
     fileEncoding = "UTF-8")
write.table(comparison.matrix, file = paste(helper.date.output.path.data,"/
     comparison.matrix.of.best.solutions.txt", sep = ""), fileEncoding = "UTF
     -8")

#Data frames
save(ga.output, file = paste(helper.date.output.path.data,"/ga.output.data.
     frame.Rda", sep = ""))
save(mds.output, file = paste(helper.date.output.path.data,"/mds.output.data.
     frame.Rda", sep = ""))

#Summary of current result
#These files can be opened by: read.table(file = "output/summary.of.recent.
     results.txt", header = TRUE, row.names = NULL) #Or similar filename.

helper.summary.output <- data.frame(title.of.input.data, scale.of.input.data,
     dummy.nodes, error.count.max.possible.error, network.output.error.count.
     best.solution, network.output.error.share.best.solution, network.output.
     fitness.share.best.solution, mds.output.error.count.best.solution, mds.
     output.error.share.best.solution, mds.output.fitness.share.best.solution,
      network.stress1.best.solution, mds.stress1.best.solution, ga.output.
     number.of.iterations, ga.popSize, ga.pcrossover, ga.pmutation, ga.elitism
     .popsize.share, ga.use.suggestions.matrix, ga.use.modified.mutation, ga.
```

```
      output.number.of.best.solutions, inputmatrix.dimension, inputmatrix.
      dimension.incl.dummies, inputmatrix.possible.links, inputmatrix.possible.
      links.incl.dummies, time.ga.in.min, time.ga.in.hours, helper.date.output)
961
962 write.table(helper.summary.output, file = paste(helper.date.output.path.data,"/
      results.txt", sep = ""), append = FALSE, col.names = TRUE, fileEncoding =
       "UTF-8")
963
964 #Add to summary table of recent results
965 if(file.exists(file = paste("output/summary.of.recent.results.txt", sep = ""))
      == TRUE) {
966   write.table(helper.summary.output, file = paste("output/summary.of.recent.
        results.txt"), append = TRUE, col.names = FALSE, fileEncoding = "UTF-8"
        ) #Important: col.names have to be switched from TRUE to false
967 } else {
968   write.table(helper.summary.output, file = paste("output/summary.of.recent.
        results.txt"), append = FALSE, col.names = TRUE, fileEncoding = "UTF-8"
        ) #Important: col.names have to be switched from TRUE to false
969 }
970
971 #This checks, whether the file already exists and changes the write.table
      option with respect to the result.
972
973
974 ###Plots
975 ##With errors
976 pdf(file = paste(helper.date.output.path.plots,"/mds.solution.show.errors.pdf",
       sep = ""), onefile=TRUE) #Saves the results of the following plot (till
      dev.off())
977 par(mfrow=c(1,1)) #Can be set plot to plot two graphs in one file.
978 if(scale.of.input.data == "metric") {PlotMetricMDS(mds.output,show.errors=TRUE)
      }
979 if(scale.of.input.data == "ordinal") {PlotOrdinalMDS(mds.output,show.errors=
      TRUE)}
980 dev.off() #Required to tell R to stop plotting to the PDF file.
981
982 pdf(file = paste(helper.date.output.path.plots,"/network.solution.show.errors.
      pdf", sep = ""), onefile=TRUE)
983 par(mfrow=c(1,1))
984 PlotNetwork(ga.output.solution.best.igraph,show.errors=TRUE)
985 dev.off()
986
987 ##Without errors
988 pdf(file = paste(helper.date.output.path.plots,"/mds.solution.pdf", sep = ""),
      onefile=TRUE) #Saves the results of the following plot (till dev.off())
989 par(mfrow=c(1,1)) #Can be set plot to plot two graphs in one file.
990 par(mar=c(4.6, 4.6, 0.1, 0.6)) #Changed margins of plot
991 if(scale.of.input.data == "metric") {PlotMetricMDS(mds.output,show.errors=FALSE
      )}
```

```
992 if(scale.of.input.data == "ordinal") {PlotOrdinalMDS(mds.output,show.errors=
        FALSE)}
993 dev.off() #Required to tell R to stop plotting to the PDF file.
994
995 pdf(file = paste(helper.date.output.path.plots,"/mds.solution.without.axes.pdf"
        , sep = ""), onefile=TRUE) #Saves the results of the following plot (till
         dev.off())
996 par(mfrow=c(1,1)) #Can be set plot to plot two graphs in one file.
997 par(mar=c(0.1, 0.1, 0.1, 0.1)) #Changed margins of plot
998 if(scale.of.input.data == "metric") {PlotMetricMDS(mds.output,show.errors=FALSE
        )}
999 if(scale.of.input.data == "ordinal") {PlotOrdinalMDS(mds.output,show.errors=
        FALSE)}
1000 dev.off() #Required to tell R to stop plotting to the PDF file.
1001
1002 pdf(file = paste(helper.date.output.path.plots,"/network.solution.pdf", sep = "
        "), onefile=TRUE)
1003 par(mfrow=c(1,1))
1004 par(mar=c(0.6, 0.6, 0.6, 0.6)) #Changed margins of plot
1005 PlotNetwork(ga.output.solution.best.igraph,show.errors=FALSE)
1006 dev.off()
1007
1008 pdf(file = paste(helper.date.output.path.plots,"/network.solution.thesis.pdf",
        sep = ""), onefile=TRUE)
1009 par(mfrow=c(1,1))
1010 par(mar=c(1.6, 1.6, 1.6, 1.6)) #Changed margins of plot
1011 PlotNetwork(ga.output.solution.best.igraph,show.errors=FALSE,manual.vertex.
        label.cex=1.6) #manual.vertex.size can also be used as input.
1012 dev.off()
1013
1014 par(mar=c(5.1, 4.1, 4.1, 2.1)) #Resetting to default margins #c(bottom, left,
        top, right)
1015
1016 ##GA iterations
1017 pdf(file = paste(helper.date.output.path.plots,"/network.ga.iterations.pdf",
        sep = ""), onefile=TRUE)
1018 par(mfrow=c(1,1))
1019 par(mar=c(4.6, 4.6, 0.1, 0.1)) #Changed margins of plot
1020 PlotGAIterations(ga.output) #TODO One might add an error message if a distance
        and hence the mean is inf
1021 dev.off()
1022
1023 par(mar=c(5.1, 4.1, 4.1, 2.1)) #Resetting to default margins #c(bottom, left,
        top, right)
1024
1025 ###Plot inputmatrix distribution histogram
1026 pdf(file = paste(helper.date.output.path.plots,"/inputmatrix.histogram.pdf",
        sep = ""), onefile=TRUE)
1027 par(mfrow=c(1,1))
```

```
1028 par(mar=c(2.6, 4.6, 0.1, 0.1)) #Changed margins of plot
1029 hist(SymMatrixToVector(inputmatrix), main = "", xlab= "")
1030 dev.off()
1031 par(mar=c(5.1, 4.1, 4.1, 2.1)) #Resetting to default margins #c(bottom, left,
       top, right)
1032
1033 pdf(file = paste(helper.date.output.path.plots,"/inputmatrix.density.plot.pdf",
        sep = ""), onefile=TRUE)
1034 par(mfrow=c(1,1))
1035 plot(density(SymMatrixToVector(inputmatrix), na.rm = TRUE))
1036 dev.off()
1037
1038 ###Plot GA Network and metric MDS solution in R
1039 par(mfrow=c(1,2)) #Set plot to plot two graphs in one plot.
1040 if(scale.of.input.data == "metric") {PlotMetricMDS(mds.output,show.errors=TRUE)
       }
1041 if(scale.of.input.data == "ordinal") {PlotOrdinalMDS(mds.output,show.errors=
       TRUE)}
1042 PlotNetwork(ga.output.solution.best.igraph,show.errors=TRUE)
1043 par(mfrow=c(1,1)) #Reset to standard.
1044
1045 #
1046 # Grooming -----------------------------------------------------------------
1047 #
1048
1049 #TODO Further no longer needed helper variables might be dropped.
1050
1051
1052 ###Dropping no longer needed functions and objects
1053 #Copy of the fitness function (is the ordinal or metric function)
1054 rm(ga.fitness)
1055
1056 #Variables that were required for the modified mutation function
1057 rm(ga.mutation.prob.1.obj, ga.mutation.prob.2.obj, ga.mutation.prob.3.obj, ga.
       mutation.prob.5.obj, ga.mutation.prob.1.percent, ga.mutation.prob.3.
       percent, ga.mutation.prob.5.percent, ga.mutation.prob.10.percent, ga.
       mutation.prob.15.percent, ga.mutation.prob.20.percent, ga.mutation.cum.
       prob.1.obj, ga.mutation.cum.prob.2.obj, ga.mutation.cum.prob.3.obj, ga.
       mutation.cum.prob.5.obj, ga.mutation.cum.prob.1.percent, ga.mutation.cum.
       prob.3.percent, ga.mutation.cum.prob.5.percent, ga.mutation.cum.prob.10.
       percent, ga.mutation.cum.prob.15.percent, ga.mutation.cum.prob.20.percent
       )
1058
1059
1060 # End of file ---------------------------------------------------------------
```

A.2. Visualisation of the Stress-1 Problem of Networks

This section shows a network representation that led to a Stress-1 value of zero, while it reached an error count of 347 out of 990 possible order violations, which is a fitness share of 0.6495. The aim of this section is to underline the problems associated with the application of Stress-1 to measure the fitness of a network representation, as described in the sections 3.4.1 and 3.6.

The related randomly generated (dis)similarity data that is represented in figure A.1 can be seen in table A.1.

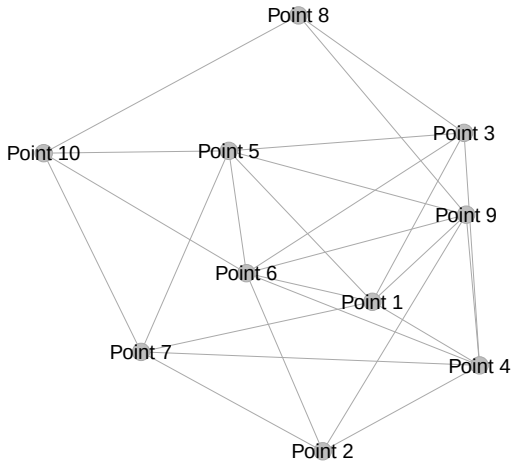

Figure A.1.: *Network representation of the distances of the ten rounded randomly generated objects from table A.1.*

	P. 1	P. 2	P. 3	P. 4	P. 5	P. 6	P. 7	P. 8	P. 9	P. 10
Point 1	0	3	6	6	5	5	7	1	6	2
Point 2	3	0	3	6	3	8	9	3	5	3
Point 3	6	3	0	6	6	6	4	6	4	4
Point 4	6	6	6	0	2	8	6	2	5	4
Point 5	5	3	6	2	0	5	6	3	5	5
Point 6	5	8	6	8	5	0	4	3	7	5
Point 7	7	9	4	6	6	4	0	3	4	6
Point 8	1	3	6	2	3	3	3	0	9	6
Point 9	6	5	4	5	5	7	4	9	0	4
Point 10	2	3	4	4	5	5	6	6	4	0

Table A.1.: *(Dis)similarity data represented as network graph in figure A.1.*

Bibliography

ALLBUS (1991): "ALLBUS Baseline Survey 1991 (German General Social Survey - Baseline Survey 1991)," DOI: http://dx.doi.org/10.4232/1.1990.

Arabie, Phipps (1991): "Was Euclid an Unnecessarily Sophisticated Psychologist?" *Psychometrika*, Vol. 56, No. 4, pp. 567–587, DOI: http://dx.doi.org/10.1007/BF02294491.

Büchel, Berno and Giuseppe Mastrangeli (2013): "Cognitive Networks – An Alternative to MDS," Mimeo, Hamburg University 2013.

Borg, Ingwer and Patrick J. F. Groenen (2005): *Modern Multidimensional Scaling – Theory and Applications*, Springer Series in Statistics: Springer, 2nd Edition, DOI: http://dx.doi.org/10.1007/0-387-28981-X.

Borg, Ingwer, Patrick J. F. Groenen, and Patrick Mair (2013): *Applied Multidimensional Scaling*, SpringerBriefs in Statistics: Springer, DOI: http://dx.doi.org/10.1007/978-3-642-31848-1.

Borg, Ingwer and Thomas Staufenbiel (2007): *Lehrbuch Theorien und Methoden der Skalierung [German]*: Verlag Hans Huber, 4th Edition.

Breiger, Ronald L., Scott A. Boorman, and Phibbs Arabie (1975): "An Algorithm for Clustering Relational Data with Applications to Social Network Analysis and Comparison with Multidimensional Scaling," *Journal of Mathematical Psychology*, Vol. 12, No. 3, pp. 328–383.

Buja, Andreas, Deborah F. Swayne, Michael L. Littman, Nathaniel Dean, Heike Hofmann, and Lisha Chen (2008): "Data Visualization

With Multidimensional Scaling," *Journal of Computational and Graphical Statistics*, Vol. 17, No. 2, pp. 444–472, DOI: http://dx.doi.org/10.1198/106186008X318440.

Cox, Michael A. A. and Trevor F. Cox (2008): "Multidimensional Scaling," in Chun-houh Chen, Wolfgang Härdle, and Antony Unwin (eds.) *Handbook of Data Visualization*: Springer, pp. 315–347, DOI: http://dx.doi.org/10.1007/978-3-540-33037-0.

Csárdi, Gábor and Tamás Nepusz (2006): "The igraph Software Package for Complex Network Research," *InterJournal Complex Systems*, Vol. 1695.

Erdős, Paul and Alfréd Rényi (1964): "On the Strength of Connectedness of a Random Graph," *Acta Mathematica Academiae Scientiarum Hungarica*, Vol. 12, No. 1-2, pp. 261–267, DOI: http://dx.doi.org/10.1007/BF02066689.

Fitch, Walter M. and Emanuel Margoliash (1967): "Construction of Phylogenetic Trees," *Science*, Vol. 155, No. 3760, pp. 279–284, DOI: http://dx.doi.org/10.1126/science.155.3760.279.

Groenen, P. J. F., W. J. Heiser, and J. J. Meulman (1999): "Global Optimization in Least-Squares Multidimensional Scaling by Distance Smoothing," *Journal of Classification*, Vol. 16, No. 2, pp. 225–254, DOI: http://dx.doi.org/10.1007/s003579900055.

Groenen, Patrick J. F., Rudolf Mathar, and Willem J. Heiser (1995): "The Majorization Approach to Multidimensional Scaling for Minkowski Distances," *Journal of Classification*, Vol. 12, No. 1, pp. 3–19, DOI: http://dx.doi.org/10.1007/BF01202265.

Henderson, Geraldine R., Dawn Iacobucci, and Bobby J. Calder (1998): "Brand Diagnostics: Mapping Branding Effects Using Consumer Associative Networks," *European Journal of Operational Research*, Vol. 111, pp. 306–327.

Hornik, Kurt (2014): "The R FAQ," URL: http://cran.r-project.org/doc/manuals/R-FAQ.pdf, accessed on 16th March 2014.

Jackson, Matthew O. (2006): "The Economics of Social Networks," in Whitney Newey Richard Blundell and Torsten Persson (eds.) *Advances in Economics and Econometrics – Theory and Applications, Ninth World Congress*, Vol. 1: Cambridge University Press, Chap. 1.

Jackson, Matthew O. (2010): *Social and Economic Networks*: Princeton University Press.

Jackson, Matthew O. (2011): "An Overview of Social Networks and Economic Applications," in Alberto Bisin Jess Benhabib and Matthew O. Jackson (eds.) *Handbook of Social Economics*, Vol. 1: North Holland Publishing Company, Chap. 12, pp. 511–585.

Kruskal, Joseph Bernard (1964): "Multidimensional Scaling by Optimizing Goodness of Fit to a Nonmetric Hypothesis," *Psychometrika*, Vol. 29, No. 1, pp. 1–27, DOI: `http://dx.doi.org/10.1007/BF02289565`.

de Leeuw, Jan (1977): "Applications of Convex Analysis to Multidimensional Scaling," in J. R. Barra, F. Brodeau, G. Romier, and B. Van Cutsem (eds.) *Recent Developments in Statistics*, pp. 133–145: North Holland Publishing Company.

de Leeuw, Jan (2001): "Multidimensional Scaling," in N. J. Smelser and P. B. Baltes (eds.) *International Encyclopedia of the Social and Behavioral Sciences*: Elsevier, pp. 13512–13519.

de Leeuw, Jan, Kurt Hornik, and Patrick Mair (2009): "Isotone Optimization in R: Pool-Adjacent-Violators Algorithm (PAVA) and Active Set Methods," *Journal of Statistical Software*, Vol. 32, No. 5, pp. 1–24, URL: `http://www.jstatsoft.org/v32/i05`.

de Leeuw, Jan and Patrick Mair (2009): "Multidimensional Scaling Using Majorization: SMACOF in R," *Journal of Statistical Software*, Vol. 31, No. 3, pp. 1–30, URL: `http://www.jstatsoft.org/v31/i03`.

McLachlan, Geoffrey, Kim-Anh Do, and Christophe Ambroise (2004): *Analyzing Microarray Gene Expression Data*: Wiley.

Scrucca, Luca (2013): "GA: A Package for Genetic Algorithms in R," *Journal of Statistical Software*, Vol. 53, No. 4, pp. 1–37, URL: http://www.jstatsoft.org/v53/i04.

Shepard, Roger N. (1980): "Multidimensional Scaling, Tree-Fitting, and Clustering," *Science*, Vol. 210, No. 4468, pp. 390–398, DOI: http://dx.doi.org/10.1126/science.210.4468.390.

Teichert, Thorsten A. and Katja Schöntag (2010): "Exploring Consumer Knowledge Structures Using Associative Network Analysis," *Psychology and Marketing*, Vol. 27, No. 4, pp. 369–398, DOI: http://dx.doi.org/10.1002/mar.20332.

Venables, William N. and Brian D. Ripley (2002): *Modern Applied Statistics with S*: Springer, 4th Edition, URL: http://www.springer.com/statistics/computational+statistics/book/978-0-387-95457-8.

Weise, Thomas (2009): *Global Optimization Algorithms – Theory and Application*: it-weise.de (self-published): Germany, 2nd Edition, URL: http://www.it-weise.de/projects/book.pdf, accessed on 13th February 2014.

Wooldridge, Jeffrey M. (2013): *Introductory Econometrics – A Modern Approach*: South-Western, Cengage Learning, 5th Edition.

Acknowledgements

I would like to thank Prof. Dr. Gerd Mühlheußer for the opportunity to write this thesis and his advice. I would also like to express my deep gratitude to Dr. Berno Büchel for the valuable guidance and the constructive suggestions. Without his enthusiastic encouragement this thesis would not be the same.

My grateful thanks are also extended to Andrea Gosewisch, Fransziska Kösling, Johanna Behr, Kerstin Menke, Kevin West, Philipp Müller, Renate Menke, and Torben Menke for their valuable support.

Finally, I wish to thank my family for their support and encouragement throughout my studies.

Statement of Originality

Herewith, I confirm that I have written the thesis to be found above independently and without help from another party. I have not used any material or sources apart from those which have been indicated on the list of references. All internet sources have been listed. Furthermore, I confirm that I have not submitted this thesis to any previous examination procedure and that the submitted printed version is identical to the electronic version submitted.

Hamburg, 14th May 2014

Björn Menke