Designing Pixel-Oriented Visualization Techniques: Theory and Applications

Daniel A. Keim

Abstract—Visualization techniques are of increasing importance in exploring and analyzing large amounts of multidimensional information. One important class of visualization techniques which is particularly interesting for visualizing very large multidimensional data sets is the class of pixel-oriented techniques. The basic idea of pixel-oriented visualization techniques is to represent as many data objects as possible on the screen at the same time by mapping each data value to a pixel of the screen and arranging the pixels adequately. A number of different pixel-oriented visualization techniques have been proposed in recent years and it has been shown that the techniques are useful for visual data exploration in a number of different application contexts. In this paper, we discuss a number of issues which are of high importance in developing pixel-oriented visualization techniques. The major goal of this article is to provide a formal basis of pixel-oriented visualization techniques and show that the design decisions in developing them can be seen as solutions of well-defined optimization problems. This is true for the mapping of the data values to colors, the arrangement of pixels inside the subwindows, the shape of the subwindows, and the ordering of the dimension subwindows. The paper also discusses the design issues of special variants of pixel-oriented techniques for visualizing large spatial data sets. The optimization functions for the mentioned design decisions are important for the effectiveness of the resulting visualizations. We show this by evaluating the optimization functions and comparing the results to the visualizations obtained in a number of different application.

Index Terms—Information visualization, visualizing large data sets, visualizing multidimensional and multivariate data, visual data exploration, visual data mining.

1 Introduction

Information visualization techniques are becoming increasingly important for the analysis and exploration of large multidimensional data sets. A major advantage of visualization techniques over other (semi)automatic data exploration and analysis techniques (from statistics, machine learning, artificial intelligence, etc.) is that visualizations allow a direct interaction with the user and provide an immediate feedback, as well as user steering, which is difficult to achieve in most nonvisual approaches. The practical importance of visual data mining techniques is therefore steadily increasing and basically all commercial data mining systems try to incorporate visualization techniques of one kind or the other (usually rather simple ones). In the visualization community, a considerable number of advanced visualization techniques for multidimensional data have been proposed. Examples of visual data exploration approaches include geometric projection techniques such as parallel coordinates [28], [29], icon-based techniques (e.g., [51], [12]), hierarchical techniques (e.g., [46], [54], [56]), graph-based techniques (e.g., [21], [15]), pixel-oriented techniques (e.g., [31], [37], [39]), and combinations thereof ([8], [7]). In general, the visualization techniques are used in conjunction with some interaction techniques (e.g., [17], [10], [3]) and, sometimes, also some distortion techniques [55], [45] (cf. Section 2 for details).

When considering the proposed visualization techniques, however, the question arises: Are the techniques just some fancy ad hoc ideas or do they have a more formal basis? Related questions are: Is there a systematic way of developing them? And, if so, what are the specific design goals and is there a formal way of describing them? The questions are not only of intellectual interest, but important for the field since they a allow a better understanding of the existing techniques, a systematic development of new techniques, and a more formal way of evaluating them. In this paper, we describe the design goals behind developing the different variants of pixel-oriented visualization techniques. It is interesting that most of the design goals can be formalized as optimization problems and the proposed techniques are actually (possibly suboptimal) solutions of the optimization problems. The formalization of the design goals also make it possible to formally evaluate the quality of the obtained solutions.

The rest of the article is organized as follows: In Section 2, we provide an overview and classification of techniques for visualizing large amounts of multidimensional data, including a brief introduction to the basic idea of pixel-oriented visualization techniques. Then, we discuss the design issues which are important in developing the different variants of the pixel-oriented technique: In Section 3, we discuss the color mapping, in Section 4, the arrangement of pixels inside the dimension subwindows, in Section 5, the shape of the dimension subwindows, and, in Section 6, the ordering and arrangement of the dimension subwindows. We show that the design issues are complex optimization problems and that the different variants of pixel-oriented techniques optimize different criteria. In Section 7, we discuss the special case of data sets with some two-dimensional semantics. Again, the design goals can be formally specified and the potential solutions be

• *The author is with the Institute of Computer Science, University of Halle, Kurt-Mothes-Str. 1, 06120 Halle, Germany. E-mail: keim@informatik.uni-halle.de.*

For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number 110867.

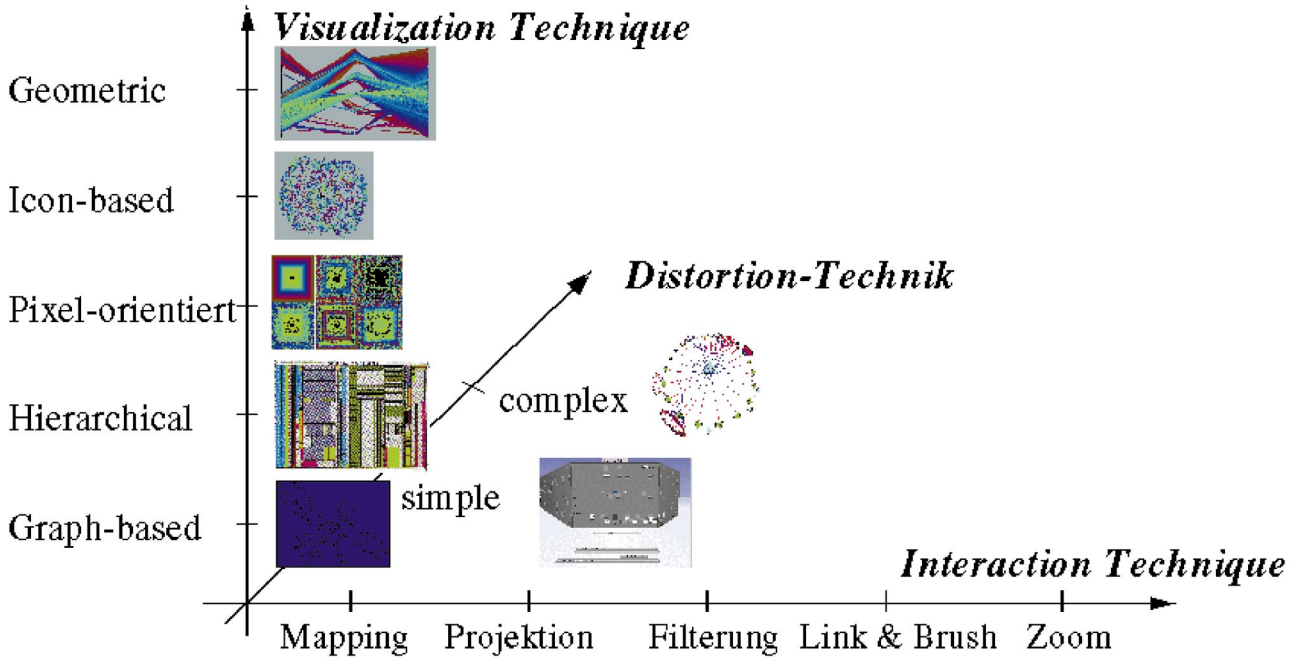


Fig. 1. Classification of multidimensional visualization techniques.

evaluated against them. Section 8 summarizes important aspects of the article and discusses their potential impact on future work in the area of information visualization.

2 Visualizing Large Amounts of Multidimensional Data

In contrast to most other work in the visualization area, in multidimensional visualization, there is no standard mapping into the Cartesian coordinate system since the data does not have some inherent two- or three-dimensional semantics. In this section, we provide a brief overview of multidimensional visualization techniques and introduce a classification of the existing techniques (cf. Section 2.1). Then, we discuss the basic ideas of the pixel-oriented techniques in more detail (cf. Section 2.2).

2.1 Classification of Multidimensional Visualization Techniques

Visualization of data which have some inherent two- or three-dimensional semantics has been done even before computers were used to create visualizations. In his well-known books [58], [59], Tufte provides many examples of visualization techniques that have been used for many years. Since computers are used to create visualizations, many novel visualization techniques have been developed and existing techniques have been extended to work for larger data sets and make the displays interactive. For most of the data stored in databases, however, there is no standard mapping into the Cartesian coordinate system since the data has no inherent two- or three-dimensional semantics. In general, relational databases can be seen as multidimensional data sets with the attributes of the database corresponding to the dimensions. The techniques for visualizing multidimensional data sets can be best classified using three orthogonal criteria: the visualization technique, the distortion technique, and the interaction technique (cf. Fig. 1). Orthogonality means, in this context, that any of the visualization techniques can be used in conjunction with any of the distortion, as well as any of the interaction techniques. The visualization techniques can be divided into geometric projection, icon-based, pixel-based, hierarchical, and graph-based techniques. Well-known examples of geometric projection techniques include scatterplot matrices and coplots [5], [18], landscapes [62], prosection views [23], [57], hyperslice [61], and parallel coordinates [28], [29]; examples of icon-based techniques are stick figures [51], shape-coding [12], and color icons [43], [37]; examples of pixel-oriented techniques are the spiral [37], [33], recursive pattern [39] and circle segment techniques [4]; examples of hierarchical techniques are dimensional stacking [46], treemap [56], [30], and cone-trees [54]; and examples of graph-based techniques are cluster- and symmetry-optimized, as well as hierarchical graph visualizations [15], [14]. In addition to the visualization technique, for an effective data exploration, it is important to use some interaction and distortion techniques. The interaction techniques allow the user to directly interact with the visualization. Examples of interaction techniques include interactive mapping [16], [11], projection [8], [11], filtering [7], [20], [24], zooming [13], [9], and interactive linking and brushing [60], [63]. Interaction techniques allow dynamic changes of the visualizations according to the exploration objectives, but they also make it possible to relate and combine multiple independent visualizations. Note that connecting multiple visualizations by linking and brushing, for example, provides more information than considering the component visualizations independently. The last criterion of the classification helps in the interactive process of exploration by providing means for focusing while preserving an overview of the data. The basic idea of distortion techniques is to show portions of the data with a

		Clust-ering	multi-variate hot spot	no. of vari-ates	no. of data items	cate-gorical data	visual over-lap	learn-ing curve
Geometric Techniques	Scatterplot Matrices	++	++	+	+	-	o	++
	Landscapes	+	+	-	o	o	+	+
	Prosection Views	++	++	+	+	-	o	+
	Hyperslice	+	+	+	+	-	o	o
	Parallel Coordinates	o	++	++	-	o	--	o
Icon-based Techniques	Stick Figures	o	o	+	-	-	-	o
	Shape Coding	o	-	++	+	-	+	-
	Color Icon	o	-	++	+	-	+	-
Pixel-oriented Techniques	Query-Independent	+	+	++	++	-	++	+
	Query-Dependent	+	+	++	++	-	++	-
Hierarchical Techniques	Dimensional Stacking	+	+	o	o	++	o	o
	Treemap	+	o		o	++		o
	Cone Trees	+	+	o	+	o	+	+
Graph-based Techniques	Basic Graphs	o	o	-	+	o	o	+
	Specific Graphs	++	+	-	+	o	+	+

Fig. 2. An attempt at comparing multidimensional visualization techniques (++: very good, +: good, o: neutral, -: bad, –: very bad).

high level of detail while others are shown with a much lower level of detail. A number of simple and complex distortion techniques may be used for this purpose [42]. Examples are the perspective wall [49], bifocal lens [6], table lens [52], fisheye view [25], [55], hyperbolic tree [44], [45], [47], and hyperbox techniques [2].

This brief introduction of our classification and the enumeration of examples is aimed at providing a more structured understanding of the large number of available multidimensional visualization techniques. It can also be used as a starting point to compare the available techniques, to improve existing techniques, and to develop new techniques. To provide a starting point for such a comparison, in Fig. 2, we provide a preliminary and subjective comparison table[1] which is trying to compare a number of visualization techniques. The comparison of the visualization techniques is based on their suitability for certain

- **data characteristics** such as number of dimensions (attributes), number of data objects, and suitability for categorical data,
- **task characteristics** such as clustering and multivariate hot spots,
- **visualization characteristics** such as visual overlap and learning curve.

A more detailed description of the classification and examples can be found in tutorial notes on visual data exploration [34], [35]. In the following, we introduce the class of pixel-oriented techniques in more detail.

1. **Disclaimer:** The comparison table expresses the authors personal opinion obtained from reading the literature and experimenting with several of the described techniques. Many of the ratings are arguable and largely depend on the considered data, the exploration task, experience of the user, etc. In addition, implementations of the techniques in real systems usually avoid the drawbacks of the single techniques by combining them with other techniques, which is also not reflected in the ratings.

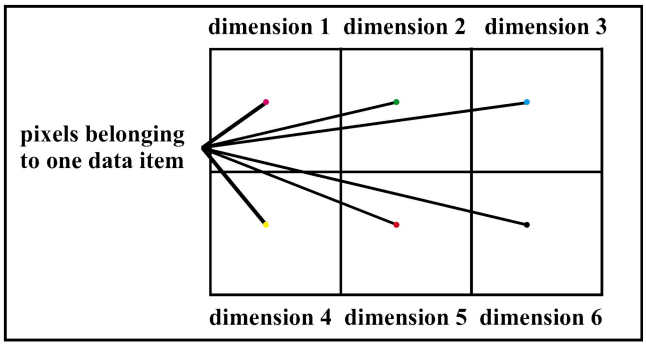


Fig. 3. Basic arrangement of subwindows for data with six dimensions.

2.2 Pixel-Oriented Techniques

The basic idea of pixel-oriented techniques is to map each data value to a colored pixel and present the data values belonging to one dimension (attribute) in a separate subwindow (cf. Fig. 3). Since, in general, our techniques use only one pixel per data value, the techniques allow us to visualize the largest amount of data which is possible on current displays (up to about 1,000,000 data values). All pixel-oriented techniques partition the screen into multiple subwindows. For data sets with m dimensions (attributes), the screen is partitioned into m subwindows—one for each of the dimensions. In the case of a special class of pixel-oriented techniques—the query-dependent techniques—an additional $(m+1)$th window is provided for the overall distance. Inside the windows, the data values are arranged according to the given overall sorting, which may be data-driven for the query-independent techniques or query-driven for the query-dependent techniques. Correlations, functional dependencies, and other interesting relationships between dimensions may be detected by relating corresponding regions in the multiple windows.

To achieve that objective a number of design problems have to be solved. The first problem is the mapping of data values to colors. A good mapping is obviously very important, but has to be carefully engineered to be intuitive. A second important question is how the pixels are arranged inside the subwindows. The arrangement depends on the data and the task of the visualization and, therefore, different arrangements are useful for different purposes. As we discuss in Section 4, the arrangement problem can be described formally as an optimization problem and different visualization techniques optimize different variants of the optimization problem. A third question is the shape of the subwindows. With the rectangular shape of the subwindows as given in Fig. 3, for data sets with a large number of dimensions (attributes), the subwindows for the different dimensions are quite distant and, therefore, it becomes difficult to find interesting relationships between the dimensions. Again, shape of subwindows can be seen as an optimization problem and, in Section 5, we introduce a visualization technique which better solves this problem. The next question in designing the pixel-oriented techniques is how to order the subwindows for the dimensions (attributes). In most applications, there is no natural ordering of the dimensions. To detect dependencies and correlations between the dimensions represented in the

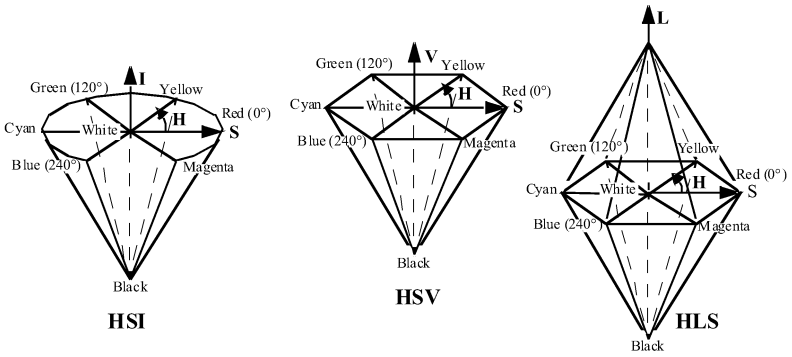


Fig. 4. The HSI color model compared to the HSV and HLS models. (Adapted from [38] ©IFIP.)

subwindows, it is best to place related dimensions next to each other. This again is a difficult optimization problem and, in Section 6, we show that this problem is even NP-complete and propose a heuristic solution. Since there are a large number of applications where data sets with some two-dimensional semantics arise, in Section 7, we consider this special class of applications. We formally describe their design goals as optimization problems, propose a number of potential solutions, and evaluate them against the design goals.

3 Color Mapping

Visualizing the data values using color corresponds to the task of mapping a single parameter distribution to color. The advantage of color over gray scales is that the number of just noticeable differences (JNDs) is much higher [27]. Finding a path through color space that maximizes the number of JNDs, but, at the same time, is intuitive for the application domain is a difficult task. For our purpose of mapping the distances to color, we can restrict the task to a simpler, more solvable problem. From a perceptual point of view, brightness is the most important characteristic for distinguishing colors corresponding to a single parameter distribution. Therefore, for our purpose, it is sufficient to find a color scale with a monotonically increasing (decreasing) brightness while using the full color range. The parameters of the color mapping should therefore use a monotonically decreasing brightness (intensity, lightness, value), a color ranging over the full color scale (hue), and a constant (full) saturation. We found experimentally that a colormap with colors ranging from yellow over green, blue, and red to almost black is a good choice for the colormap to be intuitive.

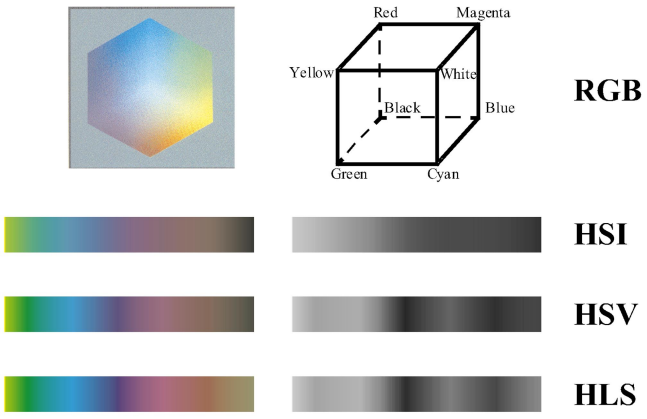


Fig. 5. Color scales generated using different color models. (Adapted from [38] ©IFIP.)

For generating color scales, we used a linear interpolation between a minimum and a maximum value for hue, saturation, and value (intensity, lightness) within the various color models. We first used the standard HSV and HLS color models. The HSV color model is best described by a single-hexcone and the HLS model by a double-hexcone (see Fig. 4). Linear interpolations within these color models, however, do not provide color scales with a monotonically decreasing brightness. Fig. 5 shows HSV and HLS color scales which are produced using a linear interpolation algorithm. The color scales are generated using a constant saturation, a decreasing value (intensity, lightness), and a hue varying over the complete range. If the generated HSV and HLS color scales are used in conjunction with our visualization techniques, brighter colors in the visualizations do not necessarily denote lower data values. If HSV and HLS color scales are mapped to gray-scale, their nonmonotonicity becomes obvious (see Fig. 5).[2]

Unfortunately, linear interpolations within the HSV and HLS color models do not produce color scales with monotonically decreasing brightness. A closer consideration of the HSV and HLS color models showed that one of the reasons for the problems of those color models is the discontinuities at the corners of the hexcones. We therefore developed our own color model, which we call the HSI model (H: Hue, S: Saturation, I: Intensity). The HSI color model is a variation of the HSV model. In contrast to color scales generated according to the HSV model, linear interpolation within the HSI model provides color scales whose lightness ranges continuously from light to dark (see Fig. 5). This is achieved by using a circular cone instead of the hexcone used in the HSV model (see Fig. 4), which means that colors with constant intensity and saturation form a circle. The hue is defined as the angle α between red

2. The standard mapping from color to gray-scale used by the X-windows system is a linear combination of the RGB-values $[Grey(r,g,b) = 0.34 * r + 0.5 * g + 0.16 * b]$ which tries to reflect the perceived brightness. Note that the X-windows mapping to gray does not correspond to the gray portion in either of the color models, which is defined as the corresponding portion of the gray axis (center of the color cones).

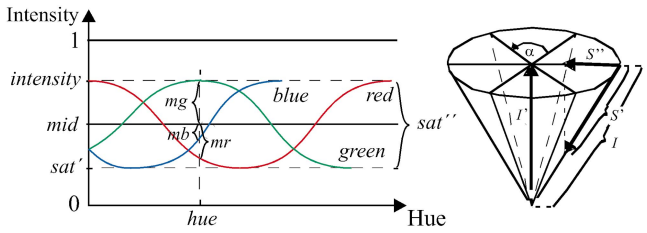


Fig. 6. The HSI color model. (Adapted from [38] ©IFIP.)

and the chosen color. When moving on a circle of the HSI color cone, the red, green, and blue portion of the color follow cosine-curves with phase-shifts of $\frac{2}{3}\pi$ (see Fig. 6). If intensity and saturation are both equal to 1, the cosine-curves for each of the RGB-color-portions run between 0 and 1. Reducing the intensity means reducing the maximum of the cosine-curve. Reducing the saturation means lifting the minimum of the cosine-curve (sat'). In terms of the HSI color cone, the intensity is defined as the Euclidean distance to the origin and the saturation as percentage of the intensity ($\frac{S'}{I}$). In the HSV model, intensity and saturation are determined by using the maximum and minimum of (red, green, blue). In Fig. 6, I and S' are marked in the HSI color cone. Since I is proportional to I' and S' proportional to S'', the HSI color cone can also be described with I' and S'' as defining axes, as done in Fig. 4.

In the following, we briefly describe the mathematical definition of the HSI parameters (hue, saturation, intensity) in terms of the RGB color components. The intensity can be determined directly from the (red, green, blue)-components. Since the cosine-curves for red, green, and blue have phase-shifts of $\frac{2}{3}\pi$, the square sum of (mr, mg, mb) is constant and proportional to $(intensity - mid)^2$. The proportionality constant $(\frac{3}{2})$ may be determined by using the special case: $saturation = 1$.

$$(mr^2 + mg^2 + mb^2) = const$$
$$\Rightarrow (mr^2 + mg^2 + mb^2) = \frac{3}{2} \times (intensity - mid)^2$$
$$\Rightarrow \textbf{intensity} = mid + \sqrt{\frac{2}{3} \times (mr^2 + mg^2 + mb^2)}.$$

To determine the saturation, we have to consider the lower limit of the cosine-curves (c.f. sat' in Fig. 6). sat' only yields values between 0 and $intensity$. To allow values between 0 and 1, sat' has to be normalized ($\frac{sat'}{intensity}$). Increasing this value causes a shrinking of the amplitude of the cosine-curves, which means that the equal portion of red, green, and blue and, therefore, the white-portion of the color increases. This is the inverse of the normal use of the term $saturation$. The saturation is therefore defined as

$$\textbf{saturation} = 1 - \frac{sat'}{intensity} = \frac{sat''}{intensity} = \frac{2 \times (intensity - mid)}{intensity}.$$

The hue is determined by using the scalar product between the vector from the gray axis to red and the vector from the gray axis to the color-point. The point on the gray axis corresponding to a color point (red, green, blue) is calculated as $\frac{red+green+blue}{3}$. The vector from the gray axis $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ to red $(1, 0, 0)$ results in $(\frac{2}{3}, -\frac{1}{3}, -\frac{1}{3})$, which is equivalent to $(2, -1, -1)$ when used in the scalar product. The vector from the gray axis (mid, mid, mid) to the color-point $(red, green, blue)$ results in (mr, mg, mb). The angle α is calculated as

$$\cos\alpha = \frac{(2, -1, -1) \times (mr, mg, mb)}{|(2, -1, -1)| \times |(mr, mg, mb)|}$$
$$\Rightarrow \textbf{hue} = \arccos\left(\frac{(2 \times mr - mg - mb)}{\sqrt{6} \times \sqrt{mr^2 + mg^2 + mb^2}}\right).$$

The algorithms for generating HSI color scales and for converting HSI to RGB and vice versa are provided in [38]. The parameters for generating the color scales presented in Fig. 5—including the HSI color scale used for the visualizations presented in rest of this article—are shown in Table 1. Since the usefulness of colormaps varies depending on the user and the application, we allow the users to define their own colormaps and use them instead of our standard colormap.

TABLE 1
Parameters for Generating Color Scales Presented in Fig. 5

	hue	saturation	intensity
MinHsi	1.5 (= LightGreen)	1.0	0.4
MaxHsi	1.0 (= Yellow)	1.0	1.0

4 Arrangement of Pixels

The second and very important question is how the pixels are arranged within each of the subwindows. This is important since, due to the density of the pixel displays, only a good arrangement will allow a discovery of clusters and correlations among the dimensions. For the arrangement problem, we have to distinguish between data sets which have a natural ordering of data objects (such as in time-series data) and data sets, without inherent ordering (as in the case of query responses).

4.1 Naturally Ordered Arrangement

For naturally ordered data sets, we assume having an ordered sequence of n data objects $\{a_1, \ldots, a_n\}$, each consisting of k data values $\{a_i^1, \ldots, a_i^k\}$. In one of the subwindows, we want to present all the values $\{a_1^i, \ldots, a_n^i\}$. The problem can then be formally defined as

Definition 1 (Pixel Arrangement Problem of naturally ordered data). *The* pixel arrangement problem *(of naturally ordered data) is the problem of finding a mapping of the data objects $\{a_1^k, \ldots, a_n^k\}$ to a subwindow of size $(w \times h)$, i.e., a bijective mapping $f : \{1 \ldots n\} \rightarrow \{1 \ldots w\} \times \{1 \ldots h\}$ such that*

$$\sum_{i=1}^{n}\sum_{j=1}^{n} \Big| d\big(f(i), f(j)\big) - d\Big((0,0), \Big(w \cdot \sqrt{\frac{|i-j|}{n}}, h \cdot \sqrt{\frac{|i-j|}{n}}\Big)\Big) \Big|$$

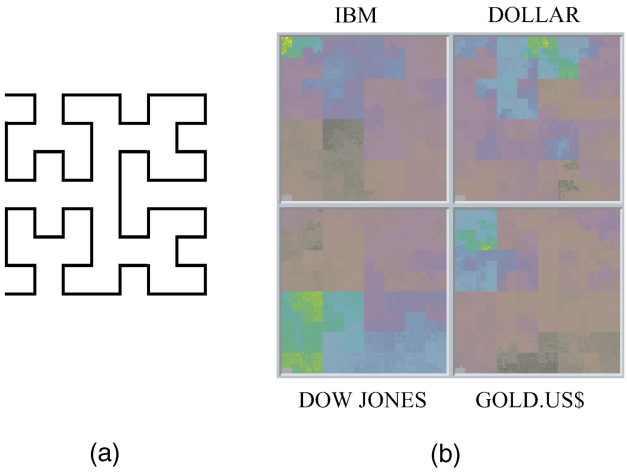


Fig. 7. Peano-Hilbert arrangement. (a) Peano-Hilbert curve. (b) Example visualization. (Adapted from [33] ©American Statistical Association.)

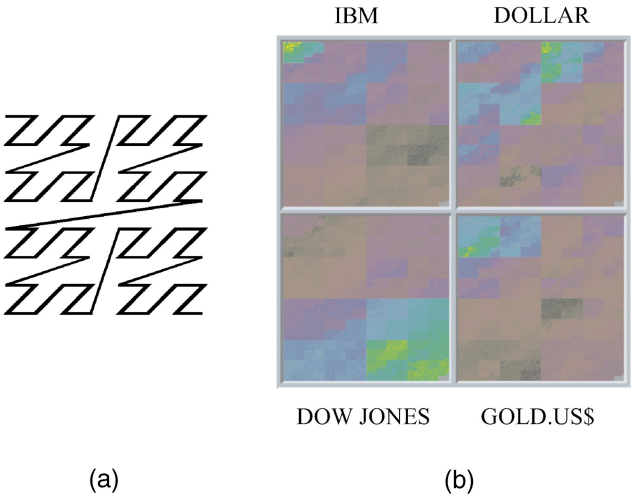


Fig. 8. Morton arrangement. (a) Morton curve. (b) Example visualization. (Adapted from [33] ©American Statistical Association.)

is minimal, where $d(f(i), f(j))$ is the L^p-distance of the pixels belonging to a_i and a_j.

The definition describes the pixel arrangement problem as an optimization problem which tries to determine the arrangement of pixels which best preserves the distance of the one-dimensional ordering in the two-dimensional arrangement. The optimization formula therefore sums up over the L^p-distances of all pixels and normalizes them by the minimum distance of the two pixels in a rectangular subwindow of proportions $(w \times h)$ (second part of the equation).

Mappings of ordered one-dimensional data sets to two dimensions have already attracted the attention of mathematicians long before computers came into existence. So-called space-filling curves try to solve exactly the above optimization problem and it is well-known that the Peano-Hilbert curve [50], [26] is among the space-filling curves which provide the best optimization of the above formula. In our first experiments, we therefore used the Peano-Hilbert curve (cf. Fig. 7a) to present the data. In Fig. 7b, we show an example of a visualization of financial data using the Peano-Hilbert curve. The database contains the prices of the IBM stock, Dow Jones index, and Gold, as well as the exchange rate between the US-Dollar and the German mark, from September 1987 to February 1995, with nine data items referring to one day. The database consists of 64,800 data values (16,200 data entries per dimension). Although the Peano-Hilbert curve provides a good clustering of the data, in general, it is difficult to follow the curve and, therefore, it is also difficult to relate the subwindows. We therefore also used the Morton curve [48] which has more regularity and is much easier to follow (cf. Fig. 8a). The resulting visualizations (cf. Fig. 8b), however, show that the Morton curve does not provide convincing results since the arrangement is still not intuitive and the distances between neighboring points are not preserved well enough.

Our solution to this problem, as first proposed in [39], is based on the idea of generalizing a line and column-wise arrangement by allowing the user to provide input in order to obtain a semantic arrangement. If simple left-right or top-down arrangements are used on the pixel level, in general, the resulting visualizations do not provide useful results. One possibility to improve the visualizations is to organize the pixels in small groups and arrange the groups to form some higher-order pattern. The basic idea of the **recursive pattern visualization technique** is based on a general recursive scheme which allows lower-level patterns to be used as building blocks for higher-level patterns. In the simplest case, the patterns for all recursion levels are identical. In many cases, however, the data has some inherent structure which should be reflected by the patterns in the visualization. Consider, for example, time series data, measuring some parameters several times a day over a period of several years. It would be natural to group all data objects belonging to one day in the first level pattern, those belonging to one week in the second level pattern, those belonging to one month in the third level pattern, and so on. This, however, means that the technique must be defined in a generic fashion, allowing user-provided parameters for defining the structure of the patterns for the recursion levels.

The recursive pattern visualization technique is based on a simple back and forth arrangement: Let w_i be the number of elements arranged in the left-right direction on recursion level i and h_i be the number of rows on recursion level i. Then, the algorithm can be described as follows:

> First, w_i patterns of recursion level $(i-1)$ are arranged in left-right direction and this is repeated h_i times in top-down direction.

The pattern on recursion level i consists of level$(i-1)$-patterns and the maximum number of pixels that can be presented on recursion level k is given by $\prod_{i=1}^{k} w_i \times h_i$. In general, the visualizations get more expressive by using more recursion levels. Suppose a data set consists of data values measured nine times a day for three consecutive weeks. To visualize this data set using the "recursive pattern" algorithm, the user may enter the parameters $(w_1, h_1) = (3, 3)$ and $(w_2, h_2) = (3, 7)$, with the level(1)-pattern describing a day and the level(2)-pattern representing the three weeks. For larger data sets, the user may

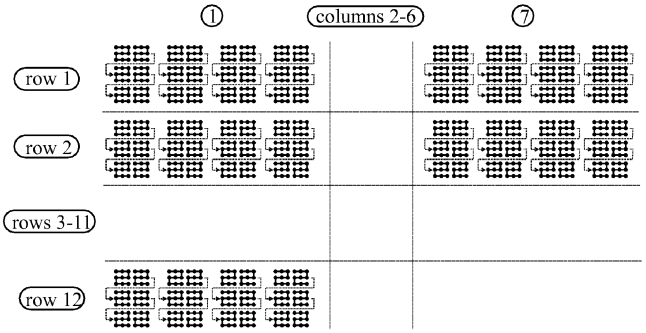


Fig. 9. Schematic representation of a highly structured arrangement $[(w_1, h_1) = (3, 3),\ (w_2, h_2) = (2, 3),\ (w_3, h_3) = (4, 1),\ (w_4, h_4) = (1, 12),\ (w_5, h_5) = (7, 1)]$. (Adapted from [39] ©IEEE.)

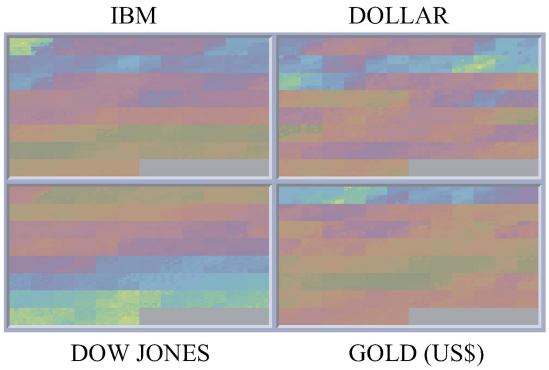


Fig 10. Five level recursive arrangement of financial data (September 1987-February 1995). (Adapted from [39] ©IEEE.)

repeat this procedure, either by enlarging the size of the second level pattern (e.g., $(w2, h2) = (28, 54)$, which corresponds to four weeks per row) or by adding additional recursion levels denoting months, years, decades, and so on. A schematic example for a highly structured arrangement using five recursion levels is provided in Fig. 9 and the resulting visualization is shown in Fig. 10. The data used in the example is the same as shown in Fig. 7b and Fig. 8b. The parameter settings of the recursive pattern visualization technique represent a semantic arrangement, i.e., they are chosen such that the level(1)-pattern represents one day, the level(2)-pattern one week, the level(3)-pattern one month, and the level(4)-pattern one year. In the resulting visualization, the eight horizontal bars correspond to the eight years and the subdivision of the bars to the 12 months within each year. By having this structure in the visualization, it is easy to get detailed information from the dense pixel display. The user may, for example, easily see that the gold price was very low in the sixth year, that the IBM price quickly fell after the first one and a half months, that the US-Dollar exchange rate was highest in June 1989, etc. These are only a few examples for useful information which can be directly derived from the visualization.

4.2 Query Dependent Arrangement

Even if each pixel of the display is used to represent one data value, the amount of information that can be represented using pixel-oriented techniques is still rather limited. The basic idea of query dependent visualization techniques is to visualize only the data which is relevant in the context of a specific query. Simple queries against the database can be described as regions in the k-dimensional space defined by the k dimensions (attributes) of the data. If exactly one query value is specified for each dimension, the query corresponds to a point in k-dimensional space; if a query range is specified for each dimension, the query corresponds to a region in k-dimensional space. The data objects which are within the query region form the result of the query. In most cases, the number of results cannot be determined a priori; the resulting data set may be quite large, or it may even be empty. In both cases, it is difficult for the user to understand the result and modify the query accordingly. To give the user more feedback on the query, we therefore do not only present the data objects which are within the query region, but also those which are "close" to the query region and only approximately fulfill the query. For determining the approximate results, distances between the data and query values are calculated. The distance functions are data type and application dependent. For numeric types such as *integer* or *real* and other metric types such as *date*, the distance of two values is easily determined by their numerical difference. For other types such as *strings*, multiple distance functions such as the lexicographical difference, character-wise difference, substring difference, or even some kind of phonetic difference may be useful. The distance calculation yields distance tuples $(d_i^1, d_i^2, \ldots, d_i^k)$ which denote the distances of the data object a_i to the query. We extend the distance tuples by a distance value d_i^{k+1}, denoting the overall distance of a data object to the query. The value of d_i^{k+1} is zero if the data object is within the query region; otherwise, d_i^{k+1} provides the distance of the data object to the query region. For combining the distance values $(d_i^1, d_i^2, \ldots, d_i^k)$ into the overall distance value d_i^{k+1}, user-provided weighting factors $(w^1, w^2, \ldots, w^k)$ are used to weight the distance values according to their importance. The distance tuples $(d_i^1, d_i^2, \ldots, d_i^k, d_i^{k+1})$ are sorted according to the overall distance d_i^{k+1} and only the $(\frac{m}{n \cdot k})$-quantile (where m is the number of pixels of the display) of the most relevant data objects is presented to the user.

The problem now is how the dimension values of the most relevant data objects are arranged in the subwindows. In principle, any of the techniques introduced in Section 4.1 can be used since the ordering according to the overall distance d_i^{k+1} from the query may be used as a one-dimensional ordering. As first experiments with those arrangements showed, however, in this case, the existing arrangements, such as the Peano-Hilbert, Morton or recursive pattern arrangement, do not provide convincing results. The reason is that all techniques display the most relevant data objects in a corner of the subwindows while the user expects them in the center of the subwindows. The pixel arrangement problem is, therefore, actually different from the case of naturally ordered data and can be formalized as follows:

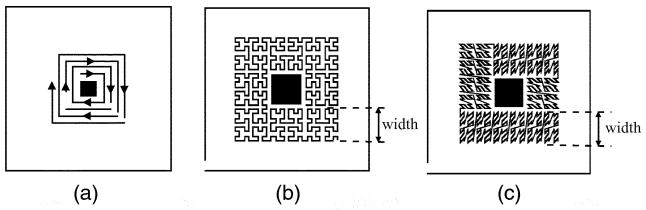


Fig. 11. Spiral and Generalized Spiral technique. (a) Spiral technique. (b) Peano-Hilbert spiral (width = 8). (c) Morton spiral (width = 8).

Definition 2 (Pixel Arrangement Problem of query dependent data). *The* pixel arrangement problem *(of query dependent data) is the problem of finding a mapping of the data objects* $\{a_1^k, \ldots, a_n^k\}$ *to a subwindow of size* $(w \times h)$*, i.e., a bijective mapping* $f : \{1 \ldots n\} \rightarrow \{1 \ldots w\} \times \{1 \ldots h\}$ *such that*

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\left| d(f(i), f(j)) - d\left((0,0), \left(w \cdot \sqrt{\frac{|i-j|}{n}}, h \cdot \sqrt{\frac{|i-j|}{n}}\right)\right)\right|$$

is minimal where $d(f(i), f(j))$ *is the* L^p*-distance of the pixels belonging to* a_i *and* a_j*, and*

$$\sum_{i=1}^{n}\left| d\left(f(i), \left(\frac{w}{2}, \frac{h}{2}\right)\right) - d\left((0,0), \left(\frac{w}{2} \cdot \sqrt{\frac{i}{n}}, \frac{h}{2} \cdot \sqrt{\frac{i}{n}}\right)\right)\right|$$

is minimal where $d(f(i), (\frac{w}{2}, \frac{h}{2}))$ *is the* L^p*-distance of the pixel belonging to* a_i *from the center.*

The first condition of Definition 2 is the same as in Definition 1 and aims at preserving the distance of the one-dimensional ordering in the two-dimensional arrangement as much as possible. The second portion adds the constraint that the distance to the center should correspond to the overall distance (i.e., the ordering of the data objects) as much as possible. The optimization formula sums up over the L^p-distances of all pixels from the center $(\frac{w}{2}, \frac{h}{2})$ and normalizes them by the minimum distance of the pixel in a rectangular subwindow of proportions $(w \times h)$.

A good solution for the second condition is a simple Spiral Arrangement as first proposed in [40], [37]. The idea of the simple spiral arrangement is to center the most relevant data objects (data objects fulfilling the query) in the middle of the window and less relevant data objects (data objects approximately fulfilling the query) are arranged in a rectangular spiral-shape to the outside of the window (cf. Fig. 11a). Although the Spiral technique already provides interesting results [37], it does not optimize the first condition, with the result that the local clustering properties of the spiral are rather week. Since the spiral is only one pixel wide, it is perceptually impossible to detect small clusters. The reason for this problem is that the mapping from the ordered sequence of data objects to the position of the pixels on the two-dimensional display does not preserve locality. More specifically, the probability that two pixels which are close together on the screen are also close together in the one-dimensional ordered sequence of data objects is rather low, which is exactly the constraint expressed by the first condition of Definition 2. From Section 4.1, we already know that arrangements which do provide a maximum of locality preservation are space-filling curves (Peano-Hilbert and Morton curve) or the recursive pattern technique. A problem of those techniques, however, is that the most relevant data objects are placed in one corner of the visualization and that the ordering of data objects does not become clear in the visualization. So, the question remains: What is a good solution optimizing both conditions?

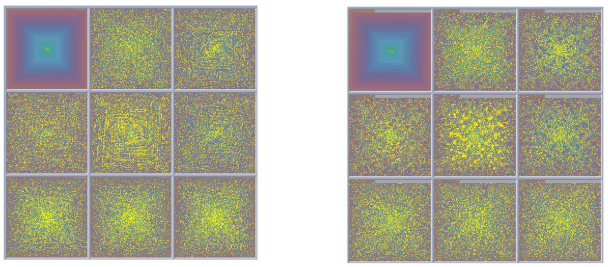


Fig. 12. Advantage of Generalized Spiral over Spiral technique. (a) Spiral technique. (b) Generalized Spiral technique. (Adapted from [33] ©American Statistical Association.)

Our solution is a combination of the Spiral and the Peano-Hilbert or Morton techniques. The Generalized Spiral-techniques retain the overall arrangement of the original Spiral-technique, centering the most relevant data objects in the middle of the screen, but enhancing the clustering properties of the arrangement by using screen-filling curves locally. This means, in case of the Generalized Spiral technique, that the primary arrangement remains a rectangular spiral shape (cf. Fig. 11a). In contrast to the original technique, however, the spiral is composed of small Peano-Hilbert- or Morton-like curves which make up the secondary arrangement (cf. Fig. 11b and Fig. 11c). The width of the spiral is determined by the size of the "small" curves. Note that the structure of the Peano-Hilbert and Morton curve does not allow arbitrary widths since the width is determined by the order of the Peano-Hilbert and Morton curves $(width = 2^{order})$.

The advantage of improving the local clustering of query-dependent visualization techniques by screen-filling curves can be easily verified using visualization examples. In Fig. 12, we provide an example visualization showing the effect of using the Generalized Spiral technique over the Spiral technique. The data set used consists of about $24,000$ test data objects with eight dimensions. Most of the data set ($20,000$ data objects) is randomly generated in the range $[-100, 100]$. The remaining $4,000$ data objects split up into two clusters which are only defined on the first five dimensions. The query used is $[-20, 20]$ for each of the dimensions. Fig. 12a shows the visualization generated by the Spiral-technique and Fig. 12b shows the visualization generated by the Generalized Spiral technique with a width of six pixels. In Fig. 12a, almost no clustering is visible, while, in Fig. 12b, the clustering becomes quite obvious by the similar structure in the first five dimensions.

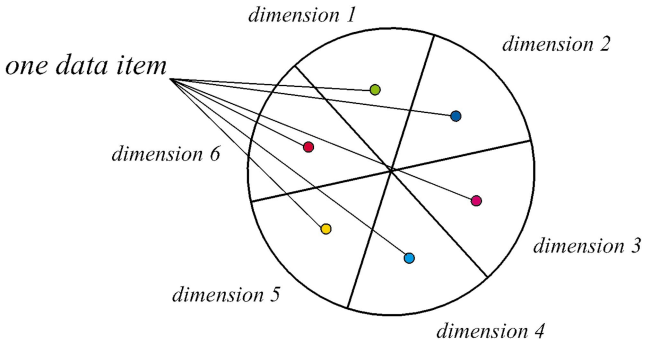


Fig. 13. Alternative shape of subwindows—circle segments.

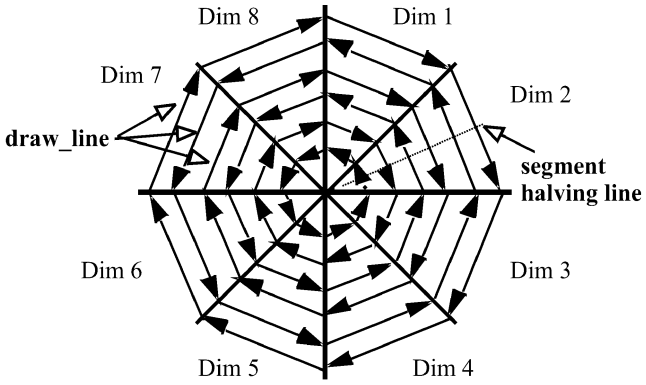


Fig. 14. Circle segment technique for eight-dimensional data.

At this point, it should be mentioned that there are a number of other query dependent arrangements such as the Axes or Grouping techniques. The idea of the Axes technique is to partition the data sets into four subsets according to the direction of the distance for two dimensions: For one dimension, negative distances are arranged to the left, positive ones to the right and, for the other dimension, negative distances are arranged to the bottom, positive ones to the top. The Axes-technique may also be improved using the same idea. The interested reader is referred to [37] for the details of those techniques and [32] discusses generalizations of those techniques and provides more examples.

5 Shape of Subwindows

The next important question is whether there exists an alternative to the regular partitioning of the screen into rectangular subwindows. The rectangular shape of the subwindows allows a good screen usage, but at the same time leads to a dispersal of the pixels belonging to one data object over the whole screen. Especially for data sets with many dimensions, the subwindows for the dimensions are rather far apart, which makes it difficult to detect clusters, correlations, etc. In the optimization functions described so far, the distance between the pixels belonging to one data object is not taken into account. This, however, is necessary in order to find alternative shapes for the dimension subwindows. The optimization goal may be expressed by the following optimization function:

Definition 3 (Subwindow Shape Problem). *The* subwindow shape problem *is the problem of finding an appropriate shape of the subwindows such that*

$$\frac{1}{n}\sum_{l=1}^{n}\left(\frac{1}{k}\sum_{i=1}^{k}\sum_{j=1}^{k} d\Big(f\big(a_l^i\big), f\Big(a_l^j\Big)\Big)\right)$$

is minimal where $d(f(a_l^i), f(a_l^j))$ is the L^p-distance $(p = 1, 2)$ of two pixels a_l^i and a_l^j belonging to two different dimensions.

The optimization problem tries to minimize the average distance between the pixels belonging to the dimension of one data object. The formula sums up all pairwise distances between the pixels belonging to one data object and then averages by the number of dimensions k to obtain an average distance of all pixels belonging to one data object. The outer sum runs over all data objects of the data set to obtain an average value of the inner sum. In the case of the rectangular arrangement used by the recursive pattern and generalized spiral techniques, the inner sum is identical for all data objects and, therefore, the outer sum is actually not necessary. If the distances are, however, different for the data objects of the data set, the outer sum is crucial to obtain a useful optimization function.

An idea for an alternative shape of the subwindows which optimizes the above function is the circle segments technique. The fundamental idea of the "circle segments" visualization technique is to display the data dimensions as segments of a circle (cf. Fig. 13). If the data consists of k dimensions, the circle is partitioned into k segments, each representing one data dimension. The data objects within one segment are arranged in a back and forth manner along the so-called "draw_line," which is orthogonal to the line that halves the two border lines of the segment (cf. Fig. 14). The "draw_line" starts in the center of the circle and draws the pixels from one border line of the segment to the other. Whenever the "draw_line" hits one of the border lines, the "draw_line" is moved in parallel along the segment-halving line to the outside of the circle and the direction of the "draw_line" changes. This process is repeated until all data objects of one dimension are visualized and, then, the whole procedure is restarted for the remaining dimensions.

Fig. 16a provides an example of a circle segments visualization showing 50 stock from the Frankfurt stock index (FAZ) over a period of 20 year, resulting in about $265,000$ data values. Because of the high degree of overlap, "line graphs" are not suitable for visualizing this many stacks (cf. Fig. 16c for a line graph of eight stocks with value aggregated for one week). In Fig. 16b, we compare the "circle segments" technique with the "recursive pattern" technique (cf. Fig. 16b). The main advantage of our new technique is that the overall presentation of the whole data set is better perceivable—including potential dependencies, analogies, and correlations between the dimensions. The advantage can be easily seen in the comparison of Fig. 16, but it can also be evaluated in terms of the optimization function as expressed in Definition 3. If we compute the average distance of the pixels belonging to one data object for the circle segments and the recursive pattern techniques, we see that the circle segments technique provides a better optimization, especially for larger dimensionalities (k) (cf.

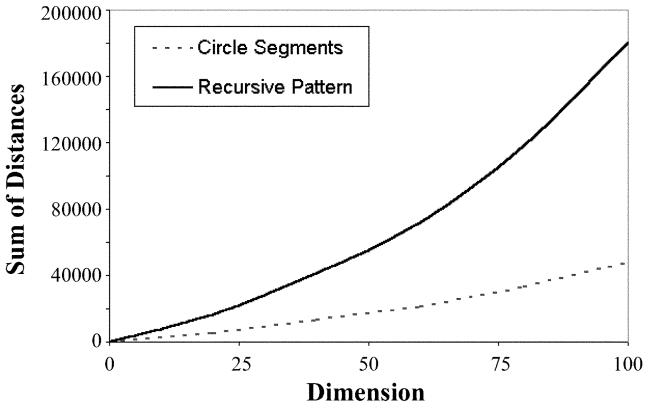


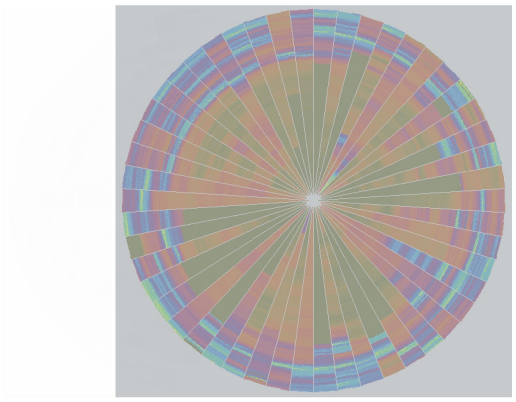
Fig. 15. Evaluation of the optimization function.

Fig. 15). This result shows the correspondence of the mathematical forms and experimental results and confirms the soundness of the optimization goal.

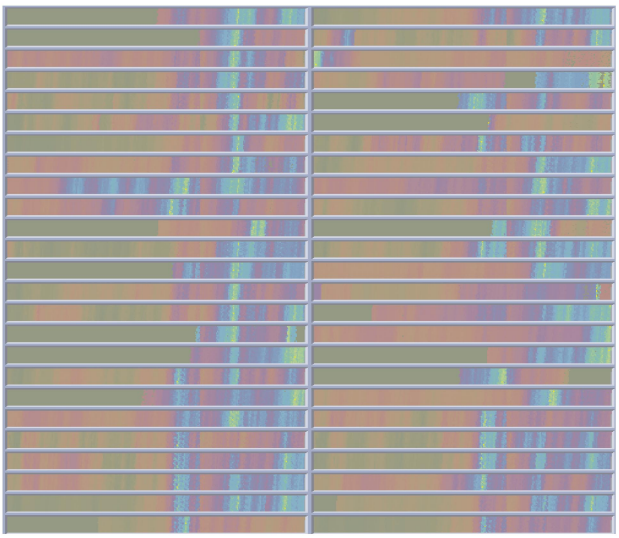
6 Ordering of Dimensions

The next question to consider is the ordering of dimensions. This problem is actually not just a problem of pixel-oriented techniques, but a more general problem which arises for a number of other techniques, such as the parallel coordinates technique, as well. The basic problem is that the data dimensions have to be positioned in some one- or two-dimensional ordering on the screen and this is usually done more or less by chance—namely, in the order in which the dimensions happen to appear in the data set. The ordering of dimensions, however, has a major impact on the expressiveness of the visualization. Consider, for example, the parallel coordinates technique [28], [29]. If one chooses a different order of dimensions, the resulting visualization becomes completely different and allows different conclusions to be drawn. Techniques such as the parallel coordinates technique and the circle segments technique require a one-dimensional ordering of the dimensions. In case of other techniques—such as the recursive pattern technique or the generalized spiral technique—a two-dimensional ordering of the dimensions is required.

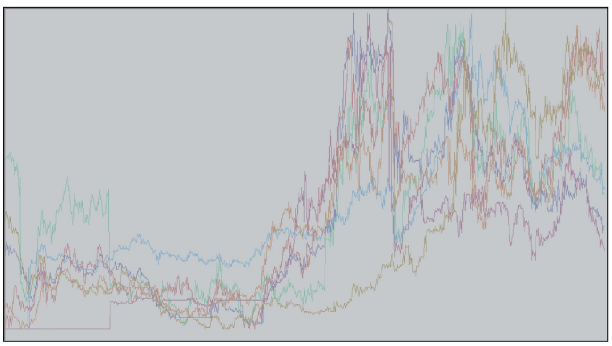
The basic idea of our approach for finding an effective order of dimensions is to arrange the dimensions according to their similarity. For this purpose, we first have to define similarity measures which determine the similarity of two dimensions. These similarity measures may be based on a partial or global similarity of the considered dimensions (cf. Section 6.1). For determining the similarity, a simple Euclidean or more complex (e.g., Fourier-based) distance measures may be used. Based on the similarity measure, we then have to determine the similarity ordering of dimensions. After formally defining the one- and two-dimensional ordering problems, we show that all variants of the ordering problem are computationally hard (i.e., NP-complete) problems (cf. Section 6.2). For solving the problems, we therefore have to use heuristic algorithms which are shown to work effectively for the circle segments and recursive pattern techniques (cf. Section 6.3).



(a)



(b)



(c)

Fig. 16. Twenty years of daily data of the FAZ Index (January 1974-April 1995). (a) Circle segments visualization of 50 stocks. (b) Recursive pattern visualization of 50 stocks. (c) Line graph visualization of eight stocks (one week aggregated into one value).

6.1 Similarity of Dimensions

The problem of determining the similarity of dimensions may be characterized as follows: Let A_i be the set of all data values $\{a_1^i, \ldots, a_N^i\}$ for dimension i $(0 \leq i < k)$. The similarity of two dimensions $S : \mathbb{R}^N \times \mathbb{R}^N \rightarrow \mathbb{R}$ is a function $S(A_i, A_j)$ which determines a value for the similarity of A_i

and A_j.[3] All meaningful similarity measures S must have the following properties $(0 \leq i < k)$:

1. positivity: $\forall A_i, A_j \in \mathbb{R}^k : S(A_i, A_j) \geq 0$.
2. reflexivity: $\forall A_i, A_j \in \mathbb{R}^k : (A_i = A_j) \Leftrightarrow S(A_i, A_j) = 0$.
3. symmetry: $\forall A_i, A_j \in \mathbb{R}^k : S(A_i, A_j) = S(A_j, A_i)$.

As outlined in [1], similarity is highly application dependent. Depending on the application, one has to consider global or partial similarity and invariance with respect to translation or scaling. An example for a global similarity measure which is translation invariant, is

$$S_{trans}(A_k, A_l) = \sqrt{\sum_{i=0}^{N-1} \Big(\big(a_i^k - mean(A_k)\big) - \big(a_i^l - mean(A_l)\big) \Big)^2},$$

where

$$mean(A_i) = \frac{1}{N} \sum_{k=0}^{N-1} a_i^k.$$

If one additionally demands invariance against scaling, the dimension can be scaled independently such that the maximum value of a dimension becomes 1 and the minimum becomes -1. Thus, scaling invariant global similarity can be computed as

$$S_{scaling}(A_k, A_l) = \sqrt{\sum_{i=0}^{N-1} (b_i^k - b_i^l)^2},$$

where[4]

$$b_j^i = \frac{a_j^i - MIN(A_i)}{MAX(A_i) - MIN(A_i)}.$$

For most real-life applications, partial similarity measures are more appropriate than global ones. Imagine two stock rates over time, say AT&T and IBM. Of course, there will be weeks, or even months, where the two stocks show a similar behavior, e.g., because some global development (such as a black Friday) is going on. However, it is very unlikely that the AT&T and IBM stocks behave similarly over a period of 10 years. Therefore, we are actually interested in periods where the AT&T and IBM stocks behaved similarly. Thus, given the two dimensions A_k and A_l, in the most simple case, we are looking for

$$S_{sync}(A_k, A_l) = \max_{i,j} \left\{ (j-i) \mid (0 \leq i < j < N) \wedge \sqrt{\sum_{z=i}^{j} (b_z^k - b_z^l)^2} \right\} < \varepsilon,$$

where b_y^x is defined as above and ε is some maximum allowed dissimilarity. This partial similarity measure uses the length of the longest sequence, which is at least ε-similar (under scaling and translation invariance).

3. One might also call S a dissimilarity measure because large numbers mean high dissimilarity, whereas zero means identity.

4. In order to become more robust against outliers, instead of using MAX (the 100 percent-quantile) and MIN (the 0 percent-quantile), we use the 98 percent and 2 percent quantile of A_i.

6.2 Similarity Ordering of Dimensions

The mapping of the dimensions into the visual representation is fundamental for the perception of the user. The ordering of dimensions especially plays a significant role, e.g., for the detection of functional dependencies and correlations. It is therefore important to adequately arrange the dimensions. In the following, we define the dimension ordering problem mathematically as an optimization problem which ensures that the most similar dimensions are placed next to each other.

Depending on the considered visualization technique, we have to distinguish between the one-dimensional and the two-dimensional ordering problem. The one-dimensional ordering problem occurs, for example, for the circle segment techniques and the two-dimensional problem occurs, for example, for the recursive pattern and generalized spiral techniques.[5] In case of the one-dimensional ordering problem, there are two slightly different variants of the problem—the linear and the circular problem (cf. Fig. 17). In the case of the linear one-dimensional ordering problem, the first and last dimensions do not have to be similar, whereas, in the case of the circular problem, the dimensions form a closed circle, i.e., first and last dimension have to be similar. In the following, we assume to have a symmetric $(k \times k)$ similarity matrix

$$S = \begin{bmatrix} S(A_0, A_0) & \dots & S(A_{k-1}, A_0) \\ \vdots & \ddots & \vdots \\ S(A_0, A_{k-1}) & \dots & S(A_{k-1}, A_{k-1}) \end{bmatrix},$$

where $S(A_i, A_j) = S(A_j, A_i)\ \forall i, j = 0, \dots, (k-1)$ and $S(A_i, A_i) = 0\ \forall i = 0, \dots, (k-1)$. $S(A_i, A_j)$ describes the similarity between dimension i and dimension j. The similarity matrix is the result of applying the global or partial similarity measures introduced in Section 6.1. In addition, we need a $(k \times k)$ neighborhood matrix

$$N = \begin{bmatrix} n_{00} & \dots & n_{(d-1)0} \\ \vdots & \ddots & \vdots \\ n_{0(k-1)} & \dots & n_{(k-1)(k-1)} \end{bmatrix},$$

which describes the neighborhood relation between the dimensions in the ordering. The matrix N is also symmetric (i.e., $(n_{ij} = n_{ji} \ \wedge \ n_{ii} = 0)\ \forall i, j = 0, \dots, (k-1)$) and

$$n_{ij} = \begin{cases} 1 & \text{if dimensions } i \text{ and } j \text{ are neighbors} \\ 0 & \text{otherwise.} \end{cases}$$

Now, we are able to define the general ordering problem as follows:

Definition 4 (General Ordering Problem). *For a given similarity matrix S, the optimal ordering of dimensions is given by a neighborhood matrix N such that*

5. Note that the same problem also occurs for other visualization techniques which do not work on a pixel-oriented basis such as the parallel coordinate technique.

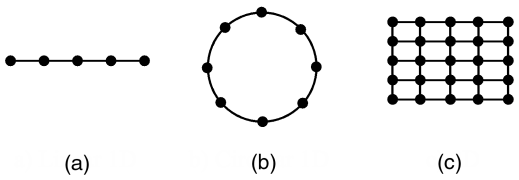


Fig. 17. One- and two-dimensional arrangement problems. (a) Linear 1D. (b) circular 1D. (c) 2D.

$$\sum_{i=0}^{k-1}\sum_{j=0}^{k-1} n_{ij} \cdot S(A_i, A_j)$$

is minimal.

This definition is a general notion of the problem which defines the optimal ordering of dimensions. The specific one- and two-dimensional ordering problems of the existing visualization techniques, such as the parallel coordinates, circle segments, and spiral techniques, are instantiations of the problem. In the case of the one-dimensional ordering problem, the neighborhood matrix reflects either the linear (cf. Fig. 17a) or the circular ordering of the dimensions (cf. Fig. 17b). The linear ordering problem occurs, for example, in the case of the parallel coordinate technique and the circular ordering problem occurs, for example, in the case of the circle segments technique.

Definition 5 (One-Dimensional Ordering Problem). *For a given similarity matrix S, the optimal one-dimensional ordering of dimensions is a permutation $\{\pi(0), \ldots, \pi(k-1)\}$ of the dimensions such that:*

1. *Circular Case:* $\sum_{j=0}^{k-1} S\left(A_{\pi(i)}, A_{\pi((i+1) \bmod k)}\right)$ *is minimal.*
2. *Linear Case:* $\sum_{j=0}^{k-2} S\left(A_{\pi(i)}, A_{\pi(i+1)}\right)$ *is minimal.*

In the case of the two-dimensional ordering of dimensions, without loss of generality, we assume $k = k_1 \cdot k_2$, where k_1 corresponds to the number of rows and k_2 corresponds to the number of columns of the arrangement. Then, the two-dimensional ordering problem can be defined as follows:

Definition 6 (Two-Dimensional Ordering Problem). *For a given similarity matrix S, the optimal two-dimensional ordering of dimensions is the arrangement $\pi(i,j)$ $(i = 1 \ldots k_1,\ j = 1 \ldots k_2)$ such that*

$$\sum_{i=0}^{k_1-2}\sum_{j=0}^{k_2-1} S\left(A_{\pi(i,j)}, A_{\pi(i+1,j)}\right) + \sum_{i=0}^{k_1-1}\sum_{j=0}^{k_2-2} S\left(A_{\pi(i,j)}, A_{\pi(i,j+1)}\right)$$

is minimal.

The first portion of the formula corresponds to the sum of the distances in the rows and the second portion to the sum of the distances in the columns of the two-dimensional ordering.

In the following, we briefly discuss the complexity of the one- and two-dimensional ordering problems. We show that even the one-dimensional ordering problems are computationally hard problems, i.e., they are NP-complete.

Lemma 1 (NP-Completeness of the Circular 1D Problem). *The circular variant of the one-dimensional ordering problem according to Definition 5 is NP-complete.*

Proof. The problem is obviously equivalent to the well-known traveling salesman problem (TSP), which is known to be NP-complete. We just have to map the dimensions to cities, the similarity between the dimensions to the cost of traveling between cities, and the solution back to the ordering of dimensions. □

In the case of the linear one-dimensional and the two-dimensional ordering problems, the proofs of the NP-completeness are more complex and will therefore be provided in the appendix.

Lemma 2 (NP-Completeness of the Linear 1D Problem). *The linear variant of the one-dimensional ordering problem, according to Definition 5, is NP-complete.*

Proof. See Appendix. □

Lemma 3 (NP-Completeness of the 2D Ordering Problem). *The two-dimensional ordering problem, according to Definition 6, is NP-complete.*

Proof. See Appendix. □

6.3 Dimension Ordering Algorithm

Since the dimension ordering problems are NP-complete, we have to use heuristic algorithms to solve the problem. Since the problems are all similar to the traveling salesman problem, we can use variants of the existing heuristic algorithm which have been proposed for the traveling salesman problem, such as memetic and genetic algorithms, tabu search, ant colony optimization, neural networks, space-filling heuristics, or simulated annealing. For an overview of these approaches, including an extensive bibliography, see [53].

In our implementation, we use a variant of the ant system algorithm, which is inspired by the behavior of real ants [19]. Ants are able to find good solutions to shortest path problems between a food source and their home colony. Ants deposit a certain amount of pheromone while walking and each ant probabilistically prefers to follow a direction rich in pheromone. The pheromone trail evaporates over time, i.e., it looses intensity if no more pheromone is laid down by other ants.

In our variant of the algorithm, which was first proposed in [1], we have transferred three ideas from natural ant behavior to our artificial ant colony: 1) the trail mediated communication among ants, 2) the preference for paths with a high pheromone level, and 3) the higher rate of growth of the amount of pheromone on shorter paths. An artificial ant is an agent which moves from dimension to dimension on the neighborhood graph where the length of the edges equals to the distance (dissimilarity S) between the corresponding dimension nodes. Initially, m artificial ants are placed on randomly selected dimensions. At each time step, they move to new dimensions and modify the pheromone trail on the edges passed. The ants choose the next dimension by using a probabilistic function depending on both the trail accumulated on edges and on a heuristic value which is chosen as a function of the edge length.

Obviously, the ants must have a working memory used to memorize the dimensions already visited. When all ants have completed a tour, the ant which made the shortest tour modifies the edges belonging to its tour by adding an amount of pheromone trail which is inversely proportional to the tour length. This procedure is repeated for a given number of cycles.

In our version of the ant colony system, an artificial ant k at dimension r chooses dimension s to move to (s is among the dimensions which do not belong to its working memory M_j) by applying the following probabilistic formula:

$$s = \begin{cases} \max_u \left\{ [\tau(r,u)] \cdot [\eta(r,u)]^\beta \right\} & \text{if } (q \leq q_0) \\ T & \text{otherwise,} \end{cases}$$

where $\tau(r,u)$ is the amount of pheromone trail on edge (r,u), $\eta(r,u)$ is a heuristic function which is chosen to be the inverse of the distance between dimensions r and u, β is a parameter which weighs the relative importance of pheromone trail and of closeness, q is a value chosen randomly with uniform probability in $[0,1]$, q_0 $(0 \leq q_0 \leq 1)$ is a parameter, and T is a random variable selected according to the following probability distribution, favoring dimensions with small distances and higher levels of pheromone trail:

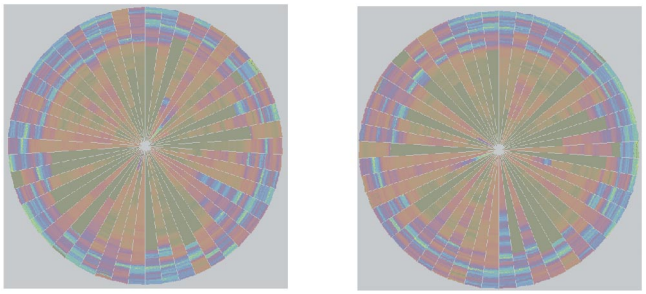
$$p_j(r,s) = \begin{cases} \frac{[\tau(r,u)] \cdot [\eta(r,u)]^\beta}{\sum_{u \notin M_j} [\tau(r,u)] \cdot [\eta(r,u)]^\beta} & \text{if } (s \notin M_j) \\ 0 & \text{otherwise,} \end{cases}$$

where $p_j(r,s)$ is the probability that ant j chooses to move from dimension r to dimension s.

We applied this heuristic to arrange the dimensions according to their distances. In the one-dimensional ordering case, the only difference between the linear and the circular variant is that the tour consists of one more dimension and that the ants move back to the starting dimension. For the two-dimensional ordering problem, we have to slightly modify the algorithm described above. Let k_1 be the number of rows and k_2 be the number of columns of the two-dimensional ordering and let us assume that we map the sorted dimensions on the ordering in a row-wise manner, always filling the rows from the left to right. Thus, the $k = k_1 \cdot k_2$ ordered dimensions are mapped to the ordering such that the dimension number j is mapped to column number $1 + ((j-1) \bmod k_2)$ and to row number $\lceil n/k_2 \rceil$. Let $S(A_i, A_j)$ be the distance between dimension A_i and dimension A_j and $M_j(m)$ be the dimension in the mth position in the working memory. Then, we modify the heuristic function as

$$\eta(r,u,n) = \begin{cases} \frac{1}{S(A_r,A_u)} & \text{if } \lceil n/k_2 \rceil = 1 \\ \frac{1}{S(A_u,A_{M_j(n+1-k_2)})} & \text{if } (n-1) \bmod k_2 = k_2 - 1 \\ \frac{1}{2} \cdot \left(\frac{1}{S(A_r,A_u)} + \frac{1}{S(A_u,A_{M_j(n+1-k_2)})} \right) & \text{else.} \end{cases}$$

In the two-dimensional version of the algorithm, the heuristic function $\eta(r,u,n)$ also depends on n which is the number of dimensions already in working memory. This function results in the inverse of the distance to the next dimension in case of arranging the first uppermost row. The second condition is fulfilled if a dimension for the first or last column is chosen. In this case, we consider the inverse of the distance to the dimension located in the same column one row above. In all other cases, we consider the average of the inverse of the distances to its already known neighbors.



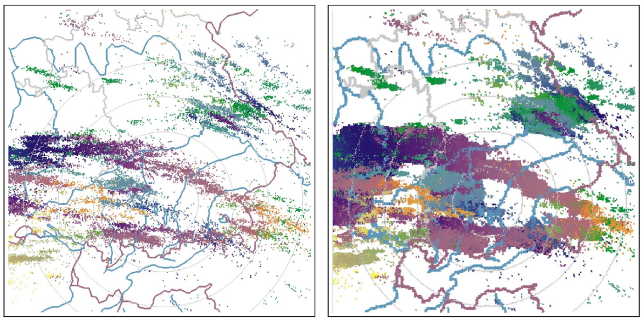
(a) (b)

Fig. 18. Visualizations generated using the circle segments visualization technique. (a) Sequential arrangement. (b) Similarity arrangement. (Adapted from [1] ©IEEE.)

In Fig. 18, we demonstrate the advantage of the similarity ordering using our stock exchange database introduced in Section 5. The similarity measure used is based on the translation- and scaling-invariant partial similarity measure described in Section 6.1. In comparing the sequential (cf. Fig. 18a) with the similarity arrangement (cf. Fig. 18b), our new ordering allows the user to see clusters, correlations, and functional dependencies more easily. The segments on the right side of the circle, for example, all seem to have a peak (light color) at the outside, which corresponds to approximately the same period of time. Seven dimensions on the upper left side seem to have their peaks in a different period of time and—because they are placed next to each other—it is easy to compare them and to find differences between them. More examples are provided in [1].

7 Geometry-Related Data

There are a large number of applications where geometry-related data arises. Examples include weather measurements, such as temperature, rainfall, wind-speed, etc., measured at a large number of locations, use of connecting nodes in telephone business, load of a large number of internet nodes at different locations, air pollution of cities with a certain number of inhabitants, etc. Visualizing this type of information requires representing the data values (e.g., air pollution) and their spatial location. A natural way to visualize the data would be, for example, to represent the data values as colored pixels on a screen position which directly correlates to the spatial location of the data. Since the spatial locations of the data are not uniformly distributed in a rectangular data space, however, the display will usually be sparsely populated in some regions, while, in other regions of the display, a high degree of overplotting occurs. Consider, for example, the air pollution example from above. The cities with more than $10,000$ inhabitants cluster in few places (such as North America, Europe, Asia, etc.), while large portions of the earth are only



(a) (b)

Fig. 19. Lightening strike data. (a) With overlap. (b) Without overlap. (Adapted from [36] ©IEEE.)

sparsely inhabited. In addition, if the data is presented on a world map, the large portion of the screen which corresponds to oceans is not used, while only a few data values corresponding to European cities may be displayed. This results in a loss of potentially important information.

A simple but intuitive idea to avoid this problem is to present data values which cannot be presented at the correct position on the screen at nearby unoccupied positions. If the values are presented in an appropriate way, the visualization naturally reflects the spatial location of the data and the loss of information can be avoided. In addition, as many pixels of the display as necessary are used while still reflecting the spatial nature of the data. In Fig. 19, we show a data set of lightning strikes in southern Germany over a period of time. On the left, the data set is presented without overlapping data points, whereas, on the right, overlapping data points are placed on unoccupied pixels close to their original position, thereby avoiding less information.

7.1 The Problem of Visualizing Geometry-Related Data

The problem of visualizing spatially referenced data can be described as a mapping between the multiset of original positions and the set of new positions. Let A be the data set of original positions $A = \{a_0, \ldots, a_{N-1}\}$ with $a_i = (a_i^x, a_i^y)$, where it is possible that $a_i = a_j$ for an arbitrary i and j. Let the data space (or, better, screen space) $DS \subseteq \mathbb{Z}^2$ be defined as $DS = \{0, \ldots, x_{max} - 1\} \times \{0, \ldots, y_{max} - 1\}$, where x_{max} and y_{max} are the maximal extension of the screen.

Definition 7 (Problem of Visualizing Geometry-related Data). *The goal in visualizing geometry-related data is to determine a solution set $S = \{s_0, \ldots, s_{N-1}\}$ of new positions, with s_i being the new position of a_i, such that*

$$i \neq j \Rightarrow s_i \neq s_j \qquad \forall i, j \in \{0, \ldots, N-1\}$$

and the following criteria are optimized:

1. *absolute position-preservation*

$$\sum_{i=0}^{N-1} d(a_i, s_i) \rightarrow \min,$$

2. *relative position-preservation*

$$\sum_{i=0}^{N-1} \sum_{j=0, j\neq i}^{N-1} d(s_i, s_j) - d(a_i, a_j) \rightarrow \min,$$

3. *relative distance-preservation*

$$\sum_{i=0}^{N-1} \sum_{j=0, j\neq i}^{N-1} \frac{d(s_i, s_j)}{d(a_i, a_j)} \rightarrow \min,$$

where d is an arbitrary distance metric in 2D such as $d(a_i, a_j) = |a_i^x - a_j^x| + |a_i^y - a_j^y|$ or the Euclidean metric

$$d(a_i, a_j) = \sqrt{(a_i^x - a_j^x)^2 + (a_i^y - a_j^y)^2}.$$

The goal is that the resulting visualization should be as similar as possible to the visualization of the original data. The similarity may be defined by the absolute distance of the data points to their original positions (cf. first condition) or by the relative distance (cf. second condition) or relative position (cf. third condition) between the data points. The optimization goals make sure that as little as possible of the spatial information is lost. Which of the three optimization goals is most important and should be fulfilled first depends on the application. The formal description of the problem indicates that finding a mapping which fulfills the above properties is a typical optimization problem. Most typical optimization problems are NP-complete and we assume that our mapping problem also belongs to the class of NP-complete problems. A formal proof, however, has not yet been found.

7.2 The Nearest-Neighbor, Curve, and Gridfit Algorithms

In [36], we proposed three solutions to solve the problem described in Definition 7. Let us briefly recall the three algorithms: All three algorithms work in two steps. In Step 1, all data points a_i which have unique positions (i.e., $a_i \neq a_j \quad \forall a_j \in A_j \neq i$) are placed on the display. In the second step, a new position which is as close as possible to their original position is determined for the remaining data points. More formally, the general idea of the algorithm can be described as follows: The set of points S is the set of data points with unique positions and the set of points T is a temporary set to be placed in the second step.

Step 1:

- $S_0 = \{a_0\},\ T_0 = \emptyset$
- $$S_{i+1} = \begin{cases} S_i \cup \{a_{i+1}\} & \text{if } (a_{i+1} \neq s)\ \forall s \in S_i \\ S_i & \text{otherwise} \end{cases}$$
- $$T_{i+1} = \begin{cases} T_i & \text{if } (a_{i+1} \neq s)\ \forall s \in S_i \\ T_i \cup \{a_{i+1}\} & \text{otherwise} \end{cases}$$

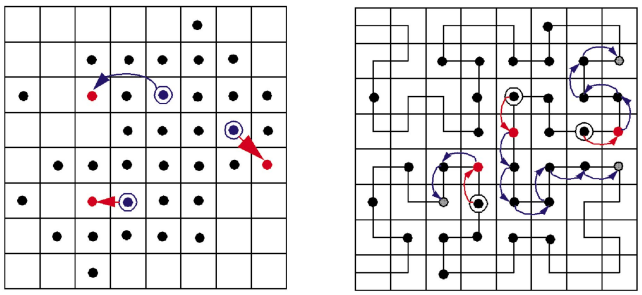


Fig. 20. Nearest-neighbor and curve-based algorithms. (a) NN-algorithm. (b) Curve-based algorithm. (Adapted from [36] ©IEEE.)

Step 2:

- $S'_0 = S_{N-1}$, $T'_0 = T_{N-1}$
- $T'_{i+1} = T'_i - \{a\}$ for an arbitrary $a \in T'_i$
- $S'_{i+1} = S'_i \cup \{a_{\text{new}}\}$ where a_{new} is the new position corresonding to a which is determined differently by the three algorithms.

7.2.1 Nearest-Neighbor Algorithm

In the case of the nearest-neighbor algorithm, the new position for the data points which do not have a unique position and have therefore not been placed in the first step is determined by simply placing the data points on the nearest unoccupied positions (cf. Fig. 20a). More formally, the new position a_{new} of a data point a is determined as

$$a_{\text{new}} = \{s' \in DS \setminus S_i \mid d(a, s') \leq d(a, s)\ \forall s \in DS \setminus S_i\}.$$

An advantage is that the new position can usually be determined locally and therefore, in general, the algorithm works rather efficiently. For a very dense display, however, the efficiency and effectiveness suffer from the fact that the new position may be rather far from the original position.

7.2.2 Curve-based Algorithm

In the case of the curve-based algorithm, the new position for the data points which do not have a unique position and have therefore not been placed in the first step is determined by computing the nearest unoccupied positions on a given screen-filling curve and shifting all data points between the overlapping data point and the unoccupied position in that direction (cf. Fig. 20b). Screen-filling curves, such as the Hilbert-curve [50], [26] or Z-curve [48], provide a bijective mapping between a position on a one-dimensional line and a two-dimensional position. The advantage is that data points which are close in 1D are usually mapped to nearby points in 2D and vice versa. The idea of the curve-based algorithm is to shift the data points along the screen-filling curve (1D) which in general also corresponds to nearby positions in 2D.

7.2.3 The Gridfit Algorithm

The basic idea of the Gridfit algorithm is to hierarchically partition the data space. In each step, the data set is partitioned into four subsets containing the data points which belong to four equally sized subregions. Since the data points may not fit into the four equally sized subregions, we have to determine a new extend of the four subregions (without changing the four subsets of data points) such that the data points in each subset can be visualized in the corresponding subregion. For an efficient implementation of the algorithm, a quadtree-like data structure is used to manage the required information and to support the recursive partitioning process. The partitioning process works as follows: Starting with the root of the quadtree, in each step, the data space is partitioned into four subregions. The partitioning is made such that the area occupied by each ubregion (in pixels) is larger than the number of pixels belonging to the corresponding subregion (cf. Fig. 21). For the details of the three algorithms, the reader is referred to [36].

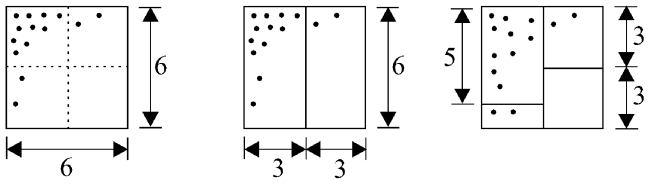


Fig. 21. Partitioning of a node. (Adapted from [36] ©IEEE.)

7.3 Evaluation

All algorithms introduced in Section 7.2 have been implemented as part of the **VisualPoints** system. The system is implemented in C++ and is running under HP-UX and LINUX. We used the **VisualPoints** system to evaluate and compare the different algorithms. We evaluated not only the efficiency, but also their mathematically defined absolute and relative position- and distance-preservation and their visual effectiveness.

7.3.1 Efficiency

The theoretical time and space complexities of the three algorithm are all similar. In all three cases, the space complexity is $O(N)$ and the time complexity is between $O(N)$ in the best case and $O(N^2)$ in the worst case. An experimental evaluation based on different realistic data sets, however, clearly shows the advantage of the Gridfit algorithm. Since the number of data points which on the average would be positioned at the same pixel plays an important role in all algorithms, we used data sets with a

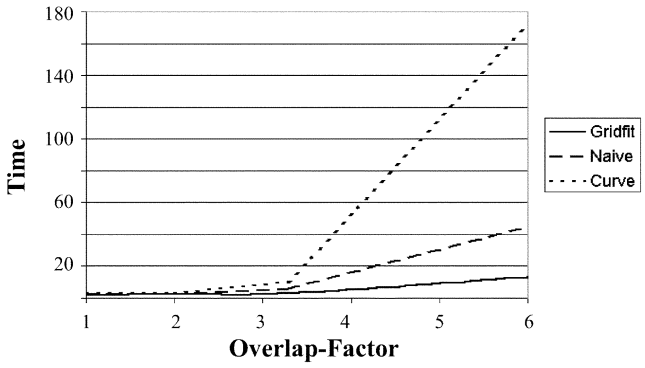


Fig. 22. Comparison of the efficiency. (Adapted from [36] ©IEEE.)

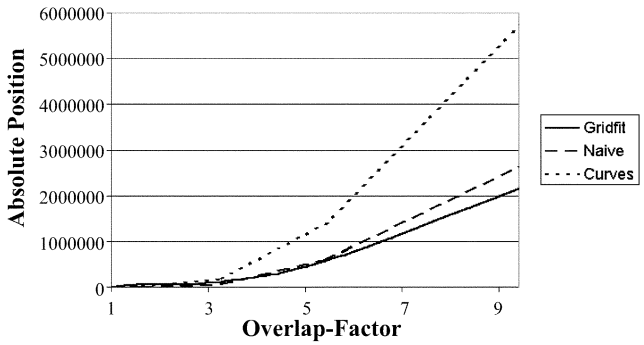


Fig. 23. Absolute position measure. (Adapted from [36] ©IEEE.)

different overlap-factor (OF). In Fig. 22, we provide the performance curves for a varying overlap-factor. It is clear that, for all algorithms, the time increases significantly with an increasing overlap-factor, with the Gridfit algorithm being the clear winner, closely followed by the Nearest-Neighbor algorithm.

7.3.2 Effectiveness

More important than the efficiency, however, is the effectiveness of the visualizations. The effectiveness can be determined by visually comparing the generated visualizations, but it can also (at least partially) be determined mathematically according to our optimizations goals. Before presenting the visual comparison, we therefore briefly present the measured effectiveness, i.e., the absolute and relative position and distance preservation (cf. Definition 7) for an L^1 distance function (d).

In Fig. 23, we present the absolute position measure of the three algorithms depending on the overlap-factor. Fig. 23 clearly shows that the Gridfit algorithm provides a smaller average deviation from the original position than the nearest-neighbor and curve-based algorithms, especially for higher overlap factors, which can also be confirmed by the visual comparison. Fig. 24 presents the development of the relative position and relative distance measure with increasing overlap-factor. It measures how good the relative position of the data points is preserved in the visualization (a smaller value means a better relative position preservation). Here, the advantage of the Gridfit algorithm over the nearest-neighbor and curve-based algorithms becomes even more impressive. The improvement is up to 390 percent over the nearest-neighbor and up to 870 percent over the curve-based algorithm. Fig. 24b shows the relative distance measure of the three algorithms for an overlap-factor of about 10. In this case, all three algorithms provide about the same performance and the value of the Gridfit algorithm is between the nearest-neighbor and the curve-based algorithms.

All formal effectiveness measures as defined by the absolute and relative position and distance are of limited value if they do not correspond to improvements in the generated visualizations. We therefore performed a detailed visual comparison of the three techniques which, in general, confirms the measured effectiveness criteria.

Our first comparison uses a grayscale world data map with simulated points distributed over the surface of dry

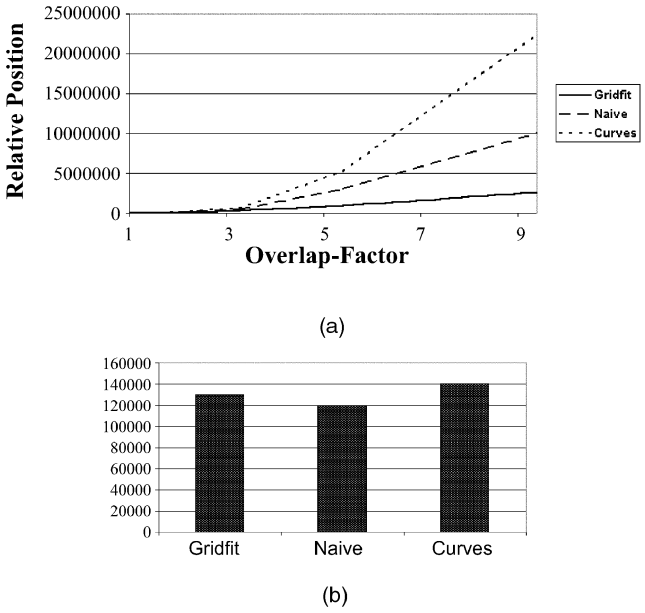


Fig. 24. Relative position and relative distance measure. (a) Relative position measure. (b) Relative distance measure. (Adapted from [36] ©IEEE.)

land. Fig. 25 shows the result of visualizing the data using our nearest-neighbor, curve-based, and Gridfit algorithms on two different resolutions. The nearest-neighbor algorithm provides nice results, at least for the portion of data which can be placed at its original position in the first step of the algorithm. The contours of the continents are clearly visible in their original size. All data points which cannot be placed in the first step, however, do not show any structure (cf. right portion of a Fig. 25a). In the case of the curve-based algorithm, the continents are rather distorted and their contours are barely visible. In the lower resolution picture (cf. right portion of Fig. 25b), there seems to be no similarity to the original image. This result corresponds to the result of our theoretical effectiveness comparison,, which showed that the curved-based algorithm is worse than the other two approaches. The visualization generated by the Gridfit algorithm also confirms the results of the theoretical effectiveness comparison, namely that the Gridfit algorithm provides significantly better results than the other two approaches (cf. right portion of Fig. 25c). In contrast to the nearest-neighbor algorithm, the Gridfit algorithm enlarges and distorts the contours such that all points can be placed close to their original position. As a result, the visualization retains the spatial locality of the data points as much as possible, which results not only in a better (absolute and relative) position-preservation but also in a better visual representation of the data.

To analyze the properties of three algorithms in more detail, we designed a synthetic data set consisting of a number of objects with different properties (cf. Fig. 26a). Besides a few simple objects, such as circles, straight and curved lines, we also include vertical and horizontal bars and text, as well as rectangular and circular patterns. In the visualizations generated by our three algorithms, some of the properties of the algorithms get clearer. Again, the curve-based algorithm provides the poorest results for all

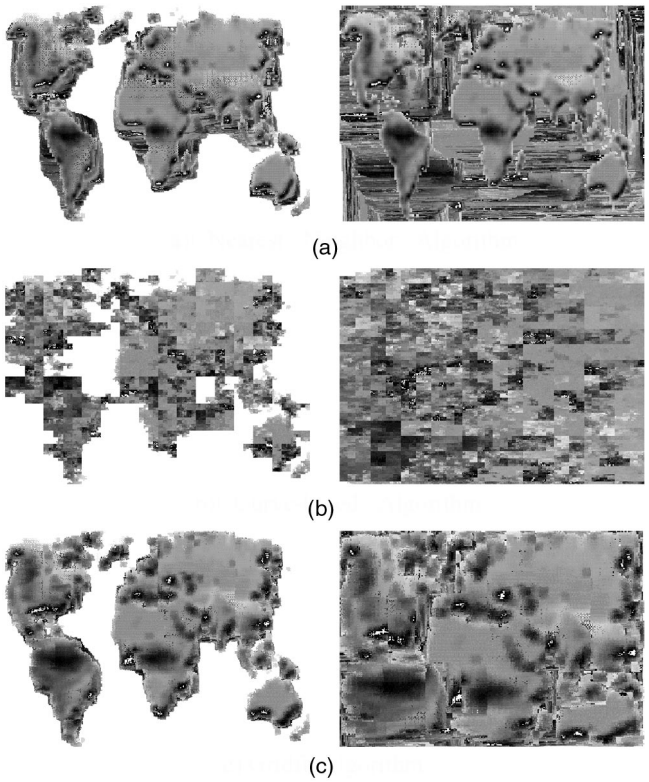


Fig. 25. Comparison of the effectiveness. (a) Nearest-neighbor algorithm. (b) Curve-based algorithm. (c) Gridfit algorithm. (Adapted from [36] ©IEEE.)

types of objects (cf. Fig. 26c). The nearest-neighbor algorithm provides rather good results, especially for the text (cf. Fig. 26b). The main problem of the algorithm, however, are the rectangular and circular patterns, which show the correct result in the center, but no structure for the overlapping data points positioned in the second step of the algorithm. Since patterns are very important in data exploration, this turns out to be major drawbacks of the nearest-neighbor algorithm. In contrast, the Gridfit algorithm performs nicely for all types of objects (cf. Fig. 26d). Note that the rectangular and circular patterns are enlarged, which is a desired effect for preserving the spatial locality as much as possible.

8 Summary and Conclusions

Pixel-oriented visualization techniques have been shown to be useful for the exploration and analysis of large databases to find interesting data clusters and their properties. So far, most of the techniques seem to be ad hoc solutions without any formal basis. In this paper, we show that underlying the development and design of pixel-oriented techniques there are a number of serious optimization problems which have to be solved. We provide the formal problem definitions and show how the optimizations of different criterions lead to the different variants of pixel-oriented techniques.

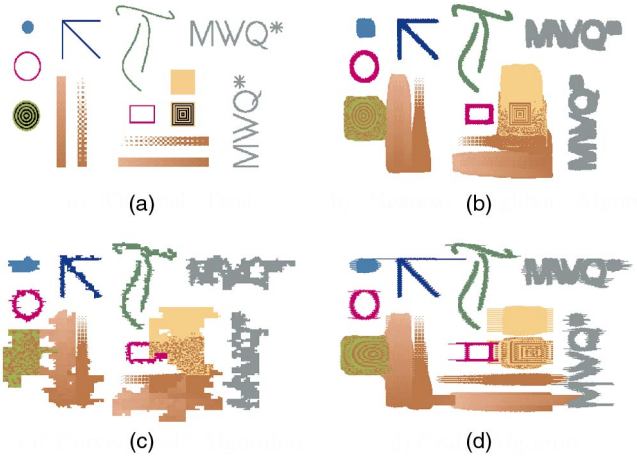


Fig. 26. Visualization of synthetic test data using the three algorithms. (a) Original data. (b) Nearest-neighbor algorithm. (c) Curve-based algorithm. (d) Gridfit algorithm. (Adapted from [36] ©IEEE.)

Appendix

Proofs of NP-Completeness

For the proofs of the NP-completeness of the linear one- and the two-dimensional ordering problem, we need to recall the notion of "polynomial reduction" and the "reduction lemma" from complexity theory.

Definition 8 (Polynomial Reduction). *A problem $P_1 \subseteq \Sigma_1^*$ can be polynomially reduced to a problem $P_2 \subseteq \Sigma_2^*$ (notation $P_2 \leq P_1$) if there exists a transformation $f: \Sigma_1^* \rightarrow \Sigma_2^*$ which can be determined in polynomial time such that $\forall x \in \Sigma_1^* : x \in P_1 \Leftrightarrow f(x) \in P_2$.*

Lemma 4 (Reduction [22]).

$$P_1 \in \text{NP} \ \wedge \ P_2\text{NP-complete} \ \wedge \ P_2 \leq P_1 \Rightarrow P_1 \ \text{NP-complete}.$$

The principle idea of the reduction is to show that the problem can be reduced to a known NP-complete problem. A precondition is that the new problem P_1 can be solved in nondeterministic polynomial time. If we assume that we have a solution of the problem P_1 and show that, in this case, we can use the solution to also solve the NP-complete problem P_2, then it implies that P_1 is at least as complex as P_2 and, therefore, P_1 also has to be NP-complete. Note that the transformation of the problem and solution in the reduction step have to be of polynomial time and space complexity.

Lemma 2 (NP-Completeness of the Linear 1D Problem). *The linear variant of the one-dimensional ordering problem according to Definition 5 is NP-complete.*

Proof. For proving the NP-completeness of the problem, we have to show that 1) the problem can be solved in nondeterministic time, and 2) we have to find a related NP-complete problem and a polynomial transformation (reduction) between the original and the NP-complete problem.

1. To show that the problem can be solved in nondeterministic time, we have to define the corresponding decision problem:

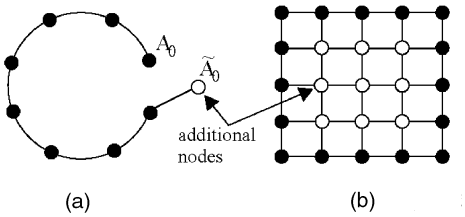


Fig. 27. Ideas of the NP-completeness proofs. (a) Linear 1D arrangement. (b) 2D arrangement.

Given an ordering $\{\pi(0), \ldots, \pi(k-1)\}$ and some real number X. Decide whether

$$\sum_{j=0}^{k-1} S\big(A_{\pi(i)}, A_{\pi((i+1) \bmod k)}\big) \leq X.$$

This problem is obviously in NP (we can nondeterministically guess a solution and then calculate the sum in polynomial time). If we are able to solve this problem, we can also solve the original problem in nondeterministic polynomial time since we can use a binary partitioning for the X value range and iteratively apply the decision problem to determine the correct X which corresponds to the correct solution.

2. A related NP-complete problem is the TSP problem. The reduction, however, is not straightforward. We have to show that the linear problem is at least as complex as the TSP problem, i.e., if we can solve the linear problem, then we also have a solution of the TSP problem. Let us assume that we have an algorithm for solving the linear problem. For solving the TSP problem (for an arbitrary set of dimensions $A = \{A_0, \ldots, A_{k-1}\}$ with an arbitrary similarity matrix S), we now define a transformation

$$f(A, S) = (\tilde{A}, \tilde{S}),$$

where $\tilde{A} = A \cup \tilde{A}_0$ and $\tilde{S}$ is a $(k+1) \times (k+1)$ matrix which is defined as

- $\tilde{S}(A_i, A_j) = S(A_i, A_j) \quad \forall i, j = 0, \ldots, (k-1)$,
- $$\tilde{S}(\tilde{A}_0, A_i) = \tilde{S}(A_i, \tilde{A}_0) = S(A_0, A_i) \quad \forall i = 0, \ldots, (k-1),$$
- $\tilde{S}(A_0, \tilde{A}_0) = \tilde{S}(\tilde{A}_0, A_0) = LARGE$, where

$$LARGE = \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} S(A_i, A_j) + 1.$$

Without loss of generality, we split A_0 such that A_0 becomes the start dimension and the additional dimension $\tilde{A}_0$ becomes the end dimension of the linear solution (cf. Fig. 27a). The distance (similarity) values of the new dimension $\tilde{A}_0$ are set to the same values as the distances for A_0 and the distance between A_0 and $\tilde{A}_0$ is set to a very high value $(LARGE)$, which is larger than all similarity values in the similarity matrix together. By this choice, we ensure that the path between A_0 and $\tilde{A}_0$ will not become part of the solution and therefore, A_0 and $\tilde{A}_0$ will be the start and end point. If we now use the linear algorithm to determine a solution, then we also have a solution of the TSP problem since, in the back transformation, we just have to ignore the $\tilde{A}_0$ dimension and connect A_0 directly to the neighbor of $\tilde{A}_0$. The transformation between the linear problem and the TSP problem, as well as the back transformation of the solution can be done in polynomial time and space. □

Lemma 3 (NP-Completeness of the 2D Ordering Problem). *The two-dimensional ordering problem, according to Definition 6, is NP-complete.*

Proof. The structure of the proof is analogous to the proof of Lemma 3. Again we have to show that 1) the problem can be solved in nondeterministic time, and 2) we have to find a related NP-complete problem and a polynomial transformation (reduction) between the original and the NP-complete problem.

1. Analogously to the proof of Lemma 3, we have to define the corresponding decision problem and, then, the rest works as shown in proof of Lemma 3. The decision is:

   Given a two-dimensional ordering $\{\pi(0,0), \ldots, \pi(k_1-1, k_2-1)\}$ and some real number X. Decide whether

$$\sum_{i=0}^{k_1-2} \sum_{j=0}^{k_2-1} S\big(A_{\pi(i,j)}, A_{\pi(i+1,j)}\big) + \sum_{i=0}^{k_1-1} \sum_{j=0}^{k_2-2} S\big(A_{\pi(i,j)}, A_{\pi(i,j+1)}\big) \leq X.$$

   The first portion of the formula corresponds to the sum of the distances in the rows and the second portion to the sum of the distances in the columns of the two-dimensional ordering.

2. Again, we use the TSP problem as the related NP-complete problem. In this case, the reduction, however, gets more complex. Again, let us assume that we have an algorithm for solving the two-dimensional ordering problem. Without loss of generality, we assume that the two-dimensional ordering consists of k_1 rows and k_2 columns and we assume $k = 2 \cdot (k_1 + k_2) - 4$.[6] For solving the TSP problem (for an arbitrary set of dimensions $A = \{A_0, \ldots, A_{k-1}\}$ with an arbitrary similarity matrix S), we now define a transformation

6. This assumption is only necessary to technically simplify the proof since otherwise we would have to introduce additional dimensions to fill up the gap and we would have do define specific distances to ensure an appropriate ordering of those dimensions.

$$f(A, S) = (\tilde{A}, \tilde{S}),$$

where $\tilde{A} = A \cup \{A_k, \ldots, A_{k_1 \cdot k_2 - 1}\}$ and $\tilde{S}$ is a $(k_1 \cdot k_2) \times (k_1 \cdot k_2)$ matrix which is defined as

- $$\tilde{S}(A_i, A_j) = S(A_i, A_j) + LARGE \quad \forall i, j = 0, \ldots, (k-1),$$
- $$\tilde{S}(A_i, A_j) = \tilde{S}(A_j, A_i) = 2 \cdot LARGE \quad \forall i = 0, \ldots, (k-1) \quad \forall j = k, \ldots, k_1 \cdot k_2 - 1,$$
- $\tilde{S}(A_i, A_j) = 0 \quad \forall i, j = k, \ldots, k_1 \cdot k_2 - 1.$

The basic idea of the proof is to introduce $(k_1 - 2) \cdot (k_2 - 2)$ new dimensions, for which the distances (similarity values) are chosen such that those dimensions will be positioned by the two-dimensional ordering algorithm as inner nodes of the ordering, while the dimensions of the original problem will be positioned as outer nodes (cf. Fig. 27b). This is achieved by giving the new dimensions very small distances to all other new dimensions while the distances of the outer dimensions are increased by a high value $(LARGE)$ so that they do not interfere with the inner dimensions. The distance between inner and outer dimension is set to a very high value $(2 \cdot LARGE)$ to prevent a jumping between the inner and outer dimensions.

If the algorithm for the two-dimensional ordering problem is now applied, we also obtain a solution of the TSP problem since, in the back transformation, we just have to ignore the additional dimensions $\{A_k, \ldots, A_{k_1 \cdot k_2 - 1}\}$. Again, the transformation between the two-dimensional ordering and the TSP problem, as well as the mapping between the solutions, can be done in polynomial time and with polynomial space since at most $O(k_1 \cdot k_2) = O(k^2)$ dimensions are added and since the summations can also be done in polynomial time. Therefore, if we have a solution for the two-dimensional ordering problem, we are able to construct a solution of the TSP problem in polynomial time and space. Thus, the two-dimensional ordering problem must also be NP-complete. □

ACKNOWLEDGMENTS

The author would like to thank all the people who contributed to the research presented in this paper, especially J. Porada who implemented the first prototype of the VisDB system, M. Ankerst who developed the Recursive Pattern and Circle Segments techniques, and R. Gansel who implemented the VisualPoints system.

REFERENCES

[1] M. Ankerst, S. Berchtold, and D.A. Keim, "Similarity Clustering of Dimensions for an Enhanced Visualization of Multidimensional Data," *Proc. Int'l Conf. Information Visualization '98*, pp. 52-60, 1998.
[2] B. Alpern and L. Carter, "Hyperbox," *Proc. Visualization '91*, pp. 133-139, 1991.
[3] V. Anupam, S. Dar, T. Leibfried, and E. Petajan, "DataSpace: 3-D Visualization of Large Databases," *Proc. Int'l Symp. Information Visualization*, pp. 82-88, 1995.
[4] M. Ankerst, D.A. Keim, and H.-P. Kriegel, "Circle Segments: A Technique for Visually Exploring Large Multidimensional Data Set," *Proc. Visualization '96*, 1996.
[5] D.F. Andrews, "Plots of High-Dimensional Data," *Biometrics*, vol. 29, pp. 125-136, 1972.
[6] M. Apperley and I.T. Spence, "A Bifocal Display Technique for Data Presentation" *Proc. Eurographics*, pp. 27-43, 1982.
[7] C. Ahlberg and B. Shneiderman, "Visual Information Seeking: Tight Coupling of Dynamic Query Filters with Starfield Displays," *Proc. Human Factors in Computing Systems CHI '94 Conf.*, pp. 313-317, 1994.
[8] D. Asimov, "The Grand Tour: A Tool For Viewing Multidimensional Data," *SIAM J. Science and Statistical Computing*, vol. 6, pp. 128-143, 1985.
[9] C. Ahlberg and E. Wistrand, "IVEE: An Information Visualization and Exploration Environment," *Proc. Int'l Symp. Information Visualization*, pp. 66-73, 1995.
[10] C. Ahlberg, C. Williamson, and B. Shneiderman, "Dynamic Queries for Information Exploration: An Implementation and Evaluation," *Proc. ACM CHI Int'l Conf. Human Factors in Computing*, pp. 619-626, 1992.
[11] A. Buja, D.F. Swayne, and D. Cook, "Interactive High-Dimensional Data Visualization," *J. Computational and Graphical Statistics*, vol. 5, no. 1, pp. 78-99, 1996.
[12] J. Beddow, "Shape Coding of Multidimensional Data on a Microcomputer Display," *Proc. Visualization '90*, pp. 238-246, 1990.
[13] B. Bederson, "Pad++: Advances in Multiscale Interfaces," *Proc. Human Factors in Computing Systems CHI '94 Conf.*, p. 315, 1994.
[14] G.D. Battista, P. Eades, R. Tamassia, and I. Tollis, *Graph Drawin: Algorithms for the Visualization of Graphs.* Prentice Hall, 1999.
[15] R.A. Becker, S.G. Eick, and A.R. Wilks, "Visualizing Network Data," *IEEE Trans. Visualization and Computer Graphics*, vol. 1, no. 1, pp. 16-28, Mar. 1995.
[16] C. Beshers and S. Feiner, "AutoVisual: Rule-Based Design of Interactive Multivariate Visualizations," *IEEE Computer Graphics and Applications*, vol. 13, no. 4, pp. 41-49, 1993.
[17] A. Buja et al., "Interactive Data Visualization Using Focusing and Linking," *Proc. Visualization '91*, pp. 156-163, 1991.
[18] W.S. Cleveland, *Visualizing Data.* Summit, N.J.: Hobart Press, 1993.
[19] M. Dorigo and L. M. Gambardella, "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem," *IEEE Trans. Evolutionary Computation*, vol. 1, no. 1, 1997.
[20] S.G. Eick, "Data Visualization Sliders," *Proc. ACM UIST*, pp. 119-120, 1994.
[21] S. Eick and G.J. Wills, "Navigating Large Networks with Hierarchies," *Proc. Visualization '93*, pp. 204-210, 1993.
[22] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freemann, 1979.
[23] G.W. Furnas and A. Buja, "Prosections Views: Dimensional Inference through Sections and Projections," *J. Computational and Graphical Statistics*, vol. 3, no. 4, pp. 323-353, 1994.
[24] K. Fishkin and M.C. Stone, "Enhanced Dynamic Queries via Movable Filters," *Proc. Human Factors in Computing Systems CHI '95 Conf.*, pp. 415-420, 1995.
[25] G. Furnas, "Generalized Fisheye Views," *Proc. Human Factors in Computing Systems CHI '86 Conf.*, pp. 18-23, 1986.
[26] D. Hilbert, "Über stetige Abbildung einer Linie auf ein Flächenstück," *Math. Annalen*, vol. 38, pp. 459-460, 1891.
[27] G.T. Herman and H. Levkowitz, "Color Scales for Image Data," *Computer Graphics and Applications*, pp. 72-80, 1992.
[28] A. Inselberg, "The Plane with Parallel Coordinates, Special Issue on Computational Geometry," *The Visual Computer*, vol. 1, pp. 69-97, 1985.
[29] A. Inselberg and B. Dimsdale, "Parallel Coordinates: A Tool for Visualizing Multi-Dimensional Geometry," *Proc. Visualization '90*, pp. 361-370, 1990.
[30] B. Johnson, "Visualizing Hierarchical and Categorical Data," PhD thesis, Dept. of Computer Science, Univ. of Maryland, 1993.

[31] D.A. Keim, "Visual Support for Query Specification and Data Mining," Aachen, Germany: Shaker-Publishing Company, 1995.
[32] D.A. Keim, "Enhancing the Visual Clustering of Query-Dependent Database Visualization Techniques Using Screen-Filling Curves," *Proc. Workshop Database Issues for Data Visualization,* 1995.
[33] D.A. Keim, "Pixel-Oriented Visualization Techniques for Exploring Very Large Databases," *J. Computational and Graphical Statistics,* vol. 5, no. 1, pp. 58-77, 1996.
[34] D.A. Keim, "Visual Database Exploration," tutorial, *Proc. Int'l Conf. Knowledge Discovery in Databases (KDD '97),* 1997.
[35] D.A. Keim, "Visual Data Mining," tutorial, *Proc. Conf. Very Large Databases,* 1997.
[36] D.A. Keim and A. Herrmann, "The Gridfit Algorithm: An Efficient and Effective Algorithm to Visualizing Large Amounts of Spatial Data," *Proc. IEEE Visualization Conf.,* pp. 181-188, 1998.
[37] D.A. Keim and H.-P. Kriegel, "VisDB: Database Exploration Using Multidimensional Visualization," *IEEE Computer Graphics & Applications,* pp. 40-49, Sept. 1994.
[38] D.A. Keim and H.-P. Kriegel, "Issues in Visualizing Large Databases," *Visual Database Systems,* pp. 203-214, Chapman & Hall Ltd., 1995.
[39] D.A. Keim, H.-P. Kriegel, and M. Ankerst, "Recursive Pattern: A Technique for Visualizing Very Large Amounts of Data," *Proc. Visualization '95,* pp. 279-286, 1995.
[40] D.A. Keim, H.-P. Kriegel, and T. Seidl, "Supporting Data Mining of Large Databases by Visual Feedback Queries," *Proc. 10th Int'l Conf. Data Eng.,* pp. 302-313, 1994.
[41] D.A. Keim and H.-P. Kriegel, "Visualization Techniques for Mining Large Databases: A Comparison," *IEEE Trans. Knowledge and Data Eng.,* vol. 8, no. 6, pp. 923-938, Dec. 1996.
[42] Y. Leung and M. Apperley, "A Review and Taxonomy of Distortion-Oriented Presentation Techniques," *Proc. Human Factors in Computing Systems CHI '94 Conf.,* pp. 126-160, 1994.
[43] H. Levkowitz, "Color Icons: Merging Color and Texture Perception for Integrated Visualization of Multiple Parameters," *Proc. Visualization '91,* Oct. 1991.
[44] J. Lamping and R. Rao, "Laying Out and Visualizing Large Trees Using a Hyperbolic Space," *Proc. UIST,* pp. 13-14, 1994.
[45] J. Lamping, R. Rao, and P. Pirolli, "A Focus + Context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies," *Proc. Human Factors in Computing Systems CHI '95 Conf.,* pp. 401-408, 1995.
[46] J. LeBlanc, M.O. Ward, and N. Wittels, "Exploring N-Dimensional Databases," *Proc. Visualization '90,* pp. 230-239, 1990.
[47] T. Munzner and P. Burchard, "Visualizing the Structure of the World Wide Web in 3D Hyperbolic Space," *Proc. VRML '95 Symp.,* pp. 33-38, 1995.
[48] G.M. Morton, "A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing," IBM Ltd., Ottawa, Canada, 1966.
[49] J.D. Mackinlay, G.G. Robertson, and S.K. Card, "The Perspective Wall: Detail and Context Smoothly Integrated," *Proc. Human Factors in Computing Systems CHI '91 Conf.,* pp. 173-179, 1991.
[50] G. Peano, "Sur une courbe qui remplit toute une aire plaine," *Math. Annalen,* vol. 36, pp. 157-160, 1890.
[51] R.M. Pickett and G.G. Grinstein, "Iconographic Displays for Visualizing Multidimensional Data," *Proc. IEEE Conf. Systems, Man, and Cybernetics,* pp. 514-519, 1988.
[52] R. Rao and S.K. Card, "The Table Lens: Merging Graphical and Symbolic Representation in an Interactive Focus+Context Visualization for Tabular Information," *Proc. Human Factors in Computing Systems CHI '94 Conf.,* pp. 318-322, 1994.
[53] G. Reinelt, "The Traveling Salesman—Computational Solutions for TSP Applications," *Lecture Notes in Computer Science,* vol. 840, Springer-Verlag, 1994.
[54] G.G. Robertson, J.D. Mackinlay, and S.K. Card, "Cone Trees: Animated 3D Visualizations of Hierarchical Information," *Proc. Human Factors in Computing Systems CHI '91 Conf.,* pp. 189-194, 1991.
[55] M. Sarkar and M. Brown, "Graphical Fisheye Views," *Comm. ACM,* vol. 37, no. 12, pp. 73-84, 1994.
[56] B. Shneiderman, "Tree Visualization with Treemaps: A 2D Space-Filling Approach," *ACM Trans. Graphics,* vol. 11, no. 1, pp. 92-99, 1992.
[57] R. Spence et al., "Visualization for Functional Design," *Proc. Int'l Symp. Information Visualization (InfoVis '95),* pp. 4-10, 1995.
[58] E.R. Tufte, *The Visual Display of Quantitative Information.* Cheshire, Conn.: Graphics Press, 1983.
[59] E.R. Tufte, *Envisioning Information.* Cheshire, Conn.: Graphics Press, 1990.
[60] M.O. Ward, "XmdvTool: Integrating Multiple Methods for Visualizing Multivariate Data," *Proc. Visualization '94,* pp. 326-336, 1994.
[61] J.J. van Wijk and R.D. van Liere, "Hyperslice," *Proc. Visualization '93,* pp. 119-125, 1993.
[62] W. Wright, "Information Animation Applications in the Capital Markets," *Proc. Int'l Symp. Information Visualization,* pp. 19-25, 1995.
[63] A. Wilhelm, A.R. Unwin, and M. Theus, "Software for Interactive Statistical Graphics—A Review," *Proc. Int'l Softstat '95 Conf.,* 1995.

Daniel A. Keim received his diploma (equivalent to an MS degree) in computer science from the University of Dortmund in 1990 and his PhD in computer science from the University of Munich in 1994. Currently, he is an associate professor at the Institute for Computer Science of the Martin-Luther-University Halle, Germany. He is working in the area of information visualization and data mining, as well as similarity search and indexing in multimedia databases. In the field of information visualization, he developed several novel techniques which use visualization technology for the purpose of exploring large databases. He was the chief engineer in designing the VisDB system—a visual database exploration system focusing on pixel-oriented visualization techniques. He has published extensively on information visualization and data mining, he has given tutorials on related issues at several large conferences including SIGMOD, VLDB, KDD, and AVI, and he was program cochair of the IEEE Information Visualization Symposia 1999 and 2000.