On the Efficient Construction of Rectangular Grids from Given Data Points

Jan Poland, Kosmas Knödler and Andreas Zell

Universität Tübingen, WSI-RA, Sand 1, D - 72076 Tübingen, Germany
poland@informatik.uni-tuebingen.de,
http://www-ra.informatik.uni-tuebingen.de

Abstract. Many combinatorial optimization problems provide their data in an input space with a given dimension. Genetic algorithms for those problems can benefit by using this natural dimension for the encoding of the individuals rather than a traditional one-dimensional bit string. This is true in particular if each data point of the problem corresponds to a bit or a group of bits of the chromosome. We develop different methods for constructing a rectangular grid of near-optimal dimension for given data points, providing a natural encoding of the individuals. Our algorithms are tested with some large TSP instances.

1 Introduction

Many combinatorial optimization problems provide their data in an input space with a given dimension d. For example, the classical traveling salesman problem (TSP) consists of n points in the two-dimensional plane. An efficient representation of a tour for solving the TSP with a genetic algorithm is the adjacency coding (see [1], [2] or [3]): A tour is defined by its adjacency matrix or, equivalently, by a list containing the successor for each city. This *locus*-based representation has shown to be much more appropriate for use with genetic algorithms than a time-based representation containing a permutation of the cities. It is characterized by the fact that each city corresponds to a certain part of the chromosome.

When using a locus-based representation together with a standard N-point crossover operator that does not permute the chromosomes, the arrangement of the points in the representation is important. For example in the case of the TSP, arranging the cities in the order given by a rough approximation of the TSP results in a better performance of the genetic algorithm than using an arbitrary arrangement, as shown in [1]. This is due to the fact that the crossover operator can exploit neighbourhood relations of the preordered cities.

However, since the original problem is two-dimensional, a two-dimensional representation as suggested by Bui and Moon ([4]) would be more natural. They present a d-dimensional N-point crossover operator that applies to d-dimensional grids instead of one-dimensional strings. If this idea is employed for the TSP with a locus-based representation, the cities have to be allocated to the grid points. This is a preliminary task that has to be completed before the start of the genetic

algorithm and should not consume too much time. On the other hand, a good arrangement may increase the performance of the genetic algorithm.

Clearly, the grid encoding is not the only possibility for a d-dimensional representation, one could, for example, try a graph-based encoding. But the grid encoding has the advantage of a very simple and efficient crossover procedure and is therefore very convenient for a genetic algorithm that uses crossover as an important operator.

Hence, the problem considered here can be defined as follows. Given are n data points $(x_1, \ldots, x_n)$ in the d-dimensional space. Find an appropriate grid size $(g_1, \ldots, g_d)$ and an unambiguous (injective) allocation $\imath : \{1 \ldots n\} \mapsto \{1 \ldots g_1\} \otimes \{1 \ldots g_2\} \otimes \ldots \otimes \{1 \ldots g_d\}$, such that the allocation is good in the following sense: Points that are close to each other should be mapped to neighbouring grid points, while distant points should be mapped to distant nodes in the grid. Thus, the transformation to the grid throws away as little neighbourhood information as possible, and one can hope that a genetic algorithm is able to use this information for faster convergence. Note that the injectivity of $\imath$ implies $g_1 \cdot g_2 \cdot \ldots \cdot g_d \geq n$. On the other hand, this product should be as small as possible, since each unused grid point means unused space in the chromosomes and therefore poorer performance of the genetic algorithm.

In [5], B. Moon and C. Kim study a similar problem, the two-dimensional embedding of graphs. Their assumptions are weaker, since they exploit only the adjacency information (i.e. distance in the TSP case). In the contrary, we will exploit the locus information.

2 Calculating the grid size

We start with a brief discussion of different approaches. Basically, there are two possible ways: One can either fix the grid size before allocating the points, or one can determine the grid size while arranging the points. Suppose we want to do the latter. An algorithm could perform a depth first search, in analogy to the graph embedding algorithm in [5]. In each step, the grid is extended by an additional point. But the decision where to extend the grid and in which direction is difficult in general, since it depends crucially on points that are not yet processed. Furthermore, if $n < g_1 \cdot g_2 \cdot \ldots \cdot g_d$, then unused grid points have to be inserted at some places. Suboptimal decisions that result in an ineffective allocation are almost certain, which carries the need for repairing algorithms. Hence, constructing the grid without previously fixing the grid size can become very expensive, in particular for large data sets (for example $n \approx 1000$).

For a concrete example, consider the points displayed in Fig. 1 (a). A step by step arrangement may lead to the situation shown in Fig. 1 (b). It would be relatively easy to repair this arrangement by inserting two empty positions, thus arriving at Fig. 1 (c). In contrast, all modifications that could result in the optimal allocation Fig. 1 (d) are quite expensive. Another situation: If point 6 is placed to the right of point 5 instead of below, the result is Fig. 1 (e) or, if point

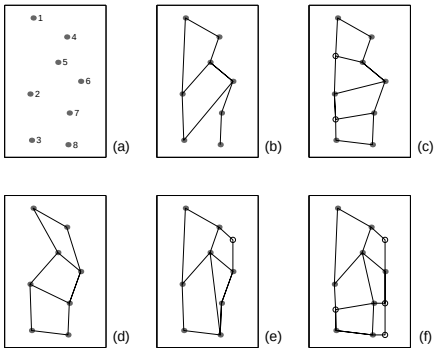

Fig. 1. (a) Eight data points, (b) - (f) their allocation to different grids, (d) is optimal

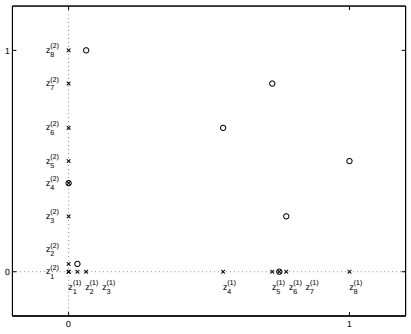

Fig. 2. The points have been normalized and projected to the coordinate axis

7 is choosen to be below point 5, Fig. 1 (f). Note that for the last arrangement four additional empty positions have to be inserted, which is not optimal at all.

In general, we are interested in a reasonable minimum of grid points. This ensures a minimum of unused space for the individuals, while Fig. 1 suggests that the neighbourhood relation is fairly well preserved by different arrangements (each of the arrangements (c) through (f) would pass this criterion). Therefore, we pursue the other way and fix the grid size before allocating the points.

We assume that the data points cover a region that has approximately the shape of a d-dimensional hypercube with balanced aspect ratio, i.e. that the data set is not poorly scaled. Otherwise, the scaling information should be exploited for calculating the grid size.

Consider again Fig. 1 (a). We are looking for a method that calculates the grid size for this set of points. A 2×4 grid would be optimal, as we can see in the figure, a 3×3 grid would be tolerable, too.

To this aim, we first normalize the points to $[0,1]^d$ and name the resulting points $(y_1, y_2, \ldots, y_n)$. Then we project the points to each of the d coordinate axis, obtaining d vectors of length n $\{(y_1, \ldots, y_n)^{(k)}, k = 1 \ldots d\}$, see Fig. 2. Sorting $(y_1, \ldots, y_n)^{(k)}$ for each $k \in \{1 \ldots d\}$ *separately* in ascending order yields $(z_1, \ldots, z_n)^{(k)}$, $k \in \{1 \ldots d\}$, having $z_1^{(k)} = 0$ and $z_n^{(k)} = 1$ for each k. We define

$$r_k = \frac{1}{\sum_{j=1}^{n-1} (z_{j+1}^{(k)} - z_j^{(k)})^2} + 1 \quad \text{for each} \quad k \in \{1 \ldots d\}.$$

This number r_k estimates the requested number of points for each dimension k.

To motivate this statement, consider more closely the projections onto the x-axis in Fig. 2. We have eight different points $z_1^{(1)}, \ldots, z_8^{(1)}$, but they are not equidistantly distributed. Instead, $z_1^{(1)}$, $z_2^{(1)}$ and $z_3^{(1)}$ form a cluster, and the same is true for $z_5^{(1)}$, $z_6^{(1)}$ and $z_7^{(1)}$. Thus one should expect $r_1 = 4$ rather than $r_1 = 8$.

Consider n sorted points $z_1, \ldots, z_n \in [0,1]$ with $z_1 = 0$ and $z_n = 1$. Suppose that each z_j is one of the $m \leq n$ points $\{\frac{i}{m-1} \ : \ 0 \leq i \leq m-1\}$, which are *equidistantly* located. If $m < n$, then at least two points coincide. Let X be a random variable with uniform distribution on $[0,1]$ and define $\delta(\xi)$ as the distance of ξ to the closest point z_j:

$$\delta(\xi) = \min_{1 \leq j \leq n} |z_j - \xi| \quad \text{for} \quad \xi \in [0,1].$$

Then the expectation of $\delta(X)$ can be computed:

$$\begin{aligned} \mathrm{E}(\delta(X)) = \int_0^1 \delta(\xi) d\xi \ &= \ \sum_{j=1}^{n-1} \int_{z_j}^{z_{j+1}} \min_{i \in \{j, j+1\}} |z_i - \xi| \ d\xi \\ = \sum_{j=1}^{n-1} \frac{1}{4} (z_{j+1} - z_j)^2 \ &= \ \sum_{j=1}^{m-1} \frac{1}{4(m-1)^2} \ = \ \frac{1}{4(m-1)}. \end{aligned}$$

Thus, we can reconstruct the number of distinct points m by means of $\mathrm{E}(\delta(X))$:

$$m = \frac{1}{4 \cdot \mathrm{E}(\delta(X))} + 1 = \frac{1}{\sum_{j=1}^{n-1} (z_{j+1} - z_j)^2} + 1.$$

This is the formula stated above. Note that this expression is continuous in each z_j, so if you take a point z_j that coincides with its successor and move it a little to the left, the left-hand side changes only a little, too. Hence a cluster that is for example formed by $z_1^{(k)}$, $z_2^{(k)}$ and $z_3^{(k)}$ in Fig. 2 yields almost the same value as a cluster of three coinciding points. This property enables the formula to be a good estimate for the number of different points. In our example (Fig. 2) the calculations yield $r_1 = 4.1045$ and $r_2 = 7.0624$.

Now one notes that the product of the requested grid sizes is in general much larger than n: We have $r_1 \cdot r_2 = 28.9880$ and $n = 8$ in our example. A simple reflexion explains this fact: If n points are equidistantly placed on the line through $(0,0)$ and $(1,1)$, the result will be $r_1 = n$ and $r_2 = n$, hence $r_1 \cdot r_2 = n^2$.

At this stage, we recall our assumption that the data set is not poorly scaled. Thus, each dimension can be treated in the same way, and we set

$$s_k = r_k \cdot \sqrt[d]{\frac{n}{r_1 \cdot r_2 \cdot \ldots \cdot r_d}} \quad \text{for each} \quad k \in \{1, \ldots, d\}$$

and obtain the desired grid size for each dimension having $s_1 \cdot s_2 \cdot \ldots \cdot s_d = n$, while the relations are preserved.

Unfortunately, $s_1, \ldots, s_d$ are no integer numbers yet. We could obtain integers by setting $g_k = \lceil s_k \rceil$. But since we are interested in a grid as small as possible, this is not a good choice. Thus, we simply try each reasonable combination $g_1, \ldots, g_d$. Since the dimension d is normally quite small, this is no drawback for the performance. For large d (about $d \geq 15$), a different method has to be used instead.

In order to select the best grid size, we define

$$q_1 \;=\; \max_{1\le k\le d} |\log(\frac{g_k}{s_k})| \quad \text{and} \quad q_2 \;=\; \log(\frac{g_1\cdot\ldots\cdot g_d}{n}).$$

The ratio q_1 can be considered as the bias of the desired size relations, while q_2 is the factor by which the grid is too large. We define the optimal grid size as the vector $(g_1,\ldots,g_d)$ for which $\max\{q_1,q_2\}$ attains its minimum while $g_1\cdot\ldots\cdot g_d \ge n$. The number of grid points $g_1\cdot\ldots\cdot g_d$ will be denoted by n_{grid} in the sequel. In our example (Fig. 2), the algorithm found the 2×4 grid to be optimal.

We point out that this method for calculating the grid size is most appropriate when the projections to the coordinate axis form significant clusters, otherwise the algorithm will produce a nearly quadratic grid. This is a desirable behaviour for data sets that are reasonably scaled and oriented. If the orientation is not appropriate, i.e. not roughly parallel to the coordinate axis, a transformation using the eigenvectors of the covariance matrix (see Section 4) can fix that problem. The algorithm is not appropriate for exploiting the scaling information of a poorly scaled data set, but this is normally a simple task which can be done by multiplying the requested grid sizes r_k with the corresponding scaling factors.

3 Allocation of the points

Once the grid size has been computed, the allocation of the points is performed by a simple heuristic, similar to the heuristic that is often used for the TSP. We define the n_{grid} grid points $\gamma_1,\ldots,\gamma_{n_{grid}}$ as an arbitrary enumeration of the set

$$\left\{\left(\frac{1}{2}+\frac{i_1-1}{g_1},\ldots,\frac{1}{2}+\frac{i_d-1}{g_d}\right) \quad : \quad 1\le i_1\le g_1,\ldots,1\le i_d\le g_d\right\}.$$

Then, we start with a random allocation of the points, see Fig. 3 (a). Each step of the heuristic selects randomly two edges, i.e. two existing connections between a point and a grid point, and exchanges the allocation if this reduces the sum of the distances. Instead of two edges, the heuristic can select one edge and one unused grid point (they exist if $n_{grid} > n$) and exchange the allocation if the distance is reduced. The heuristic aborts after $5000\cdot n_{grid}$ steps maximum. Fig. 3 illustrates the function of the heuristic in our example. Note that the result (Fig. 3 (b)) is the optimal arrangement (Fig. 1 (d)).

Instead of the distances, the heuristic can also minimize the *square* distances. This results in a slightly different behaviour and is a little faster, since no square root has to be computed. If the standard grid defined above does not yield satisfactory results, a different grid as described in the next section can improve the performance.

4 Data preprocessing and Self Organizing Maps

If the shape that is covered by the data points (x_j) differs from a (scaled) hypercube, both the calculation of the grid size and the heuristic that uses a rectangular grid (γ_i) may yield poor results. In many cases, a simple rotation of the data

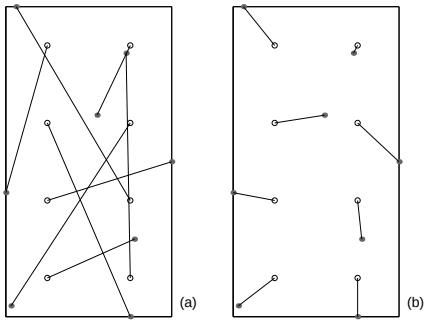

Fig. 3. Allocation of the points to the grid: (a) random initialization, (b) final result of the heuristic

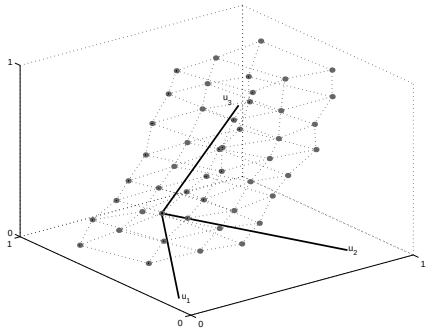

Fig. 4. A data set in $\mathbb{R}^3$ and the coordinate system formed by the eigenvectors of the covariance matrix

is sufficient to fix this problem. To this aim, we apply a very efficient statistical method that makes use of the covariance matrix of the data. Let $\bar{x} = \frac{1}{n}\sum_{j=1}^{n} x_j$ and $X = \big((x_1 - \bar{x}) \ \ldots \ (x_n - \bar{x})\big)$. Since XX' is symmetric, its eigenvectors $u_1, \ldots, u_d$ form an orthogonal basis. We set $U = (u_1 \ \ldots \ u_d)$ and $y_j = U'x_j$ for each $1 \leq j \leq n$. This transformation yields a data set that is oriented mainly parallel to the coordinate axis (Fig. 4).

If the data set has a more complex shape e.g with a curvature, the use of a Self Organizing Map (SOM) may be appropriate. This is a class of Neural Networks introduced by Kohonen, see [6] or [7] for details. Self Organizing Maps provide a powerful tool for classifying large sets of points in the d-dimensional space with regard to neighbourhood relations.

We take a rectangular SOM with a highly unbalanced aspect ratio and only few codebook vectors. Thus, the SOM can reproduce the curvature of the data, when it is trained with the points (x_j), see Fig. 5 (a). In the resulting map, we can define a coordinate system for each codebook vector (in the figure, this is shown for the third point from the left). With this information, we can "straighten" the SOM and obtain a transformation to a nearly rectangular grid, while the neighbourhood relations are preserved.

A grid defined by a SOM can be also very appropriate as a base for the heuristic, instead of the standard grid. For this aim, we build a SOM of size $g_1 \times \ldots \times g_d$ and train it with the data points (x_j), see Fig. 5 (b). The resulting codebook vectors define the grid (γ_i), which is rectangular and contains the neighbourhood information, but is adapted to the data shape. For such a grid, the heuristic that minimizes the square distances has shown to be best suitable.

5 Practical tests

The single steps have been described by now. The complete algorithm for constructing a rectangular grid from given data points can be summarized as follows.

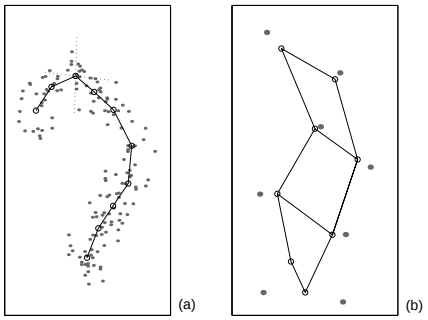

Fig. 5. Different applications of a SOM: (a) a SOM for data preprocessing, (b) the SOM defines the grid

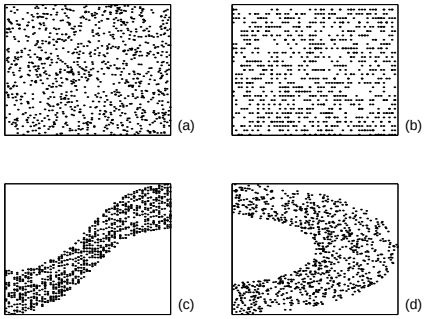

Fig. 6. The test data sets, each defines 1000 cities for a TSP

Algorithm.

1. *Data preprocessing.* Choose either no preprocessing, rotation according to the eigenvectors of the covariance matrix, or transformation using a SOM.
2. *Grid size calculation.*
3. *Grid definition.* Use either a standard grid or define the grid by a SOM.
4. *Choice of the heuristic.* Choose the sum of the distances or the sum of the square distances to be minimized.

The algorithm is implemented in the MATLAB environment. This implies that the performance is lower than, for example, C-code would be. This is tolerable since there are no time-consuming loops, except for the heuristic, which is therefore directly coded in C. For the SOM features, we use the SOM toolbox, see [8]. The genetic algorithm is the MATLAB implementation presented in [9].

For testing our algorithm, we used four rather large traveling salesman instances with 1000 cities each. The cities are shown in Fig 6. The first data set (a) simply consists of randomly placed points. The second (b) is defined by randomly placed points aligned to a grid. The third data set (c) is the second transformed to a sigmoid curve, and the last (d) is the first transformed to a half circle.

To estimate the performance of the grid allocations, there is a direct and an indirect way. The former consists in defining a measure for the preservation of neighbourhood relations under the grid transformation. To this aim, consider the distance matrix D^x defined by $D^x_{ij} = \|x_i - x_j\|$ $(1 \leq i, j \leq n)$ and the distance matrix D^γ that contains the distances after transformation to a standard rectangular grid. D^x is normalized such that $\sum_i D_{ij} = 1$ for each j. Moreover, we define a weight matrix W to be inverse proportional to the square of the distances and normalized: $W_{ij} = c \cdot (D^x_{ij})^{-2}$ for $i \neq j$ and $W_{ij} = 0$ for $i = j$ and $\sum_i W_{ij} = 1$ for each j. Then, a measure M_j for the preservation of neighbourhood relations for one point x_j is defined as the weighted mean of all distance differences that arise from the grid transformation. Thereby the grid distances

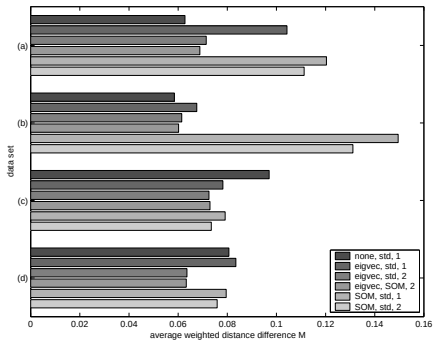

Fig. 7. Direct performance comparison: A lower value means less differences and thus a better performance

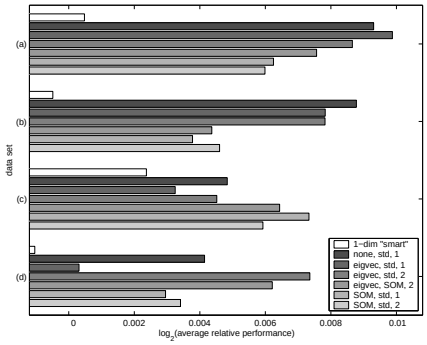

Fig. 8. Indirect performance comparison: A higher value means a shorter tour and therefore a better performance

must be scaled in order to achieve the best fit to the distances. As an explicit formula, this gives

$$M_j = \min_{s \in \mathbb{R}^+} \sum_{i=1}^{n} W_{ij} \cdot |D^x_{ij} - s \cdot D^\gamma_{ij}| \quad \forall\ 1 \leq j \leq n.$$

The measure M for the overall preservation of the neighbourhood relations is defined as the mean of all M_j.

Fig. 7 shows this evaluation of the grid transformation for the four test data sets and several allocation strategies with different preprocessing, grid definition and heuristic. Each combination corresponds to a color in the figure which is explained in the legend by a comma separated list: The first entry is the preprocessing method (none, eigenvectors or SOM), the second the grid definition (standard or SOM) and the third is the heuristic (1 for absolute distances and 2 for square distances). We observe that the evaluation for the grid allocation varies not very much, except for a larger value for the SOM preprocessing for the first two data sets. This is not unexpected since these data sets cover already a quadratic area, thus a rectangular SOM is likely to generate a worse arrangement.

The indirect evaluation of a grid arrangement is given in terms of the shortest city tour computed by a genetic algorithm using the grid arrangement for encoding. As a mutation operator, we employ the well known TSP-heuristic (see e.g. [10]) that randomly performs edge exchange and single point insertion. We use the same combinations for preprocessing, grid definition and heuristic as above. In addition, for reference purpose, we try a "smart" one-dimensional arrangement according to a run of the TSP-heuristic as suggested in [1] as well as an arbitrary one-dimensional arrangement. For each setting, 20 GA runs have been performed, from which we take the averages. The following GA settings

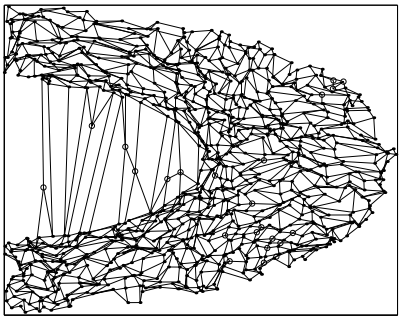

Fig. 9. Grid defined by a SOM with eigenvector preprocessing and quadratic heuristic for data set (d)

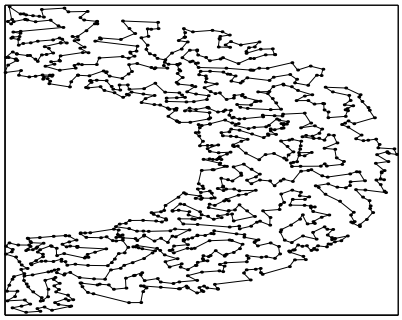

Fig. 10. The optimal path found with the coding shown in Fig. 9

were used: population size $\mu = 40$, number of generations $t_{max} = 100$, nonlinear ranking selection ($q = 7$), crossover and mutation probablity $p_{cross} = 0.5$ and $p_{mut} = 0.1$. Fig. 8 shows the resulting indirect performances relative to the average performance of the arbitrary one-dimensional arrangement. We use logarithmic (base 2) scale, where a positive value means a shorter average tour length.

The performance differences are minimal (less than 1%), but reproducible. The two-dimensional arrangement is always better than both the arbitrary and the smart one-dimensional arrangement. These relations remain valid also in earlier generations of the GA run, e.g. after $t = 30$, 50, or 80 generations. Thus, our coding implies a faster GA convergence. It is interesting to observe that the smart one-dimensional arrangement is sometimes worse than the reference. Further, there is no outstanding correlation between the direct and the indirect performance.

Fig. 9 and Fig. 10 show examples of a grid gained with a SOM and the resulting optimal path for the last test data set.

The time complexity of each of the steps is clearly linear in the number of points n. In the practical experiments with $n = 1000$ cities, the time for the first step ranges from 0 (no preprocessing) over 0.01 sec (eigenvector transformation) up to 0.8 sec (SOM). The second step (calculation of the grid size) needs about 0.01 sec. The standard grid definition takes no measurable time, while the SOM grid calculation costs 99 sec. This is due to the fact that a large amount of codebook vectors have to be trained. Nevertheless, even this is not much compared to the running time of the genetic algorithm which is in this case about 48 min. The last step, the heuristic, takes 2.3 sec for minimizing the distance sum and 1.1 sec for minimizing the square distance sum. In any case, the overall algorithm takes little time in relation to the following genetic algorithm.

6 Conclusions

The presented methods allow the efficient arrangement of a given set of data points in a rectangular grid under preservation of the neighbourhood relations. This can increase the performance of genetic algorithms using a locus-based representation, when each data point corresponds to a part of the chromosome.

The algorithms developed in this paper can be applied also in completely different situations. For example, to define the grid size of a SOM, the ideas from Section 2 can be used.

The natural encoding results in a small performance improvement for the TSP. Tested with other problems, the improvements gained by a d-dimensional encoding were similarly measurable, but small. On the other hand, there is no obvious relation between the direct performance measure (neighbourhood preservation) and the genetic performance. These facts raise some questions: Is there a direct performance measure that is correlated to the indirect performance? Are there (d-dimensional) encoding schemes that yield more improvement? What are characteristics of an optimal encoding scheme for a locus-based representation?

Acknowledgments. We thank Alexander Mitterer, Thomas Fleischhauer and Frank Zuber-Goos for helpful discussions. This research has been supported by the BMBF (grant no. 01 IB 805 A/1).

References

1. T. N. Bui and B. R. Moon. A new genetic approach for the traveling salesman problem. In *International Conference on Evolutionary Computation*, pages 7–12, 1994.
2. J. Grefenstette, R. Gopal, B. Rosmaita, and D. Van Gucht. Genetic algorithms for the travelling salesman problem. In *Proceedings of the first International Conference on Genetic Algorithms and Application*, pages 160–168, 1985.
3. A. Homaifar, S. Guan, and G. E. Liepins. A new approach on the traveling salesman problem by genetic algorithms. In *5th International Conference on Genetic Algorithms*, pages 460–466, 1993.
4. T. N. Bui and B. R. Moon. On multidimensional encoding/crossover. In *6th International Conference on Genetic Algorithms*, pages 49–56, 1995.
5. B. R. Moon and C. K. Kim. A two-dimensional embedding of graphs for genetic algorithms. In *7th International Conference on Genetic Algorithms*, pages 204–211, 1997.
6. T. Kohonen. *Self-Organization and Associative Memory.* Springer-Verlag, 3rd edition, 1989.
7. A. Zell. *Simulation neuronaler Netze.* Addison-Wesley, Bonn, 1994.
8. J Vesanto, J. Himberg, E. Alhoniemi, and J. Parhankangas. SOM Toolbox for Matlab 5. Technical Report A57, Helsinki University of Technology, http://www.cis.hut.fi/projects/somtoolbox/, April 2000.
9. J. Poland and K. Knödler et al. A genetic algorithm with variable alphabet coding for a new NP-complete problem from application. Preprint, 2000.
10. K. Knödler, J. Poland, A. Mitterer, and A. Zell. Optimizing data measurements at test beds using multi-step genetic algorithms. Preprint, 2000.