Efficient optimal overlap removal

DOI: 10.1111/cgf.13722
Eurographics Conference on Visualization (EuroVis) 2019
M. Gleicher, H. Leitte, and I. Viola
(Guest Editors)

Volume 38 (2019), Number 3

Efficient Optimal Overlap Removal: Algorithms and Experiments

W. Meulemans[†]

TU Eindhoven, the Netherlands

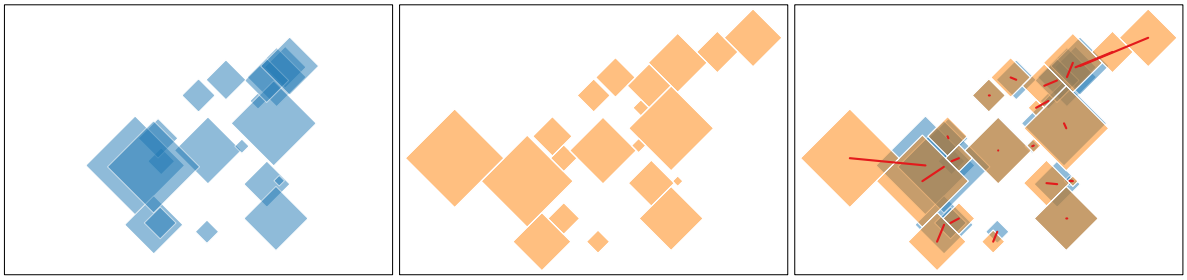

Figure 1: *Left: example input of overlapping diamonds. Middle: output of the suggested linear program, having minimal displacement and maintaining the orthogonal order. Right: overlay visualizing the displacement of each diamond.*

Abstract

Motivated by visualizing spatial data using proportional symbols, we study the following problem: given a set of overlapping squares of varying sizes, minimally displace the squares as to remove the overlap while maintaining the orthogonal order on their centers. Though this problem is NP-hard, we show that rotating the squares by 45 degrees into diamonds allows for a linear or convex quadratic program. It is thus efficiently solvable even for relatively large instances.

This positive result and the flexibility offered by constraint programming allow us to study various trade-offs for overlap removal. Specifically, we model and evaluate through computational experiments the relations between displacement, scale and order constraints for static data, and between displacement and temporal coherence for time-varying data. Finally, we also explore the generalization of our methodology to other shapes.

CCS Concepts

•***Human-centered computing*** → *Visualization;*

1. Introduction

Proportional symbol maps are a common way of visualizing scalar values of events or measurements at their geographic location. That is, each data item is visualized as a symbol (typically a simple shape such as a circle or square) placed on its associated location on a map, such that the symbol size represents the scalar value. Locations are often close to each other and thus placing the symbol precisely on its location results in greatly reduced symbols sizes, overlapping symbols or the omission of symbols. We can consider the algorithmic optimization problem for each of those solutions. Symbol sizes can be optimized trivially. Symbol overlap can be countered by computing the drawing order with optimal visibility of the symbols [CHvKS10]. Omitting a minimal number of symbols is NP-hard, even for circular symbols of equal sizes [CCJ90], though approximation algorithms exist [EJS05]. If the symbols are associated with regions rather than points, several placement strategies can be considered [vKSW04].

However, none of these solutions is particularly effective for obtaining a complete, legible visualization with many symbols. An alternative is to displace the symbols, such that the new locations are suitable for appropriately sized, non-overlapping symbols. The objective is then to minimize displacement, to ensure that the used locations are still informative of the actual geospatial situation. Beyond minimizing displacement, additional constraints can be useful to ensure that important geospatial characteristics carry over to

[†] Supported by the Netherlands eScience Center (NLeSC, 027.015.G02)

the repositioned symbols. A common constraint is the *orthogonal order*: if symbol A is left of symbol B in the input, then symbol A should be left of symbol B after repositioning, and likewise for the "above" relation. We use the centers of the symbols to decide this order. This constraint helps retain spatial relations, that may be crucial for certain application domains [vGPNB17].

The removal of overlap between nodes finds also numerous applications in other visualization problems, such as visualizing disjoint glyphs on a map [vGPNB17], computing network layouts [DMS05, MSTH03], positioning nonspatial data [SSS*12, GNCNT13] and computing Demer's and Dorling cartograms [Dor96]. However, this problem of overlap removal is NP-hard in most settings, leading to a wealth of heuristic approaches. Specifically also the variant described above with squares and the orthogonal order is NP-hard [vG18, Section 7.2]. For minimizing the maximum distance, Fiala et al. [FKP05] show that the problem is NP-hard for circles, disks and diamonds without orthogonal order constraints – even if they all have the same size.

Contributions. We show that the problem admits a linear program (LP), if we change the symbol shape from a square to a diamond, i.e., a square rotated by 45 degrees (Section 3). We leverage polyhedral distance functions (see Section 2) to approximate the Euclidean distance to obtain linear equations. Linear programs are efficiently solvable, even for relatively large instances; Fig. 1 illustrates the result on a small instance. Hence, this variant may be particularly useful for interactive applications. Of course, we may equally well keep the square symbols, but rotate the orthogonal order: in other words, we maintain the sorted order along the two diagonal directions. We also discuss the modification to minimizing squared Euclidean distances via convex quadratic programming (CQP), and two techniques to reduce the number of constraints in the constraint programs. We perform a simple experiment to investigate the efficacy of these techniques and the efficiency of solving such problems. We achieve (near-)interactive average running times for 1 000 symbols for both constraint programs.

We subsequently go through extensions and modifications of the constraint programs. We show how to deal with scale concerns (Section 4) and investigate the trade-off between displacement and maintaining the orthogonal order (Section 5). In Section 6 we extend our method to handle time-varying data and consider the trade-off between stability ("displacement between time steps") and displacement with respect to the original locations. Finally, we examine the possibility to use other symbol shapes (Section 7) and briefly discuss other extensions (Section 8).

Related work. We observe that the use of constraint programming is not new for overlap removal. Dwyer et al. [DMS05] and Marriot et al. [MSTH03] treat the dimensions independently to be able to solve the problem, but this is not necessarily optimal. It has also been used to avoid occlusion in 3D city models, using overlap removal as a soft constraint [HWAT13]. To the best of our knowledge, none has yet observed that rotating part of the problem makes it much easier to solve, even with perfect overlap removal.

The overlap removal problem also bears similarity to label placement for cartographic maps. Labels can be placed according to several models, see e.g. [vKSW99]. However, the points themselves are not displaced: the models typically do not allow for large gaps between label and point. Thus, optimization is more concerned with maximizing the number of placed labels within the constraints of the model [KR92, vKSW99]. This contrasts the problem studied here as our placement model allows symbols to stray arbitrarily far from their original location.

Disjoint placement for squares or rectangles, bearing resemblance to the original map, is also considered in grid maps [EvKSS15] and spatial treemaps [WD08]. For the former, placement on a grid is typically desired and the squares are of equal size. This leads to efficient algorithms to optimize displacement, or a 4-approximation for minimizing the number of changes to the orthogonal order [EvKSS15]. Spatial treemaps [WD08] partition a rectangle in smaller rectangles of a given size. In contrast, our solution here is not aligned on a grid and as such allows for gaps in the visualization, through which an original map can still be observed to provide context. The use of gaps to reduce geospatial distortion has been proposed by Meulemans et al. [MDS*17]. They also observe that such techniques can be useful to find clutter-free variants of other spatial distributions, such as those of a scatter plot. Our techniques can also be used similarly, but then for symbols of varying size; see Section 7 for its application to barycentric plots.

2. Preliminaries

Polyhedral distances. A (convex) polyhedral distance function δ is a metric distance between two points, in our setting in two dimensions. Characteristically, the unit "circle" of δ, that is, the locus of points at distance 1 from the origin, is the boundary of a convex polygon that strictly contains the origin. Distance $\delta(p,q)$ can be computed as $\max_{c\in C_\delta} c\cdot(q-p)/\|c\|^2$, where C_δ contains, for each edge e of the unit circle of δ, the point closest to the origin on the line spanned by e. Two common polyhedral distance functions are L_1 (Manhattan distance), with a diamond as unit circle and $L_1(p,q) = |x_p - x_q| + |y_p - y_q|$, and L_∞, with a square as unit circle and $L_\infty(p,q) = \max\{|x_p - x_q|, |y_p - y_q|\}$; here, x_p and y_p denote respectively the x- and y-coordinate of point p.

Data generation. The upcoming sections contain computational experiments to investigate facets of the overlap-removal problem. The data sets were generated according to three parameters: the number of points N, a weight parameter W, and a density parameter D. We vary N from 100 to 1 000, using increments of 100.

Parameter W determined the range from which weights are sampled. More precisely, weights are drawn uniformly at random from the range $[1,W]$. In our experiments, we use either *Uniform* ($W = 1$) or *Weighted* ($W = 12$) problem instances.

Density D determines the side length $h = \sqrt{N/D}\cdot(W+1)/2$ of the square in which the points are generated. Parameter D should be a positive value and represents the expected fraction covered of the square of area h^2 after overlap removal. In our experiments, we use either *Sparse* ($D = 0.12$) or *Dense* (0.60) problem instances.

With these parameters, we generate the point locations in one of two ways, *Random* and *Clustered* (see Fig. 2 for examples). For the former, the points are distributed randomly over the square. For the latter, we first generate $\sqrt{N/10}$ helper points in the square,

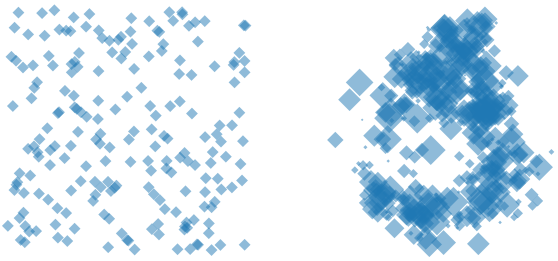

Figure 2: *Two example inputs. Left:* Sparse, Uniform, Random, $N = 200$. *Right:* Dense, Weighted, Clustered, $N = 400$.

and compute the Euclidean minimum spanning tree on the helper points. Then, we iteratively add the edge that reduces the dilation between two helper points most ($\sqrt{N/50}-1$ times). Both numbers are rounded to the nearest integer value. We call the result a *skeleton*; we use it to generate the actual point set. For each point, we first randomly select an edge (a,b) from the skeleton. Let $r = a-b$ denote the vector between its endpoints and r' the vector r rotated by $\pi/2$. We then place the point at $a+\lambda\cdot r+\mu\cdot r'$ where λ is drawn uniformly at random from $[-0.1, 1.1]$ and μ from a standard Gaussian distribution with mean 0 and standard deviation 1.

3. Constraint Programming

In this section we explore the core problem of this paper: given a set of weighted points in the plane, to be represented by suitably sized diamonds without overlap, compute the placement that minimizes the distortion. We first describe the basic linear program, which uses polyhedral distance functions. We then turn to the Euclidean distance, followed by two methods that help us to greatly reduce the number of constraints. Finally, we discuss the results of some experiments that investigate the practical efficiency, as well as the efficacy of the constraint-reduction schemes.

3.1. An explanatory note

At a glance, it seems curious that rotating the problem by 45 degrees suddenly makes an NP-hard problem tractable. It is important to observe that the orthogonal order is *not* rotated. Hence, we can indeed rotate the linear program described below by 45 degrees and obtain an LP that solves overlap removal for axis-aligned squares, but with a rotated orthogonal order: rather than preserving the order of x and y, we must then preserve the order of $x+y$ and $x-y$.

But why does this rotation make such a large difference? The explanation for that lies in studying the feasible space for the placement of one symbol with respect to the other. Whereas for squares this space is not convex, it is convex for the diamond case (see Fig. 3). This space is bounded by three lines, which capture the main aspects of the linear program for these two symbols.

3.2. The Linear Program

Problem statement. Let $P = \{p_1, \dots p_n\}$ denote a set of n points, where each point p_i has an x-coordinate x_i, a y-coordinate y_i and a weight w_i. Moreover, let δ denote a convex polyhedral distance function. We wish to find a position p'_i for each point p_i in P, such that: (1) diamonds drawn at p'_i with radius w_i are pairwise disjoint, that is, $L_1(p'_i, p'_j) \geq w_i + w_j$ for each $1 \leq i < j \leq n$; (2) the orthogonal order is maintained, that is, $x'_i \leq x'_j$ if and only if $x_i \leq x_j$ and analogously for y-coordinates; (3) the sum of displacements, that is, $\sum_{i=1}^{n} \delta(p_i, p'_i)$ is minimized.

Variables. For each point $p_i \in P$, we introduce three variables in the linear program: x'_i, y'_i and d_i. The first two represent the new position p'_i of point p_i, whereas the latter represents the displacement of the point with respect to its original location, as measured via distance function δ.

Objective function. We wish to minimize the total displacement, measured as the sum of distances using δ. Hence, the objective function is:

$$\text{minimize} \sum_{i=1}^{n} d_i$$

Constraints. First, we add constraints to ensure that the new positions respect the orthogonal order of the input points. Let $h_1, \dots, h_n$ denote the indices of P in sorted x-order; and $v_1, \dots, v_n$ in sorted y-order. For all $1 \leq i < n$ we add the following constraints:

$$x'_{h_i} \leq x'_{h_{i+1}} \text{ and } y'_{v_i} \leq y'_{v_{i+1}}$$

Second, we must add constraints to ensure that the resulting diamond shapes are disjoint. We present the constraint below for two points p_i and p_j such that p_i is to the left and below p_j; the other constraints are analogous. We must ensure that the L_1 distance (as its unit circle is a diamond) between the two centers is at least the sum of their weights. That is, we want that $L_1(p'_i, p'_j) = |x'_i - x'_j| + |y'_i - y'_j| \geq w_i + w_j$. Though the absolute value is not linear, we can use the fact that the orthogonal order is maintained. Since $x_i \leq x_j$, we know that $x'_i \leq x'_j$ in the output: $|x'_i - x'_j|$ is hence equal to $x'_j - x'_i$. Concretely, we add for all

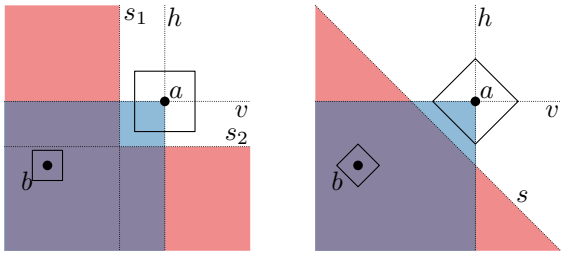

Figure 3: *Feasible space (purple) for the placement of b with respect to a, for squares (left) and diamonds (right). The blue+purple region is dictated by the orthogonal order: b must start left of h for the horizontal order and below of v for the vertical order. In this quadrant, b must remain left of s_1 or below s_2 in the square case to avoid overlap (red+purple region), causing a nonconvex feasible space. For the diamond case, it must simply remain below separator s. Note that these separators are at distance $w_a + w_b$ from a, measured with L_∞ (square) and L_1 (diamond).*

$1 \leq i < j \leq n$ with $x_i \leq x_j$ and $y_i \leq y_j$ the constraint

$$x'_j - x'_i + y'_j - y'_i \geq w_i + w_j.$$

Finally, we must add constraints that relate the displacement variables to the actual positions. We know that $\delta(p_i, p'_i)$ is computed as $\max_{c \in C_\delta} c \cdot (p'_i - p_i)/\|c\|^2$. Thus, we add the constraint

$$d_i \geq x/\|c\|^2 (x'_i - x_i) + y/\|c\|^2 (y'_i - y_i)$$

for all $1 \leq i \leq n$ and $c = (x, y) \in C_\delta$. This ensures that d_i is at least this maximum; the minimization objective ensures that d_i is equal to this maximum. Since we know the orthogonal order, we know which $c \in C_\delta$ may yield a positive value for $c \cdot (p'_i - p_i)$ and need to add only those constraints.

3.3. Euclidean distances

Approximation. The most natural distance metric in the plane is the Euclidean distance. As this metric is nonlinear, it cannot be incorporated directly into the linear program. However, L_1 overestimates the Euclidean distance by at most a factor $\sqrt{2}$; similarly, L_∞ underestimates the Euclidean distance by at most a factor $\sqrt{2}$. Hence, using $\delta = L_1$ or $\delta = L_\infty$ gives us a $\sqrt{2}$-approximation of the problem under the Euclidean distance.

Furthermore, we can approximate the unit circle of the Euclidean distance, using a regular k-gon. To get a $(1+\varepsilon)$-approximation, we need to use $k = O(\varepsilon^{-1/2})$. Proof of this claim is provided in Appendix A of the online material.

Heuristically, L_∞ tends to give better results than L_1. This is explained as follows (refer to Fig. 4). Consider the placement of two overlapping symbols. Under the L_∞ metric, the closest point to the original location is unique and the same as the point achieving the best Euclidean distance. Under L_1 however, all points along the separator s in the appropriate quadrant are indistinguishable.

Convex quadratic program. However, regardless of approximation ratio, the sum of Euclidean distances may not make a reasonable trade-off between significantly displacing one diamond and displacing many diamonds only a little. To penalize large displacements more heavily, we may want to use the sum of *squared* Euclidean distances instead. Again, this is nonlinear, but it can be computed via a convex quadratic program.

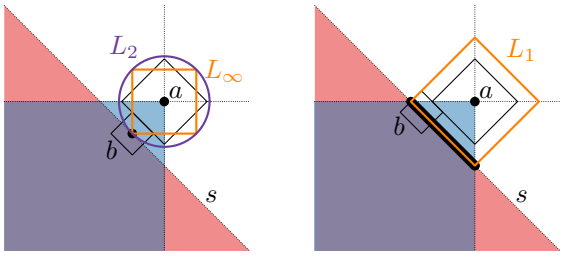

Figure 4: *Consider the placement of just two overlapping symbols. Left: the closest point to a is the same for L_2 (Euclidean distance) and L_∞. Right: for L_1, all center positions on the thick black line have the same quality.*

To do so, we set up the same constraints as for the linear program. However, we now omit the variables d_i and their corresponding constraints. Moreover, we change the optimization function to minimize $\sum_{i=1}^n (x_i - x'_i)^2 + (y_i - y'_i)^2$. We observe that the matrix describing the quadratic part of the optimization function is the identity matrix: in other words, the quadratic terms always involve the same variable twice, rather than being the product of two different variables. Thus, this matrix is trivially a positive definite matrix – a requirement for using convex quadratic programming.

3.4. Constraint-reduction techniques

The constraint programs enforce the symbols to be disjoint through a diagonal separating line. The orientation of this line is known beforehand, due to the orthogonal order constraints. The basic program described above introduces such a separation constraint for each pair of points in P, thus resulting in $\sum_{i=1}^n (i-1) = n \cdot (n-1)/2 = O(n^2)$ constraints. Below, we briefly describe two ways to reduce the number of constraints, while still being able to guarantee that the solution is free of overlap. The effectiveness of these approaches is considered in Section 3.5.

Lazy constraints. We may expect many intersection constraints to be satisfied automatically. Hence, we suggest using a technique often employed for integer linear programming: lazy constraints. Concretely, we initially add disjointness constraints only for points p_i, p_j with $L_1(p_i, p_j) \leq 2 \cdot (w_i + w_j)$, that is, only for points that can intersect if they are displaced less then the sum of their weights. We then solve the program and check for intersections; if any are found, they are added as constraints and we repeat the process.

Dominance. Let $\prec_U^*$ denote a relation on P, such that $p \prec_U^* p'$ if p' lies in the top-right quadrant of p ($p_x \leq p'_x$ and $p_y \leq p'_y$). We call $\prec_U^*$ the *dominance* relation. Now, consider three points p, p', p'', such that $p \prec_U^* p' \prec_U^* p''$. This means that for all three pairs the separating diagonal has slope -1. Moreover, we know that p' effectively guarantees that p and p'' are disjoint: otherwise, the orthogonal order must be violated, or p' also intersects p or p''. As such, we know that we never need the separation constraint for p and p''. To leverage this, we compute the *minimal* dominance relation $\prec_U$, such that its transitive closure is $\prec_U^*$. Any separation constraints in $\prec_U^*$ are guaranteed by constraining only those in $\prec_U$.

We can compute all pairs in $\prec_U$ as follows. First, we sort P lexicographically. Then, for each point $p_i \in P$, we compute all points p_j with $p_j \prec_U p_i$ (and hence $j < i$), by going through the points from p_{i-1} back to p_1. Whenever a point p_j is found with $y^* < p_{j,y} \leq p_{i,y}$, we add $p_j \prec_U p_i$ to the relation. Here, y^* denotes the maximum y-coordinate of earlier points below $p_{i,y}$ (initially $-\infty$). This simple computation runs in $O(n^2)$ time and thus adds no asymptotic overhead to setting up $O(n^2)$ constraints.

Symmetrically, we consider a relation $\prec_D^*$ for separators with slope $+1$, that is, $p \prec_D^* q$ if $p_x \geq p'_x$ and $p_y \leq p'_y$. As only the x-coordinate has reversed its role, we can re-sort P using an inverted x-axis to compute the minimal relation $\prec_D$.

As maintaining the minimal versions implies maintaining the full dominance relations, the constraints in $\prec_U$ and $\prec_D$ are sufficient to

ensure that the result contains no overlapping symbols. Though in the worst-case $\prec_U$ or $\prec_D$ can still contain $O(n^2)$ pairs, we expect this number to be significantly less in practice.

3.5. Experiment: efficiency

To investigate the scalability of the basic LP and CQP solutions, we implemented these methods (with $\delta = L_\infty$ for the LP). We solved 10 instances for each condition (see Section 2). We measured the elapsed time, including the time necessary to setup the programs and read the results, on a HP ZBook with an Intel Core i7-6700HQ CPU, 24 GB RAM and running Windows 8.1. CPLEX 12.7.1 with default settings was used to solve the constraint programs.

Dominance. Fig. 5 shows the results obtained by measuring the number of constraints in $\prec_U$ and $\prec_D$, with respect to the regular case of $n(n-1)/2$ constraints. As we can see, at least 85% of the constraints are eliminated without any need for further consideration, and this effect increases as N increases. Moreover, there seems little variation in the size of the dominance relations. Though we indeed know that quadratic remains a worst-case bound, this does not occur with the generation methods used. There does seem to be a very small effect related to the point placement method (in favor of the more realistic *Clustered*), but this was not investigated further. Note that these numbers are dependent only on the point locations and thus independent of other aspects, such as the distance metric.

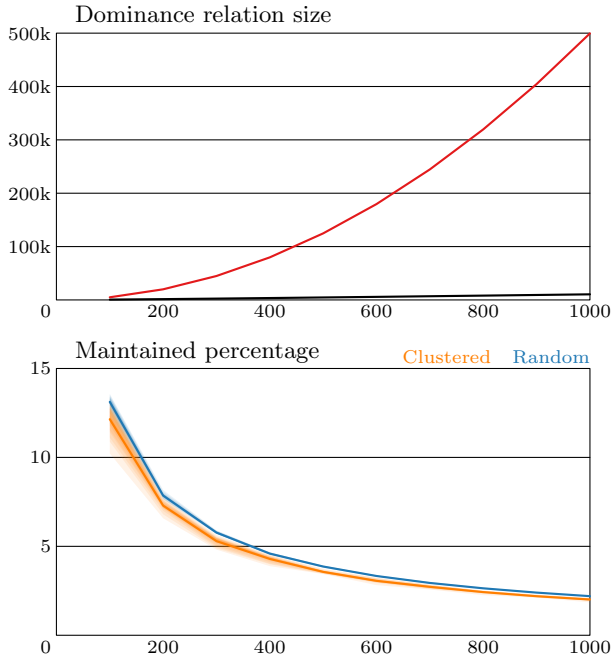

Figure 5: *Efficacy of reducing the number of constraints via the dominance relations. Top: constraints remaining (black line) and the quadratic upper bound (red). Bottom: percentage of remaining constraints as a function of N, with breakdown per generation method. Note the cropped y-axis.*

Time. The main results in terms of running time efficiency for the LP are shown in Fig. 6; other breakdowns and results for the CQP can be found in Fig. C.2 through Fig. C.5 of the online material.

Without using constraint-reduction methods, the average running time is approximately 13.40s for $N = 1\,000$. With dominance,

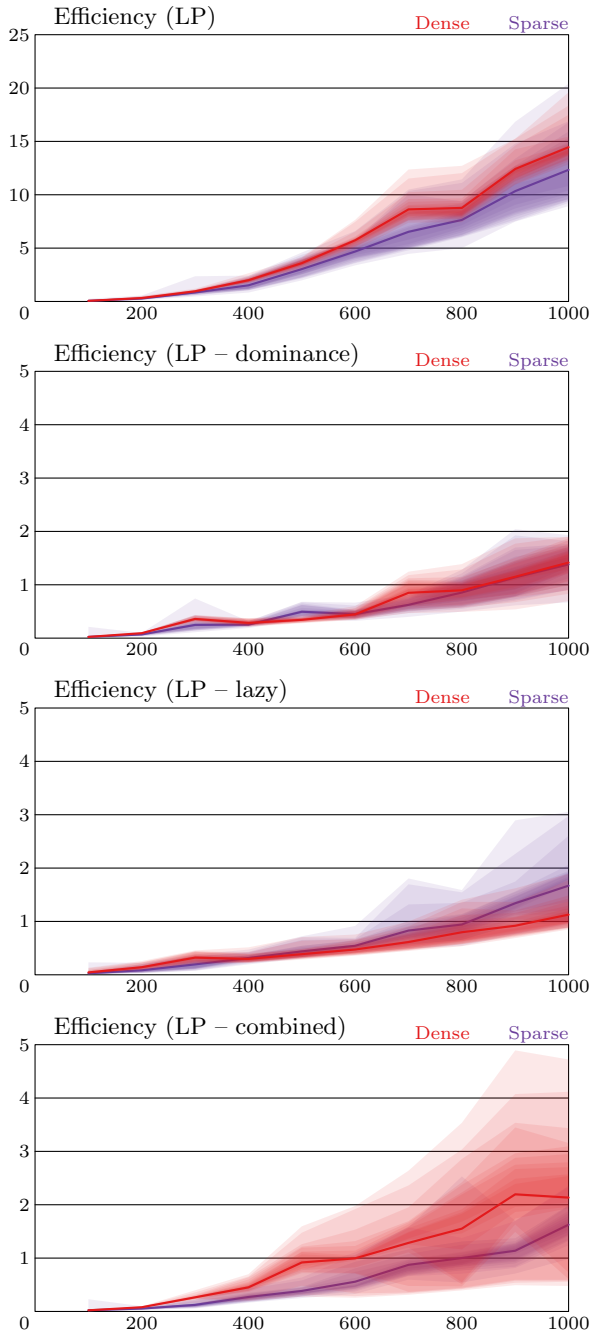

Figure 6: *Running times in seconds for the 30 runs of the LP using constraint-reduction techniques, as a function of N, split between* Dense *and* Sparse *data sets.*

this drops down to 1.40s. Using only lazy constraints has a similar effect, resulting in 1.40s average running time for $N = 1\,000$. However, we see that the combined techniques yields 1.88s. The maximum time with the dominance-based reduction is smaller; hence, to get the highest reliability, the best methodology for the linear program is to use only the dominance technique. The added effort in setting up and checking the lazy constraints may be costly.

Notably, density in the dominance case has little effect, but its effect in the unoptimized and combined case is inverse with respect to the lazy case. This can be attributed to the number of times lazy constraints need to be added: on average 1.2 times in the *Sparse* case, yet only 0.2 times in the *Dense* case. A similar situation arises for the point-placement method (see Fig. C.5): whereas random cases are solved faster in unoptimized runs, clustered cases solve faster when dominance is used – we attribute to the dominance reduction working better for clustered cases.

For the CQP, the effects are slightly different. For the unoptimized case, we obtain running times in excess of 6 minutes already for $N = 200$. But we get average running times for $N = 1\,000$ of 10.01s (dominance), 11.05s (lazy constraints) and 2.21s (combined). Here it turns out to be worthwhile to use both techniques simultaneously. The difference is possibly caused by the increased complexity of solving a CQP. That is, the time saved by not accounting for the lazy constraints initially weighs more strongly with respect to the overhead of potentially running another iteration. With a vast majority of cases (78.6%) solving correctly without adding any lazy constraints, this may be favorable for the average.

We conclude that (near-)interactive speeds are indeed achievable at least for data sets up to 1 000 points, for both the LP and the CQP.

4. Scale

The constraint programs sketched above (theoretically) use the infinite plane in which the symbols can be placed. As such, uniformly scaling all the coordinates eventually removes all overlap, if a large enough scaling factor is used. But once we have to use the solution in an actual visualization system, the result has to be scaled down again to fit a target frame, effectively shrinking the symbol sizes. In this section, we briefly consider how we can directly incorporate such scale concerns into the solution.

4.1. Modeling scale

The easiest way to incorporate scale is to create a rectangular frame (or any other convex polygon) that represents the graphical space in which the resulting overlap-free visualization must be placed eventually. We can then constrain each symbol to remain inside this frame. Assume that the left side of this frame has x-coordinate L. Then, we can restrict $L \leq x'_i - w_i$ for each point p_i with weight w_i. Analogously, we add constraints for each side of the frame.

The problem with this solution is that it may result in infeasible instances. That is, there may be no placement of the symbols such that all constraints are met. To remedy this, we allow the program to uniformly scale-down all symbols. To do so, we introduce a single variable ς with $0 \leq \varsigma \leq 1$. We then replace each occurrence of a weight w_i by ςw_i. All constraints remain linear, as the weight is never multiplied with a variable in the original constraint programs.

Of course, using $\varsigma = 0$ then trivially solves any instance, degenerating each symbol into an infinitesimally small point. Hence, we should discourage the use of small values for ς. To this end, we add to the minimization function the term $-\varsigma \cdot \Gamma$ for some sufficiently large value Γ. In case we do not have reasonable symbol sizes already, we can also allow ς to be larger than 1, thereby admitting solutions that increase the symbol size.

We can also drop the frame and use smaller values of Γ to model a trade-off between displacement and symbol size, which can be seen as a proxy for legibility. But since this proxy does not readily capture the aspect ratio of the resulting visualization, we may expect that such a variant is more cumbersome to work with in practice, needing a careful selection of Γ.

4.2. Experiment: displacement versus scale

We wish to investigate how symbol size (scale) influences displacement. To this end, we run the original linear program using a *fixed* value for ς, between 0.01 and 2 and repeat this 10 times for each condition. In other words, we scale all symbol sizes by ς prior to setting up and solving the LP, and do not use a frame as described above. This is to avoid the aspect ratio from influencing the results. Note that the original symbol sizes can be expected to greatly influence these results. Hence, we also increase their sizes to allow exploring the effect with less dependency on the generated weights.

Fig. 7 shows the resulting sum of Euclidean distances (normalized to a fraction of the $\varsigma = 2$ case) as a function of ς. We observe that there is a strong difference depending on the (initial) density parameter D. Specifically, the *Sparse* case seems to increase at an increasing rate, whereas the *Dense* case increases mostly linearly. However, we may consider the *Dense* input as a *Sparse* input with $\varsigma = \sqrt{5}$, and thus provides us with a view of what is likely to happen for the *Sparse* instances if we increase ς further. Fig. C.6 in the online material shows that there is an additional effect of point-placement methodology, where *Clustered* inputs scale worse than *Random* inputs – this likely can be explained by the *locally* higher density of the former.

The linear behavior suggests that all symbols are affected equally

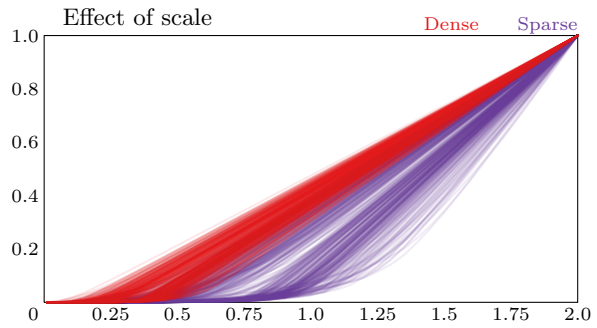

Figure 7: *Results showing the sum of Euclidean distances, as a ratio to the $\varsigma = 2$ case, as a function of ς. Each line is one input.*

with the increase in scale. As such, the visualization seems "saturated" and the scale is of little influence. Likely, this behavior indicates that a certain maximum effective scale has been reached. Scaling it up further does not structurally change the solution and fitting it to a graphical container then yields the same result. Fig. C.6 in the online material shows that weak order constraints (see Section 5) move this saturation value to larger values of ς.

5. The cost of order

The orthogonal order ensures that the solution space of our problem is convex. Essentially, it allows us to derive exactly which of the potential diagonal lines separating two diamond shapes is relevant. At the same time, the orthogonal order may be a needlessly strong restriction to the solution. In this section we investigate the use of the orthogonal order in determining the separators, but relax the constraints on the placement of the actual centers. This relaxation may greatly decrease the displacement of the centers and thus allow for more compact results. This allows a trade-off between maintain spatial relations (orthogonal order) between the data elements and the spatial distortion (displacement) of each data element.

5.1. Relaxing the order constraints

The described linear program remains valid even if we drop the orthogonal order constraints. That is, the disjointness constraints remain functional, but dictate a (weaker) placement constraint. This is illustrated in Fig. 8: in the original setting, the placement of the smaller diamond's center must be in the purple region with respect to the large diamond's (new) position; in this version with *weak* order constraints, it may also be placed in the red areas. This essentially eliminates the order constraints.

Rotating order constraints. As an alternative to removing the order constraints completely, we can also change the two lines defining the order constraints. Consider two points $a, b \in P$, such that $b \prec_U a$ (that is, a is above and to the right of b). Instead of requiring that the new position of b is to the left of the vertical line through the new position of a, we rotate the line by α degrees in counterclockwise direction around a. Similarly, we rotate the horizontal line that b should stay below by angle α in clockwise direction. The rationale here is that we want to maintain the orthogonal order more strictly for points placed near to each other. but as the distance between points increases, we allow more flexibility. In this model, $\alpha = 0$ corresponds to the orthogonal order and $\alpha = \pi/2$ to having weak order constraints.

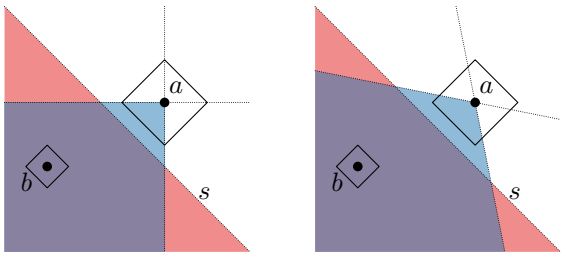

Figure 8: *Left: orthogonal order requires b to be placed in the blue+purple region w.r.t. a; the disjointness constraint requires it to be placed in the purple+red area. Right: rotating the order constraints, with* $\alpha = \pi/16$.

Observe that the $2n-2$ constraints to capture the orthogonal order no longer suffice, because $\prec_U$ and $\prec_D$ use different lines. Rather, we need $O(n^2)$ constraints, two for each pair of points. But we can re-use the concept of dominance (Section 3.4) to significantly reduce the number of necessary constraints.

Alternatives. The order constraints can be relaxed in various other ways. Rather than or in addition to rotating the lines, we can also translate them to allow for more (or less) flexibility. We can even change the strictness of the order constraints per pair of points. For example, points that are nearby in the input must maintain exact orthogonal order, but points far away use a more flexible version.

Other separator heuristics. When we fully drop the order constraints, we can use other heuristics to determine the orientation of the separating line as well; see e.g. [BKS*11] for a method of determining a high-quality separation orientation for geographic regions. But care must be taken that the computed separators are compatible, that is, they do not yield (cyclic) constraints that are impossible to satisfy. The orthogonal order can be seen as such a heuristic that yields compatible constraints. Further consideration of other heuristics is out of scope for this paper.

5.2. Experiment: displacement versus order

Now that we can configure how strictly the "orthogonal" order is maintained, we investigate the trade-off between how much deformation is caused by the overlap removal and how much by the strictness of order preservation. Specifically, for one instance in each setting, we run the algorithm using α ranging from 0 (orthogonal order) to $\pi/2$ (weak order constraints), using the rotational model. We use a polyhedral distance function based on the regular 8-gon for the LP to get a better approximation of the Euclidean distance. We then measure the sum of Euclidean distances (LP) or sum of squared Euclidean distances (CQP). The results are shown in Fig. 9, as a function of α. We observe that a little flexibility already achieves a great reduction in displacement; for $\alpha > \pi/3$, there is little improvement for increasing the flexibility.

There is some hint that the *Sparse* inputs benefit more; this is likely caused the the nonlocal effects of the orthogonal order. That is, two symbols on opposite sides of the region that happen to have very similar y-values effect each other. *Dense* inputs are more "saturated" (see Section 4.2 and Fig. C.6) and thus these nonlocal effects are to a lesser extent also captured by local effects. As also argued in Section 4.2, this also may correlate to the point-placement method due to the local density of *Clustered* being higher. This is indeed the case, as shown in Fig. C.7 in the online material.

6. Time-varying data

Temporal dimensions of data become ever more important, with data sets being captured via regular measurements over time. Visualizing time-varying data can be done in a variety of ways, most

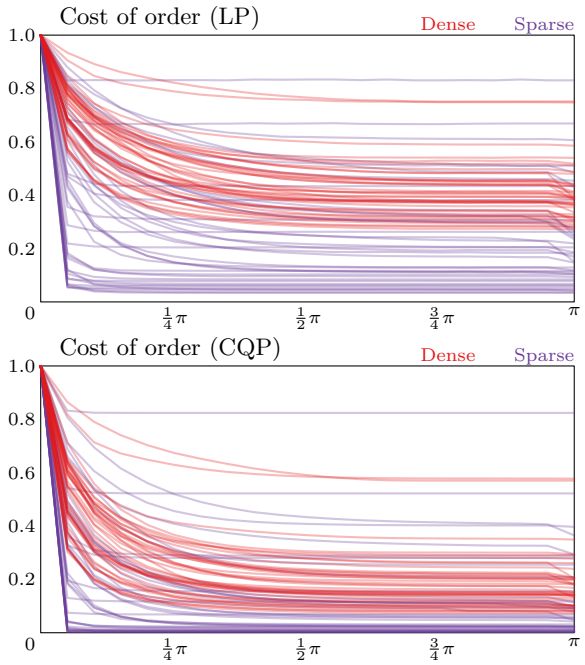

Figure 9: *Displacement as a function of* α*, for the two constraint programs. Vertical axis expresses the displacement at* α *as a ratio of the displacement for* $\alpha = 0$*. Each line is an instance.*

prominently used are animations and small multiples. Regardless of the visualization technique, *stability* is important. That is, small changes in the data should correspond to small changes in the visualization, to ensure coherence between the representation of the different time steps. In this section we briefly explore how we can capture this concept within our constraint programs and how the trade-off between stability and displacement behaves.

Our input of weighted points can vary in multiple ways. We interpret time-varying data as the change of weights; kinetic data as the movement of point locations; and dynamic data as the addition or removal of points. In our exposition, we focus on time-varying data but observe that the methodology can easily be adapted to work for kinetic and dynamic data as well. Note that kinetic data may change the orthogonal order between time steps. The online material contains a video showcasing the results for real-world, time-varying, dynamic data; Appendix B provides more details.

Here, another advantage of our technique can be found. As the point locations dictate the separating lines, these separators have the same slope for all time steps, at least for time-varying and dynamic data. As such, even if we solve each time step individually without regarding stability, we can obtain an animation without intermediate overlap between the two consecutive time steps, simply by linearly interpolating the output positions and weights.

6.1. Modeling stability

Stability can be captured in various ways, but most natural in our setting is to measure the displacement of each symbol from one time step to the next. Consider a time-varying point set P, with T time steps, denoted by P_t for integer $t \in [1,T]$. Consider two disjoint placements of the weighted symbols for consecutive time steps, P'_t and P'_{t+1}. For each point $p_i \in P$, the displacement of p_i between the two time steps is $\delta(p'_{t,i}, p'_{t+1,i})$ for some distance metric δ, where $p'_{t,i}$ denotes the position of point p_i according to P'_t. We can then measure stability from time step t to $t+1$ as $\sum_{i=1}^{n} \delta(p'_{t,i}, p'_{t+1,i})$. Again, using the Euclidean distance for δ is nonlinear; we use a polyhedral distance function to approximate it. For the time-varying LP, we use our original LP for each time step with separate variables: for each of the original variables, the subscript t indicates the time step it relates to. But additionally, we introduce a variable $s_{t,i}$ for each time step $t \in [1,T)$ and point $p_i \in P$. We then constrain $s_{t,i} \geq x/\|c\|^2(x'_{t,i} - x'_{t+1,i}) + y/\|c\|^2(x'_{t,i} - x'_{t+1,i})$ for all $1 \leq i \leq n$ and $c = (x,y) \in C_\delta$. By minimizing $\sum_{t=1}^{T-1}\sum_{i=1}^{n} s_{t,i}$, we can then optimize stability.

We must, however, note that optimizing for stability alone does not yield satisfactory results. Specifically, the placement of the symbols is not constrained to remain near their original positions. The LP therefore always admits an optimal solution with cost 0, by uniformly scaling the instance sufficiently such that the distance between each pair of points exceeds their maximal sum of weights over all time steps. Hence, we always place the stability criterion to be optimized together with the original objective function. That is, we minimize $(1-\sigma)\sum_{t=1}^{T}\sum_{i=1}^{n} d_{t,i} + \sigma\frac{T}{T-1}\sum_{t=1}^{T-1}\sum_{i=1}^{n} s_{t,i}$, for some constant $\sigma \in [0,1)$. The value $\sigma = 0$ represents no stability: each time step is optimized separately; $\sigma = 1-\varepsilon$ for some small $\varepsilon > 0$ indicates primarily optimizing stability, with as secondary concern minimizing the distance to the original location. Note that, since the stability term sums over one less time step, we normalize the two terms by multiplying the second by $T/(T-1)$.

The program becomes large rather quickly as we have data that varies over multiple time steps. To simplify it, we may run the program *incrementally*, that is, for each time step separately. The first step $t = 1$ is identical to the original problem. For any subsequent step $t > 1$, we solve the program sketched above, by using the positions obtained in the previous step. That is, we treat the variables $x'_{t-1,i}$ and $y'_{t-1,i}$ as constants instead, and measure only the displacement with respect to the original position p for time step t and stability with respect to p'_{t-1}. This incremental approach is likely to be faster for larger data sets, but is also likely to decrease stability as it deals with the now-fixed solution of the previous iteration. That is, earlier solutions are not adapted to ensure more stability later. As such, $\sigma = 1$ does not guarantee a cost of 0 anymore.

6.2. Experiment: displacement versus stability

With our stability model in place, we investigate two questions: what does the trade-off look like between displacement and stability? And how does the incremental approach compare to the nonincremental approach? We generate point sets using the same methodology as before, but have to generate a weight $w_{t,i}$ for each time step for each point p_i. We first generate a weight $w_{1,i}$ and then generate weights $w_{t,i}$ for $2 \leq t \leq T$ uniformly at random in the interval $[w_{t-1,i} - \Delta W, w_{t-1,i} + \Delta W] \cap [1,W]$. Here, ΔW is a parameter that controls how erratic the weights change over time. Specifically, we use $\Delta W = 1$ for smoothly changing weights (*Smooth*) and

$\Delta W = 12$ for uncorrelated weights (*Erratic*). Note that we use only the case $W = 12$: for $W = 1$ the weights are constant, making the solution for every time step identical – this is thus inherently stable. Moreover, we use N varying from 100 to 300 in increments of 50 and perform 10 trials for each configuration.

Fig. 10 shows the results for the nonincremental case. We observe that the stability cost steadily reduces as σ increases, though initially the displacement does not increase much. With $\sigma = 0.4$, we improve stability by 49.1% on average (26.6% at least) while increasing distortion by 2.6% (16.7% at most). Prominently visible is that the erratic case mostly covers the worst cases, and the smoothly varying data behaves much better: stability improves more quickly and the displacement suffers less. Density was found to have a similar effect, though its effect size is much smaller than the separation by ΔW; the point-placement method has little influence on this result (see Fig. C.8 of the online material).

Fig. C.9 of the online material shows additional results for the incremental case and the case with weak order constraints (see Section 5). For the latter, we see very similar patterns emerge. However, for the former, the stability ratio decreases less quickly. The tipping point is at $\sigma = 0.4$ ($= 0.5\frac{T-1}{T}$) where maintaining the stability starts to dominate, causing the new frames to be optimized mostly for stability rather than displacement; the increase in distortion is mostly limited to erratically changing weights. Note, however, that the incremental approach is not able to reduce the stability to being perfect, contrasting the nonincremental method.

But perhaps most interesting for the incremental method is how it performs with respect to the nonincremental method. Contrary to expectations, 92 of 8 400 cases see an increase of displacement of over 4% (maximum increase: 38.1%); with weak order constraints, this is for 538 cases, with a maximum increase of 28.1% increase. Similarly, the incremental method improves stability by over 10% in 224 and 362 cases respectively for the LP and CQP. Closer investigation shows that this happens mostly for σ just above the tipping point. For low values of σ, the methods are fairly similar. For high values, their displacement is also similar, but the reduction in stability can be arbitrarily large. This is caused by the nonincremental case converging to perfect stability. However, the motivation for the incremental method and indeed its main gain are its efficiency: measured on the same machine as in Section 3.5, the average running time is decreased by 78.7% and the entire set of trials was computed 85.9% faster. Using weak order constraints, these numbers are 77.5% and 90.4% respectively.

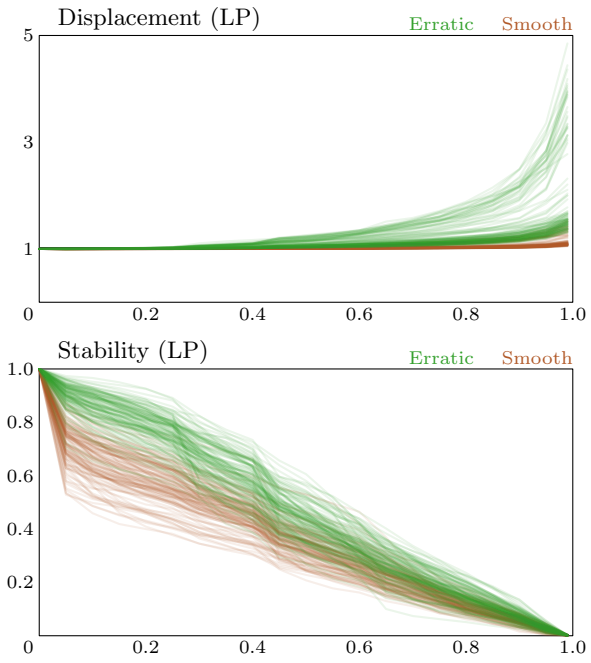

Figure 10: *Stability measurements. Horizontally, σ varies from 0 to 0.99. Vertically, the two plots show the ratio of displacement (top) and stability (bottom) to the unstable case ($\sigma = 0$).*

We conclude that the nonincremental method allows a genuine trade-off between stability and displacement. The incremental method is significantly faster, but is not suitable for modeling the trade-off. It can be used only if stability is a secondary concern. We did not, however, investigate the CQP for stability due to efficiency reasons. We may expect that the incremental method allows a better trade-off when optimizing for squared Euclidean distance.

7. Other shapes

We used an orthogonal order and diamond shapes. Of course, we can similarly solve problems where we use square shapes by rotating the orthogonal order by 45 degrees. This has no computational effect, but does have a representation change. Squares may look more natural for some applications. We can further add extra constraints to also maintain the actual orthogonal order of the squares, though this is likely to constrain the placement so much that the necessary displacement increases disproportionately.

We can also go beyond squares and consider other shapes. The primary observation that makes our method possible is that the orthogonal order implies the orientation of the separating line. If we decide on some methodology to determine this slope based on the input, then we can use any symbol shape, using the convex hull of the shape to determine its extremal points towards the fixed-slope separating line. Such solutions are not necessarily optimal in a more general sense, only optimal under the given separating lines.

We can reverse the line of thought, and wonder for a given symbol shape, which "order" constraints allow us to derive a unique necessary slope for a separating line in an optimal solution. For any convex shape S, positioned with respect to some center point c, taking all lines that pass through c and at least one vertex of S gives us such constraints. Specifically, a regular k-gon then has k constraining lines for odd k and $k/2$ constraining lines for even k. Indeed, the diamond is a regular 4-gon, and the horizontal and vertical constraining lines belonging to the orthogonal order pass precisely through its corners. This implies that 2 is the smallest number of such constraining lines, and as such, using diamonds is "minimal" in the sense of imposing order constraints that lead to optimality. Higher values for k impose ever-increasing strictness on the order. It seems unlikely that these are of general use for visualization, unless there is a specific meaning to the order constraints that are natural to the visualization. This idea is illustrated below.

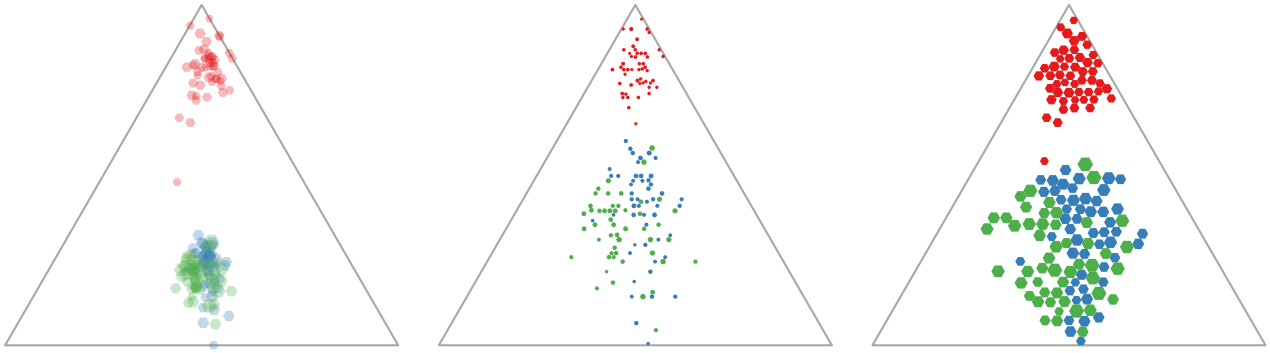

Figure 11: *Left: Iris data in barycentric coordinates, each of the attributes normalized to* $[0,1]$. *Middle: minimal displacement for hexagonal symbols and order constraints, using a regular* 6*-gon as* δ. *Right: minimal displacement with weak order constraints.*

Barycentric plots. We consider the use of our techniques for non-spatial data. In particular, we look at barycentric plots (also refered to as ternary plots), which are used in various visualization systems, e.g. [CLG16, GFG*14, NRW16]. We consider barycentric plots using an equilateral triangular frame with horizontal bottom side. If we use hexagonal shapes, we obtain three order constraints, with angles $0, \pi/3$ and $2\pi/3$ with respect to the positive x-axis. These order constraints are orthogonal to the three directions of the barycentric coordinates. Specifically, this implies that maintaining the correct order imposed by the hexagon in fact translates to ensuring that the relative order of all three coordinates is maintained for all data items. Using the triangular frame to constrain the scale (see Section 4), we thus obtain a method for removing overlap in barycentric plots, using hexagonal symbols. Fig. 11 illustrates the concept using the Iris data set (`https://en.wikipedia.org/wiki/Iris_flower_data_set`, accessed December 2018). Following the same rationale as for L_∞ (Section 3.3), we use the hexagon rotated by $\pi/6$ for δ.

Our method causes a loss of precision in the barycentric coordinates, but has the advantage that symbols are more clearly identifiable, easier to interact with, and possibly useful as frames in a small multiples visualization – similar to [MDS*17] proposing the use of 2D distributions to decide on the grid-layout of a small multiples visualization. Naturally, we can take similar steps as for the diamond case. Rather than imposing the order constraints, we can also use them only to determine the slope of the separating line. This can be interpreted as ensuring that the largest coordinate difference is maintained in the right order. This is illustrated in Fig. 11(right). This makes a further trade-off for larger symbols, at the cost of a less accurate representation of the original barycentric coordinates.

8. Other extensions

The constraint program admits other modifications. Some are described below, but full consideration is out of scope for this paper.

Min-max. In the linear program, it is straightforward to minimize the maximum displacement, rather than the sum of displacements over all symbols. To achieve this, we need to add only a variable d, constrained by $d \geq d_i$ and then minimize d in the objective function. Note the stark contrast to the NP-hardness of these problems without order constraints [FKP05].

This min-max solution is likely not unique. We can steer the linear program to a "nice" solution by maintaining the original minimization term ($\sum_{i=1}^{n} d_i$) multiplied by a small constant.

In this min-max case, the Euclidean case and squared Euclidean are identical. But we cannot bound the maximum squared Euclidean displacement directly, as it would result in quadratic inequalities which are not allowed for a CQP. Hence, we can only approximate it, using polyhedral distance functions.

Symbol importance. The constraint programs can be extended to allow for weighing the importance of symbols. That is, we can modify the objective function by multiplying each d_i by some constant c_i, where c_i may vary per point. A high value for c_i ensures that displacing p_i is penalized more heavily, compared to moving another point p_j with a low value c_j. This may be useful to prioritize the position of key data elements or characteristic.

9. Conclusion

It was already known that overlap removal for squares under the orthogonal order is NP-hard. Yet, we showed that one minor alteration – rotating the squares into diamonds – allows for efficient solutions via constraint programming. We showed how such solutions can be implemented efficiently, to achieve (near-)interactive running times for instances up to 1 000 points.

With the flexibility offered by constraint programming, we explored more facets of this problem. Specifically, we looked at ensuring that the solution fits a graphical container and at the effect of overall symbol size (scale) on the result. Moreover, we considered relaxing the requirements set by the orthogonal order, to allow for solutions with less displacement. This showed that even slightly relaxing these constraints already gives a major reduction in displacement. We also investigated supporting time-varying data with two different methods: one supports a clear trade-off between stability and displacement, whereas the other is significantly faster but does not allow for a high-quality trade-off. Finally, we considered other extensions, specifically also to other shapes and suggested the use of overlap removal using hexagons for barycentric plots.

Future work. We demonstrated that the overlap-removal problem, using diamond-shaped symbols and the orthogonal order, is efficiently solvable. It raises the question whether specialized algorithms exist that solve the problem optimally and are more efficient than constraint programming.

Though our method allows a trade-off between stability and displacement, it remains to be established how to effectively use this trade-off to choose good values for σ. Specifically in animation, the symbols may visually move when using smaller values of σ, whereas the actual location does not change. How does this affect interpretation of the data? Does such movement weigh up against the lower distortion in each frame?

Symbol maps without displacement can be optimized for symbol visibility through their drawing order [CHvKS10]. We may be able to significantly reduce displacement while keeping larger symbols, by allowing some limited overlap between symbols and choosing a good drawing order. If we do these two steps in sequence, we can simply use the techniques in this paper (effectively we are scaling down symbols) combined with those presented by Cabello et al. [CHvKS10]. Yet, it is exactly at the boundary that overlap is being introduced, whereas the boundary is specifically important for symbol-size estimation [CHvKS10]. Thus, these two processes likely effect each other: can we simultaneously optimize these? And how does this compare to solving the steps in sequence?

Spatial distortion knows many facets. We studied the effect of order and displacement, but other components can be studied as well. For example, topological effects have not been included. How can we effectively capture such aspects for overlap removal? This is particularly interesting when the symbols represent geographic regions which have a natural definition of topology.

We may want to ensure that the displacement vector of a symbol is similar to those of nearby symbols. That is, we may want to ensure that spatially close symbols are displaced similarly. The order constraints relate to this criterion only partially and do not immediately allow for optimizing this spatial similarity. In the context of pattern matching, such a problem has been studied without symbol sizes and order constraints [KKS11]. It shows that a trade-off between the local similarity of displacement vectors and their length is possible in that specific scenario. Can we incorporating such considerations also in this technique? This will likely require careful consideration of a definition for neighborhood, and of the importance given to this local similarity of transformation.

References

[BKS*11] BUCHIN K., KUSTERS V. J. J., SPECKMANN B., STAALS F., VASILESCU B.: A splitting line model for directional relations. In *Proc. 19th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems* (2011), pp. 142–151. 7

[CCJ90] CLARK B. N., COLBOURN C. J., JOHNSON D. S.: Unit disk graphs. *Discrete mathematics 86*, 1-3 (1990), 165–177. 1

[CHvKS10] CABELLO S., HAVERKORT H. J., VAN KREVELD M. J., SPECKMANN B.: Algorithmic aspects of proportional symbol maps. *Algorithmica 58*, 3 (2010), 543–565. 1, 11

[CLG16] CAO N., LIN Y.-R., GOTZ D.: UnTangle Map: Visual analysis of probabilistic multi-label data. *IEEE Transactions on Visualization and Computer Graphics 22*, 2 (2016), 1149–1163. 10

[DMS05] DWYER T., MARRIOTT K., STUCKEY P. J.: Fast node overlap removal. In *Proc. International Symposium on Graph Drawing* (2005), LNCS 3843, pp. 153–164. 2

[Dor96] DORLING D.: *Area Cartograms: their Use and Creation*, vol. 59 of *Concepts and Techniques in Modern Geography*. University of East Anglia, 1996. 2

[EJS05] ERLEBACH T., JANSEN K., SEIDEL E.: Polynomial-time approximation schemes for geometric intersection graphs. *SIAM Journal on Computing 34*, 6 (2005), 1302–1323. 1

[EvKSS15] EPPSTEIN D., VAN KREVELD M., SPECKMANN B., STAALS F.: Improved grid map layout by point set matching. *International Journal of Computational Geometry & Applications 25*, 02 (2015), 101–122. 2

[FKP05] FIALA J., KRATOCHVÍL J., PROSKUROWSKI A.: Systems of distant representatives. *Discrete Applied Mathematics 145*, 2 (2005), 306–316. 2, 10

[GFG*14] GANUZA M. L., FERRACUTTI G., GARGIULO M. F., CASTRO S. M., BJERG E., GRÖLLER E., MATKOVIĆ K.: The Spinel Explorer – interactive visual analysis of spinel group minerals. *IEEE Transactions on Visualization and Computer Graphics 20*, 12 (2014), 1913–1922. 10

[GNCNT13] GOMEZ-NIETO E., CASACA W., NONATO L. G., TAUBIN G.: Mixed integer optimization for layout arrangement. In *Proc. Conference on Graphics, Patterns and Images* (2013), pp. 115–122. 2

[HWAT13] HIRONO D., WU H.-Y., ARIKAWA M., TAKAHASHI S.: Constrained optimization for disoccluding geographic landmarks in 3D urban maps. In *Proc. IEEE Pacific Visualization Symposium* (2013), pp. 17–24. 2

[KKS11] KNAUER C., KRIEGEL K., STEHN F.: Non-uniform geometric matchings. In *Proc. International Conference on Computational Science and Its Applications* (2011), LNCS 6784, pp. 44–57. 11

[KR92] KNUTH D. E., RAGHUNATHAN A.: The problem of compatible representatives. *SIAM Journal on Discrete Mathematics 5*, 3 (1992), 422–427. 2

[MDS*17] MEULEMANS W., DYKES J., SLINGSBY A., TURKAY C., WOOD J.: Small multiples with gaps. *IEEE Transactions on Visualization and Computer Graphics 23*, 1 (2017), 381–390. 2, 10

[MSTH03] MARRIOTT K., STUCKEY P., TAM V., HE W.: Removing node overlapping in graph layout using constrained optimization. *Constraints 8*, 2 (2003), 143–171. 2

[NRW16] NGUYEN H., ROSEN P., WANG B.: Visual exploration of multiway dependencies in multivariate data. In *Proc. 2016 ACM SIGGRAPH ASIA Symposium on Visualization* (2016), p. 2. 10

[SSS*12] STROBELT H., SPICKER M., STOFFEL A., KEIM D., DEUSSEN O.: Rolled-out Wordles: A heuristic method for overlap removal of 2D data representatives. *Computer Graphics Forum 31*, 3pt3 (2012), 1135–1144. 2

[vG18] VAN GARDEREN M.: *Pictures of the Past – Visualization and visual analysis in archaeological context*. PhD thesis, Universität Konstanz, 2018. 2

[vGPNB17] VAN GARDEREN M., PAMPEL B., NOCAJ A., BRANDES U.: Minimum-displacement overlap removal for geo-referenced data visualization. *Computer Graphics Forum 36*, 3 (2017), 423–433. 2

[vKSW99] VAN KREVELD M. J., STRIJK T., WOLFF A.: Point labeling with sliding labels. *Computational Geometry 13*, 1 (1999), 21–47. 2

[vKSW04] VAN KREVELD M., SCHRAMM É., WOLFF A.: Algorithms for the placement of diagrams on maps. In *Proc. 12th Annual ACM International Workshop on Geographic Information Systems* (2004), pp. 222–231. 1

[WD08] WOOD J., DYKES J.: Spatially ordered treemaps. *IEEE Transactions on Visualization and Computer Graphics 14*, 6 (2008), 1348–1355. 2