On minimal-displacement overlap removal

On Minimum-Displacement Overlap Removal

Wouter Meulemans*†
TU Eindhoven

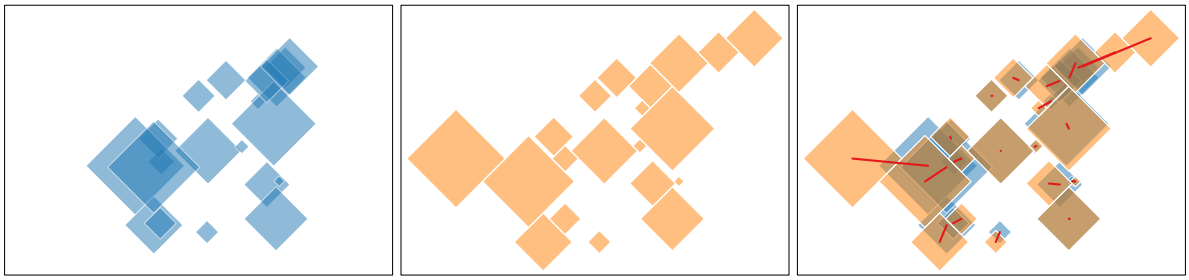


Figure 1: Left: example input of overlapping diamonds. Middle: output of the suggested linear program, having minimal displacement and maintaining the orthogonal order. Right: overlay visualizing the displacement of each diamond.

ABSTRACT

In the context of visualizing spatial data using proportional symbols, the following problem often arises: given a set of overlapping squares of varying sizes, reposition the squares as to remove the overlap while minimizing the displacement of the squares, constrained to maintain the orthogonal order. 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 and is thus efficiently solvable even for relatively large instances.

Index Terms: Human-centered computing—Visualization

1 INTRODUCTION

Proportional symbol maps are a common way of visualizing scalar values in 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. However, locations are often close to each other and thus placing the symbol precisely on its location results in overlapping symbols. greatly reduced symbols sizes or the omission of symbols: none of these solutions is particularly effective for obtaining a complete, legible visualization. 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, one could pose additional constraints to ensure that important geospatial characteristics carry over to the repositioned symbols. A common constraint is the *orthogonal order*: if symbol A is left of symbol B in the geospatial situation, then symbol A should be left of symbol B after repositioning, and likewise for the "above" relation. This constraint helps retain spatial relations, that may be crucial for certain application domains [6].

The removal of overlap between nodes finds also numerous applications in other visualization problems, such as visualizing disjoint glyphs on a map [6], computing network layouts [2, 4], positioning nonspatial data [3, 5] and computing Demer's and Dorling cartograms [1]. However, this problem of overlap removal is NP-hard in most settings, in particular also the variant described above, leading to a wealth of heuristic approaches.

*e-mail: w.meulemans@tue.nl
†Supported by the Netherlands eScience Center (NLeSC, 027.015.G02)

Contribution We show that the problem admits a linear program, if we change the symbol shape from a square to a diamond, i.e., a square rotated by 45 degrees. We leverage polyhedral distance functions (see preliminaries) 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.

The solution can be adapted to stronger and weaker order constraints, other objective functions, allowing uniform & nonuniform rescaling, weighing importance of symbols, and even minimizing the sum of squared Euclidean distances via convex quadratic programming. Though a full consideration of these variants is out of scope, it opens an interesting road forward to examine the trade-offs and their implications on the use of the resulting visualizations.

We observe that the use of constraint programming is not new for node removal. In [2, 4] they treat the dimensions independently to be able to solve the problem, but this is not necessarily optimal. To best of our knowledge, none has yet observed that rotating part of the problem makes it much easier to solve.

Preliminaries 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 $L_1(p,q) = |p_x - q_x| + |p_y - q_y|$, and L_∞, with a square as unit circle and $L_\infty(p,q) = \max\{|p_x - q_x|, |p_y - q_y|\}$; here, p_x and p_y denote respectively the x- and y-coordinate of point p.

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 disjoint; (2) the orthogonal order is maintained, that is, $x'_i \le x'_j$ if and only if $x_i \le x_j$ 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: 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.

Objective function We minimize the total displacement, measured as the sum of distances: $\sum_{i=1}^{n} d_i$.

Constraints First, we add constraints that ensure that the new positions respect the orthogonal order of the input points. Let $h_1, \ldots, h_n$ denote the indices of P in sorted x-order; and $v_1, \ldots, v_n$ in sorted y-order. For all $1 \leq i < n$ we add the following constraints: $x'_{h_i} \leq x'_{h_{i+1}}$ and $y'_{v_i} \leq y'_{v_{i+1}}$

Second, we must add constraints that ensure that the resulting diamond shapes are disjoint. However, since we know the orthogonal order of two diamonds based on the given positions, we can easily formulate the constraint. 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 then 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. Concretely, we add for all $1 \leq i < j \leq n$ with $x_i \leq x_j$ and $y_i \leq y_j$ the constraint $(x'_j - x'_i) + (y'_j - y'_i) \geq w_i + w_j$.

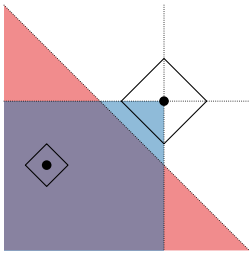
Finally, we must add constraints that relate the displacement variables to the actual positions. We know that $\delta(p_i, p'_i)$ is computed as $\max_{c \in C_\delta} c \cdot (p'_i - p_i)/\|c\|^2$. Thus, we add the constraint $d_i \geq x/\|c\|^2(x'_i - x_i) + y/\|c\|^2(y'_i - y_i)$ for all $1 \leq i \leq n$ 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 Discussion

Euclidean distance The most natural displacement metric is the Euclidean distance. As this metric is nonlinear, it is not amenable to incorporate directly into the linear program. However, as L_1 overestimates the Euclidean distance by at most a factor $\sqrt{2}$, using $\delta = L_1$ gives us immediately 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 $k = O(\varepsilon^{-1/2})$.

Alternatively, it is also possible to instead use convex quadratic programming, omitting the variables d_i and their corresponding constraints, and instead optimizing for $\sum_{i=1}^{n} (x_i - x'_i)^2 + (y_i - y'_i)^2$. Note that this optimizes for sum of squared Euclidean distances, rather than sum of Euclidean distances.

Orthogonal order The orthogonal order helps us solving the problem efficiently, but may not always be necessary to maintain as strictly. 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 the figure: in the original setting, the placement of the smaller diamond's center must be in the purple region w.r.t. the large diamond's (new) position; in this weaker version, it may also be placed in the red areas.



Efficiency To investigate the scalability, we implemented the above linear program with $\delta = L_\infty$, solved with CPLEX 12.7.1 on a modern laptop. We generated random instances with up to 1000 points and measuring the running time for 20 runs. The results are given in Fig. 2(top). We note that density ("average weight per pixel" in the input) has quite an impact.

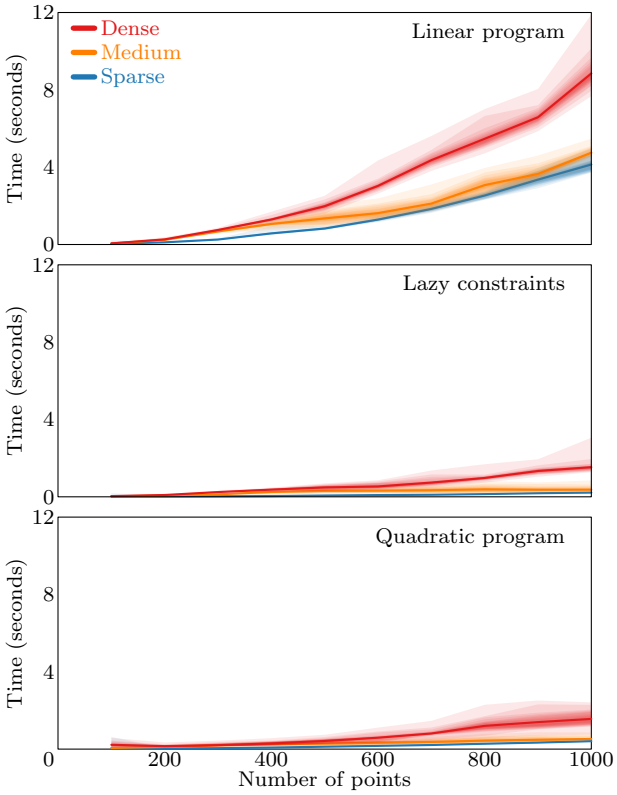


Figure 2: Running time per density for three settings.

The running time of around 10 seconds for 1000 points is still somewhat high. This can likely be attributed to the quadratic number of disjointness constraints. As many of these will automatically be satisfied, we suggest using a technique often employed for integer linear programming: lazy constraints. Concretely, we initially add only disjointness constraints 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 linear program and check for intersections; any intersections found are added as constraints after which we repeat the process. This significantly reduces the running time, with the average time dropping from 6 to 0.71 seconds for 1000 points.

Finally, we measured the efficiency of the convex quadratic program with lazy constraints. Its running time is similar to the linear program with lazy constraints; if we omit the lazy constraints, however, the running time increases drastically (not shown in figure).

We conclude that (near-)interactive speed is achievable.

References

[1] D.Dorling. *Area Cartograms: their Use and Creation*, vol. 59 of *Concepts and Techniques in Modern Geography*. 1996.

[2] T. Dwyer, K. Marriott, and P. J. Stuckey. Fast node overlap removal. In *Proc. Int. Symp. on Graph Drawing*, LNCS 3843, pp. 153–164, 2005.

[3] E. Gomez-Nieto, W. Casaca, L. G. Nonato, and G. Taubin. Mixed integer optimization for layout arrangement. In *Proc. Conf. on Graphics, Patterns and Images*, pp. 115–122, 2013.

[4] K. Marriott, P. Stuckey, V. Tam, and W. He. Removing node overlapping in graph layout using constrained optimization. *Constraints*, 8(2):143–171, 2003.

[5] H. Strobelt, M. Spicker, A. Stoffel, D. Keim, and O. Deussen. Rolledout Wordles: A heuristic method for overlap removal of 2D data representatives. *Computer Graphics Forum*, 31(3pt3):1135–1144, 2012.

[6] M. van Garderen, B. Pampel, A. Nocaj, and U. Brandes. Minimum-displacement overlap removal for geo-referenced data visualization. *Computer Graphics Forum*, 36(3):423–433, 2017.