Algorithmic Aspects of Proportional Symbol Maps

S. Cabello[1], H. Haverkort[2], M. van Kreveld[3], and B. Speckmann[2]

[1] Department of Mathematics, Institute for Mathematics, Physics and Mechanics, Ljubljana, Slovenia. sergio.cabello@imfm.uni-lj.si
[2] Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands, cs.herman@haverkort.net and speckman@win.tue.nl
[3] Institute for Information and Computing Sciences, Utrecht University, The Netherlands, marc@cs.uu.nl

Abstract. Proportional symbol maps visualize numerical data associated with point locations by placing a scaled symbol—typically opaque disks or squares—at the corresponding point on a map. Overlapping symbols need to be drawn in such a way that the user can still judge their relative sizes accurately.
We identify two types of suitable drawings: *physically realizable drawings* and *stacking drawings*. For these we study the following two problems: Max-Min—maximize the minimum visible boundary length of each symbol—and Max-Total—maximize the total visible boundary length over all symbols. We show that both problems are NP-hard for physically realizable drawings. Max-Min can be solved in $O(n^2 \log n)$ time for stacking drawings, which can be improved to $O(n \log n)$ or $O(n \log^2 n)$ time when the input has certain properties. We also experimented with four methods to compute stacking drawings: our solution to the Max-Min problem performs best on the data sets considered.

1 Introduction

Proportional symbols maps, also known as *graduated symbol maps*, are a well established cartographic tool to visualize quantitative data that is associated with specific (point) locations. A symbol, most commonly a disk or a square, is scaled such that its area corresponds to the data value associated with a point and then placed at exactly that point on a geographic map. The spatial distribution of the data can then be observed by studying the spatial distribution of the differently sized symbols. Typical data

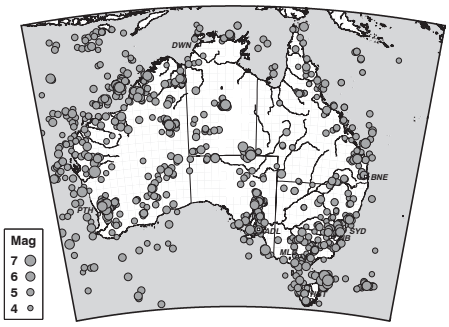

Fig. 1. A proportional symbol map depicting Australian earthquakes of size > 4.0 on the Richter scale [12].

that are visualized in this way include the magnitude of earthquakes (see Fig. 1), the production of oil wells, or the temperature at weather stations.

A proportional symbol map communicates its message via the sizes of its symbols—both the actual size of the symbols and the ratio between symbol sizes. There exists a large amount of theory and user studies that discuss which sizing communicates the difference between quantities in the most effective way. See the books by Dent [5] and Slocum et al. [13] for an extensive overview.

While it is commonly agreed upon that a map should appear "neither 'too full' nor 'too empty'" [13] it is unclear how much the symbols on a proportional symbol map should overlap. Small symbols create little or no overlap but spatial patterns are difficult to detect. On the other hand, large symbols result in a cluttered map where it is difficult to identify and judge individual symbols. Determining the ideal size for the symbols is a major issue when constructing proportional symbol maps, but every "good" map will contain at least some overlapping symbols (see the discussion in [13]).

In principle any two- or three-dimensional shape can be used as a symbol on a proportional symbol map. However, circles (transparent) and disks (opaque) are used most frequently, since they are visually stable, they conserve map space, and users do prefer them. Also squares and triangles are occasionally seen. Although opaque symbols obscure each other and also the map below them, users indicate a preference for opaque symbols [8].

Clearly there are many different ways to arrange opaque symbols with respect to each other and any choice of (partial) order makes some symbols more visible than others. In this paper we address the algorithmic question how to arrange a given set of overlapping disks or squares such that all of them can be seen "as well as possible".

Definitions and notation. Before we can formally state the problem we first introduce some definitions and notation. To simplify the presentation we give all definitions for disks, but they naturally extend to opaque squares. Let S be a set of n disks $D_1, \ldots, D_n$ in the plane. We denote by $\mathcal{S}$ the arrangement formed by the boundaries of the disks in S. A *drawing* $\mathcal{D}$ of S is a sub-arrangement of $\mathcal{S}$ which is drawn on top of the filled interiors of the disks in S.

Not every drawing is suitable for the use on a proportional symbol map. A suitable drawing needs to include at least the boundary of the union of the disks in S. It should be locally correct at the vertices: every vertex v of the drawing is formed by the intersection of the boundaries of two disks D_i and D_j; a drawing

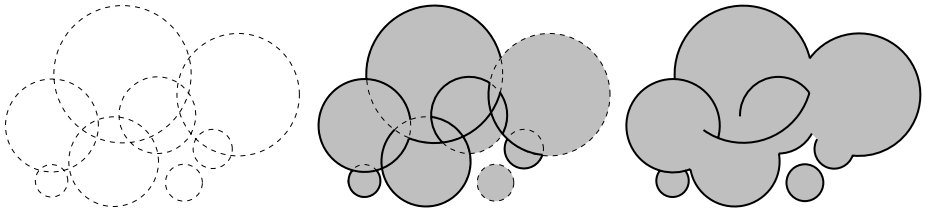

Fig. 2. An arrangement $\mathcal{S}$, a drawing with $\mathcal{S}$ visible, and a bounded drawing.

is locally correct at v if it corresponds locally around v to stacking D_i onto D_j or vice versa. Furthermore, a suitable drawing must have only correct faces: a face of the drawing is correct if there is an order in which all disks in S that contain the face can be drawn on top of each other such that the face appears. We call drawings that satisfy these conditions *face correct*.

Figure 3 shows that even a face correct drawing can still have an "Escher-like" quality which we would like to avoid on a proportional symbol map. Hence we need to enforce even stronger requirements on what constitutes a proper drawing. We consider two types of drawings.

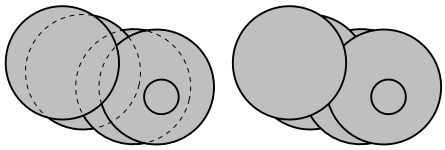

Fig. 3. A face correct drawing shown with and without $\mathcal{S}$ visible.

Physically realizable drawings. A face correct drawing is physically realizable if and only if for every face f of the arrangement $\mathcal{S}$ there exists a total order on the disks in S_f (the disks in S that contain f) such that the topmost disk is visible and the orders associated with any two faces of $\mathcal{S}$ do not conflict. That is, the order in which the disks in S are stacked upon each other is uniquely determined at every face of $\mathcal{S}$ and no two of such orders conflict. In particular, any two or more disks that have a common intersection have a unique ordering.

We can show that this definition is in fact equivalent to the following. We associate a *pr-disk* D'_i with every disk D_i in S. D'_i is a surface patch that is the image of a continuous function of the points in the input disk, that is, $(x, y) \in D_i$ maps to $(x, y, f_i(x, y))$ where $f_i(\cdot, \cdot)$ is continuous. The boundary of D'_i is a closed curve that lies in a cylinder erected vertically up on the boundary of D_i. A drawing $\mathcal{D}$ is physically realizable if functions $f_1, \ldots, f_n$ exist so that the pr-disks $D'_1, \ldots, D'_n$ are disjoint and the view vertically down from infinity is $\mathcal{D}$. That is, if we imagine that we are working with actual physical disks then we are allowed to warp them in a "Dali-like" fashion, but we cannot cut them.

Stacking drawings. A stacking drawing is a natural restriction of a physically realizable drawing and also the one most frequently found on proportional symbol maps. A physically realizable drawing $\mathcal{D}$ is a stacking drawing if there exists a total order on the disks in S such that $\mathcal{D}$ is the result of stacking the disks in this order.

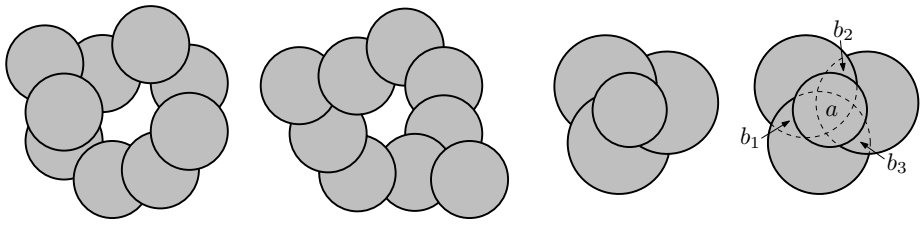

Fig. 4. A stacking drawing (left), a physically realizable drawing that is not a stacking drawing (middle), a drawing that may seem physically realizable, but is not—any order for face a will conflict with one of b_1, b_2, or b_3 (right).

Quality of a drawing. Intuitively, a good drawing should enable the viewer to see at least some part of all symbols and to judge their sizes as correctly as possible. The accuracy with which the size of a symbol can be judged is proportional to the portion of its boundary that is visible. This leads us to the following two optimization problems. Assume that we are given a set S of n opaque symbols $S_1, \ldots, S_n$.

Max-Min: Find a physically realizable or a stacking drawing that maximizes the minimum visible boundary length of each symbol, that is, $\max \min_{1 \leq i \leq n} \{\text{visible length of the boundary of } S_i\}$.

Max-Total: Find a physically realizable or a stacking drawing that maximizes the total visible boundary length over all symbols.

Figure 5 illustrates why we consider only visible boundary length and not visible area of symbols. The boundary of the center disk is completely covered but a significant part of its area is still visible. It is, however, impossible to judge its size or to determine the location of its center. Figure 6 shows that a stacking drawing can be arbitrarily much worse than a physically realizable drawing with respect to the Max-Min problem. At least half of the boundary of every disk in Figure 6 (left) is visible, whereas the lowest disk in any stacking drawing is covered by its two neighbors and hence has only a very short visible boundary.

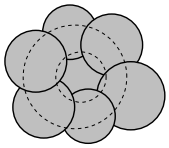

Fig. 5. Visible perimeter is more important than visible area.

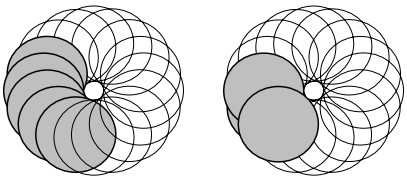

Fig. 6. An optimal physically realizable drawing (left), an optimal stacking drawing for the same disks (right).

Formal problem statement. Assume that we are given a set S of n disks or opaque squares that overlap. Construct a physically realizable drawing or a stacking drawing for the elements of S that either maximizes the minimum visible boundary of each symbol (Max-Min) or maximizes the total visible boundary over all symbols (Max-Total).

Results. We show in Section 2 that for physically realizable drawings both the Max-Min and the Max-Total problems are NP-hard. For stacking drawings the Max-Min problem can be solved in $O(n^2 \log n)$ time. If the symbols are disks and have the property that no point in the plane is covered by more than $O(1)$ disks, then it can be solved in $O(n \log n)$ time. If the symbols are unit-size squares it can be solved in $O(n \log^2 n)$ time. These algorithmic results are presented in Section 3. The status of the Max-Total problem for stacking drawings is open. We performed experiments to compare the results of four different methods that compute a stacking drawing. One of these is our solution to the Max-Min problem, and this one performs best on our data sets. These results are presented with various figures in Section 4.

2 NP-hardness

We show that the Max-Min and the Max-Total problem are NP-hard for physically realizable drawings. It is better for our discussion to use the standard RAM model of computation and consider disks or squares with integer radius and centers located at integer coordinates. In this model, the visible boundary of a set of squares is easily computable, since it involves only integer numbers. However, for problems involving disks, the visible perimeter is a sum of the lengths of circular arcs, and therefore, it is unclear if they belong to the class NP. This is not surprising, since many basic geometric problems are not known to be in NP [4, 6]. Both reductions are from planar 3-SAT, which was proved NP-hard by Lichtenstein [10]. Since the ideas are standard and often used (see for example [1, 7, 9]), our discussion concentrates mostly on the gadgets.

Theorem 1. *It is NP-hard to decide if a given collection of congruent disks has a physically realizable drawing where at least some given length of the perimeter of each disk is visible.*

Proof. We sketch a construction with disks of perimeter 1 and radius $1/2\pi$, such that it is NP-hard to decide if there is a physically realizable drawing with at least 3/4 of each disk's boundary visible. We explain in the full paper how an equivalent construction can be made in polynomial time in the RAM model.

A Boolean variable x_i is represented by an even cycle of disks, as shown in Figure 7. We say that two disks overlap for a fraction f if a fraction f of the boundary of one disk is covered by the other disk. Any two adjacent disks overlap for 1/8 or 1/4, such that any disk overlaps for 1/8 with one neighbor and for 1/4 with the other neighbor. Hence, to achieve that each disk has 3/4 of its perimeter visible, the cycle must be either clockwise overlapping (signifying that x_i is TRUE) or counterclockwise overlapping (signifying that x_i is FALSE).

In the TRUE state, every second disk has 1/4 of its boundary covered by the next disk in the cycle—if 3/4 of the boundary of each of these disks is to remain visible, no more disks (other than those in the cycle) must cover them. For the other half of the disks in the cycle, only 1/8 of their boundaries are covered and disks outside the cycle may cover another fraction 1/8 of them. In the FALSE state, it is precisely the other set of disks that can be covered for another 1/8.

A channel for x_i may start at a disk that has 1/8 overlap with a disk in the cycle that can take another 1/8 overlap in the TRUE state. A channel for $\overline{x_i}$ may

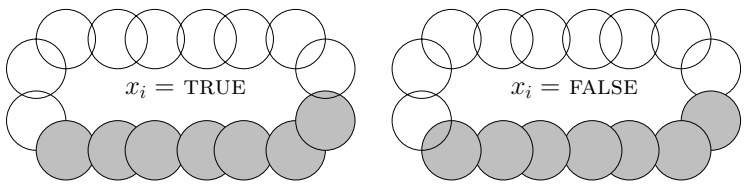

Fig. 7. Representation of a Boolean variable and its TRUE and FALSE states.

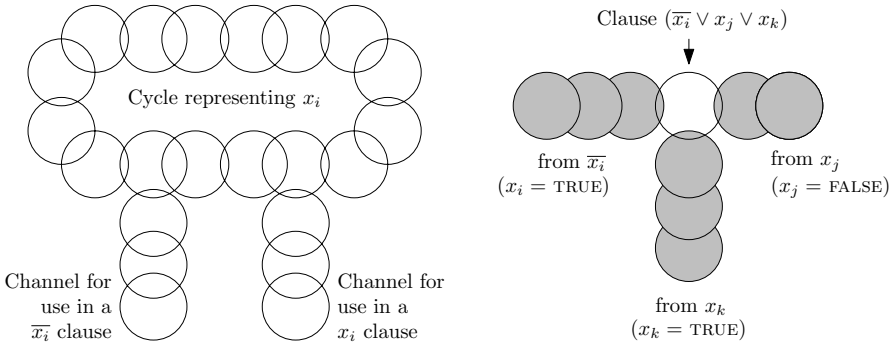

Fig. 8. Representation of a clause.

start at a disk that has 1/8 overlap with a disk in the cycle that can take another 1/8 overlap in the FALSE state. If the variable has enough disks in its cycle, then any number of channels can be connected and in any order for x_i and $\overline{x_i}$.

At a clause like $(\overline{x}_i \vee x_j \vee x_k)$, the channels for $\overline{x}_i$, x_j, and x_k come close and are connected with a single disk that has overlap 1/8 with each of the last disks of the channel, see Figure 8. For the clause disk to be uncovered for at least 3/4, at least one of the three channels must represent TRUE, and the last disk of that channel can go under the clause disk.

The details of the construction are easily filled in. The disks have a physical realization with a free perimeter of at least 3/4 if and only if the planar 3-SAT formula is satisfiable, and only a polynomial number of disks are needed in the reduction. Moreover, note that if it is not possible to achieve a free perimeter of at least 3/4 in all disks, then some disk has free perimeter of exactly 5/8. □

Note that in the proof, no point in the plane is contained in more than two disks. Furthermore, the decision whether a physically realizable drawing exists with free perimeter strictly between some values A and B is equally difficult as the decision whether a physically realizable drawing exists with free perimeter at least A, for some constants A and B. This shows that we cannot expect to find an approximation algorithm that finds a solution better than a constant factor from the optimum in polynomial time. In the given construction, with an appropriate scaling, the constant can be brought arbitrarily close to 6/5, but with some fine-tuning it can be raised.

A reduction for Max-Total when the input consists of squares can be found in the full version of this paper. Here we only state the result.

Theorem 2. *It is NP-complete to decide if a given collection of bounded-size squares has a physically realizable drawing whose visible perimeter is at least a given value T.*

Our construction uses coordinates and squares of polynomial size. Therefore, there is no fully polynomial time approximation scheme (FPTAS) for the optimization problem, unless P=NP.

3 Algorithms

We can compute the stacking order that maximizes the minimum of the visible boundary in polynomial time. We present the algorithms in this section. We first give a general algorithm for disks, then we deal with special cases and squares.

The general idea to compute a stacking order of n disks that maximizes the minimum of the boundary length uncovered is simple: for each disk, we determine how much boundary would be seen if it were the bottommost disk. We choose that disk with the maximum value, make it the bottommost disk, and then recurse on the $n-1$ remaining disks. To implement this greedy approach efficiently, we maintain for each disk a data structure that represents all of its uncovered boundary intervals. For technical reasons, we consider a disk boundary c_i to be an interval from its topmost point clockwise around. Any other disk d_j intersects c_i in zero, one or two intervals (two if d_j contains the topmost point of c_i). All intersection points on c_i define a set of elementary intervals between two consecutive intersection points. The data structure T_i that stores c_i is a variation of a segment tree that stores the elementary intervals in its leaves. An internal node ν corresponds to an interval $\mathit{int}(\nu)$ that is the union of elementary intervals below it in T_i. (See [3] for a detailed description of segment trees.)

Every node (internal and leaf) stores the boundary length of $\mathit{int}(\nu)$ and a counter that stores the number of other disks that contain $\mathit{int}(\nu)$, but not $\mathit{int}(\text{parent}(\nu))$. It also stores an interval *vis-int*(ν) that is the visible boundary length of $\mathit{int}(\nu)$ that would remain if only the disk intervals of other disks that occur in the subtree rooted at ν would hide parts of $\mathit{int}(\nu)$ from view. Disk intervals at ancestors of ν may still cause that no part of $\mathit{int}(\nu)$ is actually visible. The root of T_i stores the total perimeter length of d_i if it were placed bottommost in *vis-int*$(\text{root}(T_i))$.

Initially, we construct a segment tree T_i for each disk d_i, storing the disk intervals for all disks d_j with $j \neq i$. By inspecting *vis-int*(root) for all trees $T_1, \ldots, T_n$, we determine the one with the largest boundary length if it were bottommost, and select it. When a disk d_j is chosen, we delete the disk interval of d_j from all structures T_i of disks d_i that intersect d_j and were not yet chosen. To this end, we find the canonical nodes of the disk interval of d_j in T_i. For each canonical node ν, we lower the counter. When the counter becomes 0, we also update *vis-int*(ν) by taking the *vis-int*(..) values of the two children of ν, and adding their values. By the standard analysis of segment trees and tree augmentation, deletion of a disk interval from T_i takes $O(\log n)$ time. Therefore, the process of choosing a disk to be placed bottommost, and updating all trees takes $O(n \log n)$ time.

Theorem 3. *Given n disks in the plane, a stacking order maximizing the boundary length of the disk that is least visible can be computed in $O(n^2 \log n)$ time.*

If no point in the plane is contained in more than C disks, where C is some constant, and the ratio in size of the largest and smallest disk is also a constant, then we can prove with a packing argument that any disk intersects only a constant number of other disks. The whole arrangement of disk boundaries has

complexity $O(n)$, and for each disk d_i, we find the ones that intersect its boundary in constant time by walking in the arrangement around the boundary of d_i and inside it. We store all disks in a priority queue, sorted on visible boundary length. This allows us to select the next bottommost disk in $O(1)$ time, find the intersecting disks in $O(1)$ time as well, recompute their visible boundary length in $O(1)$ time, and recompute their position in the priority queue in $O(\log n)$ time. In this way, the whole algorithm takes only $O(n \log n)$ time overall.

If we do not assume that the ratio in size of the smallest and largest disks are bounded, we can prove the same result. A disk may now intersect many more than $O(1)$ disks, but the arrangement still has $O(n)$ complexity, and hence all traversals to find the disks that intersect a selected disk take only $O(n)$ time in total. This implies that in total, only $O(n)$ visible boundary lengths are recomputed. We need to take care that this can be done efficiently for disks that intersect many other disks. To this end, we use the segment tree given above for the general algorithm, and we again obtain an $O(n \log n)$ time algorithm (note that all segment trees together have size $O(n)$ in this case).

Theorem 4. *Given a set of n disks in the plane such that no point is contained in more than $O(1)$ disks, a stacking order that maximizes the boundary length of the disk that is least visible can be computed in $O(n \log n)$ time.*

For unit squares we can give an $O(n \log^2 n)$ time algorithm without the assumption that any point is covered by only a constant number of squares. So the arrangement of squares may have quadratic complexity. We first compute the union of all squares, and determine for each square the visible boundary length that it contributes to the boundary of the union. We store the squares in a priority queue. Note that any square has at most one visible interval on each side. We select the next bottommost square S from the priority queue and update the union of squares explicitly. Up to four segments disappear, but possibly many more appear, see Figure 9. We find these by repeated ray shooting. Up to eight squares have a visible interval enlarged because they ended on a side of S. All other squares that have a change in their visible interval have a vertex exposed on the contour, or have an interval on a side exposed for the first time; these cases can arise at most four times for each square. Hence, the total change in intervals is $O(n)$ throughout the whole process.

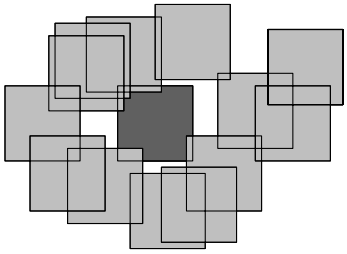

Fig. 9. Deletion of a square from the union of squares.

We preprocess all vertical sides of squares in a semi-dynamic data structure for horizontal ray shooting. Similarly, we preprocess all horizontal sides of squares in a semi-dynamic data structure for vertical ray shooting. Using an augmented segment tree we can implement this to run in $O(n \log^2 n)$ time overall.

Theorem 5. *Given a set of n unit squares in the plane, a stacking order that maximizes the visible boundary of the square that is least visible can be computed in $O(n \log^2 n)$ time.*

4 Experiments

We have examined stacking orders based on different methods experimentally. We first describe our data sets and then the stacking methods, followed by an evaluation of their quality.

Data sets. In principle there are three different types of data sets. Either all disks have the same size, or disk sizes are taken from a small number of classes, or the disk sizes are all different. Equal size disks are uncommon because they do not show a value with locations, only an occurrence. We therefore omit such data sets from our experiments.

We used several different data sets; see the table. First, we took two data sets with the cities of the USA, namely the 156 and the 538 largest ones by population. The area of each disk is proportional to the population of the city. Two other data sets consist of 602 disks corresponding to earthquakes in the world. Disks are centered at the epicenter and the areas of the disks are proportional to the magnitude (scale of Richter) and to the death count [11]. Second, we used versions of the 156 cities and the earthquake magnitudes where disk sizes were classified into five different classes.

Test data	number of disks	smallest radius	largest radius
City 156	156	1.428	12
City 538	538	0.279	6
Earthquake magnitude	602	8.075	10
Earthquake death count	602	0.123	100
City 156, classed	156	1.897	6
Earthquake mag., classed	602	4.472	10

Stacking methods. Proportional symbol maps that are published in books or on the internet do not seem to follow any method consistently. Some appear to be stacked from the left to the right, others appear to be random. For maps with differently sized disks, often the stacking order is from large to small (small on top). For disks of arbitrary sizes we compare four different stacking methods.

Left-to-right by center: The disk with leftmost center is put at the bottom of the stacking order, and the remaining disks are stacked recursively on top.

Left-to-right by leftmost: The disk with leftmost left extreme is put at the bottom, with the remaining disks stacked recursively on top.

Large-to-small: The disks are stacked from bottom to top in order of non-increasing radius.

Max-Min: We maximize the visible boundary length of the disk with least visible boundary length, using the greedy approach presented in Section 3.

All the left-to-right methods could of course also be executed from right-to-left with different results. The stacking methods and the results pertaining to the two classed data sets can be found in the full paper.

Evaluation. To evaluate the stacking drawings we measured the visible boundary length of the top-10 of the least visible disks and we measured the total boundary length that is visible. Disks that do not intersect any other disk were

	City 156	City 538	Earthquake magnitude	Earthquake death count
Left-to-right by center	0.00 / 1405	0.00 / 1533	0.00 / 14446	0.00 / 4697
Left-to-right by leftmost	2.14 / 1711	0.44 / 1815	4.96 / 15061	0.66 / 8568
Large-to-small	2.72 / 1730	0.00 / 1809	0.00 / 12126	0.78 / 9049
Max-Min	4.42 / 1759	0.88 / 1868	12.45 / 16608	0.78 / 9016

Table 1. Results on disks of arbitrary sizes, given as *average visible boundary length of top-10 / total visible boundary length*.

excluded from the top-10. For sets of disks with different sizes it makes a difference if 1 cm of the boundary of a small disk or 1 cm of the boundary of a larger disk is visible. This implies a difference in absolute visibility of a disk (in length units) and relative visibility (in percentages). We measured both absolute visibility and relative visibility. Because the comparative performance of the different methods turned out to be roughly the same in both cases, we only show results for absolute visibility and leave results for relative visibility to the full version of this paper.

Table 1 summarizes the results for the four stacking methods for the four unclassed data sets. It is clear that the Max-Min method performs best on the top-10 of least visible disks. The left-to-right by center method performs worst, except for the case where disks have roughly the same size (earthquake magnitudes), where the large-to-small method performs poorly. The same observations hold for data sets that are not shown in this paper (1260 cities, tsunami death counts (39 disks), tsunami height events (33 disks)).

Another important aspect is the visual quality of the resulting map. Since this cannot be measured, user experiments would be needed to evaluate it. In this paper, we only show a few figures for comparison purposes. Figure 10 shows the 156 largest cities of the USA with disk areas proportional to the population using the four different methods (the differences can be seen most clearly in the upper right corners of the maps). The figures correspond to the top four rows of Table 1. It is noticeable that the left-to-right methods produce maps that seem 'unbalanced" or 'asymmetric". A left-to-right structure is visible that has no cartographic meaning. This artifact can be perceived even more clearly on maps where the disk sizes vary less (not shown here).

The Max-Min method has a higher computational cost ($O(n^2 \log n)$ time) than the simple left-to-right or large-to-small methods, which require only sorting. The implementation effort is also significantly higher for the Max-Min method. However, it scores better than the other methods according to Table 1, especially for the least visible disks. Furthermore, for sets of disks with not too much difference in size, the Max-Min method is better because it does not have visual artifacts like the left-to-right methods, and it clearly outperforms the large-to-small method on visible perimeter.

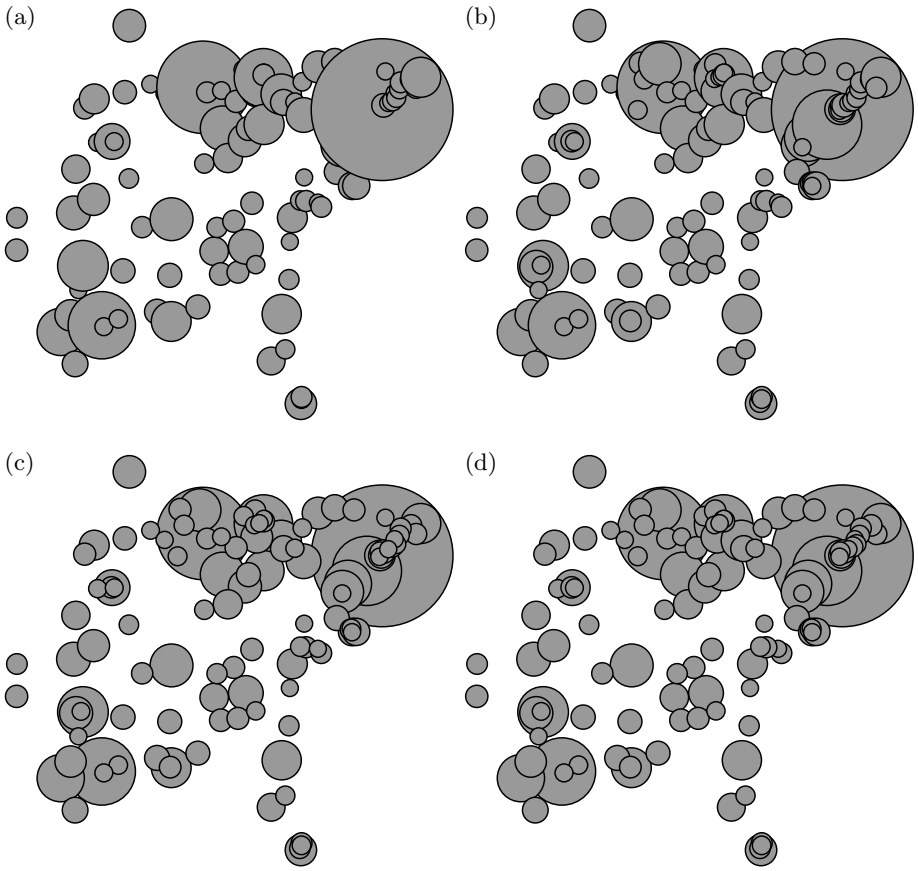

Fig. 10. USA, 156 biggest cities (only showing half of the map), stacked: (a) left-to-right by center; (b) left-to-right by leftmost; (c) large-to-small; (d) Max-Min method.

5 Conclusions and open problems

We described an algorithm that solves the Max-Min problem for stacking drawings in $O(n^2 \log n)$ time. In our experiments, comparing this algorithm with three heuristics, we found that our method performed best on our test data. However, we did not experiment with methods that compute physically realizable drawings, and it is unclear how they would perform in comparison with our methods for stacking drawings. Solving the Max-Min problem (or the Max-Total problem) for physically realizable drawings is NP-hard, and developing good heuristics for such drawings is not trivial.

Among the open problems that remain are the computation of optimal Max-Min stacking drawings in $o(n^2 \log n)$ time, the computation of optimal Max-Total stacking drawings (or approximations thereof) in polynomial time, and the development of approximation algorithms for physically realizable drawings.

Acknowledgement. The authors thank Christian Vossers for implementing the methods and running the experiments.

References

1. P. K. Agarwal and S. Suri. Surface approximation and geometric partitions. *SIAM J. Comput.*, 27:1016–1035, 1998.
2. K. C. Clarke. *Analytical and Computer Cartography*. Prentice Hall, Englewood Cliffs, NJ, 2nd edition, 1995.
3. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 2nd edition, 2000.
4. E. D. Demaine, J. S. B. Mitchell, and J. O'Rourke. The open problems project. Problem 33. `http://maven.smith.edu/ orourke/TOPP/P33.html`.
5. B. Dent. *Cartography - thematic map design*. McGraw-Hill, 5th edition, 1999.
6. L. Fortnow. Computational Complexity Blog. Post of Friday, February 14, 2003. `http://weblog.fortnow.com/archive/2003_02_01_archive.html`.
7. R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Inform. Process. Lett.*, 12(3):133–137, 1981.
8. T. Griffin. The importance of visual contrast for graduated circles. *Cartography*, 19(1):21–30, 1990.
9. D. E. Knuth and A. Raghunathan. The problem of compatible representatives. *SIAM J. Discrete Math.*, 5:422–427, 1992.
10. D. Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*, 11(2):329–343, 1982.
11. NOAA Satellite and Information Service. National geophysical data center, 2005. `http://www.ngdc.noaa.gov/`.
12. Queensland University Advanced Centre for Earthquake Studies. `http://www.quakes.uq.edu.au`.
13. T. A. Slocum, R. B. McMaster, F. C. Kessler, and H. H. Howard. *Thematic Cartography and Geographic Visualization*. Prentice Hall, 2nd edition, 2003.