Optimizing Active Ranges for Point Selection in Dynamic Maps

N. Schwartges[1], D. Allerkamp[2], J.-H. Haunert[1], A. Wolff[1]

[1]Institution: Chair of Computer Science I, University of Würzburg
http://www1.informatik.uni-wuerzburg.de/en/staff

[2]Institution: Robert Bosch Car Multimedia GmbH, Hildesheim
CM-AI/PJ-CF13, P.O. Box, 31132 Hildesheim, Germany

1. Introduction

Maps for car navigation systems need to display sufficient information to allow their users to orientate themselves, but, since driving requires attention to the road, such maps must not distract the users with too many details. Therefore, the maps need to be generalized. In this paper, we propose algorithms that, given a large set of points of interest (for example, gas stations or parking lots), select a subset of points for display.

Selection is usually considered to be one of a small number of map generalization operators (e.g., one of eight in the classification by Beard and Mackaness (1991)) and has been studied at least since the 1960s when Töpfer and Pillewizer (1966) investigated existing maps to derive guidelines about the number of features a map of a certain scale should contain. For a long time, however, research on map generalization focused on static topographic maps. Instead, we consider generalization for dynamic maps. We choose a constraint-based approach, which is generally considered to be promising for generalization (Weibel and Dutton 1998). In order to avoid that a user loses context, however, we consider a constraint for dynamic maps that keeps the number of changes during zooming small.

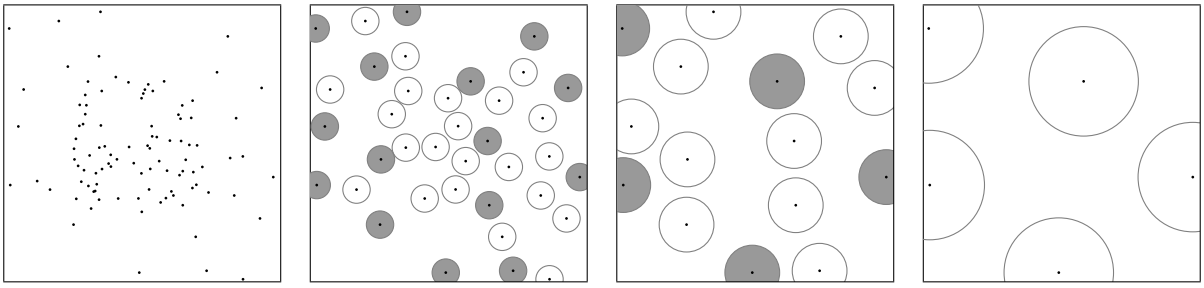
For the selection of point features, we assume that every appearance or disappearance of a point during a zooming operation may distract a user. In the worst case, points are flickering on and off. In order to model the behavior of points during zooming, we consider *active ranges*, a concept that has earlier been introduced in the context of automatic label placement (Been et al. 2010). Recall that a map of scale $1:z$ has *scale factor* z. In our model, the active range of a point p is an interval R_p of scale factors such that, for any scale factor $z \in R_p$, the map at scale $1:z$ displays p. Formally, our problem – *active range optimization* (ARO) – can be stated as follows.

Problem (ARO)**.** *Given a set P of n points in the plane, a constant $d_{\min} > 0$, and the largest scale factor $z_{\max}$ at which the map is displayed, assign active ranges to the points such that*

(C1) the active range R_p of each point $p \in P$ consists of exactly one interval $(0, z_p]$ with $z_p \leq z_{\max}$,

(C2) in a map of scale $1:z$, any two points $\{p,q\} \subseteq P$ whose active ranges contain z have distance at least $d_{\min}$, and

(C3) the overall length of all active ranges is as large as possible.



(a) $z = 0$; 100 points (b) $z = 0.1$; 37 points (c) $z = 0.2$; 15 points (d) $z = 0.4$; 5 points

Figure 1. One of our heuristics (M1) applied to a set of 100 points; one quarter is drawn randomly from the unit square $[0,1]^2$, the rest is drawn from the square $[1/4,3/4]^2$. The circles around the points for $z > 0$ visualize the minimum distance; their diameter is z. We shaded the circles of points that survive from one scale to another.

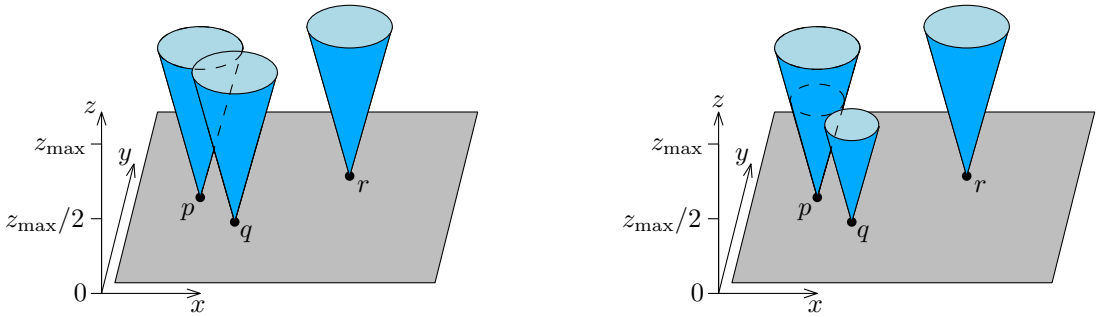
We require (C1) as a hard constraint to avoid that a point *pops*, that is, disappears and reappears during a zooming interaction. To ensure that all points are visible when zooming in far enough (that is, when the scale approaches infinity) we require $R_p = (0, z_p]$. On the other hand, when we zoom out to a scale smaller than $1 : z_p$, the point p disappears.

Additionally, our problem statement includes a legibility constraint (C2) and a preservation constraint (C3) as most constraint-based models for map generalization do (Burghardt et al. 2007). More precisely, constraint (C2) ensures that the map never contains two points that are too close to each other and thus keeps the map readable; the constant $d_{\min}$ can be thought of as, say, 2 mm on a printed map. Constraint (C3), on the other hand, tries to keep the points in the map. Obviously, if we zoom out to a very small scale, we cannot keep all points visible while satisfying constraint (C2). Therefore, our problem statement defines the preservation constraint to be soft, that is, we maximize the overall lengths of the active ranges.

Note that the active ranges are a data structure that we build for the whole map. We simply query the points to display at run time. Therefore, our approaches are real-time capable.

Our contribution is as follows. We first show how to formulate ARO as a *(mixed) integer linear program (MIP)*; see Section 3. Recall that integer programming is a global optimization technique. In general, solving a MIP is NP-hard, but there exist highly optimized commercial and non-commercial MIP solvers. We have applied integer programming to a generalization problem before, namely to area aggregation. Our article on this work (Haunert and Wolff 2010) includes an introduction to integer programming, which we recommend to readers who are not familiar with this optimization technique.

Mixed-integer programming allows us to solve small problem instances with proof of optimality. This implies that the cartographic quality of the output solely depends on whether or not we have modeled all relevant aspects of the problem appropriately. In particular, the output is *not* influenced by tuning parameters, which are common to most heuristic optimization methods. This has two advantages. First, we can verify the appropriateness of our model based on optimal solutions that we obtain for small instances. Second, such solutions allow us to assess the quality offered by more efficient heuristic methods. This helps us choose a good heuristic for large problem instances. In fact, to solve ARO efficiently (not necessarily optimally) we also present a new heuristic, the *growing-cones heuristic*, with two variants; see Section 4 and Figure 1 for an example output. The heuristic exploits that the Delaunay triangulation of the given



(a) an infeasible solution (cones c_p and c_q intersect) (b) a feasible solution obtained by truncating c_q

Figure 2. ARO can be interpreted as a labeling problem. Each label is a disk that is centered on its point. Interpreting the scale as the third dimension (in addition to the two spatial dimensions), labels become cones. In a feasible labeling, no two cones intersect.

point set contains, at any time, an edge between a closest pair. Finally, we compare solutions of our heuristic to those of the MIP and a heuristic of Been et al. (2006); see Section 5. We first review related work on map generalization and dynamic label placement; see Section 2.

2. Previous and Related Work

Computer programs for map viewing usually allow users to zoom in and out and automatically adjust the level of detail (LoD) of the map to the user-selected scale. A simple way of adjusting the LoD is to maintain multiple data layers of different LoDs; at certain scales, the software simply exchanges the current map with a less or a more detailed map. A problem with this approach is, however, that the map abruptly changes at certain scales and the user may lose context. For this reason, researchers have proposed methods of continuous generalization, which, for example, may continuously transform a polyline into a simplified polyline (van Kreveld 2001, Cecconi and Galanda 2002). While line simplification may be animated as a continuous process, other changes occurring during map generalization are inherently discrete, for example, aggregation of areas and selection. Still, it is possible to keep the discrete changes small. In the GAP-tree approach of Van Oosterom (1995), for example, areas are aggregated by selecting only one area at a time and assigning the selected area to one of its neighbors. The result of the iterative merging process is stored in a data structure (the GAP-tree) that allows a user to request a map of an arbitrary scale.

The idea of active ranges for point objects, which we apply in this paper, is similar to the GAP-tree approach in the sense that the generalization needs to be computed only once. The result may be stored in a database, from which a user can easily retrieve all points whose active ranges contain a specified scale. Interestingly, the problem ARO is closely related to a dynamic label placement problem. To see why, we observe that keeping a minimum distance $d_{\min}$ between two points (Constraint (C2)) is equivalent to drawing a disk of radius $d_{\min}/2$ around the two points and requiring that the disks for the two points do not overlap. Therefore, we can interpret ARO as a dynamic label placement problem, in which disk-shaped labels have to be placed centered on given point objects. Figure 2 illustrates this interpretation of the problem.

Been et al. (2006) introduced the notion of consistency for labeling dynamic maps and presented a data structure for displaying consistently labeled maps under panning and zooming

interactions. In order to avoid jumping labels they simply insisted that the position of a label remains the same over all scales. This allowed them to use a data structure that represents each label by a pyramid whose apex coincides with the reference point at $z = 0$. Their objective was to compute a set of pairwise (interior-) disjoint frusta, at most one per pyramid, whose total height is maximized.

Using the data structure of Been et al. (2006), the labeling during or after a user interaction can simply be obtained by intersecting the precomputed set of frusta with the horizontal rectangle that corresponds to the part of the map that the user currently sees. Hence, their data structure allows for real-time interaction. Been et al. presented a simple heuristic for consistent map labeling and showed that the problem is NP-hard in the more general case where the frusta are restricted not to pyramids, but to frusta (which means that users can specify that labels are visible only within a certain scale interval).

Been et al. (2010) refined this work from a more theoretical point of view. They showed that the problem remains NP-hard in the original frusta-in-pyramids setting. Their proof carries over to the case of disk-shaped labels (cones in scale space), which is exactly the version of ARO that we consider. They also presented two algorithms for ARO with respect to congruent square cones.

The first algorithm has an approximation ratio of 4, which means that the set of frusta it computes has total height at least $OPT/4$, where OPT is the total height of an optimal solution (which is NP-hard to compute). The algorithm runs in $O((k+n)\log^3 n)$ time, where n is the number of reference points and k is the number of pairs of intersecting cones. When applied to our version of ARO (disk-shaped labels, which are cones in scale space), the algorithm has an approximation ratio of 5. This is due to the fact that a disk can intersect at most five disjoint disks of the same size. The second algorithm has an approximation ratio of $(4+\varepsilon)$ for any $\varepsilon > 0$ and runs in $O(n \log n \cdot \log(n/\varepsilon)/\varepsilon)$ time. The second algorithm is faster than the first if the number of pairs of intersecting cones is large (say $k = \Theta(n^2)$) and ε is large (say $\varepsilon = 1$).

While Been et al. (2006, 2010) focussed on the interactions panning and zooming, Gemsa et al. (2011a) considered rotation, thus complementing the work of Been et al. As Been et al., they compute, for each given point, an active range (now an interval in $[0^\circ, 360^\circ)$) such that, when rotating the map by an angle α, the (axis-parallel congruent square) labels of all points whose range contains α are pairwise disjoint. They show that, on the one hand, this version of ARO is NP-hard; on the other hand, near-optimal solution can be computed efficiently.

Gemsa et al. (2011b) introduced a 1d-version of ARO where the input points lie on the same horizontal line. The lower edge of a label can slide along its point. Labels must not overlap. The aim is to select a single position for every label (which is then fixed over all scales) so that the sum of the lengths of the active ranges is maximized. The authors solve the problem optimally for the case that every label can be placed at a given number m of discrete positions; their algorithm runs in in $O(n^3 m^3)$ time. For the case that labels can touch their points arbitrarily, they present, for any $\varepsilon > 0$, a $(1+\varepsilon)$-approximation algorithm that runs in $O(n^3/\varepsilon^3)$ time.

Nöllenburg et al. (2010) investigated panning and zooming, but for boundary (or rather margin) labeling, where labels are placed in the margin of the map and then connected to the given points by polygonal lines, so-called *leaders*. They use rectilinear leaders with at most one bend. Given an interval of scales, they compute, for each scale, label and leader positions such that the total leader length is minimized. Their algorithm is efficient and allows for real-time interaction.

Gerrits (2013) dealt with the problem of labeling dynamic point sets as a part of his dissertation. On the one hand, he showed that all natural optimization problems for dynamic point

labeling are PSPACE-hard, this is, they are at least as hard as NP-hard problems. On the other hand, he formulated the new objective of *free-label maximization* where the goal is to maximize the number of non-overlapping labels. For the static case and unit-square labels, he developed an $O(n \log n)$-time algorithm whose approximation ratio is between 7 and 32, depending on the number of allowed label positions. He used this approach as a subroutine for labeling dynamic point sets, this is, sets in which points can be added and removed, and points can move on continuous trajectories. As he could not prove an approximation ratio, he did experiments to show that his algorithm works well in practice.

3. A Mixed-Integer Linear Program for ARO

The NP-hardness of ARO (Been et al. 2010) justifies the application of a mixed-integer linear program (MIP). Linear programming is a technique for describing an optimization problem using linear terms only. It consists of variables, an object function, and constraints. In a MIP, some variables are continuous, some are integer-valued.

Our MIP formulation is as follows. We introduce, for each $p \in P$, a continuous variable $z_p \in (0, z_{\max}]$ that encodes the upper bound of the active range of p. Now, according to (C3), our objective is to maximize $\sum_{p \in P} z_p$. Further, for each pair $\{p,q\} \subseteq P$, we introduce the binary variable y_{pq} which is 1 if $z_q \leq z_p$ and 0 otherwise. If $z_q \leq z_p$, then the distance $d_{z_q}(p,q)$ of p and q in a map of scale $1 : z_q$ is at least $d_{\min}$. Clearly, $d_{z_q}(p,q) = d_{\text{world}}(p,q)/z_q$, where $d_{\text{world}}(p,q)$ is the distance of p and q in world coordinates. This yields $z_q \leq d_{\text{world}}(p,q)/d_{\min}$. The implication "$z_q \leq z_p \;\Rightarrow\; z_q \leq d_{\text{world}}(p,q)/d_{\min}$" can be expressed by the following two constraints, which we introduce for each pair $\{p,q\} \subseteq P$:

$$z_p \leq \frac{d_{\text{world}}(p,q)}{d_{\min}} + z_{\max} \cdot y_{pq} \;\text{ and }\; z_q \leq \frac{d_{\text{world}}(p,q)}{d_{\min}} + z_{\max} \cdot (1 - y_{pq}).$$

Note that the first constraint does not have any effect if $y_{pq} = 1$ (since it holds anyway that $z_p \leq z_{\max}$). Similarly, the second constraint does not have any effect if $y_{pq} = 0$. Further note that the constraints are linear since $d_{\text{world}}(p,q)$, $d_{\min}$, and $z_{\max}$ are constants (that is, not variables) from the point of view of the MIP. This concludes the presentation of our MIP.

4. Greedy Algorithms

Due to the unpredictable running time of a MIP solver, we present two simple heuristics. Both are efficient; the results of the first are at most by a factor of 5 worse than the optimum (which is NP-hard to compute), see the argument in Section 2. We can use these heuristics for computations that rather aim for a fast computation than an exact computation, e.g., for processing large data sets or for applications with strong performance requirements.

4.1 Shrinking Cones

Algorithm 1 was introduced by Been et al. (2010). The authors used it for square labels (in scale space: square pyramids instead of cones). Recall that z_p is the upper bound of the active range R_p of the point p. Let c_p be the cone in scale space with apex p whose base is a horizontal circle of radius $d_{\min} \cdot z_p$ around the projection of p to the plane $z = z_p$; see Figure 2a.

The idea of this algorithm is to start with cones of height $z_{\max}$. These, of course, may overlap.

Algorithm 1: ShrinkingCones(P)

Input: point set P
Output: for each point $p \in P$, the upper bound z_p of its active range

let $\mathscr{D}$ be a dynamic data structure that stores cones according to their height
foreach $p \in P$ **do**
    $z_p \leftarrow z_{\max}$
    $\mathscr{D}$.insert(c_p)
while $\mathscr{D} \neq \emptyset$ **do**
    $c_p = \mathscr{D}$.extractMax()
    foreach cone $c_q \neq c_p$ that overlaps c_p **do**
        truncate c_q such that $c_p \cap c_q = \emptyset$ (by setting $z_q = d_{\text{world}}(p,q)/d_{\min}$; see Fig. 2b)
        $\mathscr{D}$.decreaseKey(c_q)

Therefore, step by step, we choose a cone c_p of maximum height that we have not yet fixed and shrink all the cones that overlap c_p. Now, we consider c_p as fixed and part of our solution.

When using a conflict graph to maintain pairs of intersecting cones and a heap that stores cones according to their height (data structure $\mathscr{D}$ in Algorithm 1), the algorithm runs in $O((k+n)\log n)$ time, where $n = |P|$ and k is the number of pairs of intersecting cones (of height $z_{\max}$). This results from an adjustment of the Bentley–Ottmann sweep line algorithm for line segments (de Berg et al. 2008) to the case of circular arcs.

Figure 3 shows the output of Algorithm 1 applied to the real world instance in Figure 4 (top).

4.2 Growing Cones

The idea of the second algorithm is to grow cones. We start with $z = 0$, i.e., initially, each point is visible. Then, we repeatedly search for the smallest scale factor $z' > z$ at which two points get too close. We remove one of the two points forming the closest pair, say p^*, by setting $z_{p^*} = z'$. We set $z = z'$ and continue until z reaches $z_{\max}$.

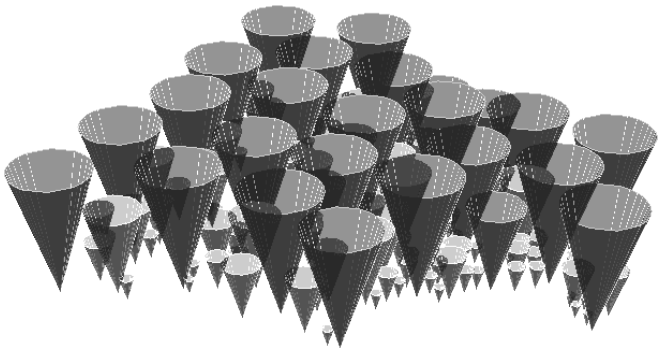


Figure 3. 3d view of the result of the shrinking-cones method applied to the real-world instance in Figure 4 (top). We added a transparency of 20% for cones whose active range is at least 70% of the maximum range.

We have implemented two methods for determining the point p^* to be deleted. Let p and q be a closest pair. The first method, M0, chooses one of these points arbitrarily. The second method, M1, considers the closest neighbors of p and q. It chooses p if the distance of p and its closest neighbor (other than q) is smaller than the distance of q and its closest neighbor (other than p).

Recall that the closest pair of a point set forms an edge in the Delaunay triangulation. Therefore, we use the Delaunay triangulation to maintain the closest pair. Since the Delaunay trian-

gulation has a linear number of edges, this immediately yields an implementation that runs in $O(n^2)$ time for a set of n points. If we make the reasonable assumption that—in the process of deleting points—the average degree of a point involved in a closest pair in the Delaunay triangulation is constant, the running time of the algorithm reduces to $O(n \log n)$ if we use a heap or a balanced binary tree to maintain the lengths of the edges in the Delaunay triangulation.

5. Experiments

We now compare the results of our algorithms to those of the MIP. We implemented all the above-mentioned algorithms in C++. For solving the linear program, we used Gurobi 5.5 (2013). For the Delaunay triangulation, we used CGAL 4.1 (2012). We executed our experiments on a Windows 7 system with a 2.3-GHz Intel quad-core processor and 4 GB RAM. We applied the Microsoft Visual Studio 2010 Professional compiler in 32-bit release mode.

We used two types of data sets. The first type is a real-world instance that consists of the locations of 249 weather stations in the central part of the U.S. The second type is synthetic. For this, we picked points uniformly distributed in the unit square $[0,1]^2$.

Figure 4 shows the outputs of our algorithms applied to the real-world data at different scales. Figure 5 shows the performance of our algorithms on the real-world and the synthetic instances. We set $d_{\min} = 600,000$ for the real-world data (this is, a distance of 600 kilometers) and $d_{\min} = 0.2$ for the synthetic data. In both cases, we set $z_{\max} = 1$. For each instance size $n = 25, 50, \ldots, 225, 249$, we randomly chose n points, ran all the algorithms, computed the performance of each algorithm with respect to the MIP, and averaged this number over five trials (one trial for $n = 249$).

When running the MIP solver, we did not wait for optimum solutions but stopped the computation as soon as the solver could prove that the current solution is at most a pre-defined percentage worse than the optimum. We used a gap of 35% since after three days of computation the gap for a single set of 225 real-world points still was 36% (so, we even went without results for $n = 225, 249$ for the real-world data). We denote the MIP solver with this gap MIP35.

The experiments show that the very simple shrinking-cone heuristic does astonishingly well. In terms of quality, its results are even for large point sets at least 95% with respect to MIP35 on both data sets. Although we used a simple brute-force implementation (running in quadratic time), it was by a factor of 100–10,000 faster than the MIP solver with its gap.

The growing-cone heuristic M1 was yet a few percent better than the cone-shrinking heuristic, but also by a factor of up to 10 slower. If quality is important, M1 is certainly the method of choice among the heuristics that we investigated. For the point set of 249 points, M1 took 0.23 seconds on both settings.

6. Weighted Points and Obstacles

The data structure that we use in the shrinking-cones algorithm (Algorithm 1) sorts the cones according to height. If we want to take priorities into account, we first sort by priority and then by height. This way, we first fix more important points. We cannot truncate fixed elements any further. Therefore, the active ranges of important points will tend to be larger than those of unimportant ones. This does not affect the asymptotic running time as the only difference to the original algorithm is the height of the truncated part. The sum over all active ranges will

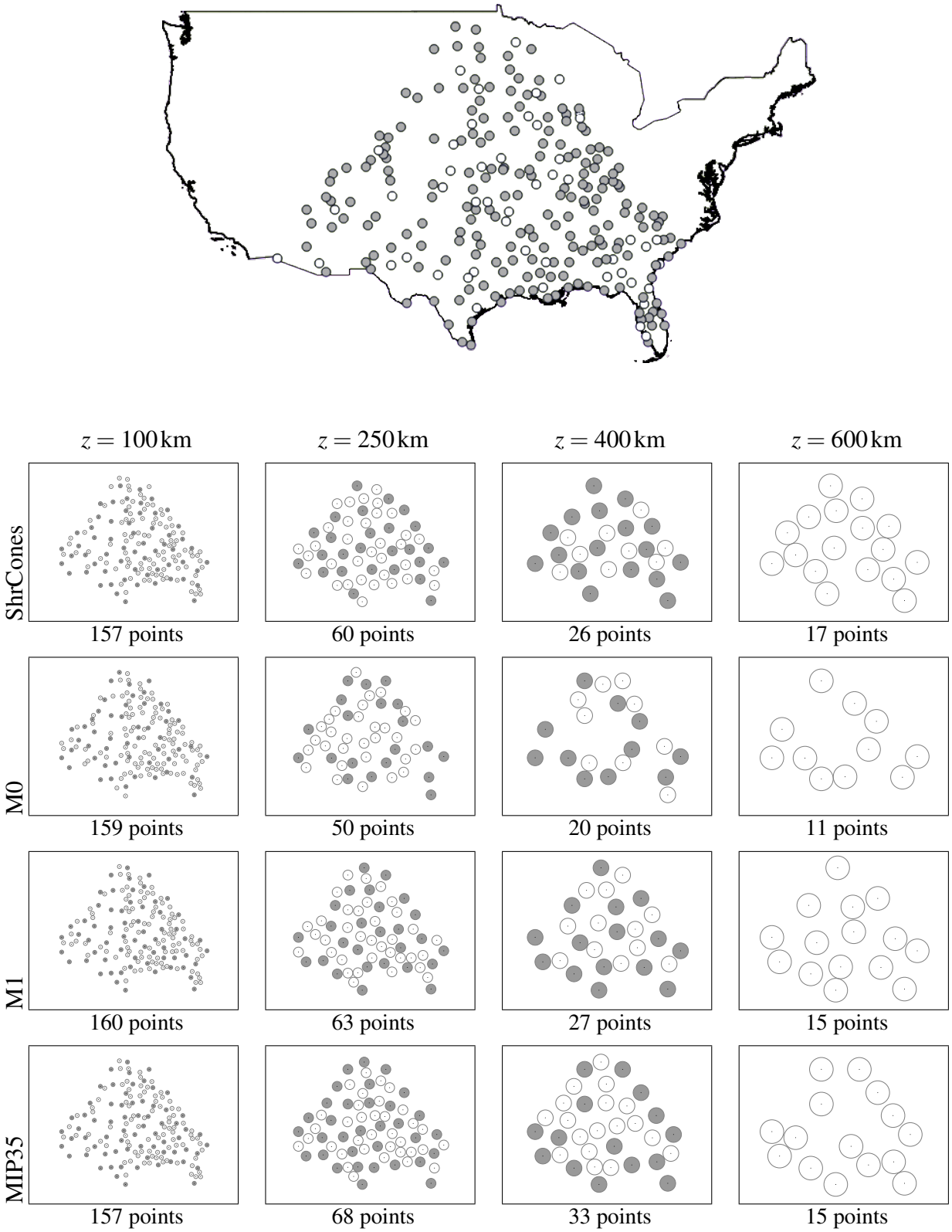
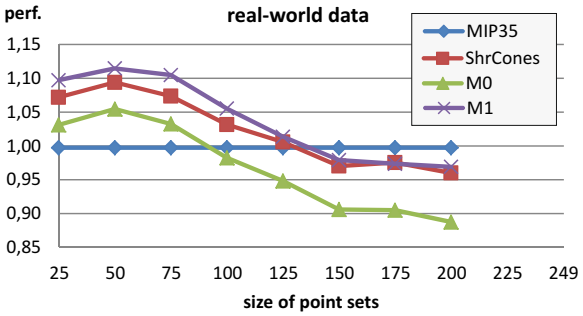
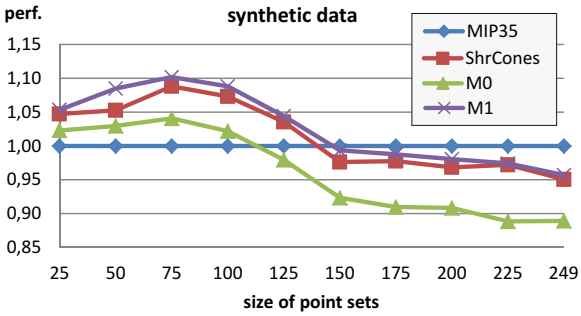


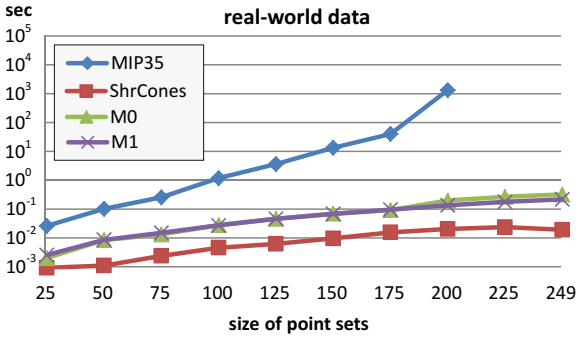
Figure 4. Our real-world instance consists of 249 whether station in the central part of the U.S. (top). We randomly chose 200 points of this instance (shaded gray) and applied all three heuristics and MIP35 to this set. We show the results at different scales (grid). The circles around the points visualize the minimum distance; their diameter is given in the top row of the grid. We shaded the circles that "survived" from one scale to another.



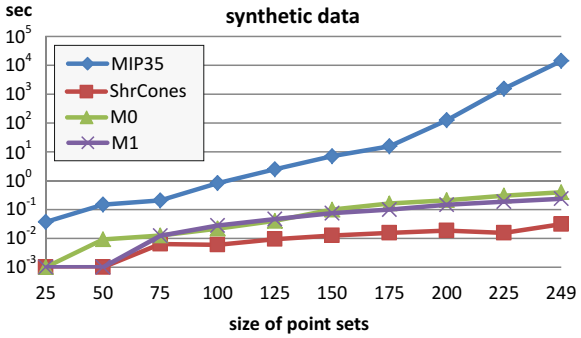
(a) Total active range height relative to MIP35 for the real-world data.



(b) Total active range height relative to MIP35 for the synthetic data.



(c) Absolute running times (in seconds; log-scale!) for the real-world data.



(d) Absolute running times (in seconds; log-scale!) for the synthetic data.

Figure 5. Quality and running time for real-world and synthetic data.

change, though.

We can also adapt the algorithm that grows cones to deal with priorities. Whenever two points get too close, we have to decide which cone keeps growing and which one stops growing. We simply let the more important cone grow further. Obviously, this approach equals M0. Therefore, it has the same asymptotic running time. We expect that the sum of active ranges is smaller than that of M1.

The adaptation for dealing with obstacles is the same for all approaches. We can simply add suitables polyhedra (for example, pyramids) and treat them like cones that we cannot truncate. This will not affect the asymptotic running time.

7. Summary and Conclusion

We introduced different algorithms for ARO in a dynamic map. Our MIP solves the problem optimally, but is too slow for larger point sets. We also introduced fast heuristics.

Concerning future work, we plan to improve the shrinking-cones algorithm; by sorting points first by their distance according to the Delaunay triangulation and, then, by the amount of overlap of the corresponding disks. We also want extend our implementation to priorities.

Acknowledgments. We thank Leon Sering for implementing the MIP and the heuristics M0 and M1.

References

K. Beard and W. Mackaness. Generalization operations and supporting structures. In *Proc. 10th Int. Symp. Computer-Assisted Cartography (AutoCarto 10)*, pages 29–42. ASPRS, 1991.

K. Been, E. Daiches, and C. Yap. Dynamic map labeling. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):773–780, 2006.

K. Been, M. Nöllenburg, S.-H. Poon, and A. Wolff. Optimizing active ranges for consistent dynamic map labeling. *Computational Geometry: Theory and Applications*, 43(3):312–328, 2010.

D. Burghardt, S. Schmid, and J. Stoter. Investigations on cartographic constraint formalisation. In *Proc. 10th ICA Workshop on Generalisation and Multiple Representation*, 2007.

A. Cecconi and M. Galanda. Adaptive zooming in web cartography. *Computer Graphics Forum*, 21(4):787–799, 2002.

CGAL 4.1, Sept. 2012. URL `http://www.cgal.org/`. Accessed Oct. 18, 2012.

M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*, chapter 2. Springer-Verlag, 3rd edition, 2008.

A. Gemsa, M. Nöllenburg, and I. Rutter. Consistent labeling of rotating maps. In F. Dehne, J. Iacono, and J.-R. Sack, editors, *Proc. 12th Int. Symp. Algorithms Data Struct. (WADS'11)*, volume 6844 of *Lecture Notes in Computer Science*, pages 451–462. Springer-Verlag, 2011a.

A. Gemsa, M. Nöllenburg, and I. Rutter. Sliding labels for dynamic point labeling. In *Proc. 23th Canadian Conf. Comput. Geom. (CCCG'11)*, pages 205–210, 2011b.

D. Gerrits. *Pushing and Pulling*. PhD thesis, Faculteit Wiskunde en Informatica, Technische Universiteit Eindhoven, 2013.

Gurobi 5.1, Jan. 2013. URL `http://www.gurobi.com/`. Accessed Aug. 6, 2013.

J.-H. Haunert and A. Wolff. Area aggregation in map generalisation by mixed-integer programming. *International Journal of Geographical Information Science*, 24(12):1871–1897, 2010.

M. Nöllenburg, V. Polishchuk, and M. Sysikaski. Dynamic one-sided boundary labeling. In *Proc. 18th Int. ACM Symp. Advances Geogr. Inform. Syst. (ACM-GIS'10)*, pages 310–319, 2010.

F. Töpfer and W. Pillewizer. The principles of selection. *The Cartographic Journal*, 3(1):10–16, 1966.

M. van Kreveld. Smooth generalization for continuous zooming. In *Proc. 20th Int. Cartogr. Conf. (ICC'01)*, pages 2180–2185, 2001.

P. van Oosterom. The GAP-tree, an approach to "on-the-fly" map generalization of an area partitioning. In *GIS and Generalization, Methodology and Practice*, chapter 9, pages 120–132. Taylor & Francis, 1995.

R. Weibel and G. Dutton. Constraint-based automated map generalization. In *Proc. 8th Int. Symp. Spatial Data Handling (SDH'98)*, pages 214–224, 1998.