

# 2 Decluttering of Icons Based on Aggregation in Mobile Maps

Stefano BURIGAT, Luca CHITTARO

HCI Lab, Department of Math and Computer Science, University of Udine, Italy

**Abstract.** In this chapter, we will deal with cluttering issues associated with the visualization of a high number of icons, representing Points of Interest, on mobile maps. Clutter is a frequently occurring problem that makes it difficult to understand the visualization and has the even more critical effect of masking data. As the user zooms out, the positions of icons increasingly converge until they partially or entirely overlap each other. In regions of high icon density, this effect impairs even the effectiveness of a close-up view. Finding ways to visualize icons in a comprehensible uncluttered fashion is a challenging task. We will first discuss techniques which have been previously proposed in the literature for the automatic placement of labels on maps, a problem which is closely related to the placement of icons. We will then propose an approach for icons - aggregation - aimed at mitigating cluttering issues when icon density is high. The approach is based on an aggregation algorithm to reduce the total number of icons that need to be displayed and on the design of icons that could function as both individuals and aggregates.

## 2.1 Introduction

In recent years, a number of map-based applications and services have been made available to users of mobile devices, with a particular emphasis on Personal Digital Assistants (PDAs). In this context, maps often need to be adapted to specific user requirements which are not known a priori. Therefore, one cannot pre-compute and store all possible maps that may be needed by users but they have to be generated in real-time. This is a particularly challenging task that requires, for example, automatic placement of text labels and icons which are essential parts of any map. Unfortunately, the design of map-based applications is constrained by device limitations: techniques and practices that are effective in the desktop scenario cannot be simply adapted to mobile devices but must be redesigned to achieve usability and performance goals on such equipment and novel solutions are often needed to cope with specific issues (Chittaro, 2006).

In this chapter, we will deal with the specific problem of visualizing a large number of icons, each one representing a Point of Interest (POI) on mobile maps. The visualization of many icons on the same screen often leads to cluttering issues, especially when users perform zoom-out operations and icons begin to touch and overlap each other. This may degrade the effectiveness of even a close-up view of a map and mask other important map features such as roads. For example, displaying the



**Fig. 2.1.** An example of icon cluttering taken from the Yahoo Maps (maps.yahoo.com) site: the displayed icons represent restaurants in the Manhattan area. Many icons overlap and have masked almost all other map features of an area.

location of restaurants as icons on a map might work very well at a large scale, but it could become useless at smaller scales when most icons overlap, as illustrated in Fig. 2.1. Typically, some simple criteria are used to limit the number of icons to be displayed. A widespread but extreme solution lies in displaying only those POIs selected by the user from a list. A slightly more flexible solution subdivides the list of all POIs (ordered by name or other criteria) in pages and displays only one page at a time (consisting of a fixed and usually limited number of POIs). However, these techniques are only suitable for the simultaneous display of map contents and a (mostly) textual list of POIs. They are hardly applicable to the mobile context characterized by limited screen space. Moreover, reducing the number of POIs to be displayed may not solve all overlapping issues, as illustrated in Fig. 2.2.

Finding ways to visualize icons in an easy-to-understand uncluttered fashion, while satisfying various users' requirements is a challenging task. The limitations of mobile devices further exacerbate the problem. For example, since only limited screen space is available, the amount of icon clutter on overviews increases with respect to desktop screens, and because of the limited processing power, the use of powerful but computationally intensive algorithms is discouraged.

While many papers address problems pertaining to the design of icons or the encoding of values with icons (Wong and Bergeron, 1997), the problem of how to properly place icons on maps avoiding clutter has been scarcely investigated. A possible strategy to tackle the problem is to adapt methods and algorithms that have been developed for the closely related problem of label placement on maps. In the first part of this chapter, we will thus describe label placement techniques which have been previously proposed in the literature, mainly for desktop map-based applications, and which are of interest for fast icon placement. We will then propose techniques based



Fig. 2.2. An example of icon cluttering with a limited number of icons, taken from the Expedia ([www.expedia.com](http://www.expedia.com)) site.

on an approach – aggregation – that aims at handling high icon density. The approach is based on an icon aggregation algorithm to reduce the total number of icons that need to be displayed and on the design of icons that could function both as individuals and aggregates in such a way that they can convey critical information on the associated POIs and possibly provide access to less critical information by tapping on them. We will conclude with a discussion of future research directions on icon decluttering techniques for mobile map-based applications.

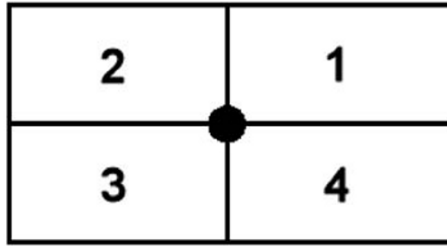
## 2.2 Label placement in map-based applications

Placing text labels on maps while avoiding overlaps with cartographic symbols and other labels is a fundamental problem in the field of cartography. Label placement is still often performed manually (Cook and Jones, 1990), although several techniques have been proposed to automate it.

Usually, there are three different label placement tasks: labeling of area features (e.g., countries, lakes, oceans), line features (e.g., roads, rivers), and point features (e.g., cities on small-scale maps, mountain peaks, points of interest). When multiple features have to be handled, these tasks share a common combinatorial difficulty in automatic label placement.

Most techniques for automatic label placement presented in the literature focus on point features. To formulate the point-feature label placement (PFLP) problem as a combinatorial optimization problem, it is necessary to define a *search space* and an *objective function*.

An element of the search space (called *labeling*) is a mapping of point features into label positions, where the potential label positions for a feature are usually taken from an explicitly enumerated set. A typical set of four possible label positions for a point feature is shown in Fig. 2.3.



**Fig. 2.3.** A set of potential label positions for a feature located in the center (black dot). Each rectangle corresponds to a region where the label may be placed and the values indicate the desirability of each position (the lower the value, the more preferred the position).

The objective function assigns to each labeling a score that corresponds to the quality of that labeling. Labeling quality depends on many factors, including map purpose and characteristics of human visual perception. A commonly used objective function for the PFLP problem was proposed by Yoeli (1972). In Yoeli's proposal, the quality of a labeling depends on the following criteria:

- The amount of overlap between labels and graphical features (including other labels);
- Cartographic preferences among potential label positions (a canonical set is shown in Fig. 2.3);
- The number of point features that are left unlabeled.

The objective function specifies how to compute a numerical score from these criteria. Lower scores are usually assigned to better labelings, so that the goal of the search is to minimize the objective function.

Unfortunately, while some simple restrictions of the label placement problem can be easily solved in polynomial time (Formann and Wagner, 1991), the general search problem and many of its interesting variants are Non-deterministic Polynomial-time hard (NP-hard) (Marks and Shieber, 1991). Therefore, a complete search algorithm could be impractical and a practical algorithm incomplete. Research efforts have then been made to find heuristic methods that work acceptably in practice, although they may not exhibit guaranteed performance.

In the following sections, we will briefly describe some of the approaches to label placement proposed in the literature. We will also introduce an established data structure, the *conflict graph*, which supports label placement and plays a fundamental role in our icon aggregation approach.

### 2.2.1 PFLP algorithms

A basic approach to PFLP is to perform an *exhaustive search* of the solution space. The algorithms based on this approach process points in a prescribed order, trying to place each label in a position that is currently unobstructed. When it is impossible to label a point, either because there are no positions without conflict or because all

available positions have been tried, these algorithms backtrack to the most recently labeled point and consider the next available position. Exhaustive search algorithms stop when an acceptable labeling has been identified or when the entire search space has been explored. Various heuristics, such as *variable ordering*, *value ordering*, *source-of-failure* and *pruning*, can be employed to improve the algorithm performance (Korf, 1988). Variable ordering consists in processing the most difficult points first, that is, those points which have fewer labeling options available. Value ordering is based on a prioritization of potential label positions for each point so that later conflicts can be avoided. A common method to achieve this goal is to always choose the label position which has the smallest number of conflicts with other features or labels. The source-of-failure heuristic backtracks to one of the points conflicting with the currently considered point, instead of backtracking to the most recently placed point. Pruning heuristics are used to eliminate whole areas of the search space from consideration. For example, it is possible to avoid considering label positions which conflict with a large number of surrounding positions. Unfortunately, regardless of these heuristics, exhaustive search is impractical as a general solution to the PFLP problem because the search space has an intrinsic exponential nature.

More efficient algorithms are based on avoiding the backtracking strategy of exhaustive search, thus limiting the scope of the search. Unlike exhaustive search algorithms, these *greedy algorithms* (Christensen et al., 1995) may leave unlabeled any point whose label cannot be placed, or may allow a certain degree of overlap among labels, thus providing a tradeoff between labeling quality and computational cost. Like exhaustive search algorithms, greedy algorithms exploit various heuristics, such as those previously described, to guide the search and help identify reasonable labelings.

The quality of labelings produced by a greedy algorithm can be considerably improved if labelings are subsequently repaired by local alteration. *Gradient-descent algorithms* (Hirsch, 1982; Christensen et al., 1995) can be used for this purpose. Starting from an initial labeling, usually obtained by placing labels randomly in any of the available candidate positions, the basic idea of gradient descent is to iteratively implement the label repositioning by moving in a descending direction towards a minimum of the objective function.

The main weakness of gradient-descent algorithms is their inability to escape from local minima of the objective function. Stochastic methods such as *simulated annealing* (Kirkpatrick et al., 1983) attempt to solve this issue by incorporating a probabilistic or stochastic element into the search, allowing movement in different directions from that of the gradient, thus including the possibility to temporarily worsen the objective function.

Various other approaches, such as *integer programming* (Zoraster, 1990), *genetic algorithms* (Verner et al., 1997) and *tabu search* (Yamamoto et al., 2002), have been proposed in the literature but their detailed analysis is beyond the scope of this chapter. For an extensive list of references on map labeling, we refer the reader to the Map-Labeling Bibliography web site (Wolff, 2006).

### 2.2.2 The conflict graph

The conflict graph is a structure that describes conflicts due to overlaps between label positions or, more accurately, between the rectangles defining label positions (see Fig. 2.3). Each node of the conflict graph corresponds to a label position while the edges of the graph link together conflicting label positions. More formally, a conflict graph can be defined as an undirected graph  $G = (N, E)$ , where  $N$  is a list of nodes and  $E$  is a list of edges. The degree of a node is the number of incident edges for that node and is a measure of the amount of conflict of the corresponding label position with other label positions. The higher the degree of a node, the more difficult it is to place the associated label on a map.

The conflict graph can be represented as an adjacency matrix, where rows and columns represent nodes and each entry is either 1 if the corresponding nodes are connected or 0 if they are not connected. In an adjacency matrix, the degree of each node is the sum of values in a line or in a column. An alternative representation of the conflict graph is an adjacency list, where each node is associated to the list of nodes it is connected to.

To build the conflict graph, it is necessary to check for overlaps among the considered label positions. A naive approach to perform this test is the all-pairs algorithm where each label position is tested against all other ones. In spite of its simple principle, this approach has a quadratic cost in the number of label positions and, therefore, is too time-consuming and inappropriate except for very small sets of labels. A more efficient algorithm is based on the sweep-line approach (de Berg et al., 1997). In general, a sweep-line algorithm detects all intersections among a set of axis-parallel rectangles in the plane by making an imaginary vertical line sweep across the plane from left to right. The algorithm is supported by two data structures, called *event-point schedule* and *sweep-line status*. Event points are the  $x$ -values where the sweep line must stop because either its status changes or intersections have to be reported. In the case of labels, event points are the  $x$ -values of the left and right edges of the rectangles defining label positions and the event-point schedule is the sorted list of these points. The sweep-line status stores intervals corresponding to the intersections of the sweep line with the given rectangles. The endpoints of the intervals are the  $y$ -values of the lower and upper edges of the input rectangles. Initially, the sweep-line status is empty. When a left edge of a rectangle is encountered during the sweep, the interval corresponding to the edge is inserted into the sweep-line status. A rectangle is reported if its interval is currently in the sweep-line status and intersects the new interval. When a right edge of a rectangle is encountered, the corresponding interval is deleted from the sweep-line status. The label overlap problem is thus reduced to the problem of maintaining a set of intervals in such a way that they can be efficiently inserted and deleted, and interval-intersection queries can be answered quickly. An interval-tree data structure (Edelsbrunner, 1980) can be employed to achieve this result. The computational complexity of the sweep-line algorithm is  $O(N \log N + I)$  where  $N$  is the number of rectangles and  $I$  is the number of intersections.

The conflict graph can be used to reformulate the map labeling problem as a maximum independent set problem (Strijk et al., 2000). In particular, finding a solution for the map labeling problem is equivalent to finding a subset  $S$  of the vertices of the graph with maximum size, for which there is no edge  $(u, v)$  in the graph that

connects vertices  $u$  and  $v$  in  $S$ . Moreover, cost values can be assigned to the vertices to model cartographic preferences for the candidate label positions and the sum of the costs of the vertices in  $S$  is minimized with respect to the size of  $S$ . The vertices in  $S$  thus indicate the candidate positions that must be used for labeling the points and, if none of the vertices related to a particular point appears in  $S$ , then the point remains unlabeled.

Various algorithms have been proposed to solve the labeling problem as a maximum independent set problem. For example, Kakoulis and Tollis (1998) propose a general approach to labeling based on the conflict graph and its connected components. They use a greedy heuristic for maximum independent set to split these components into cliques (a clique in an undirected graph  $G$ , is a set of vertices  $V$  such that for every two vertices in  $V$ , there exists an edge connecting the two). Finally they construct a graph where the identified cliques and the features of the problem (points, lines or area) become nodes. In this graph, a feature and a clique are joined by an edge if the clique contains a candidate of the feature. A maximum-cardinality matching is then used to obtain the final labeling. Wagner et al. (2001) present a fast, simple and effective algorithm that can be applied to point, line, or area labeling. The input to the algorithm is the conflict graph of the label candidates. The algorithm is organized into two phases. In the first phase, a set of rules is used to simplify the conflict graph without reducing the size of an optimal solution. The goal of this phase is to reduce as much as possible the number of label candidates of the features. In the second phase, a heuristic is employed to reduce the number of label candidates of each feature to at most one. The heuristic is conceptually simple and consists in eliminating the most troublesome candidates first, by going through all features that still have the maximum number of candidates and deleting the candidate with the maximum number of conflicts. This process is repeated until each feature has at most one candidate left and no two candidates intersect.

Conflict graphs can also be pre-computed to improve the performance of label placement algorithms when real-time generation of maps is necessary (e.g., in mobile or web-based interactive applications). Indeed, in map-based applications, different sets of labels are usually presented to the user at different zoom levels. For example, an overview of a city map might contain only labels for the most important roads and locations while more detailed views will show also labels of less important features. Usually, the set of labels varies only at predefined zoom levels, called scale levels, while it does not change at intermediate levels. As a consequence, panning and zooming operations that do not change the scale level will not change the amount of label overlaps, since no labels will be added or removed from the considered map. Therefore, it is possible to build conflict graphs (one for each scale level) in a pre-processing phase, thus improving the global performance of the map labeling process (Yamamoto et al., 2005). Pre-computation of the conflict graph is an acceptable approach for mobile map-based applications where the number of possible scale levels is fixed. A more flexible approach is presented by Petzold et al. (2004), who propose the *reactive conflict graph*, a data structure that is built in a pre-processing phase and that models possible label conflicts regardless of map scale. In particular, specific algorithms are used to identify the scale ranges at which potential conflicts among features may occur and these ranges are assigned to the edges of the graph. The reactive conflict graph is then stored in a three dimensional geometric data structure, e.g. the

3-D-R-Tree (Guttman, 1984), where the first two dimensions represent features geometry (or their bounding boxes) and the third dimension represents map scale. During labeling, the reactive conflict graph is queried to obtain a static conflict graph that contains all objects to be labeled and all possible conflicts among these objects for a specific map area and scale.

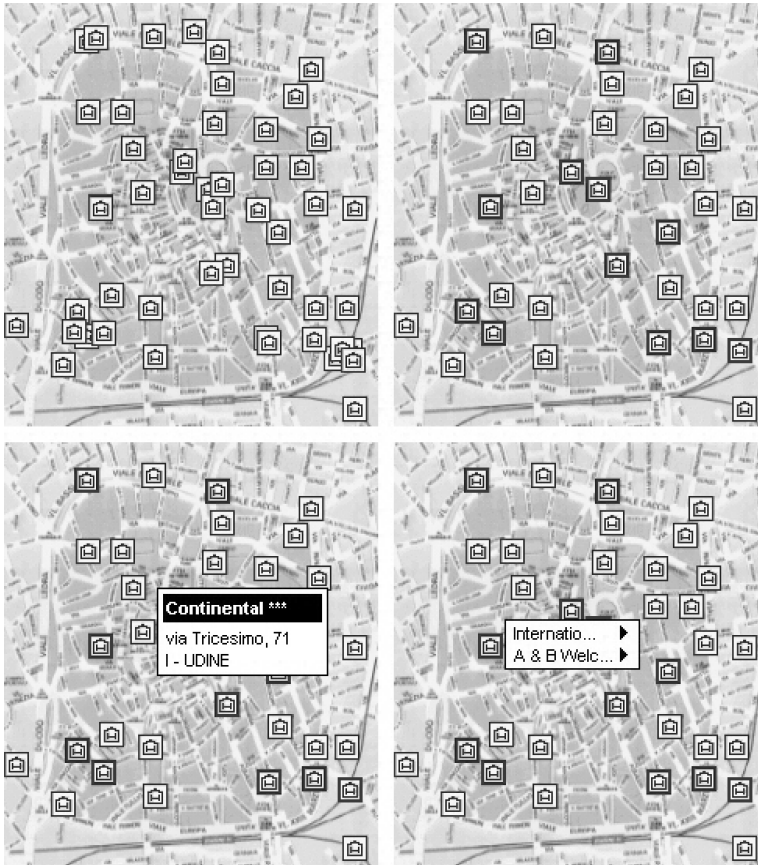
## 2.3 Decluttering of icons through aggregation

While, as shown in previous sections, a considerable effort has been devoted to automating the placement of labels on maps, much less attention has been devoted to the automatic placement of icons. This may be due to the consideration that icon placement can be regarded as a special case of label placement. Indeed, if we treat icons as labels, it is possible to use PFLP algorithms for automatic icon placement on maps. For example, Harrie et al. (2005) discuss a method of combining label and icon placement on maps created in real-time. Their method is based on reducing the search space (by choosing a fixed number of possible placements for each object) before performing a combinatorial optimization step based on simulated annealing to find a possible solution. The advantage of reducing the search space is that it allows to decrease the processing time (or improve the solution within a limited number of iterations), even if it may cause the optimal solution to be missed.

However, there are some subtle differences between the label placement and icon placement problems that can make it inappropriate to directly apply known PFLP algorithms to icons. First, labels can be removed from a map if, at the end of the labeling process, they cannot be placed without conflicts. This approach cannot be applied to icons because their purpose, in most cases, is to point out the existence of specific objects (POIs) that would be otherwise impossible to detect on a map. Their removal would thus cause fundamental information to be missing. Second, unlike labels, POI icons usually occupy a predefined fixed position with respect to their corresponding map feature, generally being placed exactly over the feature location. This constraint may be actually considered an advantage from the point of view of the computational cost of the placement algorithms. Indeed, a larger set of icon positions would result in a problem with a larger number of possible solutions and, in spite of the possibly better-looking results, the growth in problem complexity would result in a much higher computational effort. However, it must be noted that, despite the reduction in complexity, the icon placement problem still remains NP-hard and heuristic algorithms are thus needed to solve it within reasonable time.

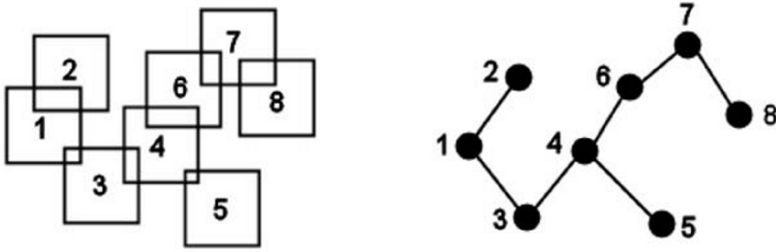
In the following sections, we will describe *icon aggregation*, the approach we propose to tackle the icon placement problem while taking into consideration its peculiarities. The idea is to identify clusters of mutually overlapping icons and replace them with special aggregator icons, whose purpose consists in pointing out the presence of aggregated icons as well as providing users with a means to access information about each of them. Indeed, selecting an aggregator icon produces a pop-up window that lists the aggregated POIs and allows the user to obtain more information on each POI. This technique can thus help to declutter the visualization, freeing map space by removing all icon clusters without causing information to be permanently lost.





**Fig. 2.4.** An example of icon aggregation: all clusters of overlapping icons that can be seen in the original map visualization in the upper left image are replaced by special icons representing aggregations (as displayed in the upper right image). Selecting an individual icon allows one to obtain detailed information on the corresponding POI, while selecting an aggregator icon produces the list of aggregated POIs.

An example of aggregation is illustrated in Fig. 2.4: the image on the upper left displays a map with 50 icons, some of which overlap; the image on the upper right displays the same map after aggregation. As shown, clusters of overlapping icons have been removed and replaced with aggregator icons that are graphically different from those used for individual POIs (in this case, hotels). As illustrated in the bottom left image, selecting an individual icon allows one to obtain detailed information on the corresponding hotel (in this case, its name, number of stars and address). Selecting an aggregator icon instead produces the list of aggregated hotels, as illustrated in the bottom right image, and each item in the list allows one to access detailed information on the corresponding hotel. It must be noted that aggregation may also highlight the presence of icons that are almost completely hidden by others, as in the case of the



**Fig. 2.5.** A set of overlapping icons (left) and the corresponding conflict graph (right).

aggregator icon to the left of the pop-up window in the bottom right image. In the original map, the fact that an icon almost completely overlapped another icon there could pass unnoticed.

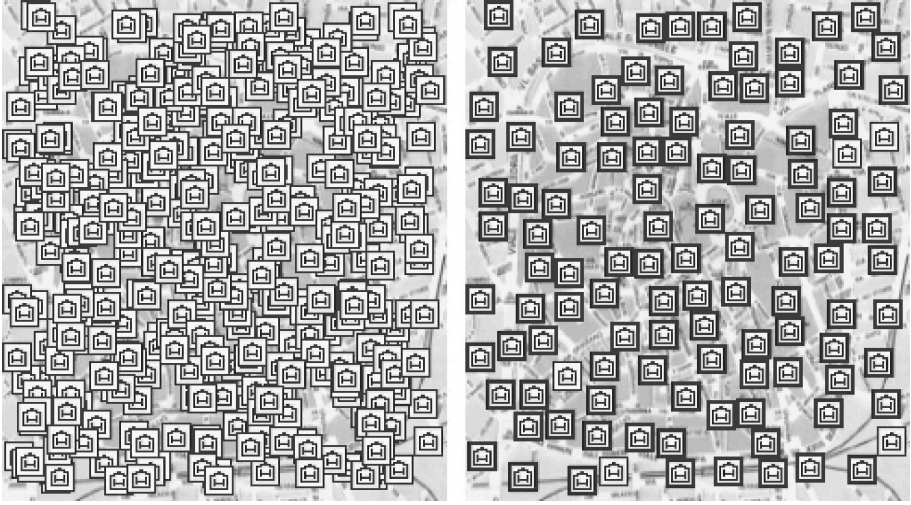
Although the idea behind aggregation is simple, different issues must be addressed to implement the approach. The most important issue consists in designing appropriate algorithms to identify clusters of icons and process them. Given the limitations of mobile devices, we need algorithms that seek a compromise between cost and benefit, with good quality (in terms of number of placed icons) and a short response time. Indeed, since we do not actually lose data when performing aggregation, we may want to sacrifice optimality of the final solution in return for improved performance. In this way, it becomes possible to update the visualization in real-time, as users perform operations (e.g., panning) that may change the set of icons to visualize.

Our implementation divides the task in two distinct subtasks that can be treated separately. The first subtask is to identify overlaps between icons and build a static conflict graph to store this information. As an example, Fig. 2.5 shows a set of overlapping icons and the corresponding conflict graph. We used the algorithm presented in section 2.2.2 to implement the conflict graph as an adjacency list. The second subtask is to process the information provided by the conflict graph so that a proper aggregation of the icons can be determined. We implemented different algorithms to perform this subtask to identify the one providing the best trade-off between performance and output quality.

### 2.3.1 Icon aggregation algorithms

The conflict graph described in the previous section is the input for the algorithm that determines how to aggregate icons. Because of the limitations in terms of processing power of the target platform on which aggregation has to take place, we chose to favor performance over quality, thus implementing only greedy but fast algorithms that could provide acceptable results.

Given a conflict graph, the *maximum aggregation algorithm* we propose produces a set  $S$  of aggregated icons without conflicts. The algorithm works as follows:



**Fig. 2.6.** An example of icon aggregation with the maximum aggregation algorithm on a map with 400 icons: the original map is displayed on the left, the result of the aggregation is displayed on the right.

- Initially, the solution set  $S$  is empty and the active node set  $ANS$  is equal to the full node set  $N$ .
- If  $ANS$  is empty, exit with  $S$  as the result. Otherwise, repeat the following steps:
  1. Calculate the degree of all nodes in  $ANS$ .
  2. Select  $n_{min}$ , the node with the smallest degree in  $ANS$ .
  3. If the degree of  $n_{min}$  is higher than 0, then mark  $n_{min}$  as aggregator and mark all nodes adjacent to  $n_{min}$  as aggregated to  $n_{min}$ .
  4. Place  $n_{min}$  in the solution set  $S$ .
  5. Remove  $n_{min}$  and all nodes adjacent to it from  $ANS$ .

An implementation of this algorithm builds a priority queue from the conflict graph, ordering icons in ascending order of degree, where the degree of each icon is the number of conflicts for that icon. At each step, the algorithm takes the icon with the smallest degree, removes it from the priority queue and places it in the solution set. The icon is marked as aggregator if its degree is higher than 0, i.e., if it overlaps other icons in the queue. Moreover, all icons that are in conflict with the aggregator are removed from the queue, they are marked as aggregated and the degrees of their neighbors that are still in the queue are decreased accordingly. The process is then repeated until the priority queue is empty.

Fig. 2.6 shows an example of the output of this algorithm on an icon placement problem involving 400 icons.

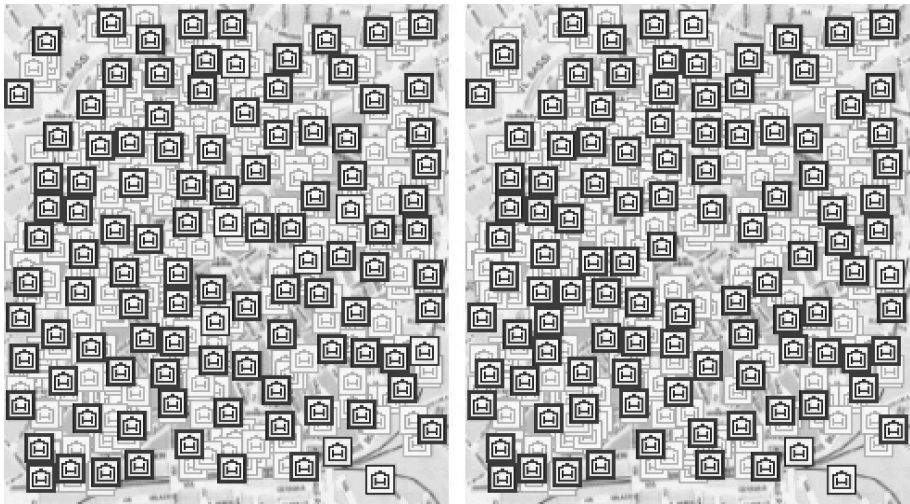
The algorithm is simple but provides good solutions in a time that depends on how the priority queue is implemented. This is because the algorithm spends most of its time adding elements to the queue, removing elements from the queue or changing the priority of elements. In particular, updating the degree of the neighbors of all

nodes adjacent to  $n_{min}$  (which is needed after having removed these nodes from *ANS*) requires an update of the priority queue and is quite time-demanding.

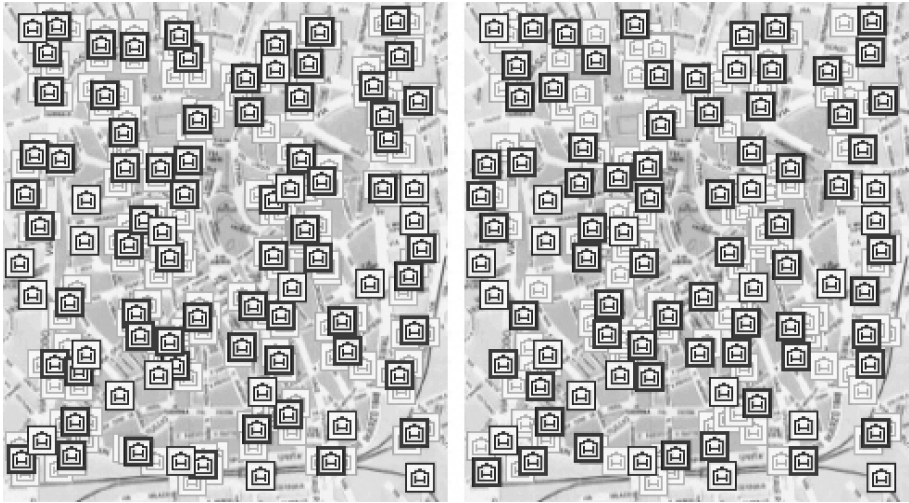
To implement the priority queue, we first tried the skip list data structure (Pugh, 1990). Skip lists are a probabilistic alternative to balanced trees (the most common data structure for implementing priority queues) that allows operations such as insertion and deletion of elements to be much simpler and faster. Unfortunately, in all our practical tests involving up to 400 icons, the cost associated to the management of a priority queue based on a skip list was higher than using a plainly ordered list. We thus employed simple lists for the final implementation of the algorithm.

An interesting effect of using the maximum aggregation algorithm is that, by considering icons in increasing order of degree, the number of icons associated to each aggregator tends to be balanced. Indeed, when icons with an initially high number of conflicts are actually processed, it is highly probable that their degree has decreased because of previous icon removals from the priority queue. This is desirable because aggregations will be composed of an approximately similar (and limited) number of elements, and users will appreciate this feature when examining the results on a map.

To further improve the performance of the aggregation process in terms of processing time, we looked for a way to simplify the maximum aggregation algorithm. In particular, since updating the priority queue after the degree of its elements has changed is time-consuming, we tried to avoid this operation entirely. The resulting *fast aggregation algorithm* goes through almost the same steps as the previous algorithm but does not need to recalculate the degree of all nodes in the active node set and to update the priority queue accordingly. By processing icons only in their initial ascending order of degree, this modified algorithm is faster than the original algorithm



**Fig.2.7.** Output of the proposed algorithms for icon aggregation on a set of 400 randomly generated icons which have been left (shaded) in the background for reference purposes. The image on the left refers to the fast aggregation algorithm and the image on the right to the maximum aggregation algorithm. Despite the differences in the solution sets, the two images contain an approximately equal number of icons (108 vs. 109).



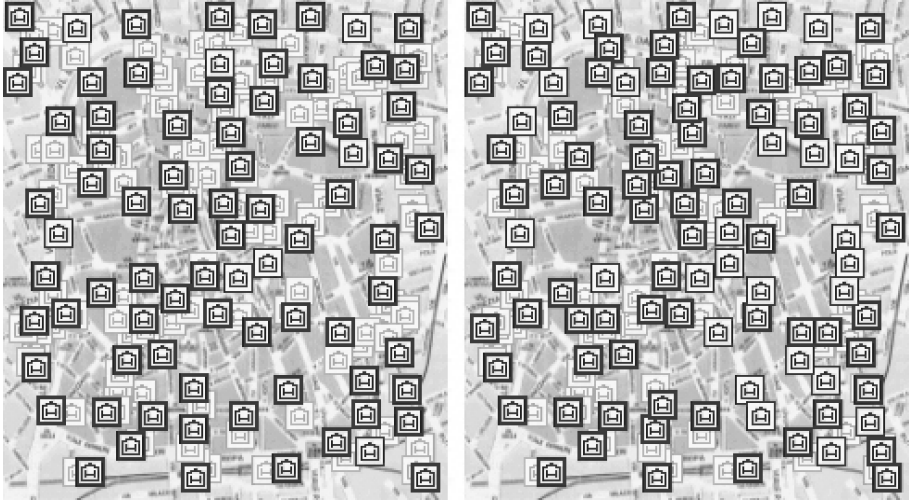
**Fig. 2.8.** Two different techniques to place aggregator icons: on the left, aggregators are placed in the centroid of each cluster of aggregated icons which have been left (shaded) in the background for reference purposes; on the right, aggregators are in the same position as one of the icons in each cluster. As it can be seen, aggregators may overlap each other with the first technique.

while achieving similar results in terms of the number of displayed icons. Fig. 2.7 shows a comparison between the results of the two algorithms on a set of 400 randomly generated icons.

A feature of these algorithms that we did not discuss so far is the placement of aggregator icons. In the current implementation, aggregator icons are placed exactly at the same position of one of the icons they aggregate. An alternative technique consists in placing aggregator icons in the centroid of each cluster of icons to be aggregated. The advantage of such a solution is that, from a geometric point of view, each aggregator is in a more representative position since its Euclidean distance from any of the aggregated icons positions is at most  $(a * \sqrt{2})/2$ , where  $a$  is the size of the icon, while in the current implementation this distance can reach  $(a * \sqrt{2})$ . Unfortunately, this technique suffers from a disadvantage that substantially reduces its usefulness and has prevented it from being adopted. Indeed, there is no guarantee that aggregator icons will not overlap each other, as illustrated in Fig. 2.8.

### 2.3.2 Relaxing the overlap constraint

In most map-based applications, the purpose of icons is simply to point out the location of specific geographic objects. Even when there are clusters of overlapping icons, there is often enough visual information for the user to be able to distinguish individual icons and determine their location on the map. However, the algorithms presented so far try to remove all overlaps among icons. While this is a fundamental requirement



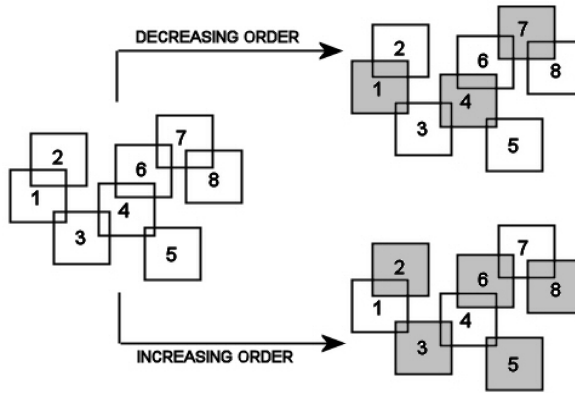
**Fig. 2.9.** Effect of relaxing the overlap constraint on icons: 20 more icons have been placed on the map on the right (where a certain degree of overlap is tolerated) with respect to the map on the left (where all overlaps have been removed).

for label placement, it might be considered too strict for icon placement. For example, looking at Fig. 2.4 it is possible to identify clusters of icons that severely overlap as well as icons that barely touch each other. In this latter case, it could be better to avoid aggregating the considered icons. In general, it would thus be useful to relax the overlap constraint, aggregating icons in such a way that a certain degree of overlap between icons is tolerated.

There are two possible ways of managing the relaxation of the overlap constraint. The simplest approach, which is the one we implemented, consists in building a custom version of the conflict graph where an arc between two nodes is added only when the degree of overlap between the corresponding icons exceeds a fixed threshold. Standard algorithms such as those presented in the previous sections could then be applied to perform the aggregation. A more flexible approach is based on associating a cost value with each arc in the conflict graph, corresponding to the amount of overlap between icons, and then modifying the aggregation algorithm so that this value is taken into consideration when processing icons. Fig. 2.9 shows how relaxing the overlap constraint leads to an increase in the number of icons that can be placed on a map when compared to the standard non-overlap approach.

### 2.3.3 Increasing map legibility

The algorithms presented in section 2.3.2 aim at maximizing the number of icons without conflicts displayed on a map, while keeping processing time short. By removing icons from the visualization, these algorithms also have the effect of enhancing the legibility of the map (i.e., the portion of map that is not hidden by icons), especially



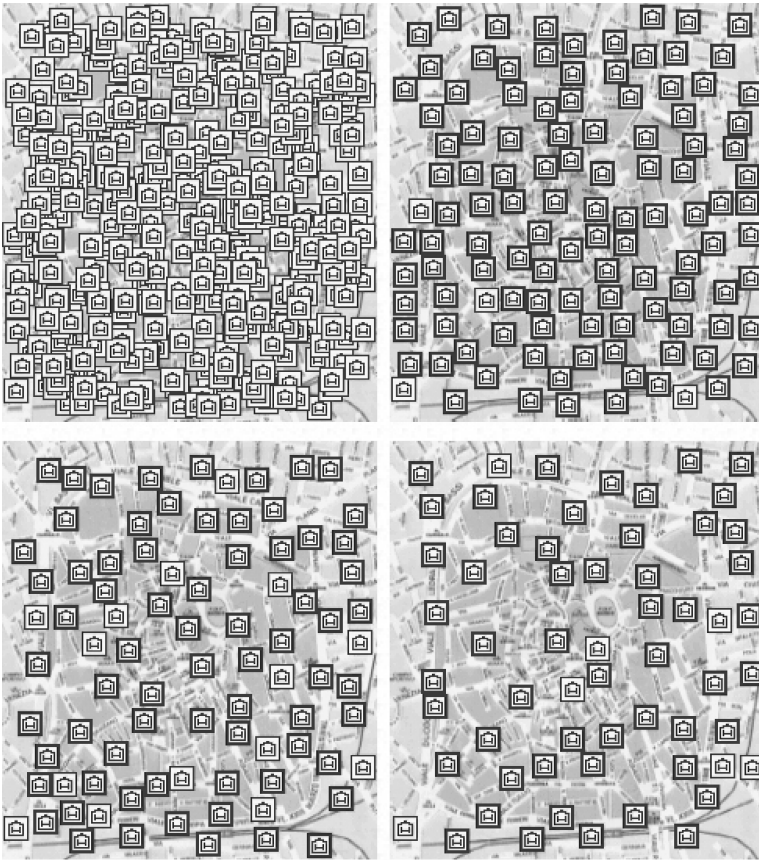
**Fig. 2.10.** Comparison of the outcome of the fast aggregation algorithm when processing nodes in decreasing or increasing order of degree. Icons in the solution set are highlighted in gray.

in highly cluttered regions, as illustrated in Fig. 2.6. However, it is possible to improve map legibility, which is a fundamental requirement in many map-based applications, by further reducing the number of icons displayed on the map. To achieve this goal, we tried two different approaches, producing two variants of the fast aggregation algorithm (similar variants for the maximum aggregation algorithm can also be produced). The two approaches can also be combined to provide better results.

The first variant is based on processing icons in decreasing (instead of increasing) order of degree, starting from icons with a higher number of overlaps. In this way, larger clusters of overlapping icons will be removed early in the process, possibly reducing the cardinality of the solution set. Fig. 2.10 shows a comparison of the outcome of the fast aggregation algorithm when processing icons in decreasing or increasing order of degree. In the considered example, when using decreasing order, the first icon that is processed is 4 (which becomes the aggregator for icons 3-4-5-6), followed by icon 1 (aggregator for 1 and 2) and icon 7 (aggregator for 7 and 8). When using increasing order, the first icon to be processed is 2 (aggregator for 1 and 2), followed by icon 5 (aggregator for 4 and 5), icon 8 (aggregator for 7 and 8), icon 3 (individual icon), and finally icon 6 (individual icon).

The second variant is based on removing all icons within a fixed distance  $K$  (in the conflict graph) from the currently processed icon. For example, for  $K = 2$ , the neighbors of an icon as well as the neighbors of those neighbors will be removed from the map and aggregated. Increasing the parameter  $K$  allows one to further reduce the number of visualized icons (until only one icon for each connected component in the conflict graph will remain) but has also the effect of increasing the maximum Euclidean distance between an aggregator and its aggregated icons.

Fig. 2.11 shows an example of the outcome of the two proposed variants compared with the original fast aggregation algorithm.



**Fig. 2.11.** Effect of different icon aggregation algorithms on map legibility. The original icon set is depicted in the upper left image, the other images show respectively the result of the fast aggregation algorithm (upper right) and its decreasing degree (lower left) and 2-distance (lower right) variants.

### 2.3.4 Evaluation

We implemented the algorithms presented in the previous sections using the C# language and the .NET Compact Framework for PocketPC devices and run them on a Dell Axim X30 device (equipped with a 624MHz processor) on different randomly generated sets of icons under the following conditions:

- Screen resolution: 240 x 268 pixels.
- Fixed icon size: 16 x 16 pixels.
- Icon set size:  $N = 50, 100, 200, 400$ .



- For each icon set size, 25 different trial configurations with random placement of icons.

**Table 2.1** Mean number of icons displayed after aggregation.

Algorithm	50	100	200	400
Min-overlap	39.8	66.2	107.3	148.4
Maximum	36.9	58.2	87	116
Fast	36.9	58.2	86.6	111.1
Fast (inverse order variant)	34.3	50.5	67.1	86
Fast (2-distance variant)	34.1	48.8	60.4	67.8

Table 2.1 reports the mean number of icons displayed after aggregation for different icon set sizes. The results show that the maximum and fast aggregation algorithms (whose goal is to maximize the number of displayed icons) are almost equivalent with 50, 100 and 200 icons, while they slightly differ with 400 icons, with the first algorithm performing better than the second. We also included results for a variant of the maximum aggregation algorithm that allows a minimal overlap (3 pixels) among icons, thus being able to place more icons for each icon set size.

Both variants of the fast aggregation algorithm (whose goal is instead to increase map visibility) reduce the number of displayed icons compared to the original algorithm with a difference that starts to be significant for icon set sizes higher or equal than 100 icons. Moreover, the 2-distance variant ( $K = 2$ ) performs better than the inverse order variant in highly cluttered configurations involving high numbers of icons.

**Table 2.2** Mean computation times (in seconds) to obtain a solution on a Dell Axim X30.

Algorithm	50	100	200	400
Maximum (skip list)	0.043	0.177	0.502	1.848
Maximum (plain list)	0.017	0.040	0.109	0.380
Fast	0.014	0.032	0.077	0.194
Fast (inverse order variant)	0.014	0.031	0.077	0.185
Fast (2-distance variant)	0.011	0.022	0.048	0.142

Table 2.2 compares the mean computation times of the algorithms. We distinguished the implementation of the maximum aggregation algorithm using the skip list data structure from the one using the plain list data structure. The results show that the fast aggregation algorithm is faster than the maximum aggregation algorithm (with plain list) but the difference starts to be significant only with 400 icons. The implementation of the maximum aggregation algorithm using the skip list is instead the slowest regardless of the number of icons, taking much more time when more than 200 icons are considered. Of the two variants of the fast aggregation algorithm, the

first one is almost equivalent to the original algorithm, while the second one performs slightly better with each icon set size.

## 2.4 Future research directions

The aggregation approach we proposed in this chapter is based on the design of proper aggregator icons to indicate clusters of individual overlapping icons. The basic solution we employed to design such aggregators consists in slightly varying the graphical appearance of individual icons by increasing the thickness of their border. Different approaches can be explored as well. For example, it is possible to indicate the number of aggregated icons by (slightly) increasing the size of aggregators. This would help users to detect areas of a map where higher numbers of icons have been aggregated.

In some map-based applications, icons are not simple placeholders that point out the presence and location of specific objects in an area but can also be used to convey information about the features characterizing these objects. For example, icons representing hotels might visually provide users with information about hotel quality, range of prices, services or other attributes. It is evident that in this case there is an even greater need to avoid overlaps among icons so that all useful information can be easily obtained by users. However, if we applied the aggregation approach as described in section 2.3, most of the visual information concerning aggregated icons would be lost. It is thus necessary to couple aggregation with additional techniques that can solve this issue. A trivial, far from optimal, solution would be to provide access to this information only through pop-up windows, by simply interacting with aggregators. Other solutions may generate aggregator icons that provide overviews of the attribute values of aggregated objects. For example, the aggregator may show the maximum value of each attribute characterizing aggregated objects so that users can understand at a glance if it is worth interacting with the aggregator to obtain more detailed information.

Icons may also refer to entities such as states, districts or parcels, and encode information about the different parameters characterizing them. Since these areas vary strongly in both shape and size, the task of positioning icons is not trivial and may require different approaches compared to the standard icon placement problem. An example of such an approach is proposed by Fuchs and Schumann (2004), who combine a simplified displacement algorithm with a suitable *focus&context* technique that interactively solves the remaining spatial conflicts.

Another variant of the icon placement problem takes into consideration icons belonging to different categories of POIs (e.g., hotels and restaurants). Since applying a standard aggregation algorithm to each set of icons separately does not guarantee that there will be no overlaps in the final outcome, proper solutions are needed to address this issue.

## 2.5 Conclusions

The requirements for icon placement on maps, although not trivial, have been scarcely studied by the research community. A number of methods for automatic label placement have been instead proposed in cartography. Unfortunately, they are focused on high-quality results, leading to long response times that are not suited for devices with limited computational resources and for interactive environments. In this chapter, we presented an approach to the icon placement problem that is based on aggregating overlapping icons, replacing them with special aggregator icons that point out the presence of aggregated icons and allow the user to retrieve information about each of them. We designed and evaluated fast algorithms to perform the aggregation and studied interesting variants of the problem.

It must be pointed out that none of the proposed algorithms can be an optimal solution to the icon placement problem for all possible map-based applications and their use scenarios. The most flexible approach, then, would be to choose the best solution according to users' needs and to the number of icons to manage. For example, when only a small number of icons have to be placed on a map, it could be better to relax the overlap constraint to maximize the amount of icons displayed. On the contrary, when a high number of icons must be considered, it could be better to adopt more aggressive aggregation algorithms, such as those presented in section 2.3.4, so that an adequate portion of map can still be visible. Thus, integrating different algorithms and selecting the most appropriate one when needed would improve the usefulness of any mobile map presented to the user.

## Acknowledgements

This work has been partially supported by the Italian Ministry of Education, University and Research (MIUR) under the PRIN 2005 Project "Adaptive, Context-aware, Multimedia Guides on Mobile Devices".

## References

- Christensen, J., Marks, J., and Shieber, S. (1995): An empirical study of algorithms for point-feature label placement. *ACM Transactions on Graphics*, Vol. 14 No. 3, pp. 203-232.
- Chittaro, L. (2006): Visualizing Information on Mobile Devices. *IEEE Computer*, Vol. 39 No. 3, pp. 40-45.
- Cook, A.C., and Jones, C.B. (1990): A Prolog rule-based system for cartographic name placement. *Computer Graphics Forum*, Vol. 9, pp. 109-126.
- de Berg, M., van Kreveld, M., Overmars, M., and Schwarzkopf, O. (1997): *Computational Geometry Algorithms and Applications*. Springer-Verlag, Berlin.
- Edelsbrunner, H. (1980): Dynamic Rectangle Intersection Searching. Institute for Information Processing Report 47, Technical University of Graz, Austria.
- Formann, M., and Wagner, F. (1991): A packing problem with applications to lettering of maps. In *Proc. Symposium on Computational Geometry*, ACM Press, pp. 281-288.

- Fuchs, G., and Schumann, H. (2004): Intelligent Icon Positioning for Interactive Map-based Information Systems. In: *Proc. International Conference of the Information Resources Management Association (IRMA 2004)*, New Orleans, USA.
- Guttman, A. (1984): R-trees: A Dynamic Index Structure for Spatial Searching. In: *Proc Conference on Management of Data*, University of California, Berkeley, USA, pp 47-57.
- Harrie, L., Zhang, Q., and Ringberg, P. (2005): A case study of combined text and icon placement. In *Proc. International Cartographic Conference (ICC 2005, CD-ROM)*, A Coruña, Spain.
- Hirsch, S. A. (1982): An algorithm for automatic name placement around point data. *American Cartographer*, Vol. 9 No. 1, pp. 5-17.
- Kakoulis, K.G., and Tollis, I.G. (1998): A unified approach to labeling graphical features. In *Proc. Symposium on Computational Geometry*, ACM Press, New York, pp. 347-356.
- Kirkpatrick, S., Gelatt Jr., C.D., and Vecchi, M.P. (1983): Optimization by simulated annealing. *Science*, Vol. 220, pp. 671-680.
- Korf, R.E. (1988): Search: A survey of recent results. In: *Exploring Artificial Intelligence: Survey Talks from the National Conferences on Artificial Intelligence*, Shrobe, H.E. (Ed). Morgan Kaufmann, pp. 197-237.
- Marks, J., and Shieber, S. (1991): The Computational Complexity of Cartographic Label Placement, Technical Report TR-05-91, Center for Research in Computing Technology, Harvard University.
- Petzold, I., Gröger, G., and Plümer, L. (2004): Modeling of Conflicts for Screen Map Labeling In *Proc. International Society for Photogrammetry and Remote Sensing Congress (ISPRS 2004, CD-ROM)*, Istanbul, Turkey.
- Pugh, W. (1990): Skip lists: A probabilistic alternative to balanced trees. *Communications of the ACM*, Vol. 33 No. 6, pp. 668-676.
- Strijk, T., Verweij, B., and Aardal, K. (2000): Algorithms for maximum independent set applied to map labelling. Technical Report UU-CS-2000-22, Department of Computer Science, Utrecht University.
- Verner, O.V., Wainwright, R.L., and Schoenefeld, D.A. (1997): Placing text labels on maps and diagrams using genetic algorithms with masking. *INFORMS Journal on Computing*, Vol. 9 No. 3, pp. 266-275.
- Wagner, F., Wolff, A., Kapoor, V., and Strijk, T. (2001): Three Rules Suffice for Good Label Placement. *Algorithmica*, Vol. 30 No. 2, pp. 334-349.
- Wolff, A. (2006): Map-Labeling Bibliography. <http://il1www.iti.uni-karlsruhe.de/~awolff/map-labeling/bibliography/>
- Wong, P.C., and Bergeron, R.D. (1997): 30 Years of Multidimensional Multivariate Visualization. In *Scientific Visualization*, Nielson, G.M., Hagen, H., and Muller, H., (Eds). IEEE Computer Society Press, pp. 3-33.
- Yamamoto, M., Camara, G., and Lorena, L.A.N. (2002): Tabu search heuristic for point-feature cartographic label placement. *GeoInformatica*, Vol. 6 No. 1, pp. 77-90.
- Yamamoto, M., Camara, G., and Lorena, L.A.N. (2005): Fast point-feature label placement for real time screen maps. In *Proc. Symposium on GeoInformatics (GEOINFO 2005)*, Campos do Jordão, Brazil.
- Yoeli, P. (1972): The logic of automated map lettering. *The Cartographic Journal*, Vol. 9 No. 2, pp. 99-108.
- Zoraster, S. (1990): The solution of large 0-1 integer programming problems encountered in automated cartography. *Operations Research*, Vol. 38 No. 5, pp. 752-759.



<http://www.springer.com/978-3-540-37109-0>

Map-based Mobile Services

Design, Interaction and Usability

Meng, L.; Zipf, A.; Winter, S. (Eds.)

2008, XIV, 352 p., Hardcover

ISBN: 978-3-540-37109-0