Fast and Reliable Map Labeling *

Frank Wagner † Alexander Wolff ‡

Abstract

We present an algorithm for the *Map Labeling Problem* that guarantees optimal approximation quality and runtime behaviour, and yields results closer to the optimum than the best heuristic known so far.

The sample data used in the experimental evaluation consists of three different classes of random problems and a selection of problems arising in the production of groundwater quality maps by the authorities of the City of München.

1 Introduction

Map lettering is one of the classical key problems that has to be solved in the process of map production. Usually the map producer does not only want to show the exact geographic positions of the features depicted but also explain properties of these features. She has to arrange this information on the map so that:
— for every piece of information it is clear which feature is described;
— the information is of legible size;
— different texts do not overlap.
These and in addition a lot of esthetic criteria are described in [Imhof 1975] in an attempt to characterize good quality map lettering having mostly manual map making in mind. Nowadays there is an increasing need for large, especially technical maps, for which legibility is much more important than beauty.

The application which brought the problem to our attention is the design of groundwater quality maps by the municipal authorities of the City of München. They have a net of drillholes spread over the city. The map has to contain the location of these holes and for every hole a block of measuring results such as the concentration of certain chemicals.

---

*This work was done at the Institut für Informatik, Fachbereich Mathematik und Informatik, Freie Universität Berlin, Takustraße 9, 14195 Berlin-Dahlem, Germany. It was supported by the ESPRIT BRA Project ALCOM II

†wagner@math.fu-berlin.de

‡awolff@inf.fu-berlin.de

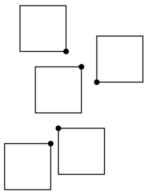

Figure 1: A valid labeling

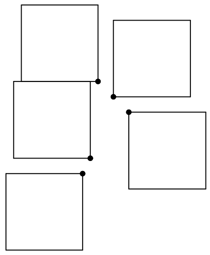

Figure 2: An optimal labeling for the example of Figure 1

The growing importance of such technical maps induces a need for the computerization of map making, the need for fully automated algorithms. Typically, labels in technical maps are axis-parallel rectangles of identical sizes. By rescaling one of the axes we can assume that the rectangles are squares. An adequate formalization is the following:

Problem MAP LABELING

Given n distinct points in the plane. Find the supremum σ_{opt} of all reals σ such that there is a set of n closed squares with side length σ, satisfying the following two properties.

1. Every point is a corner of exactly one square.

2. All squares are pairwise disjoint.

We call σ_{opt} the *optimal size*. A set of non-intersecting squares fulfilling 1. and 2. is called a *valid labeling*, see Figures 1 and 2.

In [Formann et al. 1991] the problem is shown to be $\mathcal{NP}$-hard. The main result of that paper is an approximation algorithm A that finds a valid labeling of at least half the optimal size. In addition, it is shown that presumably no polynomial time approximation algorithm with a better quality guarantee exists. In [Wagner 1994] it is shown that there is a matching lower bound on the running time.

Unfortunately A usually produces solutions not much better than 50 percent of the optimum, which makes it nearly useless for practical problems.

On the other hand there is a heuristical approach that shows very convincing results [Wagner et al. 1995a]. Three different variants H, I, and J are suggested. Method I is the winner in the experimental contest. The simplest of the heuristics, H, is used by the City of München for the application mentioned above, by the PTT Research Labs of the Netherlands to produce on-line maps for mobile radio networks, and in a computer system for the automated search for matching constellations in a star catalogue [Weber et al. 1995] as a tool to label the output on the screen.

In this paper we present an approximation algorithm B which is an integration of the two methods into a provably good and efficient algorithm of even better quality. In order to achieve A's running time efficiency of $\mathcal{O}(n \log n)$, we introduce a new way of detecting conflicts between overlapping squares. B maintains A's quality guarantee of 50 percent, and yields experimental results closer to the optimum than the best heuristic known so far. We compare A, B, and I in an experimental evaluation using three different classes of random problems and a selection of problems arising in the production of groundwater quality maps by the authorities of the City of München. For omitted details and proofs confer [Wagner et al. 1995b].

All heuristics, approximation algorithms, exact solvers, and problem generators described here, can be tested on the WWW under URL:
`http://www.inf.fu-berlin.de/~awolff/html/labeling.html`.

2 The Algorithm

Definition 1 *For a point p in the plane, a real $\sigma \geq 0$, and $i \in \{1, 2, 3, 4\}$, denote by σp_i an axis-parallel square with side length σ and p in its southwest, southeast, northeast respectively northwest corner. The enumeration is chosen like that of quadrants.*

We will call p_i a candidate *of the* site *p. Where the edge length σ is omitted, we refer to a candidate of the current label size.*

A solution of size *σ is a valid labeling with candidates of side length σ. A* label *is the candidate of a site which has been chosen to be in the solution.*

We say that two candidates overlap *or have a* conflict *if they intersect and neither contains a site. Analogously, two points are in conflict if any of their candidates are. For two candidates we define their* conflict size *as the largest edge length at which they do not intersect.*

The general structure of algorithm B is:

1. Find all *important* conflict sizes.
2. Do a binary search on these conflict sizes. Check for each size you look at, whether there is a solution or not, by going through the following three phases:

Phase I:	Preprocessing.
Phase II:	Eliminate candidates which cannot be part of the solution.
Phase III:	For those points which still have two or more candidates left, eliminate all but two, and check, whether the remaining problem is solvable by 2-SAT, as described below.

This approach is based on the fact, that a special case of the map labeling problem is efficiently solvable: If we have only two instead of four possible positions, we can consider a point p a boolean variable and associate its two candidates with the values p and $\bar{p}$ (*not* p). Thus we can generate a boolean formula consisting of clauses which encode all conflicts. Suppose square p was overlapping square $\bar{q}$ of a point q for a certain size, this would give us the clause $\overline{(p \wedge \bar{q})} = (\bar{p} \vee q)$ meaning that we do *not* want p and $\bar{q}$ to be simultanously in the solution. If we join all such clauses with the $\wedge$–operator into a single *conflict formula*, the satisfiability of the formula tells us exactly whether there is a solution of the current size. The formula is of 2-SAT type, thus its satisfiability can be decided in time proportional to its length [Even et al. 1976]. In Phases II and III we make sure that no point has more than two candidates left before we apply 2-SAT.

2.1 Finding conflict sizes

The key observation here is that it is sufficient to look at a constant number of closest neighbours of a site p in order to determine all conflict sizes which are not greater than the optimal label size.

Definition 2 *A conflict between a site s and one of its eight closest neighbours in each of the four quadrants relative to s is called* important. *The size of such a conflict analogously is an important conflict size.*

Theorem 3 *All conflict sizes which are not important, are greater than σ_{opt}.*

We do not have to consider label sizes greater than σ_{opt}, because there cannot be a solution of that size. There is no need to check a label size which is not a conflict size, since the set of conflicts does not change inbetween two consecutive conflict sizes. So it is sufficient to do the binary search in Step 2 of the algorithm on important conflict sizes.

It is obvious that the number of important conflict sizes is linear if we only have to look at a constant number of closest sites per candidate. These can be detected efficiently in time $\mathcal{O}(n \log n)$ by using a sweep-line technique. During this sweep we construct for every candidate a list of its conflicts.

2.2 Check whether there is a solution for a given σ

2.2.1 Phase I: Preprocessing

We run through all candidates. If a candidate p_i contains another point we eliminate the candidate. Otherwise we create a new list of overlap information which is an excerpt from p_i's conflict list. We use the fact that two overlapping candidates remain in conflict, until either of them contains a site if they are blown up simultanously. The elements of the new list consist of pointers to the overlap information of those candidates which actually overlap p_i for the given label size σ, and a pointer back to the candidate it belongs to. This can be done in linear time .

2.2.2 Phase II: Making Decisions

We run once through all points p and apply the following four rules:

- If p has candidates free of intersections with other candidates, we choose an arbitrary one of them (say p_i), and eliminate all other candidates p_j of p. Before a candidate's deletion, we do the following updates: we delete its list of overlap information and the symmetric entries stored with all candidates which overlap it.
- If p has only one candidate p_i left, do the same updates with all candidates q_j which overlap p_i, and then delete them.
- If p has a candidate p_i which overlaps the last two candidates of another site q, then update and eliminate p_i.
- If all candidates of p have been eliminated, we stop and return "no solution" to the program which does the binary search on the conflict list.

While we do this we maintain a stack. On this stack we put all those candidates for which we can now apply one of the rules because of the updates and eliminations we performed. Before we look at the next site p, we handle all the candidates waiting for us on the stack. Since each of the cases listed above can be detected and handled in constant time, and the number of times they occur is bounded by the number of conflicts, Phase II takes linear time.

At the end of Phase II we are done if all sites have exactly one candidate left. Otherwise we know that candidates of sites with several candidates — call them *active* — never intersect with those that are "the last of their breed", i. e. belong to sites with exactly one square left, because then the former ones would have been eliminated. So it is enough to focus on active candidates from now on. The others are already chosen as part of the solution, and do not interfere with the active ones any more.

2.2.3 Phase III

We run through all points with active candidates twice. In the first run, we only look at those with four candidates left, eliminate the one with most conflicts, and make all decisions of the type we did in Phase II. During the second run, we do the same for points which still have three active candidates. Then the remaining problem (consisting only of points with exactly two active candidates) is handed over to 2-SAT as described at the beginning of this section. This takes linear time.

2.3 Running time analysis

Phase I, II, and III can all be done in linear time, but since we have to look at $\mathcal{O}(\log n)$ conflict sizes during the binary search for the best solution, we need $\mathcal{O}(n \log n)$ time for Step 2. Finding conflicts in Step 1 takes the same time, so the whole algorithm is in $\mathcal{O}(n \log n)$.

3 Experiments

3.1 Example Generators

We run Heuristic I, and the Approximation Algorithms A and B on each of the examples produced by four problem generators. For every size we average the approximation quality and running time over 30 tests. The information about the optimal size is yielded where possible by an exact solver that was implemented by Erik Schwarzenecker [Kučera et al. 1993]. It shows exponential running time behaviour. For small examples it is very fast, for larger ones it is unreliable, so that we were forced to introduce a time limit of five minutes after which we stopped the execution. Only few of the largest *hard* and *dense* examples took less than these five minutes, and we have observed that the solution of examples beyond that bound then easily takes half an hour or much more.

Random. We choose n points uniformly distributed in a square of size $10n \times 10n$.

Dense. Here we try to place as many squares as possible of a given size σ on a rectangle. We do this by randomly choosing points p and then checking whether σp_1 intersects with any of the σq_1 chosen before. We stop when we have unsuccessfully tried to place a new square 200 times. In a last step we assign a random corner point to each of the squares we were able to place without intersection, and return its coordinates.

Hard. In principle we use the same method as for Dense, that is, trying to place as many squares as possible into a rectangle. In order to do so, we put a grid of cell size σ on it. In a random order, we try to place a square of edge length σ into each of the cells. This is done by randomly choosing a point within the cell and putting a fixed corner of the square on it.

Real World. The municipal authorities of Munich provided us with the coordinates of roughly 1200 ground water drill holes within a 10 by 10 kilometer square centred approximately on the city centre. From this list we extract a given number of points being closest to some centre point, thus getting all those lying in a square around this extraction centre.

3.2 Experimental Running Time

We measured the running time of the Approximation Algorithm B on a Sun Sparc station 20. It shows a rather stable linear behaviour with very small standard deviation. For the largest examples of about 1000 points B takes at most 5 seconds.

3.3 Approximation Quality

The approximation quality is the quotient of the solutions of the approximation algorithms and the exact solver. It is plotted for real world and hard examples (see Figure 3 and 4). The performance on random examples is nearly the same as on real world ones; on dense examples it is slightly better than on hard ones.

The standard deviation is represented by the length of the vertical bars in each point of the result plots.

The results are averaged over 30 tests per point, but only over those which the exact solver managed within the time bound. As Approximation Algorithm A guarantees 50 percent of the optimal size, twice the size of A's result gives us an upper bound on the optimum. Using this upper bound and taking into consideration the typical quality of A, we found out that B's behaviour on these examples does not differ significantly from that on the other examples.

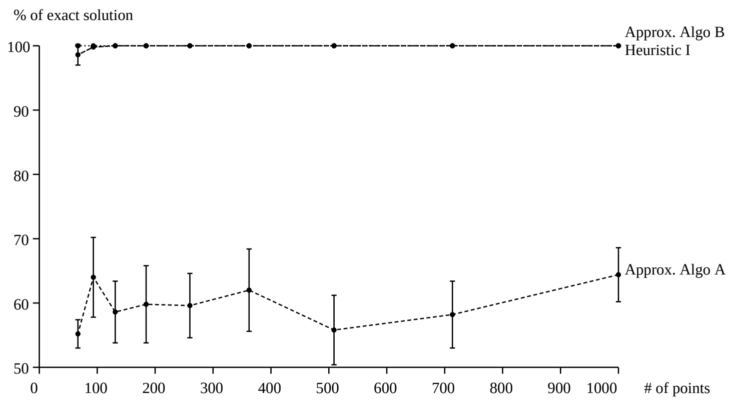

Figure 3: Quality of the algorithms on real world examples

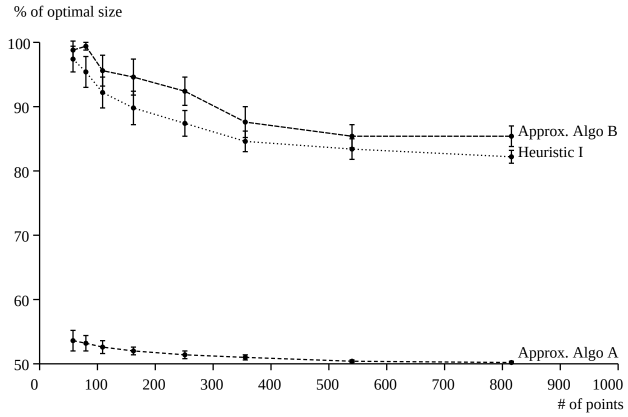

Figure 4: Quality of the algorithms on hard examples

References

[1] H. Aonuma, H. Imai, Y. Kambayashi, *A visual system of placing characters appropriatly in multimedia map databases*, Proceedings of the IFIP TC 2/WG 2.6 Working Conference on Visual Database Systems, North Holland (1989) 525–546

[2] S. Even, A. Itai, A. Shamir, *On the complexity of Timetable and Multicommodity Flow Problems*, SIAM Journal on Computing **5** (1976) 691–703

[3] M. Formann, F. Wagner, *A Packing Problem with Applications to Lettering of Maps*, Proceedings of the 7th Annual ACM Symposium on Computational Geometry (1991) 281–288

[4] E. Imhof, *Positioning Names on Maps*, The American Cartographer **2** (1975) 128-144

[5] L. Kučera, K. Mehlhorn, B. Preis, E. Schwarzenecker, *Exact Algorithms for a Geometric Packing Problem*, Proceedings of the 10th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science **665** (1993) 317–322

[6] F. Wagner *Approximate Map Labeling is in* $\Omega(n \log n)$, Information Processing Letters **52** (1994) 161–165

[7] F. Wagner, A. Wolff *Map Labeling Heuristics: Provably Good and Practically Useful*, to appear in: Proceedings of the 11th Annual ACM Symposium on Computational Geometry (1995)

[8] F. Wagner, A. Wolff *An Efficient and Effective Approximation Problem for the Map Labeling Problem* to appear in: Proceedings of the 3rd Annual European Symposium on Algorithms (1995)

[9] G. Weber, L. Knipping, H. Alt, *An Application of Point Pattern Matching in Astronautics*, Journal of Symbolic Computation **17** (1994) 321–340

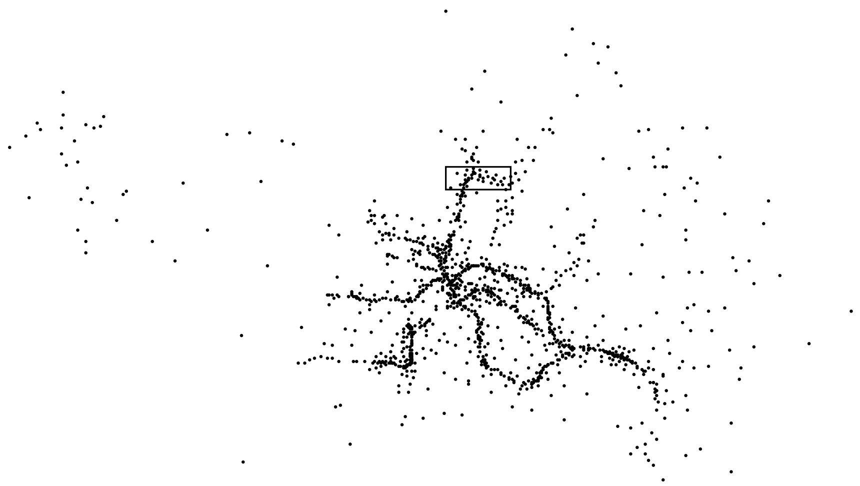

Figure 5: Map showing our sample data of approximately 1200 groundwater drillholes in Munich, and the section tested in Figures 6 and 7. There are no conflicts of interesting size between this section and the rest. The subway lines can be detected easily.

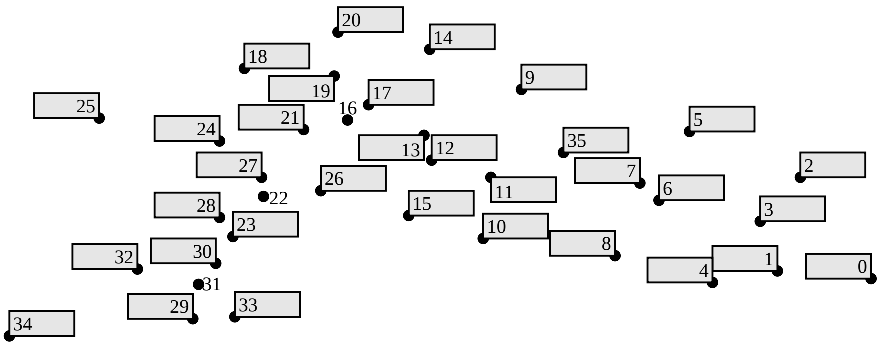

Figure 6: Solution of the program used by the authorities of the City of Munich before (label height 5000, 3 sites not labelled). It tries to maximize the number of sites labelled for a given size.

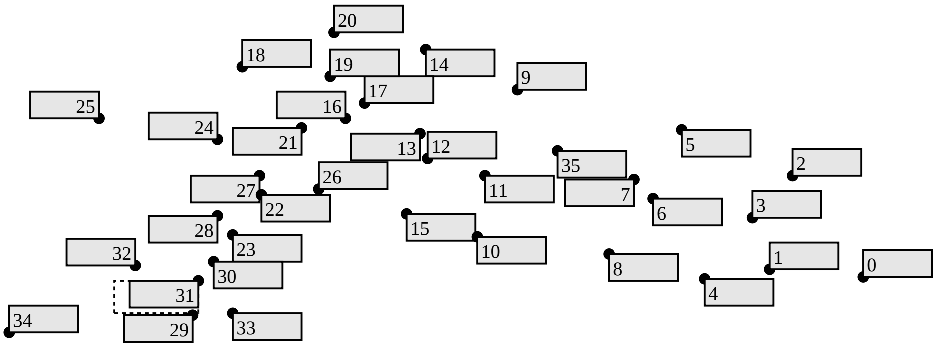

Figure 7: Solution produced by our heuristics (label height 5400, optimal).