# A Practical Map Labeling Algorithm *†

Frank Wagner ‡ Alexander Wolff ‡

Freie Universität Berlin

**Abstract**

The *Map Labeling Problem* is a classical problem of cartography. There is a theoretically optimal approximation algorithm $A$. Unfortunately $A$ is useless in practice as it typically produces results that are intolerably far off the optimal size. On the other hand there are heuristics with good practical results.

In this paper we present an algorithm $B$ that

- guarantees the optimal approximation quality and runtime behaviour of $A$, and
- yields results significantly 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 Munich.

## 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 intuitively 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 by Imhof [6] 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 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 Munich. 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 information such as measuring results or the ground water level.

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

*This paper is based on a combination of two previously published preliminary versions, [12] and [13].

†Work was partially supported by the Humboldt-Foundation and by a Heisenberg-Stipendium of the Deutsche Forschungsgemeinschaft.

‡Institut für Informatik, Fachbereich Mathematik und Informatik, Freie Universität Berlin, Takustraße 9, 14195 Berlin-Dahlem, Germany. E-mail: wagner@inf.fu-berlin.de, awolff@inf.fu-berlin.de

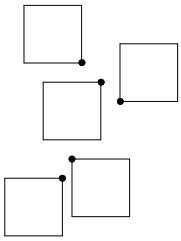

Figure 1: A valid labeling

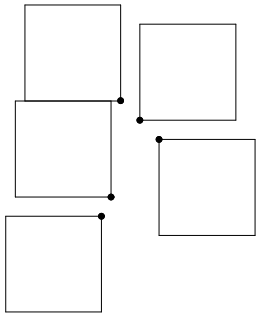

Figure 2: An optimal labeling for the example of Figure 1

rectangles. In certain cases as in the application mentioned above, they are even of identical size. By rescaling one of the axes we can then assume that the rectangles are squares. An adequate formalization is as follows:

**The MAP LABELING Problem**

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

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

(2) All squares are pairwise disjoint.

We call $\sigma_{opt}$ the *optimal size*. A set of squares fulfilling (1) and (2) is called a *valid labeling*, see Figure 1 and 2.

Formann and Wagner showed that the problem is $\mathcal{NP}$-hard [4]. 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 no polynomial time approximation algorithm with a better quality guarantee exists if $\mathcal{P} \neq \mathcal{NP}$. Related results were reported in [1] and [9]. The running time of $A$ is in $\mathcal{O}(n \log n)$. In [11] Wagner showed that there is a matching lower bound on the running time.

$A$ conceptually works as follows: It starts with infinitesimal equally sized squares attached to each point in all four possible positions. Then all squares are expanded uniformly. In order to resolve conflicts between them, we eliminate all those which would contain another point if they were twice as big. It is easy to show that after this process, a point $p$ can not have more than two squares left which overlap other squares.
If we consider $p$ a boolean variable and associate its squares with the values $p$ and $\bar{p}$, we can generate a boolean formula consisting of clauses which encode all conflicts. Suppose square $p$ was overlapping square $\bar{q}$ of a point $q$, 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, the satisfiability of the formula tells us exactly whether there is a solution of the current size. Since all clauses consist of two literals, the formula is of 2-SAT type, and can be evaluated in time proportional to its length [2], i. e., the number of conflicts.

This works only because we make sure that no point has more than two squares left after the elimination phase. In order to achieve that, we often have to eliminate both of two conflict partners, where it would have sufficed to delete one to resolve the conflict. This seems to be the reason for the practically very bad behaviour of $A$. In fact, $A$ usually produces solutions not much better than 50 percent of the optimum, which makes it nearly useless for practical problems. On the other hand this means that twice the size of $A$'s solution is a good upper bound for the optimal size.

Recently we presented a heuristical approach which performs much better in practice [12]. Instead of eliminating the squares as early as possible, it eliminates a square just when it is clear that it cannot be in any solution of the current size. The bad side effect of this is, that some points might have three or four squares left after the elimination phase. In order to handle this, we suggested three different methods $H$, $I$, and $J$ to bring their number down to two. Method $I$ was the winner of the experimental contest.
We came up with a hybrid algorithm that first runs $A$, and then uses its result to control the heuristics in two ways:

1. The approximation size of $A$ gives a lower and an upper bound on the optimal size. These bounds are used to show that there is only a linear number of conflicts of interesting size between overlapping squares. In addition to that, we use twice the upper bound as the width of a window with which we sweep across the sites to detect these conflicts in $\mathcal{O}(n \log n)$ time.

2. The result of the hybrid algorithm is the maximum of $A$'s and the heuristics' result, thus guaranteeing the optimal approximation quality.

The simplest of the heuristics, $H$, is used by the City of Munich 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 [15] as a tool to label the output on the screen. With a very similar algorithmic approach Formann and Wagner [5] were able to solve the so-called METAFONT labeling problem posed by Knuth and Raghunathan [7].

In this paper we integrate the two parts of the hybrid method into a provably good and efficient approximation algorithm $B$ 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. It is based on an algorithm to find closest neighbours, which was suggested in [3]. We maintain $A$'s quality guarantee of 50 percent without being obliged to run $A$ beforehand and use its result. In order to do so, we perform a test whether a solution can still be constructed from a subset of the squares which have not been eliminated so far during a certain phase of the algorithm. We improve this elimination phase with the help of an additional deletion rule for squares which are not needed in a valid solution.

We compare $A$, and $B$ combined with each of the heuristics 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 Munich.

# 1 The Algorithm

Before we can describe the structure of the algorithm we have to give some definitions.

**Definition 1** *For a point $p$ in the plane, a real $\sigma \geq 0$, and $i \in \{1,2,3,4\}$, denote by $\sigma p_i$ an axis-parallel square with side length $\sigma$ 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 $\sigma$ is omitted, we refer to a candidate of the current label size.*
*A* solution of size $\sigma$ *is a valid labeling with candidates of side length $\sigma$.*

For technical reasons, we will from now on consider a candidate an open square, plus the open edges incident to the site. Note that this excludes all corner points, especially the site itself. The idea is that we shrink the squares by a tiny bit, so that an optimal labeling is a valid labeling, too. $\sigma_{opt}$ then is the size of the maximum valid solution. This is equivalent to the previous definition.

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.

## 1.0 Structure

Step 1: Find all important conflict sizes.

Step 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. Then do a 2-SAT test on a subset of those which have not been eliminated. |
| Phase III: | For those points which still have two or more candidates left, choose exactly two, and check, whether this remaining problem is solvable by 2-SAT, as described in the introduction. |

## 1.1 Finding conflict sizes

We show 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. The reason for this strategy is that the $k$ closest neighbours of all $n$ sites $p$ in any of the four quadrants relative to $p$ can be found efficiently in $\mathcal{O}(kn \log n)$ time with an algorithm described in [3]. In this article, distance and proximity always refer to the $L_\infty-$ (maximum) norm.

**Definition 2** *Let* $\text{Quad}(p_i) = \infty p_i$*, that is the $i^{th}$ quadrant relative to a site $p$, $i \in \{1,2,3,4\}$. Note that this includes the border except $p$.*
*A conflict between a site $p$ and one of its eight closest neighbours in one of the four quadrants relative to $p$ is called* important*. The size of such a conflict analogously is an important conflict size. See Figure 3 for an example of a non-important conflict.*

**Theorem 1** *All conflict sizes which are not important, are greater than $\sigma_{opt}$, the size of an optimal solution.*

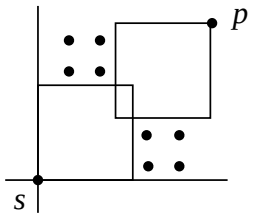


Figure 3: The conflict shown here between $s_1$ and $p_3$ is *not* important.

We do not have to consider conflict sizes greater than $\sigma_{opt}$, because there cannot be a solution of that size. Neither is there a need to check any label size which is not a conflict size, since the conflict graph of all candidates does not change in between 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 $\mathcal{O}(n \log n)$ as mentioned above. At the same time we construct for every candidate a list of its potential conflict partners. These conflict lists will be needed in Phase I. They only take linear space in total.

**Proof.** For reasons of symmetry we can focus on conflicts in which some candidate $p_1$ is involved. First, consider only sites which lie in Quad($p_1$). We want to show that the sizes of all conflicts between $p_1$ and candidates of these sites are greater $\sigma_{opt}$ if they are not important. Suppose there is a conflict of size $\sigma \leq \sigma_{opt}$ between $p_1$ and one of the candidates of its $k^{th}$ closest neighbour $s^k$. We show that there cannot be a partial solution of size $\sigma$ for $p$ and $s^1, \ldots, s^k$ if $k > 8$. This would be a contradiction to $\sigma \leq \sigma_{opt}$, because there is always a solution of size $\sigma_{opt}$, and it automatically is a solution for all smaller label sizes as well, and certainly for a subset of the sites.

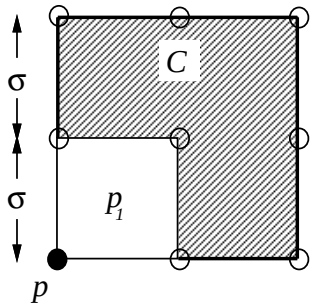


Figure 4:

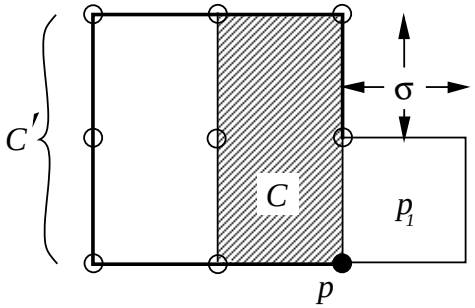


Figure 5:

The distance between $p$ and the sites with candidates in conflict with $p_1$ is bounded in the following way: $\sigma \leq ||p - s^i||_\infty \leq 2\sigma \leq 2\sigma_{opt}$ for $i = 1, \ldots, k$, otherwise $s^1$ would lie in $\sigma p_1$, or none of the candidates of $s^k$ would have a conflict of size $\sigma$ with $p_1$. The area $C$ in which the $s^i$ can therefore lie, is shaded in Figure 4. Consider the set $\mathcal{L}$ of points in $C$ which lie on a grid of size $\sigma$ with "origin" $p$. All eight points in $\mathcal{L}$ are marked by circles in Figure 4. We cannot place a site with its label such that the site lies in $C - \mathcal{L}$ and its label does not intersect $\mathcal{L}$. Remember that the two open label edges adjacent to the site belong to the label. At the same time none of the points in

$\mathcal{L}$ can be contained by more than one label, otherwise these labels would overlap. So we can only get a partial solution for sites in $C$ if there are not more than $|\mathcal{L}| = 8$ of them.

Now consider all sites which lie in the second quadrant relative to $p$, and have candidates in conflict with $p_1$. Again, let $C$ be the area in which such sites can lie. $C$ is shaded in Figure 5. Look at the area $C'$ in Quad($p_2$) of all points which have the same distance to $p$ as points in $C$. $C'$ contains eight grid points $\neq p$. Therefore, with the same argumentation as above, we get that there can only be eight sites in $C'$ such that the labels attached to them do not overlap.

Exactly the same idea holds for sites in Quad($p_4$), since it is symmetric to Quad($p_2$) relative to $p_1$. No candidate of a site in Quad($p_3$) can be in conflict with $p_1$, because it would then automatically contain $p$. Hence there cannot be a conflict of size $\sigma \leq \sigma_{opt}$ between $p_1$ and one of the candidates of its $k^{th}$ closest neighbour for $k > 8$. □

## 1.2 Check whether there is a solution for a given $\sigma$

### 1.2.1 Phase I: Preprocessing

We run through all the candidates $p_i$. If $\sigma p_i$ contains another site, we *eliminate* $p_i$, i. e., we will not consider it any more for the solution of the current label size we want to construct. Otherwise we create a new list of overlap information by extracting those conflicts from $p_i$'s conflict list, whose conflict sizes are less than $\sigma$. 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 current label size $\sigma$, and a pointer back to the candidate it belongs to. This can be done in linear time since the sum of the lengths of all conflict lists is linear, see Section 1.1. For the same reason, the lists of overlap information just need linear space.

### 1.2.2 Phase II: Eliminating Impossible Candidates

We run once through all points $p$. We look at the following four cases:

- If all candidates of $p$ have been eliminated, we stop and return "no solution of size $\sigma$" to the program which does the binary search on the conflict list.
- 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 $p_i$.

While we do this we maintain a stack. On this stack we put all those candidates which now do not intersect any other squares, or are the last candidates of their sites. Before we look at the next site $p$, we treat all those 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 us so far linear time.

**Corollary 1** *If there is a solution of the current label size $\sigma$, then there is still a subset of the candidates left after Phase II, which forms a solution.*

**Proof.** Suppose to the contrary that $p_i$ is the first candidate after whose elimination the remaining problem becomes unsolvable. Then the following statement is true:

($\star$) Every solution $\pi$ of the problem just before this elimination must contain $p_i$.

Consider the circumstances under which $p_i$ can be eliminated:

1. $p_i$ contains a site $q$. This contradicts ($\star$).

2. $p_i$ does not overlap other candidates, but the same holds for some $p_j$, and the algorithm decides to eliminate $p_i$.
   In this case we could replace $p_i$ in $\pi$ by $p_j$, contradicting ($\star$).

3. $p_i$ overlaps $q_j$ which is the last candidate of $q$.
   Then also $q_j$ must be part of $\pi$, which again contradicts ($\star$).

4. $p_i$ overlaps $q_j$ and $q_k$ which are the last two candidates of $q$.
   Then $q_j$ or $q_k$ must be in $\pi$, which is a contradiction to ($\star$).

□

At the end of this part 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.
As a consequence of Corollary 1 we also know that we have not yet returned "no solution" if there is one of size $\sigma$. So we can find a solution with the help of 2-SAT as described in the introduction if no site has more than two candidates left. Otherwise we do a test on a subset of the active candidates as follows.

### Keeping the Approximation Guarantee

How can we make sure that $B$, too, keeps the approximation guarantee of 50 percent? Its main difference from the Approximation Algorithm $A$ is that until the end of the first part of Phase II it does not destroy any candidate which might be necessary to construct a solution. $A$ on the other hand already eliminates candidates if they contain a site at twice their size. We take advantage of this approach without risking to end up with a practical behaviour as bad as $A$'s.

**Definition 3** *A candidate $p_i$ is called* $\sigma$–dead *if $\sigma p_i$ contains a site.*

**Lemma 1** *Let $\sigma$ be the current label size. Then after Phase II all sites have at most two candidates left which are not $2\sigma$–dead.*

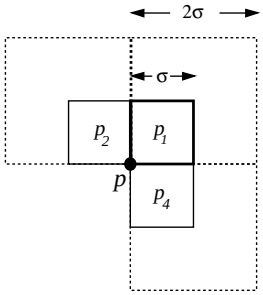


Figure 6:

**Proof.** Suppose $p$ had three such candidates, w. l. o. g. $p_1$, $p_2$, and $p_4$ as indicated in Figure 6. If $2\sigma p_1$, $2\sigma p_2$, and $2\sigma p_4$ did not contain any other site, then there could clearly be no candidate in conflict with $\sigma p_1$. This means that $p_1$ (or possibly $p_2$ or $p_4$) should have already been chosen as part of the solution in th first part of Phase II, and $p$'s other candidates eliminated. □

**Theorem 2** *If $\sigma \leq \sigma_{opt}/2$ we can efficiently construct a solution of size $\sigma$ from the candidates left after the first part of Phase II.*

**Proof.** If $\sigma \leq \sigma_{opt}/2$, then there must be a solution of size $2\sigma$. Let $\pi$ be a fixed solution of this size. Then $\pi$ is also a solution of size $\sigma$, since simultanously shrinking all squares reduces the number of conflicts. Let $i = \pi(p)$, the index of the candidate of site $p$ which is part of solution $\pi$. We want to show that $p_i$ has not yet been eliminated in Phase II – except $p$ has had a candidate without conflicts at some point. In that case, $p$'s other candidates were deleted, and we can use the one without conflicts for the solution we are about to construct. To see that $p_i$ has not yet been deleted otherwise, we just have to check the conditions under which this could have happened:

- $\sigma p_i$ cannot contain any other site, because $2\sigma p_i$ does not.
- $\sigma p_i$ cannot overlap the last two candidates of some site $q$, since $2\sigma p_i$ would then contain $q$ (see Figure 7).
- Assume that $\sigma p_i$ is overlapping $\sigma q_j$, and $q_j$ is the last candidate of $q$ (see Figure 8). Then obviously we would have to delete $p_i$ in the first part of Phase II. But suppose $p_i$ is the first candidate of solution $\pi$ which gets eliminated though $p$ has not had a candidate without conflicts. Then $q_j$ must be in solution $\pi$ because $q_j$ is the last candidate of $q$, and $q$ has never had a candidate without conflicts, otherwise its last candidate $q_j$ would not overlap $p_i$. But if $\sigma p_i$ and $\sigma q_j$ overlap, then $2\sigma p_i$ and $2\sigma q_j$ certainly do so as well, contradicting the assumption that they are part of solution $\pi$.

We have just shown that if there is a solution of size $2\sigma$, then either a site had candidates without conflicts, or it kept the candidate of solution $\pi$ all the way through Phase II. So certainly we have not stopped during the execution of Phase II and returned "no solution". In addition to that, Lemma 1 tells us that after Phase II all sites have at most two squares left which are candidates for such a solution. This means that we can use 2-SAT on the set of candidates which still have conflicts and are not $2\sigma$–dead. If 2-SAT finds a partial solution for those, we can just add all candidates without conflicts, and thus get a solution for all sites.

□

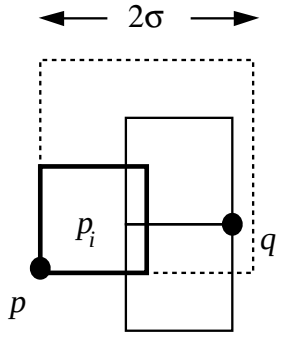


Figure 7:

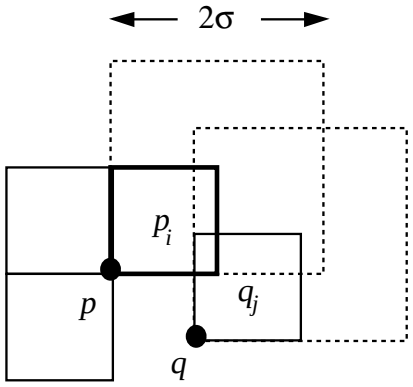


Figure 8:

If however 2-SAT returns "no solution", then there cannot be any of size $2\sigma$. That means that $\sigma > \sigma_{opt}/2$. In this case we try to find a solution in Phase III. This 2-SAT test will only be executed until it has returned a negative answer to a problem of size $\sigma_0$ to which the heuristics nevertheless found a solution afterwards. We can spare it then, because the binary search for a solution of maximal size continues only on label sizes $\sigma$ with $\sigma > \sigma_0 > \sigma_{opt}/2$. That means that the test would keep returning negative answers.

Phase II has an overall running time of $\mathcal{O}(n)$ since the first part of Phase II has linear time complexity, and 2-SAT can be implemented in time proportional to the number of conflicts [2] – which is linear.

### 1.2.3 Phase III: The Heuristics Come into Play

We enter this heuristical phase only if the test suggested in the previous section fails. Actually we could skip Phase III, return the result of this test, and would still keep the approximation guarantee due to Theorem 2. Instead of giving up when the test fails, we developed three methods which all have the same aim, that is to reduce the number of candidates per site to two without eliminating a candidate which might be essential to a solution of size $\sigma$. Because of the $\mathcal{NP}$–completeness of the decision version of the Map Labeling Problem, we cannot expect to always succeed. At last we attempt to solve the rest of the problem with the help of 2-SAT as described in the introduction.

**Heuristic** $H$ We randomly choose two of the possible four candidates left per site, before we hand them over to 2-SAT. To increase the probability of a choice which enables a solution, this process can be repeated in case of a negative answer. Three repetitions yield good results without prolonging the running time too much.
Since we look at a (hopefully small) fraction of the linear number of conflicts, we will only get a linear number of clauses, resulting in a running time of $\mathcal{O}$(n) for 2-SAT, and for this part of Heuristic $H$ as well.

**Heuristic** $I$ Here we run through all sites 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 sites which still have three active candidates. Then the remaining problem (consisting only of sites with exactly two active candidates) is handed over to 2-SAT.

This takes linear time.

**Heuristic $J$** For the third variant, we put all active candidates left into a priority queue according to the sum of all intersection areas of a candidate $p_i$. We then delete the minimum $p_i$ from the queue, and eliminate all candidates $q_j$ which overlap it, and the other active candidates $p_k$ belonging to $p$. If any of these decisions induces new ones according to the pattern used in Phase II, then these are made as well, before the next minimum is deleted from the queue. Naturally the sizes of the intersection areas, and the data structure, have to be updated accordingly. This process is repeated until either a site runs out of candidates ("no solution"), or no site has more than two of them left, so the remaining problem can be handed over to 2-SAT.
Using Fibonacci heaps to realize a priority queue that allows inserting and minimum deletions in $\mathcal{O}(\log n)$, and decreasing a key in constant time, this part of Heuristic $J$ can be implemented to run in time $\mathcal{O}(n \log n)$, since there is just a constant number of conflicts to be resolved per candidate we look at.

For Phase III of Approximation Algorithm $B$ we use Heuristic $I$ which turned out to deliver the best results in practice, confer Section 3.2.2.

# 2 Running Time Analysis

Phase I, II, and III can all be done in linear time, except for Heuristic $J$ where Phase III takes $\mathcal{O}(n \log n)$ time due to the use of a priority queue. Since we have to look at $\mathcal{O}(\log n)$ conflict sizes during the binary search for the best solution, Step 2 is in $\mathcal{O}(n \log n)$, and in $\mathcal{O}(n \log^2 n)$ for $J$. It dominates the total time complexity: Finding conflicts in Step 1 takes $\mathcal{O}(n \log n)$ as well, so Heuristics $H$ and $I$ are in $\mathcal{O}(n \log n)$, while $J$ is in $\mathcal{O}(n \log^2 n)$.

# 3 Experiments

## 3.1 Example Generators

We run the heuristics and the Approximation Algorithms $A$ on each of the examples produced by the four problem generators. For every size we average 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 from Saarbrücken. It shows exponential running time behaviour. For small examples it is very fast, for larger ones it is unreliable, so we were forced to introduce a time limit of five minutes after which we stopped the execution. Only very few of the largest *hard* and *dense* examples took less than this bound. This means that the exact algorithm is useless for large real time applications.

**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 $\sigma$ on a rectangle. We do this by randomly choosing points $p$ and then checking whether $\sigma p_1$ intersects with any of the $\sigma q_1$ 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. This method gives us a lower bound for the label size of the optimal solution. The size of the rectangle on which we place the squares is $\lfloor \alpha\sqrt{n} \rfloor \times \lceil \alpha\sqrt{n} \rceil$.

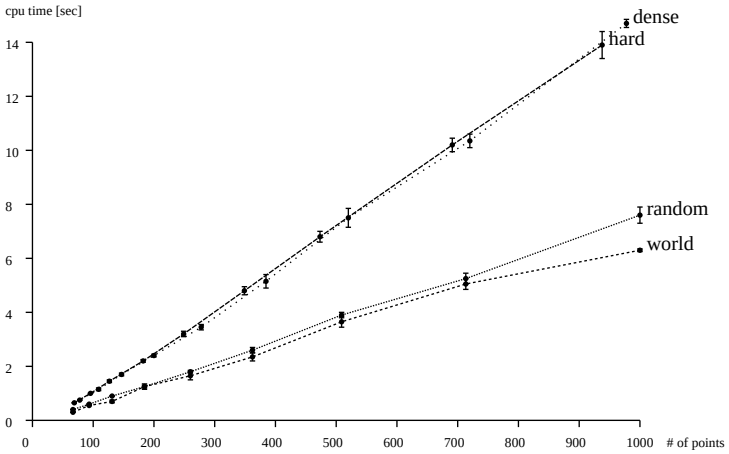


Figure 9: Running time of Approximation Algorithm $B$ on different example classes

$\alpha$ is a factor chosen such that the number of successfully placed squares is approximately $n$, the number of sites asked for.

**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 $\sigma$ on it. In random order, we try to place a square of edge length $\sigma$ 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. If it overlaps any of those chosen before, we try to place it into the same cell a constant number of times.

**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 a fixed centre point according to the $L_\infty$–norm, thus getting all those lying in a square around this extraction centre, where the size of the square depends on the number of points asked for. For our tests we chose five different centres; that of the map and those of its four quadrants in order to get results from different areas of the city with strongly varying point density. This is due to the fact that many of the holes were drilled during the construction of subway lines which are concentrated in the city centre, see Figure 16.

## 3.2 Results

We show the two classical kinds of plots; time and quality. Quality refers to the quotient of the solutions of the algorithms presented and that of the Exact Solver $X$. Time is measured in CPU time, which is sufficient since it is closely related to the number of square–square conflicts. This on the other hand determines the number of crucial steps, namely finding all conflicts once, and then extracting those valid for a certain label size in every step of the binary search; compare Graph 5 in Figure 15 with Figure 9 and 10.
The results both for time and quality are averaged only over those tests the exact solver managed within the time bound. The standard deviation is represented by the length of the vertical bars in each point of the result plots.

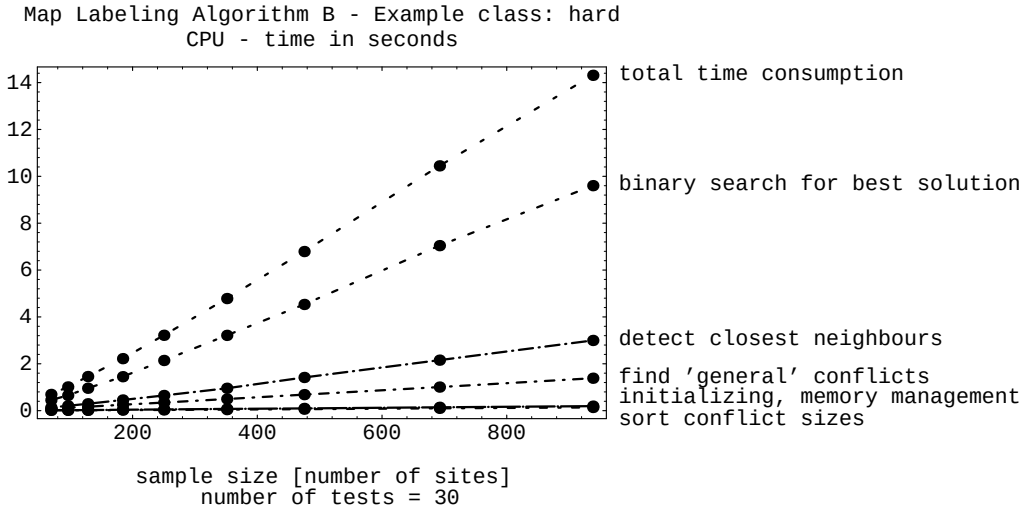


Figure 10: Running time of Approximation Algorithm $B$ on hard examples in detail

### 3.2.1 Experimental Running Time

In Figure 9 we plot the running times of Heuristic $I$ (which is used in Approximation Algorithm $A$) on the different example sets. $H$ is slightly faster, $J$ slightly slower. The test rows were performed on a Sun SPARC station 10. For a more detailed analysis of the running time of Heuristic $I$ on hard examples, see Figure 10. The time plots for other example classes look similar except for the gradient which mainly depends on the average amount of conflicts per candidate. It is lower for random or real world examples. Initially we had great difficulties in understanding the empirically linear running time until we noticed that due to the use of integer coodinates the sizes of many different conflicts were equal. Even for the largest dense and hard examples we did not get more than 200 different conflict sizes. Why? The heuristics calculate the label size at which all four candidates of a site contain other points. This is a natural upper bound for the optimal label size, so no conflict sizes beyond that have to be considered. Because of the way dense and hard examples are constructed, for large examples this upper bound is likely to be not much greater than 100, the constant side length of the squares which our generators used to throw on the plane. The fact that conflict sizes can appear in steps of half integers, explains why their number was limited to about 200.
When we changed the size of those squares to ten times the number of points of an example set, we received the linear number of conflict sizes we had expected, see Graph 4 of Figure 15. We have not yet found out why the number of conflicts summed up over all conflict sizes checked during the binary search (see Graph 5 of Figure 15), and the experimental running time still seem to be in $\mathcal{O}(n)$ instead of $\mathcal{O}(n \log n)$. It is also interesting to see that sorting which undoubtedly is in $\mathcal{O}(n \log n)$, has practically no influence on the running time for the problem sizes tested, see bottom most graph in Figure 10.

### 3.2.2 Approximation Quality

In Figures 11, 12, 13, and 14, the quality of the heuristics and Approximation Algorithm $A$ on the different example sets is plotted. On random and real world problems all heuristics yield extremely

good results. For an example, see Figure 17 and 18. On dense examples the differences between the algorithms become more clearly visible. Heuristic $I$ is the best, yielding results of very high average quality. Its behaviour on hard examples is still quite good but clearly becoming worse with an increasing number of points.
The quality of Algorithm $A$ is extremely bad on hard and dense, and still useless from a practical point of view on random and real world examples.

A remark on the examples for which $X$ did not give a result within the time bound: As mentioned above, we did not include these examples in the calculation of the quality plots. But using the upper bound provided by Approximation Algorithm $A$, and taking into consideration the typical quality of $A$, we found out that the behaviour of the heuristics on those examples does not differ significantly from that on the other examples.

# 4 Implementation

The code was written in C++, and we strongly took advantage of data structures and algorithms provided by LEDA [8]. The commands LEDA offers, helped a great deal to shorten and simplify the code. 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/map-labeling/`.

# Conclusion

Our experiences with the Map Labeling Problem and its solution can be summed up as follows: Formann and Wagner started with the purely mathematical formulation of the problem which was communicated to them by Kurt Mehlhorn from Saarbrücken, who received the problem from Rudi Krämer of the Amt für Informations- und Datenverarbeitung, Munich. They showed its $\mathcal{NP}$-hardness, and started developing an approximation algorithm when they heard of its practical relevance. They found one, analysed it, and showed its theoretical optimality. In theory the problem was solved perfectly.

Applied to real world data, the algorithm proved useless. Formann and Wagner used their insight into the problem structure gained during the design of $A$ and into the reasons for its practical failure, to develop Heuristic $H$ which produced satisfiably good results. Meanwhile Bettina Preis et. al. developed an Exact Algorithm $S$ which could solve small problems of up to about 80 points, which enabled us to estimate the quality of this heuristic. We improved $H$ to $I$, and to the even more sophisticated Heuristic $J$ which turned out to be a little worse than our champion $I$. Erik Schwarzenecker used our heuristical concept to enable the Exact Algorithm $X$ to solve larger problems in reasonable time. He also suggested the class of hard examples. Thus we were able to do a thorough experimental analysis of the quality of our heuristics [12]. We also owe thanks to Stefan Lohrum who helped us to make our heuristics accessible on the World Wide Web.

The next step was the attempt to unite the advantages of the theoretically optimal Approximation Algorithm $A$ with the strength of the heuristics — a much better practical performance. We came up with a new way of detecting conflicts which did not need $A$'s result any more, and with a new rule for eliminating candidates not needed for constructing a solution. The result of these ideas was the Approximation Algorithm $B$ which indeed shares $A$'s theoretical optimality, but delivers even better solutions to our problem sets that the best heuristic did before [13].

Our intense contacts with the practitioners were successful in two respects: We helped them to solve their problems, and they gave us the opportunity to get to know interesting related problems that come up in this context [14].

# 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, *Algorithms for Geometric Packing and Scaling Problems*, Dissertation, Fachbereich Mathematik und Informatik, Freie Universität Berlin (1992)

[4] 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

[5] M. Formann, F. Wagner, *An efficient solution to Knuth's METAFONT labeling problem*, Manuscript (1993)

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

[7] D. E. Knuth and A. Raghunathan, *The Problem of Compatible Representatives*, SIAM Journal on Discrete Mathematics **5** (1992) 422–427

[8] K. Mehlhorn and S. Näher, *LEDA: a platform for combinatorial and geometric computing*, Communications of the ACM **38** (1995) 96–102

[9] H. Imai, T. Asano, *Efficient Algorithms for Geometric Graph Search Problems*, SIAM J. Comput. **15** (1986) 478–494

[10] 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

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

[12] F. Wagner, A. Wolff, *Map Labeling Heuristics: Provably Good and Practically Useful*, Proceedings of the 11th Annual ACM Symposium on Computational Geometry (1995) 109–118

[13] F. Wagner, A. Wolff, *An Efficient and Effective Approximation Algorithm for the Map Labeling Problem*, Proceedings of the 3rd Annual European Symposium on Algorithms (1995) 420–433

[14] F. Wagner, A. Wolff, *Fast and Reliable Map Labeling*, Proceedings of the 9th International Symposium on Computer Science for Environment Protection (CSEP 95), Metropolis (1995) 667–675

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

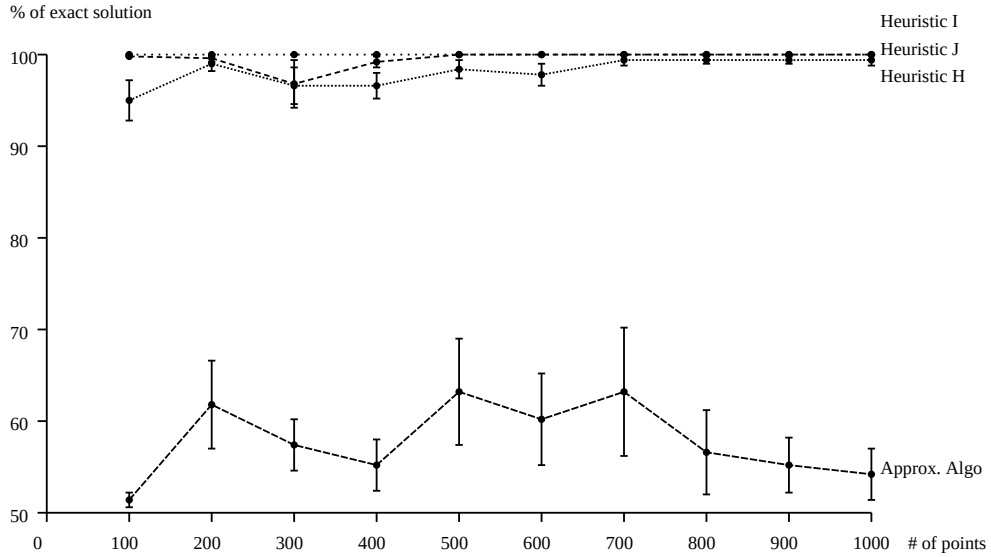


Figure 11: Quality of the algorithms on real world examples

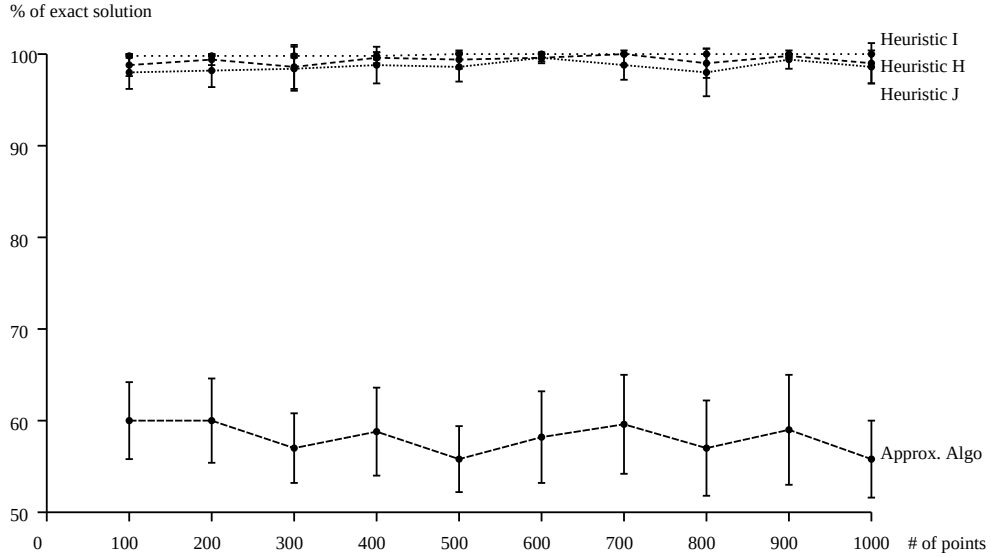


Figure 12: Quality of the algorithms on random examples

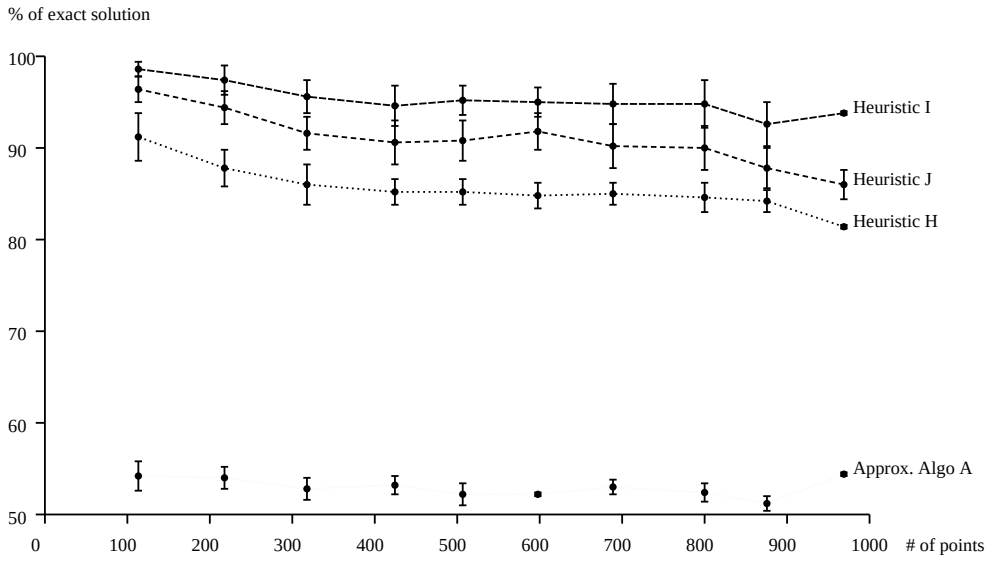


Figure 13: Quality of the algorithms on dense examples

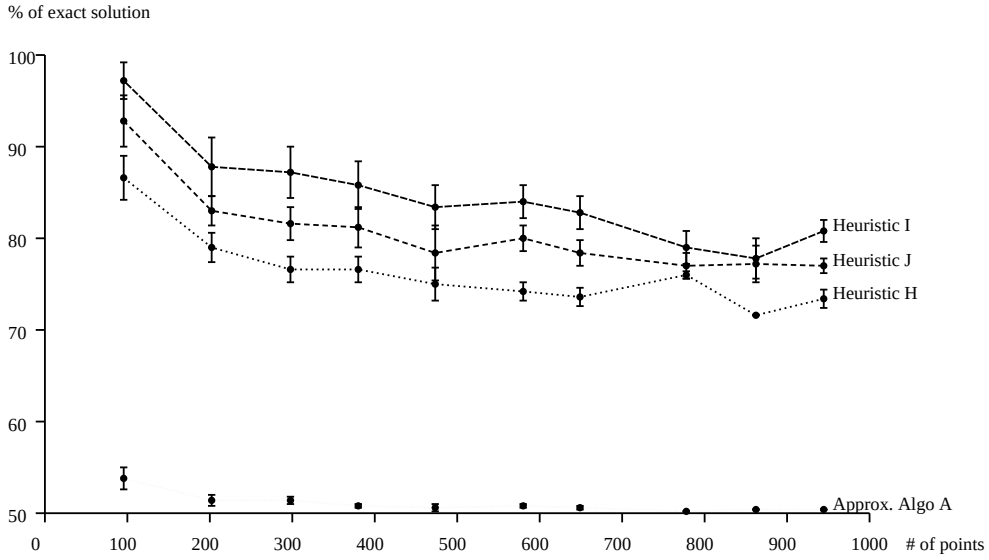


Figure 14: Quality of the algorithms on hard examples

```
Map Labeling Algorithm B - Example class: hard - number of tests = 30
1.    number of points looked at when detecting neighbours
2.    number of 'general' square - square conflicts
3.    number of conflict sizes including multiples
4.    size of array of conflict sizes used for binary search
5.    number of conflicts over all conflict sizes checked
6.    number of temporary assignments performed by 2-SAT
```

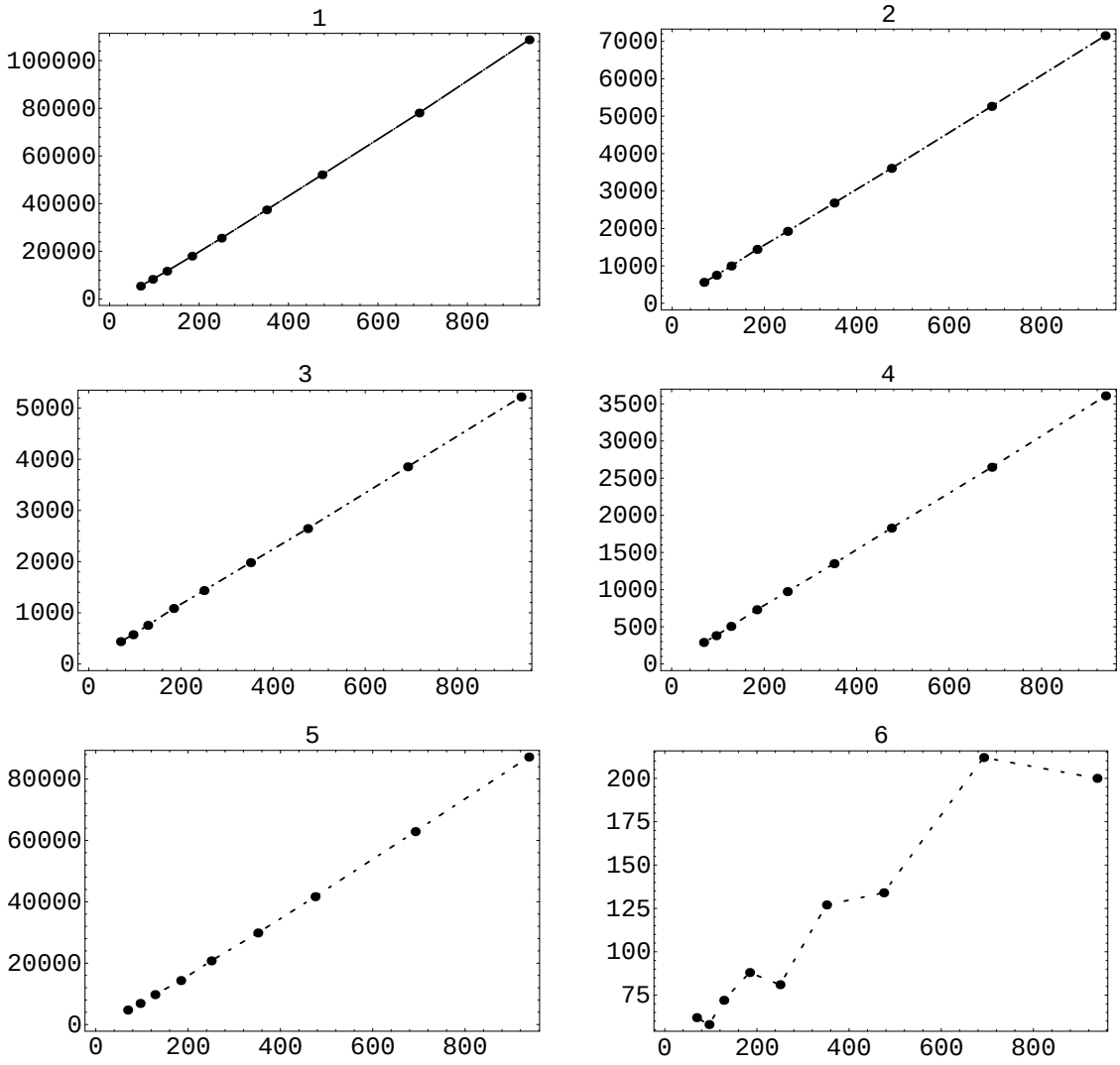


Figure 15: Average values of various counters which were installed to get a better understanding of the run time behaviour of Approximation Algoritm $B$.

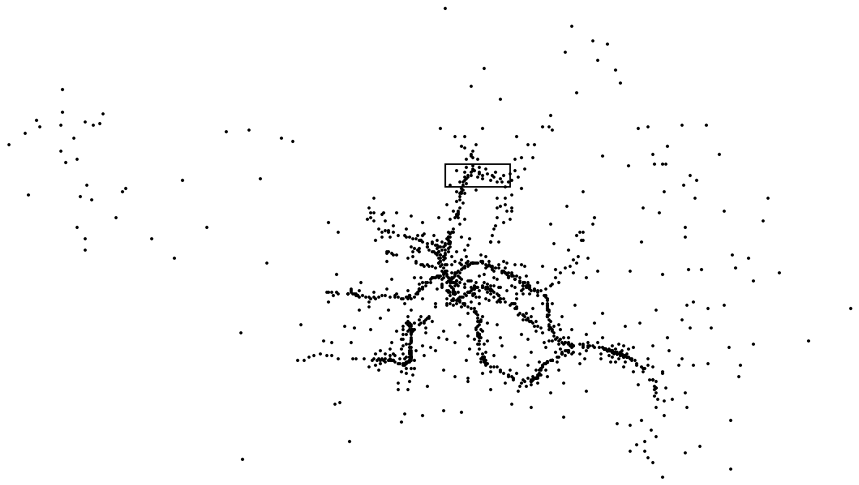

Figure 16: Map showing our sample data of approximately 1200 groundwater drillholes in Munich, and the section tested in Figures 17 and 18. There are no conflicts of interesting size between this section and the rest. The subway lines can be detected easily.

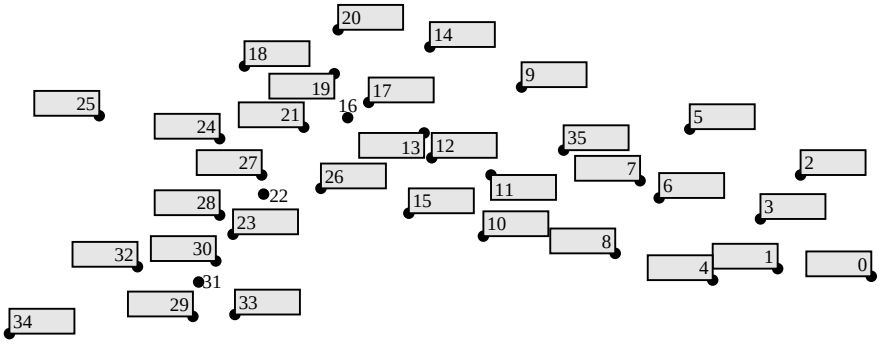


Figure 17: Solution of the program used by the authorities of the City of Munich before (label height 5000, 3 sites not labelled). It tries to maximize the number of sites labelled for a given size.

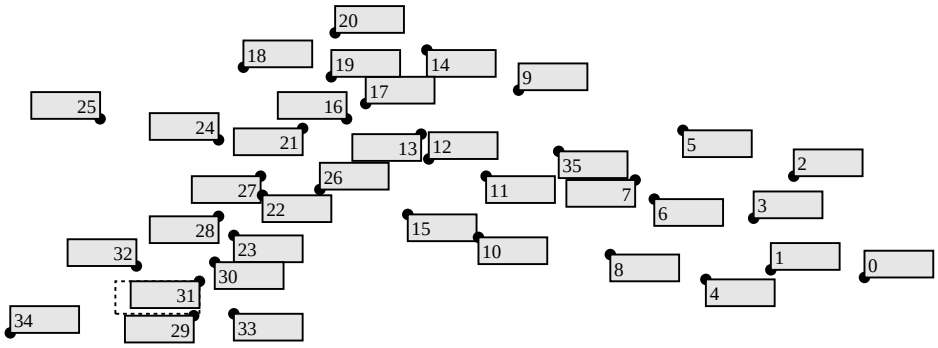


Figure 18: Solution produced by our heuristics (label height 5400, optimal).