An Efficient and Effective Approximation Algorithm for the Map Labeling Problem *

Frank Wagner † Alexander Wolff ‡

Freie Universität Berlin

Abstract

The *Map Labeling Problem* is a classical problem of cartography. There is an approximation algorithm A which is theoretically optimal: A has optimal running time and guarantees a label size of 50 percent of the maximum. Unfortunately A is useless in practice as it typically produces results that are intolerably far off the optimal size. On the other hand there is a heuristic with good practical results, which is used in real applications.

Recently a hybrid algorithm was suggested that first runs A and then uses its result to control the heuristic.

In this paper we integrate the two parts of the hybrid method into an efficient and effective approximation algorithm. In addition we include a strategy to improve the empirical quality of the results significantly. The resulting algorithm B

- 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 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 [I75] 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.

*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

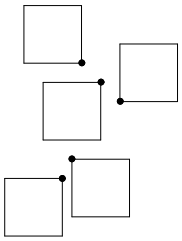

Figure 1: A valid labeling

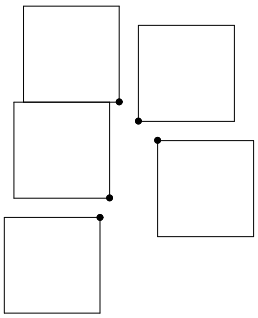

Figure 2: An optimal labeling for the example of Figure 1

The 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.

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 as follows:

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 Figure 1 and 2.

In [FW91] 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 no polynomial time approximation algorithm with a better quality guarantee exists, if $\mathcal{P} \neq \mathcal{NP}$. Related results were reported in [AIK89] and [IA86]. The running time of A is in $\mathcal{O}(n \log n)$. In [W94] it is shown that there is a matching lower bound on the running time.

A conceptually works as follows: We start 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 laterals, the formula is of 2-SAT

type, and can be evaluated in time proportional to its length [EIS76].

This works only because we make sure that no point has more than two squares left after the elimination phase. On the other hand, we often 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 there is a heuristical approach that shows very convincing results [WW95]. 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, three different methods H, I, and J are suggested to bring their number down to two. Method I is the winner in the experimental contest in [WW95].

In the same paper a hybrid algorithm is suggested 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 between overlapping squares. In addition to that, twice the upper bound is the width of a window which swept across the sites to detect these conflicts. This makes sure that the heuristics' runtime is provably optimal, too.
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 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 [WKA94] as a tool to label the output on the screen. With a very similar algorithmic approach Formann and Wagner [FW] were able to solve the so-called METAFONT labeling problem posed by Knuth and Raghunathan [KR92].

The main contribution of this paper is the integration of the two parts of the hybrid method 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. It is based on an algorithm to find closest neighbours, which was suggested in [F92]. 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 detection rule for squares which are not needed in a valid solution.

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. The best of the three, our new algorithm B, guarantees optimal approximation quality and runtime behaviour, and yields experimental results closer to the optimum than the best heuristic known so far.

2 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 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.*

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. σ_{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.

2.0 Structure

B differs from the hybrid algorithm suggested in [WW95] in the following way: It avoids to run A beforehand by finding the conflict sizes in Step 1 independently of A's result. In Phase II of Step 2, we introduce a new elimination rule that improves the performance substantially. The 2-SAT test of this phase maintains A's quality guarantee of 50 percent. Phase I and II are taken from Heuristic I.

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. 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.

2.1 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 [F92].

Definition 2 *Let* $\mathrm{Quad}(p_i) = \infty p_i$, *that is the i^{th} quadrant relative to p, $i \in \{1,2,3,4\}$. Note that this includes the border except p.*

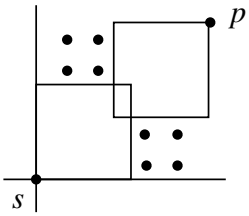

Figure 3: The conflict shown here between s_1 and p_3 is *not* important.

A conflict between a site s and one of its eight closest neighbours in one of the four quadrants relative to s is called important. *See Figure 3 for a counterexample. The size of such a conflict analogously is an important conflict size.*

A label *is the candidate of a site which has been chosen to be in the solution.*

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

We do not have to consider conflict sizes greater than σ_{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 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 $\mathcal{O}(n \log n)$ 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.

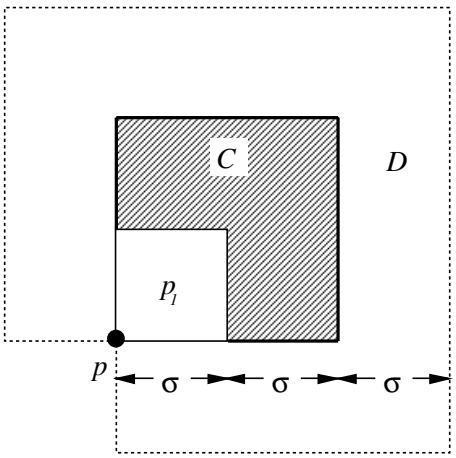

Figure 4:

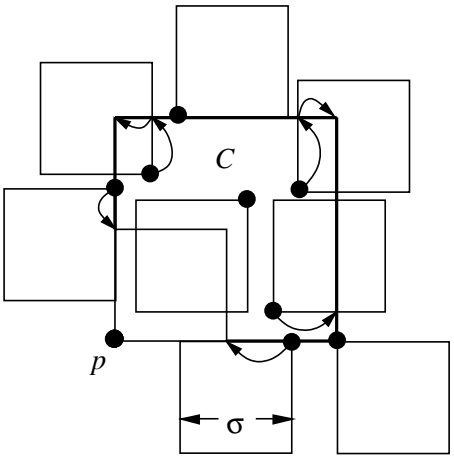

Figure 5:

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 σ_{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 σ 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 σ_{opt}, and it automatically is a solution for all smaller label sizes as well, and certainly for a subset of the sites.

It is clear that 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 σp_1, or none of the candidates of s^k would have a conflict of size σ with p_1. The area C in which the s^i can therefore lie, is shaded in Figure 4. Look at the area D which is where candidates of the sites in C can lie. The size of D is $15\sigma^2$, but the number of labels which can be placed in D, such that they are not in conflict with each other, is not 15 but 8, because the sites are restricted to C:

The idea is that we find out how many labels intersect the bold part of C's outline, and how many do not. We start with those sites whose labels contain a point of this line. We move them to a point of the outline which has either the same x- or y-coordinate — if the site is not already located on the outline. Since we move them "outwards", into a direction where no other label can lie, we avoid generating conflicts. We can then move sites along the line into its five corners. For those corners which are already contained by a label, take the site to which the label belongs, otherwise the closest site on either side of the outline. Now it is clear that apart from those in the corners not more than two sites can lie on the thick line, and another one in the interior of C, such that their labels do not overlap. Moving these labels on the one hand maximized the space in the interior of C, on the other it minimized the length of the intersection of each of the labels with the bold line. This means that in the original solution, too, not more than seven labels can have intersected the bold line, and only one can possibly have had an empty intersection.

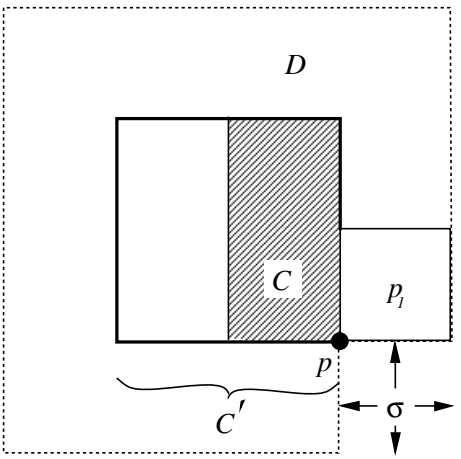

Figure 6:

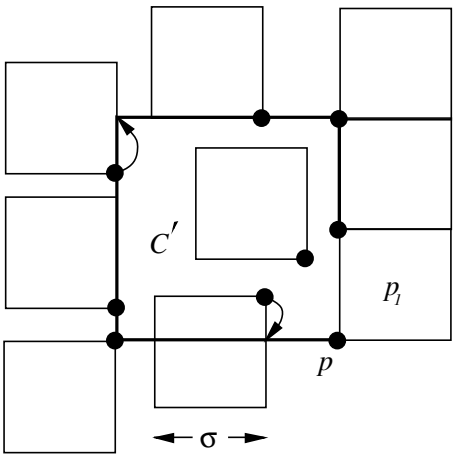

Figure 7:

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 6. Look at the area C' in Quad(p_2) of all points which have the same distance to p as points in C.

Push all sites to the outline of C' as above. Three sites can then be moved into the corners of the outline, one on each of its four sides, and again there can be only one site such that its candidate does not contain a point of the outline. This sums up to eight labels, see Figure 7.

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 $k > 8$. □

2.2 Check whether there is a solution for a given σ

2.2.1 Phase I: Preprocessing

We run through all the candidates p_i. If σp_i contains another site, we *eliminate* p_i, i. e. we will not consider it any more for our solution. 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 since the sum of the lengths of all conflict lists is linear, confer Section 2.1. For the same reason, the lists of overlap information just need linear space.

2.2.2 Phase II: Making Decisions

We run once through all points p. Heuristic I looks at the following three cases:

- 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.
- 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.

For B we add we a new rule. Refer to Figure 13 and 14 to see how this has improved the performance of B as compared to Heuristic I.

- 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 fulfill the same properties as p_i did before, i. e. do not intersect any other squares, or are

the last candidates of their sites. Before we look at the next site p, we do all the decisions 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 4 *If there is a solution of the current label size σ, then there is still a subset of the candidates left after Phase II, which forms a solution.*

Proof. This can be shown analogously to the proof of Corollary 5 in [WW95]. ❑

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 4 we also know that we have not yet returned "no solution" if there is one of size σ. 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 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 similarly bad as A's.

Definition 5 *A candidate p_i is called* σ–dead *if σp_i contains a site.*

Lemma 6 *Let σ be the current label size. Then after Phase II all sites have at most two candidates left which are not 2σ–dead.*

Proof. Suppose p had three such candidates, w. l. o. g. p_1, p_2, and p_4 as indicated in Figure 8. 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 σp_1. This means that p_1 (or possibly p_2 or p_4) should have already been chosen as part of the solution in Phase II, and p's other candidates eliminated. ❑

Theorem 7 *We can efficiently construct a solution of size σ from the candidates left after Phase II — if $\sigma \leq \sigma_{opt}/2$.*

Proof. If $\sigma \leq \sigma_{opt}/2$, then there must be a solution π of size 2σ. π is also a solution of size σ, since simultanously shrinking all squares reduces the number of conflicts. Now consider all sites p which have more than one candidate left after Phase II. Let $i = \pi(p)$. We want to show that p_i has not yet been eliminated. To see that, we just have to check the conditions under which this could have happened:

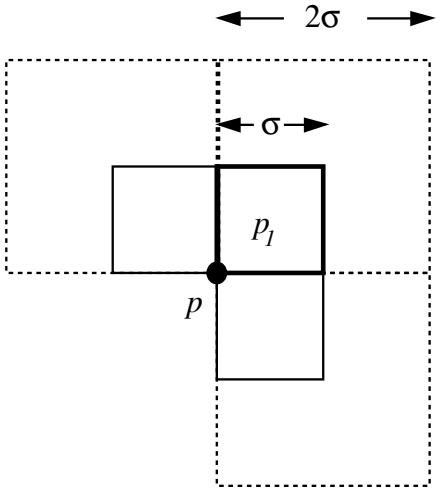

Figure 8:

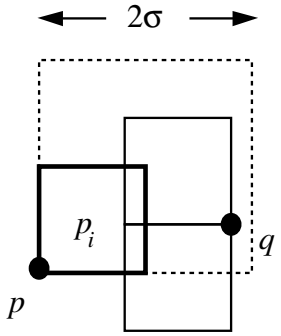

Figure 9:

- σp_i cannot contain any other site, because $2\sigma p_i$ does not.
- σp_i cannot overlap the last two candidates of some site q, since $2\sigma p_i$ would then contain q (see Figure 9).
- Suppose σp_i overlaps σq_j, and q_j is the last candidate of q (see Figure 10). Then $\pi(q) = k \neq j$, otherwise $2\sigma p_i$ and $2\sigma q_{\pi(q)}$ would overlap. The consequence of this is, that q_k also must have overlapped a candidate which is the last of its site, otherwise q_k and therefore p_i would not have been eliminated. Since the number of sites is finite, there must be a candidate r_m in this chain which overlaps a last candidate we have already looked at, say q_j. Then $2\sigma p_i$ and $2\sigma r_m$ must be disjoint, because they are part of solution π. At the same time σq_j must overlap σp_i and σr_m. This means that one corner of σq_j is contained in σp_i, and another one in σr_m. But then automatically the last two corners of σq_j must lie in $2\sigma p_i$ and $2\sigma r_m$ which are disjoint (see Figure 11). Obviously one of the corners of σq_j must be q. So q lies in either $2\sigma p_i$ or $2\sigma r_m$. This is a contradiction to the assumption that they are part of solution π.

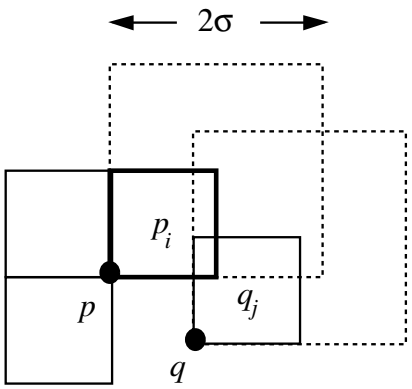

Figure 10:

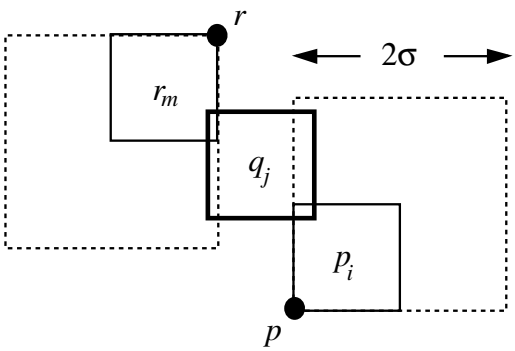

Figure 11:

We have just shown that if there is a solution of size 2σ, then all sites with active candidates have kept their candidate which is part of this solution all the way through Phase II. Lemma 6 tells us that after Phase II all sites have at most two squares left which are candidates for such a solution. So we can use 2-SAT on the set of active candidates which are not 2σ–dead. If 2-SAT finds a partial solution for those, we can just add all inactive candidates, and thus receive a solution for all sites, since the sets of active and inactive candidates do not overlap each other.

□

If however 2-SAT returns "no solution", then there cannot be any of size 2σ. 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 σ_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 σ with $\sigma > \sigma_0 > \sigma_{opt}/2$. That means that the test would keep returning negative answers.

Since the first part of Phase II and 2-SAT can be implemented in linear time [EIS76], Phase II has an overall running time of $\mathcal{O}(n)$.

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 descibed in the introduction. 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 the 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 from Saarbrücken. 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 very 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 $10n \times 10n$.

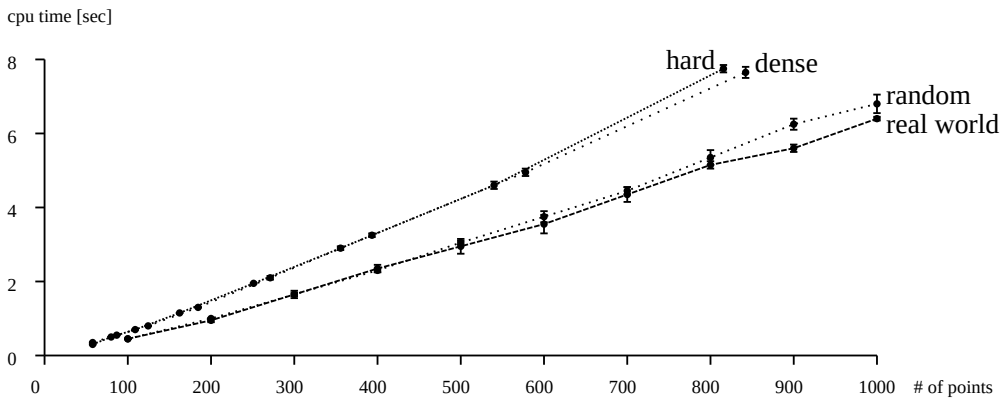

Figure 12: Running time of the Approximation Algorithm B on different example classes

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. 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 $\lfloor \alpha\sqrt{n} \rfloor \times \lceil \alpha\sqrt{n} \rceil$. α 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 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. 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 some centre point according to the L_∞–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 15.

3.2 Results

We show the two classical kinds of plots; time and quality. Quality here means the quotient of the solutions of the approximation algorithms and the exact solver. 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 σ in every step of the binary search.

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.

3.2.1 Running Time

In Figure 12 we plot the running times of the Approximation Algorithm B on the different example sets, which we measured on a Sun Sparc station 20. The plot shows a rather stable $\mathcal{O}(n)$-behaviour with very small standard deviation. So far we are not able to back the empirically linear running time by a theoretical analysis.

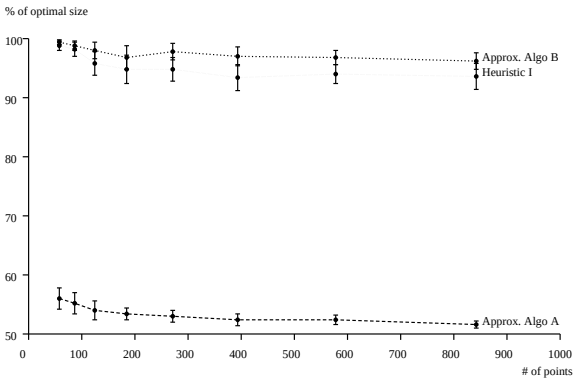

Figure 13: Quality of the algorithms on dense examples

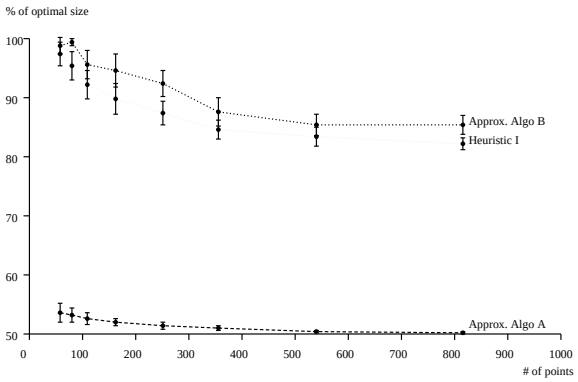

Figure 14: Quality of the algorithms on hard examples

3.2.2 Approximation Quality

The approximation quality is only plotted for dense and hard examples (see Figure 13 and 14), because on random and real world problems the performance of both B and Heuristic I was almost always 100 percent. For an example, see Figure 16 and 17.

A remark on the examples for which X did not give a result within the time bound: As mentioned above we did not include those in the calculation of the quality plots. But using the upper bound provided by the Approximation Algorithm A, and taking into consideration the typical quality of A, we found out that B's behaviour 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 [MN95]. 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/~awolff/html/labeling.html`.

References

[AIK89] 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

[EIS76] S. Even, A. Itai, A. Shamir, *On the complexity of Timetable and Multicommodity Flow Problems*, SIAM Journal on Computing **5** (1976) 691–703

[F92] M. Formann, *Algorithms for Geometric Packing and Scaling Problems*, Dissertation, Fachbereich Mathematik und Informatik, Freie Universität Berlin (1992)

[FW91] 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

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

[I75] E. Imhof, *Positioning Names on Maps*, The American Cartographer **2** (1975) 128-144

[KR92] D. E. Knuth and A. Raghunathan, *The Problem of Compatible Representatives*, SIAM Journal on Discrete Mathematics **5** (1992) 422–427

[MN95] K. Mehlhorn and S. Näher, *LEDA: a platform for combinatorial and geometric computing*, Communications of the ACM **38** (1995) 96–102

[IA86] H. Imai, T. Asano, *Efficient Algorithms for Geometric Graph Search Problems*, SIAM J. Comput. **15** (1986) 478–494

[KMPS93] 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

[W94] F. Wagner *Approximate Map Labeling is in* $\Omega(n \log n)$, Information Processing Letters **52** (1994) 161–165

[WW95] 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)

[WKA94] G. Weber, L. Knipping, H. Alt, *An Application of Point Pattern Matching in Astronautics*, Journal of Symbolic Computation **17** (1994) 321–340

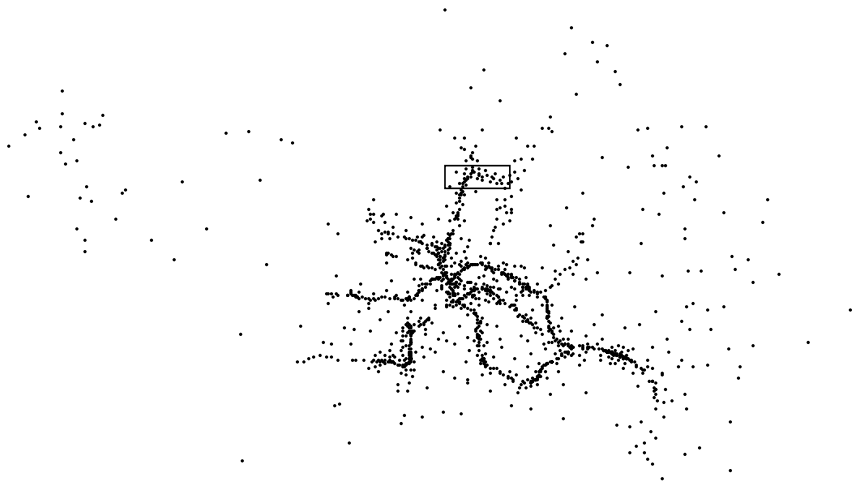

Figure 15: Map showing our sample data of approximately 1200 groundwater drillholes in Munich, and the section tested in Figures 16 and 17. There are no conflicts of interesting size between this section and the rest. The subway lines can be detected easily.

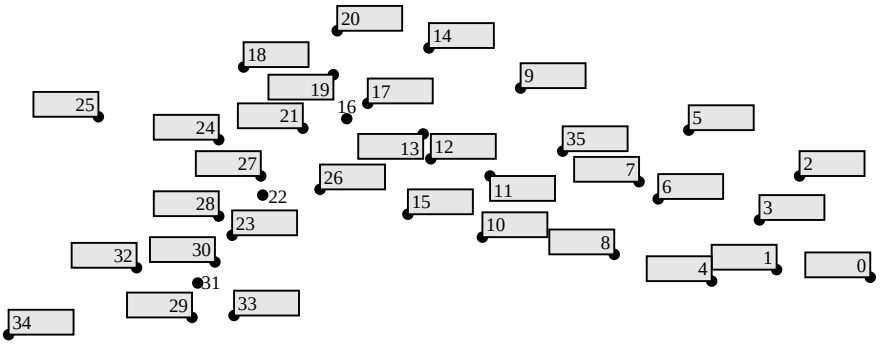

Figure 16: Solution of the program used by the authorities of the City of Munich before (label height 5000, 3 sites not labelled). It tries to maximize the number of sites labelled for a given size.

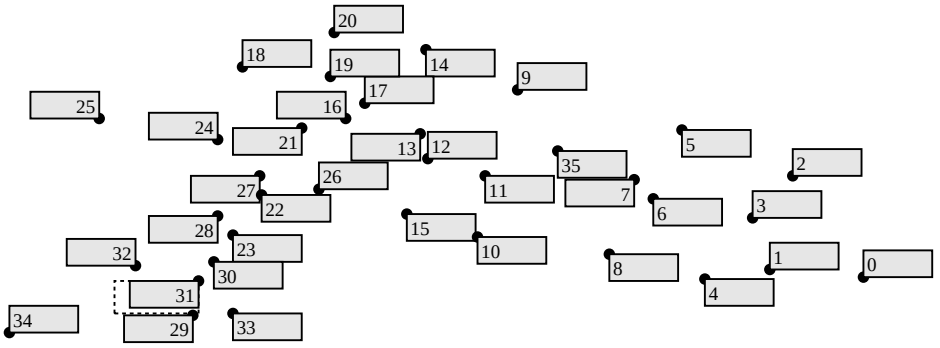

Figure 17: Solution produced by our heuristics (label height 5400, optimal).