A Combinatorial Framework for Map Labeling ⋆

Frank Wagner and Alexander Wolff

Institut für Informatik
Fachbereich Mathematik und Informatik
Takustraße 9, D-14195 Berlin
Freie Universität Berlin
{awolff,wagner}@inf.fu-berlin.de

Abstract. The general map labeling problem consists in labeling a set of sites (points, lines, regions) given a set of candidates (rectangles, circles, ellipses, irregularly shaped labels) for each site. A map can be a classical cartographical map, a diagram, a graph or any other figure that needs to be labeled. A labeling is either a complete set of non-conflicting candidates, one per site, or a subset of maximum cardinality. Finding such a labeling is NP-hard.
We present a combinatorial framework to attack the problem in its full generality. The key idea is to separate the geometric from the combinatorial part of the problem. The latter is captured by the conflict graph of the candidates and by rules which successively simplify this graph towards a near-optimal solution.
We exemplify this framework at the problem of labeling point sets with axis-parallel rectangles as candidates, four per point. We study competing algorithms and do a thorough empirical comparison. The new algorithm we suggest is fast, simple and effective.

1 Introduction

Map labeling is a classical problem of cartography. Since the first attempts of automating map production, an abundance of approaches has been applied to this problem: expert systems, 0-1 integer programming, and simulated annealing to name only a few. Map labeling is usually divided into point, line and area labeling. In recent years, especially the problem of point labeling has achieved some attention in the algorithms' community. Two interesting sub-problems have been studied. In both cases, an instance consists of a set of sites and a set of label candidates for each site.

1. *The Label Size Maximisation Problem*: Find the maximum factor σ such that each site gets a label stretched by this factor and no two labels overlap. Compute the corresponding *complete* label placement.

⋆ This work was supported by the Deutsche Forschungsgemeinschaft (DFG) under grants Wa 1066/1-1, 3-1, and 3-2

2. *The Label Number Maximisation Problem*: Find a maximum subset of the sites, and for each of these sites a label from its set of candidates, such that no two labels overlap.

The decision versions of both problems are NP-hard [5, 6]; size maximisation only if the sites have more than two label candidates.

There is an approximation algorithm which maximises the size of uniform axis-parallel square labels. It is optimal in respect to both, its approximation factor of $1/2$ and its running time of $O(n \log n)$ [5, 15]. For the same problem, there is an algorithm which keeps the theoretical optimality of the approximation algorithm, but performs close to optimal in practice [16].

For square labels of arbitrary orientation and for circular labels there are approximation algorithms maximising label size, again under the restriction that all labels are uniform, i.e. of equal size [4].

The complexity of the label *number* maximisation problem is quite different. Even for axis-parallel rectangular labels of arbitrary height and width, there is an approximation algorithm, however with a ratio of just $1/O(\log n)$ [1]. If the label height (or width) is fixed though, the problem can be approximated by a factor of $1/2$ in $O(n \log n)$ time – even if the labels are allowed to touch their sites *anywhere* on the rectangle boundary [14]. For maximising the *size* of uniform rectangular labels, this approximation factor is optimal, but for maximising the *number* of fixed-height labels, polynomial time approximation schemes [1, 14] are known.

Very recently, Kakoulis and Tollis suggested a more general approach to labeling [10]. They compute the candidate conflict graph and its connected components. Then they use a simple heuristic to split these components into cliques. Finally they construct a bipartite "matching graph" whose nodes are the cliques of the previous step and the sites of the instance. In this graph, a site and a clique are joined by an edge if the clique contains a candidate of the site. A maximum cardinality matching yields the labeling. Due to the last step, their algorithm takes $O(n^{2.5})$ time.

The algorithm for label number maximisation we present in this paper has the following advantages compared to previously suggested algorithms. Our algorithmic approach does not depend on the shape of labels, can be applied to point, line, or area labeling (even simultaneously) if a finite set of label candidates has been precomputed for each site, is easy to implement, runs fast, and returns good results in practice.

The input to our algorithm is the conflict graph of the label candidates. The algorithm is divided into two phases similar to the first two phases of the algorithm for label size maximisation described in [16]. In phase I, we apply a set of rules to all sites in order to label as many of them as possible and to reduce the number of label candidates of the others. These rules do not destroy a possible optimal placement. Then, in phase II, we heuristically reduce the number of label candidates of each site to at most one.

For the rules we apply in phase I, there is a more general concept discussed in the artificial intelligence community under the name *constraint satisfaction* which was independently introduced into the discrete mathematics community by Knuth and Raghunathan under the name *problem of compatible representatives* [11]. The difference of our approach to that of the artificial intelligence community is that we try to maximise the number of variables (sites) with a conflict-free assignment, while their objective is to either list *all* assignment tuples without conflicts [12], to minimise the number of conflicts [7], or to find the maximum weighted subset of constraints which still allows an assignment.

This paper is structured as follows. In Section 2 we describe our ideas within the framework of constraint satisfaction. In Section 3 we specialise this general concept to the context of point labeling and give the details of our algorithm. Section 4 describes our experiments. We compare our algorithm to simulated annealing and a greedy method.

Part of the examples, on which we do the comparison, are benchmarks that were already used in [16]. We added examples for placing rectangular labels of varying size, both randomly generated and from real world data. The algorithm is used to label shops on-line on a tourist map of Berlin and by the city authorities of Munich to label drill-hole maps. All example generators, real world data and algorithms are available on the World Wide Web[1].

Our tests differ from experiments performed by other researchers [8, 3, 2, 14, 10] in that we included example classes where we could measure our results with respect to tight bounds on the optimal solution.

2 Framework

A constraint satisfaction problem (CSP) is defined as follows. Given a set of n variables $v_1, \dots, v_n$, each associated with a domain D_i and a set of relations constraining the assignment of subsets of the variables, find all possible n-tuples of variable assignments that satisfy the relations [12]. Often variable domains are restricted to discrete finite sets, and only binary relations are considered.

Graph colouring is a special case of a CSP where the variables are nodes, the domains a given set of colours, and binary relations express the fact that a node cannot have the same colour as any of its neighbours. Since graph colouring is NP-complete, one cannot expect to solve general CSPs in polynomial time. For this reason, the class of network consistency algorithms has been invented. These algorithms use local arguments to exclude values from the domain of a variable that cannot be part of a global solution. Network consistency algorithms can be seen as a preprocessing step to backtracking since they often reduce the search space very effectively.

An m-consistency algorithm removes all inconsistencies among m of the given n variables. In the special cases of $m = 1$, 2, and 3, these algorithms are called node, arc, and path consistency algorithms, respectively. Mackworth and Freuder have shown that arc consistency can be achieved in $O(a^3k)$ where a is the size of the variable domains and k the number of binary relations [12].

[1] Refer to `http://www.inf.fu-berlin.de/map-labeling/`

This framework can be used nearly one-to-one for attacking the label *size* maximisation problem. When maximising simultaneously the sizes of all labels, one can do a binary search on *conflict sizes*, i.e. label sizes for which label candidates start to touch. For each conflict size, one then tries to find a complete labeling. Obviously, a site can be seen as a variable, the set of label candidates of a site then corresponds to the variable domain and intersections between label candidates are the constraining binary relations. Instead of computing *all* satisfying variable assignments, finding *one* is usually sufficient in the map labeling context. This allows to reduce the search space dramatically since a variable can immediately be assigned an unconstrained value from its domain if there is such a value. The algorithm for label size maximisation suggested in [16] uses this.

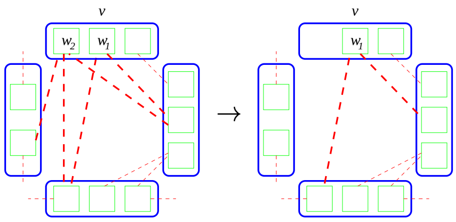

Fig. 1. applying rule G1

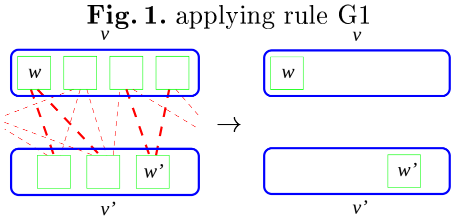

Fig. 2. applying rule G2

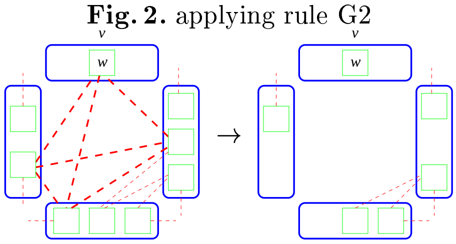

Fig. 3. applying rule G3

When maximising the number of labeled sites, label sizes are fixed and one cannot give up and try a smaller label size as soon as it turns out that there is no complete labeling for the current label size. Systems where one cannot expect to find a *complete solution*, i.e. a non-conflicting variable assignment, are called *over-constrained systems*. In such systems one has to be content with imperfect solutions. Most effort in the CSP community has been directed to finding solutions that violate as few constraints as possible [7, 9]. When labeling maps, such violations would result in label overplots and thus poor legibility. It would be possible to take the output of an algorithm which mimimises the number of violated constraints and then do some postprocessing. In order to get rid of the violations, one could drop a subset of the variables and resign from labeling the corresponding sites. Unfortunately finding the smallest subset of variables such that all constraints between the remaining variables are satisfied, corresponds to the vertex cover problem and is thus NP-hard.

A related problem, Max-CSP, has also been investigated. There, one is interested in finding a maximum (weighted) subset of the constraints such that there is an assignment that satisfies them all. In order to reduce label number maximisation to Max-CSP, one adds a new value $\triangle$ to the domain of each variable. $\triangle$ has a unary constraint of low weight; i.e. it only constrains itself. A variable which is assigned $\triangle$ then corresponds to an unlabeled site in our setting. For general Max-CSP, however, even arc consistency is NP-hard [13].

Therefore we took a different approach. We developed a weak form of local consistency for our context and then wrote a map labeling algorithm based on this concept. The algorithm first establishes local consistency. Then it repeatedly makes a heuristical decision and restores local consistency until each site is either labeled or known to constrain too many other sites and therefore not labeled.

From now on an instance will still consist of a set of variables $v_1, \ldots, v_n$, their domains and binary constraints excluding pairs of variable values, but for us an *optimal solution* is a violation-free assignment for as many variables as possible. The *size* of an optimal solution is the number of variables in the assignment.

We present a set of rules which, applied exhaustively, achieve a weak form of local consistency. We refer to it as weak since we only proof that applying these rules does not destroy an optimal solution. It would be interesting to see whether they are complete in the sense that after their application no domain of a variable can be further reduced without risking to reduce the size of the optimal solution if only subsets of 1, 2, or k variables are taken into account. This would correspond to node, arc, and k-consistency for classical CSP.

In Figure 1 to 3 typical situations before and after the application of a rule are depicted. The domain of a variable is represented by a rectangle with round corners, the values of a variable are dotted boxes, and the fact that two values of different variables exclude each other is marked by a dashed line connecting the corresponding boxes. Bold dashed lines mean that the corresponding constraints are responsible for the application of the depicted rule. Dashed lines not ending in a box indicate that the value from which they are emanating might constrain further variables.

(G1) If a variable v has two values w_1 and w_2, and all values constrained by w_1 are also constrained by w_2, then set $D_v = D_v - \{w_2\}$, see Figure 1.
• Special case: If a variable v has a value w without constraints, then set $D_v = \{w\}$.

(G2) If there is a subset V of variables $v_1, \ldots, v_l$, each with a value w_i such that w_i only constrains variables in V but does not exclude any w_j for $i \neq j$, then set $D_{v_i} = \{w_i\}$ for $i = 1, \ldots, l$.
• Special case: If a variable v has a value w that only constrains a variable v', and v' has a value w' which constrains only v and does not exclude w, then set $D_v = \{w\}$ and $D_{v'} = \{w'\}$, see Figure 2.

(G3) If the domain D_v of a variable v consists only of one value w, and the values $w_1, \ldots, w_l$ excluded by w belong to different variables $v_1, \ldots, v_l$ and pairwise exclude each other (i.e. if $w, w_1, \ldots, w_l$ form a clique in the constraint graph), then set $D_{v_i} = D_{v_i} - \{w_i\}$ for $i = 1, \ldots, l$, see Figure 3.

Note that if V is the set of all variables in the instance, then G2 yields a complete solution – if there is one. We show that our rules are conservative in the following sense.

Corollary 1. *If there is an optimal solution of size k for the given instance before applying any of the rules G1 to G3, then there is still a an optimal solution of size k after applying one of these rules.*

Proof. Assume to the contrary that the size of the optimal solution decreases after we remove a value u from the domain D_v of its variable v. Then every optimal solution π before the elimination must have assigned u to v. Consider the circumstances under which u can be removed.
– There is a value $w \neq u$ of v which excludes only a subset of the values of u (see rule G1). But then we could replace u by w in π.
– There is a subset V of variables $v_1, v_2, \ldots, v_l$, each with a value w_i ($w_1 \neq u$) such that w_i only constrains variables in V but does not exclude any w_j for $i \neq j$ (see G2). Then we could replace $\pi(v_i)$ by w_i for $i = 1, \ldots, l$ without reducing the size of π.
– The variables $v_1, \ldots, v_l$ constrained by u each have a value w_i such that $u, w_1, \ldots, w_l$ pairwise exclude each other, and there is a w_j among the w_i's which does not exclude any other value and which is the only value in the domain of its variable, i.e. $D_{v_j} = \{w_j\}$ (see G3). Then we could replace the assignment of u to v by that of w_j to v_j, again without reducing the size of π.

3 Algorithm

Our algorithm consists of two phases. In phase I, we apply a set of rules to all sites in order to label as many of them as possible and to reduce the number of label candidates of the others. These rules don't destroy a possible optimal placement. Then, in phase II, we heuristically reduce the number of label candidates of each site to at most one.

Phase I

In the first phase, we apply all of the following rules to each of the sites. Let p_i be the candidate label of site p in position i. For each of the rules we supply a sketch of a typical situation in the context of point labeling with four rectangular label candidates per point. In the pictures, we shaded the candidates which are chosen to label their site, and we used dashed edges to mark candidates which are eliminated after a rule's application.

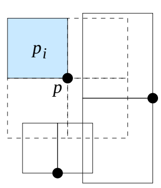

Fig. 4. Rule L1

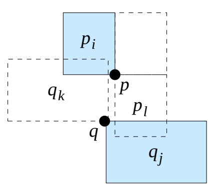

Fig. 5. Rule L2

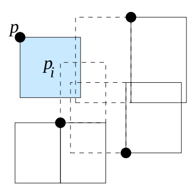

Fig. 6. Rule L3

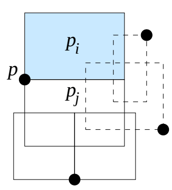

Fig. 7. Rule L4

(L1) If p has a candidate p_i without any conflicts, declare p_i to be part of the solution, and eliminate all other candidate labels of p, see Figure 4.
(L2) If p has a candidate p_i which is only in conflict with some q_k, and q has a candidate q_j ($j \neq k$) which is only overlapped by p_l ($l \neq i$), then add p_i

and q_j to the solution and eliminate all other candidates of p and q, see Figure 5.

(L3) If p has only one candidate p_i left, and the labels overlapping p_i form a clique, then declare p_i to be part of the solution and eliminate all labels which overlap p_i, see Figure 6.

(L4) If p has two neighbouring label candidates left, say p_i and p_j, and p_i is only overlapped by labels which also overlap p_j, then eliminate all labels which overlap both of them and put p_i in the solution, see Figure 7.

Rule L4 can only be applied if p_i and p_j completely share a vertical (horizontal) edge, if all labels have the same width (height) and if they are not allowed to obstruct a site. Otherwise an optimal solution can be destroyed. Due to these restrictions, we have not used rule L4 in our experiments.

We want to ensure that the rules are applied exhaustively. Therefore, after eliminating a candidate, we check whether they can be applied in its neighbourhood, i.e. to the sites of the eliminated candidate or its conflict partners.

Since the rules L1 to L4 are restrictions of the more general rules G1 to G3, it is clear that they also have the property that if there is a solution of size k (i.e. k sites can be labeled) before applying any of the rules, then this is also the case after the rule's application.

Phase II

If we have not managed to reduce the number of candidates to at most one per site in the first phase, then we must do so in phase II. Since phase II is a heuristic, we are no longer able to guarantee optimality. The heuristic is conceptionally simple and makes the algorithm work well in practice, see Section 4. The intuition is to start eliminating troublemakers where we still have a choice. Spoken more algorithmically, we go through all sites p which have the maximum number of candidates, and delete the candidate with the maximum number of conflicts among the candidates of p. This process is repeated until each site has at most one candidate left. These candidates then form the solution.

As in phase II, after eliminating a candidate, we check whether our rules can be applied in its neighbourhood.

Analysis

In order to simplify the analysis of the running time, we assume that the number of candidates per site is constant. Then it is easy to see that in phase I, rule L1 and L2 can be checked in constant time for each site. We use a stack to make sure that our rules are applied exhaustively. After we have applied a rule successfully and eliminated a candidate, we put all sites in its neighbourhood on the stack and apply the rules to these sites. Since a site is only put on the stack if one of its candidates was deleted or lost a conflict partner, this part of phase I sums up to $O(n + k)$ time, where n is the number of candidates and k the number of pairs of intersecting candidates in the instance, i.e. the number of edges in the candidate conflict graph. For rule L3, we have to check whether a candidate is intersected by a clique. In general, this takes time quadratic in the number of conflict partners. Falling back on geometry, however, can help to cut this down. In the case of axis-parallel rectangles for instance, a clique can be detected in linear time by testing whether the intersection of all conflicting rectangles is not empty. A simple charging argument then yields $O(k^2)$ time for checking L3.

The time needed for checking L4 is subsumed by that of L3. For both rules, checking can be done in constant time if we apply them only to candidates with less than a constant number of conflicts. This makes sense since it is not very likely that the neighbourhood of a candidate with many conflicts is a clique. In this case, phase I can be done in $O(n+k)$ time.

In phase II, we can afford to simply go through all sites sequentially and check whether they have the current maximum number of candidates. If so, we go through the candidates of the current site and determine the one with the maximum number of conflicts. The amount of time needed to delete this candidate and apply our rules has already been taken into account in phase I. Thus phase II needs only linear extra time.

Putting things together, we get an $O(n+k^2)$ algorithm if rule L3 can be checked in linear time, and an $O(n+k)$ algorithm if we allow only constant effort for checking L3 and L4. In our experiments, we have not bounded this effort, yet this part of the algorithm showed a linear-time behaviour. Finally, for axis-parallel rectangular labels, the conflict graph can be determined in $O(n \log n)$ time.

4 Experiments

We compare our algorithm to simulated annealing and a greedy method.

The simulated annealing algorithm we used relies on the experiments performed by Christensen et al. and follows their suggestions for the initial configuration, the objective function, a method for generating configuration changes, and the annealing schedule [3]. In order to save time, we allowed only 30 instead of the proposed 50 temperature stages in the annealing schedule. This did not seem to influence the quality of the results.

The greedy algorithm picks repeatedly the leftmost label (i.e. the label whose right edge is leftmost), and discards all candidates that intersect the chosen label. The algorithm runs in $O(n \log n)$ time and has an approximation factor of $1/(H+1)$, where H is the ratio of the greatest and the smallest label height [14].

We run our algorithm and those described above on the following instance classes. Figures 17 to 24 depict an example of each of these classes.

RandomRect. We choose n points uniformly distributed in a square of size $25n \times 25n$. To determine the label size for each site, we choose the length of both edges independently under normal distribution, take its absolute value and add 1 to avoid non-positive values. Finally we multiply both label dimensions by 10.

DenseRect. Here we try to place as many rectangles as possible on an area of size $\alpha_1\sqrt{n} \times \alpha_1\sqrt{n}$. α_1 is a factor chosen such that the number of successfully placed rectangles is approximately n, the number of sites asked for. We do this by randomly selecting the label size as above and then trying to place the label 50 times. If we don't manage, we select a new label size and repeat the procedure. If none of 20 different sized labels could be placed, we assume that the area is well covered, and stop. For each rectangle we placed successfully, we return its height and width and a corner picked at random. It is clear that all points obtained this way can be labeled by a rectangle of the given size without overlap.

RandomMap and **DenseMap** try to imitate a real map using the same point placement methods as RandomRect and DenseRect, but more realistic label sizes. We assume a distribution of 1:5:25 of cities, towns and villages. After randomly choosing one of these three classes according to the assumed distribution, we set the label height to 12, 10 or 8 points accordingly. The length of the label text then follows the distribution of the German Railway station names (see below). We assume a typewriter font and set the label length to the number of characters times the font size times 2/3. The multiplicative factor reflects the ratio of character width to height.

VariableDensity. This example class is used in the experimental paper by Christensen et al. [3]. There the points are distributed uniformly on a rectangle of size 792×612. All labels are of equal size, namely 30×7. We included this benchmark for reasons of comparability.

HardGrid. In principle we use the same method as for Dense, that is, trying to place as many labels as possible into a given area. In order to do so, we use a grid of $\lfloor \alpha_2 \sqrt{n} \rfloor \times \lceil \alpha_2 \sqrt{n} \rceil$ cells with edge lengths n. Again, α_2 is a factor chosen such that the number of successfully placed squares is approximately n. In a random order, we try to place a square of edge length n into each of the cells. This is done by randomly choosing a point within the cell and putting the lower left corner of the square on it. If it overlaps any of the squares placed before, we repeat at most 10 times before we turn to the next cell.

RegularGrid. We use a grid of $\lfloor \sqrt{n} \rfloor \times \lceil \sqrt{n} \rceil$ squares. For each cell, we randomly choose a corner and place a point with a small constant offset near the chosen corner. Then we know that we can label all points with square labels of the size of a grid cell minus the offset.

MunichDrillholes. The municipal authorities of Munich provided us with the coordinates of roughly 19,400 ground water drillholes within a 10 by 10 kilometer square centered approximately on the city center. From these sites, we randomly pick a center point and then extract a given number of sites closest to the center point according to the L_∞–norm. Thus we get a rectangular section of the map. Its size depends on the number of points asked for. The drillhole labels are abbreviations of fixed length. By scaling the x-coordinates, we make the labels into squares and subsequently apply an exact solver for label size maximisation. This gives us an instance with a maximal number of conflicts which can just be labeled completely.

In addition to these example classes, we tested the algorithms on the point sets depicted in Figure 25 and 26.

Results

We used examples of 250, 500, . . ., 3000 points. For each of the example classes and each of the example sizes, we generated 30 files. Then we labeled the points in each file with axis-parallel rectangular labels. We used four label candidates per sites, namely those where one of the label's corners is identical to the site. We allowed labels to touch each other but not to obstruct sites.

The graphs in Figures 9 to 16 show the performance of the three algorithms. The average example size is shown on the x-axis, the average percentage of labeled sites is depicted on the y-axis. Note that we varied the scale on the y-axis from graph to graph in order to show more details. The worst and the best performance of the algorithms are indicated by the vertical bars.

The example classes are divided into two groups; those that have a complete labeling and those that have not. For the former group, the percentage of labeled points expresses directly the performance ratio of an algorithm. For examples of the latter group, which consists of RandomRect, RandomMap and VariableDensity, there is only a very weak upper bound for the size of an optimal solution, namely the number of labels needed to fill the area of the bounding box of the instance completely. Thus for VariableDensity at most 2539 points can possibly be labeled. Experiments we performed with an exact solver on examples of up to 200 points showed that on an average about 85% of the points in an instance of RandomRect and usually less than 80% in the case of RandomMap can be labeled. Other than VariableDensity, these classes are designed to keep their properties with increasing point number. This is reflected by the fact that the algorithms' performance was nearly constant on these examples. It might be worth to note that we used the same set of rules as in phase I of our algorithm to speed up the exact solver.

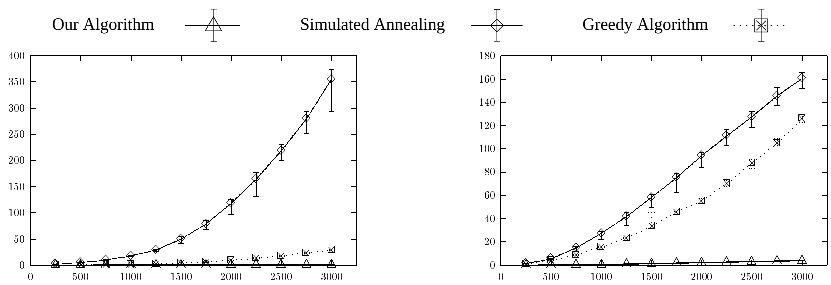

Fig. 8. MunichDrillholes (left) and VariableDensity: point number versus running time

For all examples, which have a complete labeling, our algorithm labeled between 95 and 100% of the points. Experiments on small examples hint that the same holds for larger RandomRect and RandomMap examples. The greedy algorithm performed well given that it makes its decisions only based on local information. It was outperformed clearly by our algorithm in all example classes but one. On regular grid data, it achieved 100%, followed very closely by the other algorithms. For some of the example classes, simulated annealing outperformed our algorithm by one to two percent. However, in order to achieve similarly good results, simulated annealing needed much longer.

In Figure 8 we present the running times of our implementations in CPU seconds on a Sun UltraSparc. We show the two example classes where simulated annealing performed most slowly and fastest. Our implementation of the greedy algorithm is simply based on lists and uses brute force to find the next leftmost label candidate. Given heaps and priority search trees, it would run faster. Our implementation of simulated annealing seems to be slower by a factor of 2 to 3 than that of Christiansen et al. [3]. This difference in running time may be due to the machines on which the times were measured.

References

[1] P. Agarwal, M. van Kreveld, and S. Suri. Label placement by maximum independent set in rectangles. In *Proceedings of the 9th Canadian Conference on Computational Geometry*, pages 233–238, 1997.

[2] J. Christensen, S. Friedman, J. Marks, and S. Shieber. Empirical testing of algorithms for variable-sized label placement. In *Proceedings of the 13th Annual ACM Symposium on Computational Geometry*, pages 415–417, 1997.

[3] J. Christensen, J. Marks, and S. Shieber. An empirical study of algorithms for point-feature label placement. *ACM Transactions on Graphics*, 14(3):203–232, 1995.

[4] S. Doddi, M. V. Marathe, A. Mirzaian, B. M. Moret, and B. Zhu. Map labeling and its generalizations. In *Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms*, pages 148–157, 1997.

[5] M. Formann and F. Wagner. A packing problem with applications to lettering of maps. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 281–288, 1991.

[6] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Inform. Process. Lett.*, 12(3):133–137, 1981.

[7] E. C. Freuder and R. J. Wallace. Partial constraint satisfaction. *Jour. Artificial Intelligence*, 58:21–70, 1992.

[8] S. A. Hirsch. An algorithm for automatic name placement around point data. *The American Cartographer*, 9(1):5–17, 1982.

[9] M. Jampel, E. Freuder, and M. Maher, editors. *Over-Constrained Systems.* Number 1106 in LNCS. Springer, Aug. 1996.

[10] K. G. Kakoulis and I. G. Tollis. A unified approach to labeling graphical features. In *Proc. 14th Annu. ACM Sympos. Comput. Geom.*, pages 347–356, June 1998.

[11] D. E. Knuth and A. Raghunathan. The problem of compatible representatives. *SIAM J. Discr. Math.*, 5(3):422–427, 1992.

[12] A. K. Mackworth and E. C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Jour. Artificial Intelligence.*, 25:65–74, 1985.

[13] T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In *Proc. International Joint Conference on AI*, aug 1995.

[14] M. van Kreveld, T. Strijk, and A. Wolff. Point set labeling with sliding labels. In *Proc. 14th Annu. ACM Sympos. Comput. Geom.*, pages 337–346, June 1998.

[15] F. Wagner. Approximate map labeling is in $\Omega(n \log n)$. *Information Processing Letters*, 52(3):161–165, 1994.

[16] F. Wagner and A. Wolff. A practical map labeling algorithm. *Computational Geometry: Theory and Applications*, 7:387–404, 1997.

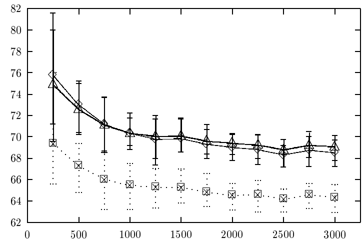

Fig. 9. RandomMap

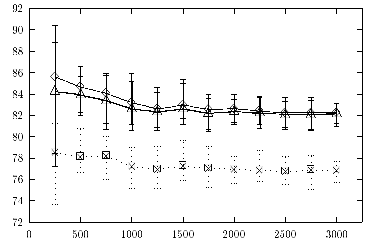

Fig. 10. RandomRect

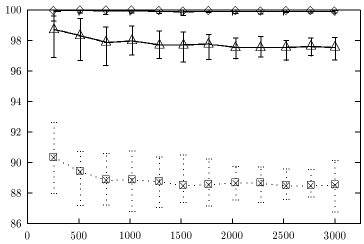

Fig. 11. DenseMap

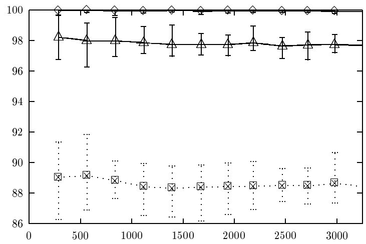

Fig. 12. DenseRect

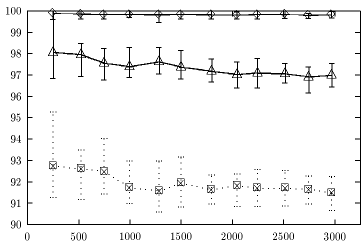

Fig. 13. HardGrid

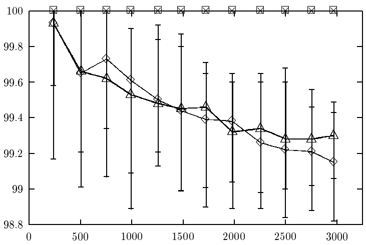

Fig. 14. RegularGrid

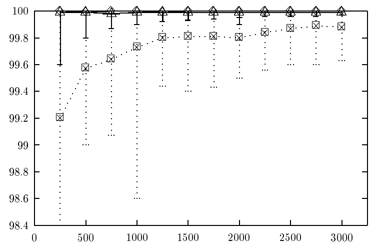

Fig. 15. MunichDrillholes

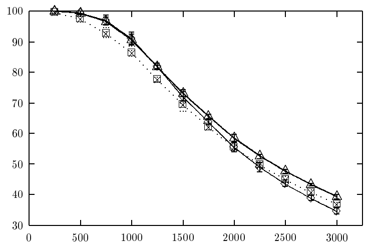

Fig. 16. VariableDensity

Our Algorithm Simulated Annealing Greedy Algorithm

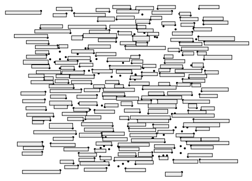

Fig. 17. RandomMap: 250 points, 193 lab.

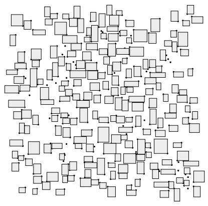

Fig. 18. RandomRect: 250 points, 212 lab.

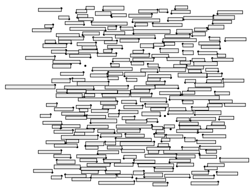

Fig. 19. DenseMap: 253 points, 249 labeled

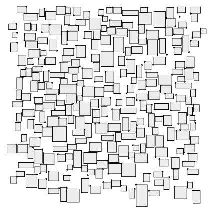

Fig. 20. DenseRect: 261 points, 258 labeled

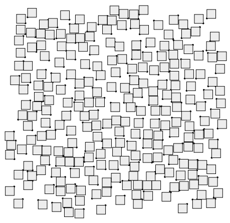

Fig. 21. HardGrid: 253 points, 252 lab.

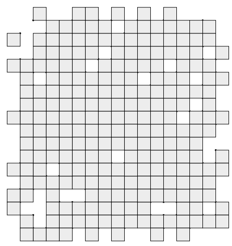

Fig. 22. RegularGrid: 240 points labeled

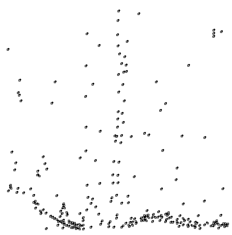

Fig. 23. MunichDrillholes: 250 points lab.

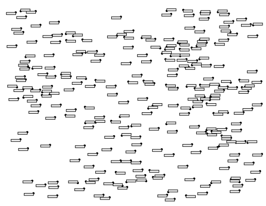

Fig. 24. VariableDensity: 250 points lab.

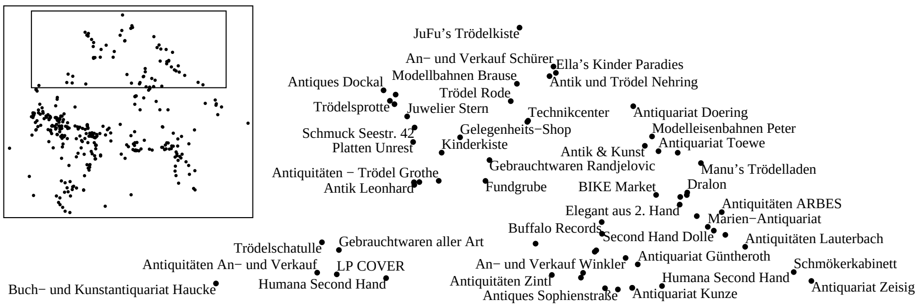

Fig. 25. left: 357 tourist shops in Berlin, right: 45 of 63 labeled.

Fig. 26. 373 German railway stations, 270 labeled.

Thanks to Vikas Kapoor, Christian Knauer, Lars Knipping, Alexander Pikovsky, Tycho Strijk, Rudi Krämer, Frank Schumacher and Karsten Weihe for data, advice, and help.