A Genetic Algorithm for Labeling Point Features

Günther R. Raidl
Institute of Computer Graphics
Vienna University of Technology
Karlsplatz 13/1861, 1040 Vienna, Austria
raidl@eiunix.tuwien.ac.at

Abstract *This paper introduces a genetic algorithm (GA) for tagging point features on images with text labels. The goal is to place all labels in a way that minimizes overlaps and simultaneously consider predefined position preferences. The proposed GA includes several problem dependent improvements: First, a preprocessing step reduces the search space in a safe way. Second, the starting population of the GA is generated in a heuristic way, which enables a faster convergence but nevertheless ensures the presence of enough diversity. Third, each newly generated solution is locally improved before its evaluation. The proposed GA is empirically compared to a very efficient simulated annealing approach using several randomly generated test cases.*

Keywords: Label Placement, Genetic Algorithm, Simulated Annealing

1 Introduction

Tagging graphical objects with text labels is a fundamental task in generating informational images. This problem arises most often in automated cartography, though it occurs also frequently in the production of many other types of informational graphics as e.g. scatterplots. A major factor affecting the clarity of the final image is the degree to which labels obscure display features including other labels as a result of spatial overlap.

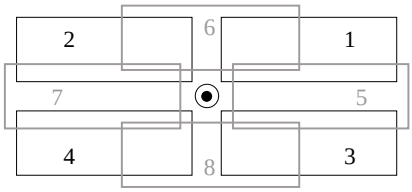


Figure 1: Possible label positions relative to a given point and their desirability.

In cartography, three different label placement tasks are distinguished [11, 7]: Labeling of point features (e.g. cities, peaks of mountains), line features (e.g. streets, rivers), and area features (e.g. countries, seas). While determining the optimal labeling of point features is a very different task from labeling lines or areas, the three categories share a common combinatorial aspect when dealing with multiple features placed near to each other: The complexity arises because the placement of a single label usually has global consequences due to label-label overlaps [3].

In this work, we concentrated on the *point-feature label placement* (PFLP) problem, which can be stated as follows: A set of n points is given, each of them must be labeled by assigning its label to one of m predefined positions. A complete label placement is represented by a vector $\vec{x} = (x_1, \ldots, x_n)$, where each component $x_i \in \{1, 2, \ldots, m\}$ $(i = 1, \ldots, n)$ identifies the assigned position of label i. The eight standard positions for text labels most commonly used in cartography [3, 4] are shown in Fig. 1. Imhof discusses in [11] many of the concerns affecting label placement. Figure 2 shows ex-

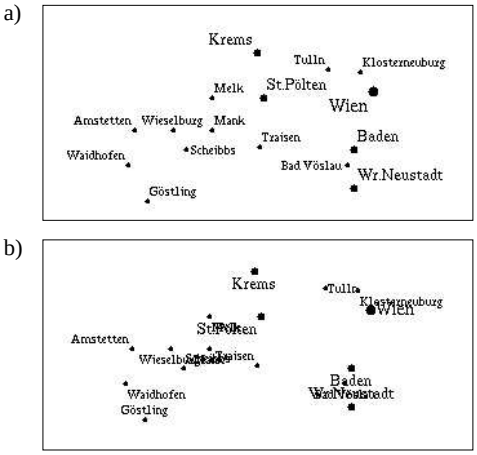


Figure 2: Small examples for (a) a good and (b) a bad point-feature label placement.

amples for good and bad labelings. Two goals stand out as being of particular importance: Minimizing the degree to which labels overlap and obscure other features and maximizing the degree to which labels are unambiguously and clearly associated with the features they identify. For a specific label placement $\vec{x}$ the first goal can be expressed in a scalar value $Conf(\vec{x})$ by counting the number of conflicting labels that at least partially overlap any other label or image feature [3, 11, 17]. The second criterion can approximately be evaluated by assigning a relative desirability value to each possible label position and basically calculating the sum of the values of all label positions used in the specific placement $\vec{x}$ [2, 3, 4]. In cartography, the upper right position is usually preferred, and desirability values correspond to the position numbers $j = 1, 2, \ldots, m$ as depicted in Fig. 1 (smaller values indicate more desirable positions).

According to these goals and corresponding to [2, 3, 4], the following objective function f, which should be minimized, is used within this work for evaluating a label placement $\vec{x}$:

$$f(\vec{x}) = Conf(\vec{x}) + \sum_{i=1}^{n} \frac{x_i - 1}{m}. \qquad (1)$$

The second term adds a position penalty determined out of the rank (desirability value) of each actual label position to the number of conflicting labels. Since the position penalty for a single label lies in $[0, (m-1)/m]$, a label overlap always counts more than assigning the label to its least desired position and avoiding any conflict.

Marks and Shieber [13] and Formann and Wagner [8] have independently shown that the PFLP and various variants of it are NP-complete. Note that NP-completeness is also already given if only label conflicts are minimized and position desirabilities are ignored.

2 Algorithms for PFLP

Various algorithms for PFLP were already published. As for any other NP-complete problem, these approaches can be divided into two categories: Exhaustive search algorithms, which are complete but too time expensive, even for moderately sized problems, and incomplete heuristics or local search algorithms, which do not guarantee to find the optimal label placement.

The approaches from Doerschler and Freeman [6], which use a rule-based system, and Jones [12], fall into the category of exhaustive search algorithms. A greedy heuristic, which avoids the high effort of backtracking needed in the exhaustive methods altogether, is discussed by Yoeli [17]. In [18], Zoraster addresses PFLP by formulating it as a 0–1 integer programming problem and applying mathematical programming techniques to find nearly optimal solutions in a reasonable amount of time. Christensen et al. presented in [4] a relatively simple discrete gradient descent method, thus a local search technique, which gives surprisingly good results. A more sophisticated gradient descent method is presented by Hirsch in [10].

In [2, 3], Christensen, Marks, and Shieber address PFLP by using a *simulated annealing* (SA) algorithm: This stochastic gradient descent method starts with a randomly created label placement $\vec{x}$. In a loop, a new solution $\vec{x}'$ is generated by copying $\vec{x}$ and reassigning

a single, randomly chosen label to a new random position. The new solution $\vec{x}'$ is accepted to be the current solution and parent for the next pass if it is better than $\vec{x}$. But even if $\vec{x}'$ is worse, it is accepted with some probability controlled by a *temperature* parameter T, which decreases over time. This behavior makes it possible for the algorithm to escape from local optima and explore the whole search space. Since the modification from one solution ($\vec{x}$) to the next ($\vec{x}'$) is only very small, the objective value of $\vec{x}'$ can be determined incrementally by calculating the objective value change Δf and adding it to the score of $\vec{x}$. This approach is far more efficient than recalculating the objective value of $\vec{x}'$ from scratch.

An experimental comparison of various, very different algorithms for PFLP is presented by Christensen, Marks, and Shieber in [4]. These tests indicated the best performance (concerning running time and quality of found solutions) for the SA approach. Furthermore, the results for Zoraster's and Hirsch's methods were slightly better than for the simple discrete gradient descent method. Empirical tests of algorithms for a variant of PFLP in which the sizes of labels may also be adapted are presented by Christensen, Friedman, Marks, and Shieber in [5]. Again, a SA approach performed best.

3 A GA for PFLP

Genetic algorithms (GAs) are known to be robust, stochastic search methods applicable to a great variety of difficult problems, see [1, 9, 14] for a general introduction. Especially for many combinatorial problems as e.g. the *traveling salesman problem* [14] or variants of the *knapsack problem* [15], GAs have proven to be very well suited and sometimes much more efficient than other known optimization techniques. Since a GA works on a population of solutions and not only with a single current solution as SA or similar techniques, it has greater potentialities to escape from local optima without completely losing found regions in the search space containing high quality solutions. Because of these properties, it seems to be interesting to apply a GA to PFLP.

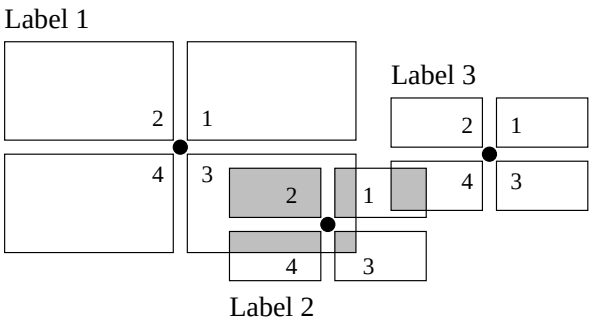
Inspired by several previous GAs for combinatorial optimization problems (especially [15]), a steady-state GA with tournament selection and a replacement scheme which eliminates the worst solution or the last one generated in case of duplicates is used as a basis instead of the traditional generational GA [9, 14]. For more details about the proposed GA, see also [16].

An essential decision for applying a GA to any problem is the way of encoding a solution. In case of PFLP, the most natural way surely is to directly use vector $\vec{x}$. This enables the usage of the traditional recombination operators as e.g. *uniform crossover* [9, 14]. Mutation is performed by setting a randomly selected x_i to a new random value (*random replacement mutation*) – this is exactly the same as the method of generating a new solution in the SA approach.

Each new solution generated by the application of selection, recombination, and mutation must be evaluated. Unfortunately, a fast incremental calculation of the objective value (as in the SA approach) is not possible because an offspring solution usually contains much more than one changed element when compared to each of its two parental solutions. To make the evaluation of solutions nevertheless as efficient as possible, a *conflict table* created in an initialization phase prior to the GA run holds informations about possible conflicts.[1] This conflict table consists of two parts, namely the conflict level array $c_{i,j}$ and the conflict references array $P_{i,j}$ ($i = 1, \ldots, n$, $j = 1, \ldots, m$):

A conflict level $c_{i,j}$ is set to 0 ("safe") if position j of label i does not overlap any fixed image feature (as another point) and can never be in conflict with any other label. If the label position (i, j) overlaps any part of a fixed image feature, the conflict level $c_{i,j}$ is set to

[1] Note that also the SA approach uses a comparable but more primitive data structure for speeding up evaluation.



$\underline{c_{i,j}}$:

$j \backslash i$	1	2	3
1	0	2	0
2	0	1	0
3	∞	1	0
4	0	1	1

$\underline{P_{i,j}}$:

$P_{2,1} = \{(1,3),(3,4)\}$
$P_{2,2} = P_{2,3} = P_{2,4} =$
$= \{(1,3)\}$
$P_{3,4} = \{(2,1)\}$

Figure 3: An example arrangement ($m = 4$) and its conflict table: conflict levels $c_{i,j}$ and conflict references $P_{i,j}$.

∞ marking the position as "hopeless". Otherwise, $c_{i,j}$ is set to the total number of positions of all other labels with which conflicts would occur. In this last case ($0 < c_{i,j} < \infty$), the label numbers and position indices of the conflicting label positions are altogether stored as conflict references set $P_{i,j}$. See Fig. 3 for an example of a conflict table.

Beside the faster detection of overlaps during evaluation, the conflict table can be utilized to reduce the whole search space in a safe way, generate a more meaningful starting population, and perform local improvements on newly generated solutions. This techniques, which altogether improve the performance of the GA essentially, are described in the following subsections.

3.1 Problem Reduction

Applying two rules to the conflict level array usually enables the reduction of the search space in advance to the GA run without the danger of overlooking the global optimum:

(a) Any label i for which $\exists j(j \in \{1,\ldots,m\} \wedge c_{i,j} = 0)$ (a safe position j exists) and $\forall k(1 \leq k < j \rightarrow c_{i,k} = \infty)$ (all more desirable positions are hopeless) can be prematurely assigned to position j. In fact, such a label needs not be considered any longer and can therefore also be omitted in the solution encoding of the GA.

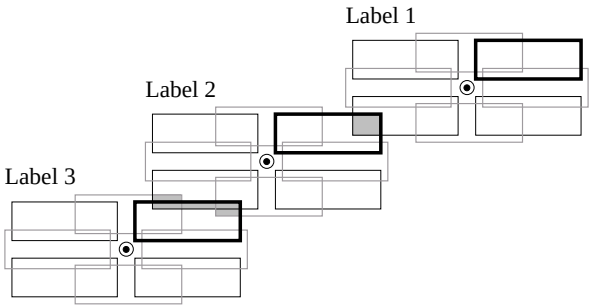


Figure 4: All labels may be prematurely fixed to the upper right position.

(b) Any position j of a label i for which $c_{i,j} = \infty$ (position is hopeless) can be dismissed from further consideration if $\exists k(1 \leq k \leq m \wedge c_{i,k} = 0)$ (a safe position exists). Such a position (i,j) is marked in some way and hereafter excluded from usage during initialization, mutation, and any other part of the GA.

Fixing labels or dismissing label positions according to these rules leads to simplifications in the conflict table: If a label i is fixed to a specific position, there is no possibility that any other label can stay in conflict with it anymore. Therefore, all conflict references from all other label positions to any position of label i should be deleted, and the corresponding conflict levels need to be decremented. Note that the affected label positions can be found easily via the conflict references sets of label i. Clearly, all entries for label i itself may also be deleted. Similarly, if a label position is dismissed by rule (b), conflict references from any other label position to it can be deleted.

Note that such a deletion of conflict references may also open the affected label positions for a further, recursive application of simplification rules. Figure 4 shows an example for such a chain reaction, in which all labels can

consecutively be fixed to the upper right position starting with label 1.

3.2 Heuristic Initialization

The convergence of the GA can be sped up by generating the starting population not in a purely random way but using some heuristics during the initialization. On the other hand, it is essential to provide enough diversity within the starting population to avoid premature convergence to bad local optima.

In our tests, the following technique proved to be well-suited for creating an initial solution: Each (not fixed) label i ($i = 1, \ldots, n$) is assigned to a position determined by choosing two (not dismissed) positions randomly and taking that with the smaller conflict level $c_{i,j}$. Thus, a kind of tournament selection is performed for each label, and positions with a smaller number of conflicts are favored.

3.3 Local Improvement

The performance of the GA could further be increased by introducing a local improvement operator, which is applied to each newly generated solution immediately before its evaluation: Each label i of the solution is checked once if it can be moved from its current position j to any more desirable position k, $k < j$, where no actual conflict would occur. Note that checking for an actual conflict can be done easily by looking into the conflict table and taking actual positions of referenced labels into account. If no better position for label i can be found in this way and the current position j actually stays in conflict with any other label or static image feature, all remaining positions $k = j + 1, \ldots, m$ (if there are any) are also checked in order to possibly circumvent the conflict.

It turned out to be essential to process all the labels in a random, always different order. In this way not the same items are favored every time, and the diversity of the population remains higher reducing the risk of premature convergence.

Table 1: Characteristics of the GA implementation.

GA:	steady state, no duplicate solutions
Selection:	tournament ($k = 2$)
Recombination:	uniform crossover ($p_c = 1$)
Mutation:	random replacement ($p_m = 0.01$ per individual)
Population size:	100
Termination:	15,000 evaluations without finding a new best solution

4 Experimental Results

Various implementation characteristics and parameters of the GA, which were determined by preliminary experiments and found to be robust and well suited for PFLP, are sub-summed in Table 1. Note that each GA run was terminated when no improvements were encountered within the last 15,000 evaluations. This condition ensures that the GA usually has enough time to converge. In general, we were primarily interested in finding high-quality solutions and only secondary in the needed CPU time. To make comparisons to SA, we also implemented this approach according to the descriptions in [2, 3].

Test problems were generated according to [4]: n point features with fixed-sized labels (40×7 units) were randomly placed on a region of size 792×612. Tests were run for $n = 50, 100, 150, \ldots, 1000$. Labels were allowed to be placed at the $m = 8$ positions around the point feature as depicted in Fig. 1.

Table 2 shows final results of the GA and SA: Note that all values are average values determined from 10 runs per problem instance and algorithm. Final objective values $f_{\min}$ and number of conflicts $Conf_{\min}$ are also depicted in Fig. 5. Note that the GA led for all problem sizes, but especially for smaller n, to slightly better solutions. Relative differences of the objective values of SA and the GA are shown in the last column of the table. The number of conflicts $Conf_{\min}$ of the GA were also always

Table 2: Final results of the GA and SA for differently sized problems: Objective values $f_{\min}$, numbers of conflicts $Conf_{\min}$, and numbers of evaluated solutions *Evals* with CPU times t_{min} until final solutions have been found (average values from 10 runs per problem).

n	**GA**				**SA**				$\frac{f_{\min}^{SA} - f_{\min}^{GA}}{f_{\min}^{SA}}$
	$f_{\min}$	$Conf_{\min}$	*Evals*	t_{min} [s]	$f_{\min}$	$Conf_{\min}$	*Evals*	t_{min} [s]	
50	0.4	0.0	100	0.01	0.5	0.0	41251	0.42	20.0%
100	1.4	0.0	100	0.08	1.5	0.0	82798	0.85	8.8%
150	4.0	0.0	242	0.20	4.2	0.0	126411	1.52	5.3%
200	6.3	2.0	464	0.41	6.8	2.0	169959	2.17	7.6%
250	17.2	8.0	2149	1.92	17.7	8.0	214107	3.14	2.7%
300	17.6	0.0	4875	5.62	18.5	0.0	261368	4.09	4.4%
350	35.7	10.0	2357	3.44	36.8	10.3	305757	5.09	3.1%
400	33.8	0.0	12151	15.38	35.8	0.6	352770	6.24	5.7%
450	45.4	4.0	11308	20.20	47.7	4.3	398063	7.37	4.8%
500	71.5	20.2	13920	29.12	74.4	21.1	440866	8.81	4.0%
550	86.4	22.3	15667	37.32	89.5	27.6	486704	10.23	3.4%
600	112.7	36.1	25884	59.47	116.9	40.1	529178	11.88	3.6%
650	155.5	57.5	26959	84.86	161.6	69.1	574111	13.82	3.8%
700	172.2	59.7	25895	87.12	177.2	72.1	618253	15.42	2.8%
750	193.5	53.7	41805	134.29	199.9	74.9	663544	16.68	3.2%
800	253.1	94.3	52674	237.16	260.5	119.9	704693	19.20	2.8%
850	281.0	101.3	79727	370.05	290.6	133.4	748523	20.80	3.3%
900	328.1	133.5	88780	446.02	337.4	164.8	791061	22.98	2.7%
950	355.7	139.2	102513	605.63	366.5	182.1	834695	25.07	2.9%
1000	431.6	207.9	121724	748.32	437.9	251.3	875286	28.27	1.4%

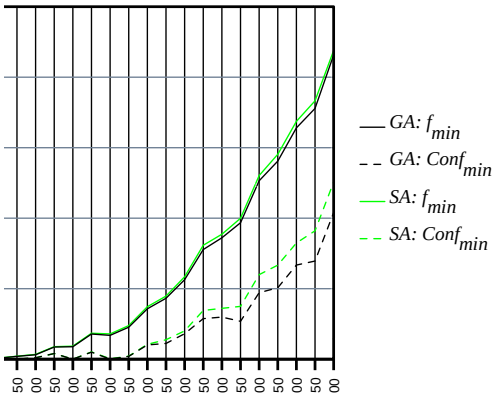


Figure 5: GA and SA: Average objective values $f_{\min}$ and numbers of conflicts $Conf_{\min}$ of final solutions for differently sized problems.

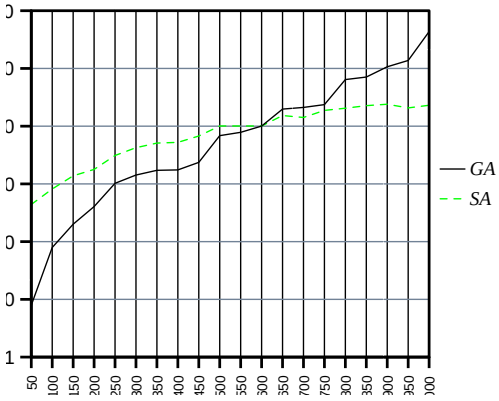
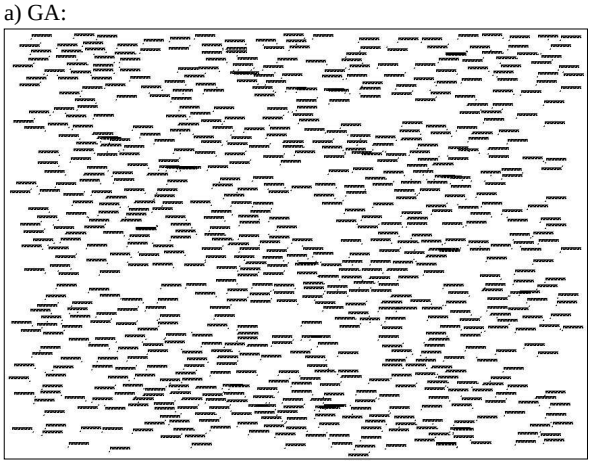


Figure 6: GA and SA: Average CPU times t needed for finding solutions with objective values smaller than given bounds.

a) GA:



b) SA:

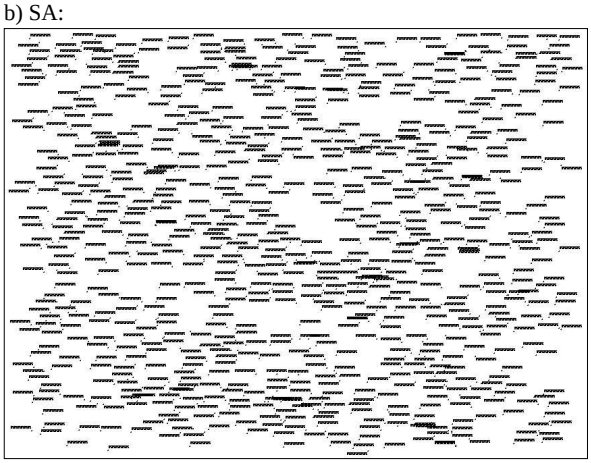


Figure 7: Final solutions of (a) the GA ($f(\vec{x}) = 191.8$) and (b) SA ($f(\vec{x}) = 201.3$) for a problem with $n = 750$ points and $m = 8$.

less than or equal to that of SA. Especially larger problems were solved with much fewer overlaps by the GA, but on the other hand, the SA solutions had smaller position penalties. Reasons for these effects seem to be the very efficient problem reduction and local improvement techniques of the GA.

Table 2 also shows the numbers of evaluations *Eval* and CPU times $t_{\min}$ needed by the two approaches to find their finally best solutions. In case of the GA and $n \in \{50, 100\}$, the finally best solutions could almost always already be found in the initial populations due to the effective interplay of problem reduction, heuristic initialization, and local improvement. In general, the numbers of needed evaluations were much larger for SA, but CPU times $t_{\min}$ show that the GA was nevertheless especially for larger problems several orders slower. Only for small problems ($n \leq 250$), the GA could find its final solution faster than SA.

For a better comparison of convergence velocity, we also measured the times needed by both approaches to find a solution with an objective value below a given, fixed bound. For each problem this bound was set to the highest final objective value observed during all runs of SA and the GA together. Average results are shown in Fig. 6. Note that the GA was particularly faster for all problems with $n \leq 600$. Thereafter, SA outperformed the GA.

See Fig. 7 for two examples of final label placements with 750 point features generated by the GA and SA.

5 Conclusions

This work shows that a GA can be a very efficient technique for finding nearly optimal solutions to PFLP problems. When compared to SA, the GA's major drawback is its time expensive evaluation function, which cannot be implemented in an incremental way as in SA. On the other hand, applying the proposed problem reduction technique and local improvement operator and starting from a heuristically generated initial population speeds up the GA essentially. Moreover, this improved GA converges nearly always to slightly better solutions, especially with fewer overlaps, than SA. The main reason for this effect is the population based model of the GA, which has greater potentialities to escape from local optima without losing found high-quality regions in the search space.

6 Future Work

Clearly, the proposed problem reduction technique can also be applied to SA. Preliminary experiments indicate slightly shorter running times without major improvements in the qualities of final solutions, but more experiments

with different annealing schedules are necessary. Furthermore, a local optimization operator similar to the one proposed for the GA may also be of interest for SA if the efficient incremental evaluation can be retained.

Another approach to PFLP would be to apply *evolutionary programming* [1, 14], which is also a population based, stochastic search method. Since a major difference to GAs is the absence of any recombination operator, an incremental evaluation of PFLP solutions may become applicable.

References

[1] T. Bäck: *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, New York, 1996.

[2] J. Christensen, J. Marks, S. Shieber: *Placing Text Labels on Maps and Diagrams*, Graphic Gems IV, edited by P. S. Heckbert, Cambridge, Academic Press, pp. 497–504, 1994.

[3] J. Christensen, J. Marks, S. Shieber: *Labeling Point Features on Maps and Diagrams*, Technical Report TR-25-92, Harvard University, 1992.

[4] J. Christensen, J. Marks, S. Shieber: *An Empirical Study of Algorithms for Point-Feature Label Placement*, ACM Transactions on Graphics, 14(3), pp. 203–232, 1995.

[5] J. Christensen, S. Friedman, J. Marks, S. M. Shieber: *Empirical Testing of Algorithms for Variable-Sized Label Placement*, to appear in Proceedings of the ACM Computational Geometry Symposium, Nice, France, 1997.

[6] J. Doerschler, H. Freeman: *A Rule-Based System for Dense-Map Name Placement*, Communications of the ACM, 35(1), pp. 68–79, 1992.

[7] S. Edmondson, J. Christensen, J. Marks, S. Shieber: *A General Cartographic Labeling Algorithm*, Cartographica, 33(4), pp. 13–23, 1997.

[8] M. Formann, F. Wagner: *A Packing Problem with Applications to Lettering of Maps*, in Proceedings of the 7th Annual Symposium on Computational Geometry, New Hampshire, pp. 281–288, 1991.

[9] D. E. Goldberg: *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison–Wesley, MA, 1989.

[10] S. A. Hirsch: *An Algorithm for Automatic Name Placement around Point Data*, The American Cartographer, 9(1), pp. 5–17, 1982.

[11] E. Imhof: *Positioning Names on Maps*, The American Cartographer, 2(2), pp. 128–144, 1975.

[12] C. Jones: *Cartographic Name Placement with Prolog*, IEEE Computer Graphics and Applications, 9(5), pp. 36–47, 1989.

[13] J. Marks, S. Shieber: *The Computational Complexity of Cartographic Label Placement*, Technical Report TR-05-91, Harvard University, 1991.

[14] Z. Michalewicz: *Genetic Algorithms + Data Structures = Evolution Programs*, Springer, Berlin, 1992.

[15] G. R. Raidl: *An Improved Genetic Algorithm for the Multiconstrained 0–1 Knapsack Problem*, to appear in Proceedings of the 5th IEEE Conference on Evolutionary Computation, Anchorage, Alaska, 1998.

[16] W. Rumplmaier: *Optimierung von Labelanordnungen mit Genetischen Algorithmen und Simulated Annealing*, diploma thesis at the Vienna University of Technology, Institute for Computer Graphics, Vienna, 1998.

[17] P. Yoeli: *The Logic of Automated Map Lettering*, The Cartographic Journal, 9(2), pp. 99–108, 1972.

[18] S. Zoraster: *The Solution of Large 0–1 Integer Programming Problems Encountered in Automated Cartography*, Operations Research, 38(5), pp 752–759, 1990.