Letting Ants Labeling Point Features

Michael Schreyer, Günther R. Raidl
Institute of Computer Graphics and Algorithms, Vienna University of Technology, Vienna, Austria
{mikes, raidl}@ads.tuwien.ac.at

Abstract - This paper describes an ant colony system (ACS) for labeling point features. A preprocessing step reduces the search space in a safe way. The ACS applies local improvement and masking, a technique that focuses the optimization on critical regions. Empirical results indicate that the ACS reliably identifies high-quality solutions which are in many cases better than those of a state-of-the-art genetic algorithm for point feature labeling.

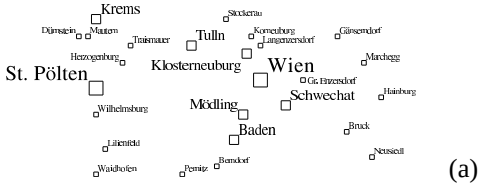
I. INTRODUCTION

Automated cartography and the graphical visualization of business or technical information are concerned with tagging graphical objects with text labels. The legibility of a final image is affected by the degree to which graphical features are obscured by overlaps as well as the degree to which labels are unambiguously associated to the feature they describe. Examples of a good and a bad labeling are shown in Fig. 1.

Different labeling tasks are distinguished in cartography [9], [12]. We focus here on the labeling of point features, e.g. cities or mountain peaks, only. More precisely, the *point feature labeling problem* (PFLP), is defined as follows.

Given a set of n point features in the Euclidean plane, each feature needs to be labeled by placing a fixed text near to it. The allowed positions are restricted to a set of p places in the feature's surrounding. Figure 2 shows the $p = 8$ standard positions for text labels which are typically used in cartography [3]. A complete labeling of all features is expressed by a vector $x = (x_1, \ldots, x_n) \in \{1, 2, \ldots, p\}^n$, in which each component x_i with $i = 1, \ldots, n$ identifies the assigned position of the label for feature i.

In [12], [18], various objectives for meaningful label placement are discussed. We concentrate on the following two



(a)

(b)

Fig. 1. Examples for a good (a) and a bad (b) labeling.

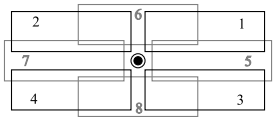


Fig. 2. A point feature's possible label positions, numbered in their order of desirability.

goals, which probably have been most often considered with the PFLP in the past [3]:

(1) The total number $conf(x)$ of "conflicting" labels that partly or completely overlap any static image feature or other label in x should be minimal.

(2) To maximize the degree to which each label is uniquely associated with the feature it represents, possible positions around a point feature are assigned different desirabilities. In Fig. 2, the numbering represents this desirability as it is common in cartography, thus the upper right position is preferred most [3].

The two goals are combined into the following objective function on x:

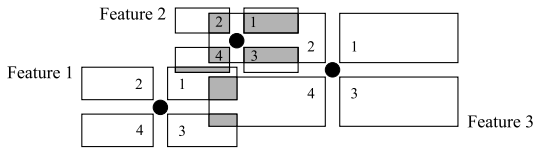
$$\text{minimize } g(x) = conf(x) + w_{\text{pos}} \cdot \sum_{i=1}^{n} \frac{x_i - 1}{p} \qquad (1)$$

The second term represents a penalty according to the desirabilities of the labels' actual positions. Constant w_{pos} controls the importance of position desirabilities over conflicts. Usually, one label overlap should count more than the highest possible position penalty for one label, thus, $0 \leq w_{\text{pos}} \leq 1$.

The PFLP is NP-hard because of the global consequences a single change of a label's position might have [10], [14]. As for every NP-hard problem, an approach for solving the PFLP either applies exhaustive search, which gives an optimum result but may be too time-consuming for larger instances, or it is of heuristic nature and optimality cannot be guaranteed.

Exact approaches include rule based systems as proposed by Doerschler and Freeman [6]. Cromley [5] and Zoraster [19] transform the PFLP into a 0/1-integer programming problem and solve it with branch-and-cut techniques. Recently, Klau and Mutzel [13] presented another 0/1-integer programming based branch-and-cut approach for the related *label number maximization problem*, in which the number of labels that can be placed without overlap is maximized.

On the side of heuristic approaches, a simple greedy heuristic [18] and a discrete gradient descent method [11] were among the first published techniques. Christensen et al. [2] described a simulated annealing approach and compared it to



$c_{i,j}$:

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

$P_{i,j}$:

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

Fig. 3. An example PFLP instance with $p = 4$ and its conflict table: conflict numbers $c_{i,j}$ and conflict reference lists $P_{i,j}$.

several other algorithms in [3]. A genetic algorithm (GA) for the PFLP has been proposed by Verner et al. [17]. It benefits from *masking*, which allows a solution to inherit a set of spatially connected high-quality alleles in a guaranteed unchanged form from a parent solution. Raidl [15] described another GA that includes a heuristic improvement operator. A more extensive review on algorithms for the PFLP can be found in [16].

Today, simulated annealing and hybrid GAs are among the best choices for large, hard PFLP instances where exact techniques are not applicable anymore. This article describes a new effective approach following the concept of *ant colony optimization* [7].

The next section presents a general preprocessing of problem data in order to reduce the search space in a safe way. Section III describes the ant colony system, and Section IV compares it empirically with simulated annealing and a GA similar to that of [15] but enhanced by masking as suggested in [17]. Results indicate that the ant colony system is a strong competitor for simulated annealing and the GA. In particular for hard, dense instances, the ant colony system often finds superior solutions. Section V draws some final conclusions.

II. PREPROCESSING

To be able to efficiently identify conflicting labels of a candidate solution during the optimization, a *conflict table* is created as a part of preprocessing. This data structure holds general information about all possible label/label or label/feature overlaps [15].

A *conflict number* $c_{i,j} \geq 0$ is assigned to each possible label position $j = 1, \ldots, p$ of each point feature $i = 1, \ldots, n$. $c_{i,j}$ is zero ("safe") if label position j of feature i does not overlap any feature and can never collide with any other label. When label position j of feature i statically overlaps any part of a feature, the conflict number $c_{i,j}$ is set to ∞ marking the position as "hopeless". In any other case, $c_{i,j}$ is the total number of positions of all other labels which at least partly overlap position j of label i, and we additionally store references to all these conflicting label positions in a *conflict reference list* $P_{i,j}$. See Fig. 3 for an example.

Having the conflict table initialized, a deterministic *problem reduction* takes place, which applies the following two rules to each feature $i = 1, \ldots, n$:

(1) If a safe label position j exists ($\exists j \mid 1 \leq j \leq p \wedge c_{i,j} = 0$) and all more desirable positions are hopeless ($\forall k = 1, \ldots, j-1 : c_{i,k} = \infty$), then label i must be assigned to position j in any optimum solution and therefore, we permanently fix this assignment and dismiss all other positions $k \neq j$ for label i.

(2) If a safe label position j exists, any hopeless position ($\{k \mid k = 1, \ldots, p \wedge c_{i,k} = \infty\}$) cannot appear in an optimal solution and is therefore permanently dismissed.

A label whose position could be prematurely fixed by rule (1) is from now on treated as static image feature, and all label positions dismissed by rules (1) or (2) are somehow marked and excluded from any further consideration. In the conflict table, all conflicts with dismissed label positions are removed and conflicts with fixed label positions are noted by setting the corresponding conflict values to ∞. Each such reduction in the conflict table may also enable further reductions. In this way, larger chain-reactions sometimes arise shrinking the search space significantly.

III. AN ANT COLONY SYSTEM FOR THE PFLP

In nature, a single ant can be seen as an autonomous agent whose actions are strongly guided by randomness. While ants are wandering around looking for food, they deposit pheromones on the ground which influence the behavior of following ants. Via this indirect communication, called *stigmergy*, a cooperative ant colony is able to efficiently determine the shortest path between its nest and a food source [1].

This principle has been adopted to attack hard combinatorial optimization problems. Dorigo and Gambardella [8] proposed an *ant colony system* (ACS) for the traveling salesman problem (TSP), which works on a weighted graph $G = (V, E)$ with node set V and edge set E. Simple autonomous agents, called artificial ants, create repeatedly independent solutions by touring the graph in parallel. At each node $r \in V$, an ant's decision which edge to follow next is a random choice biased by local parameters, namely heuristic values $\eta(r, s) \geq 0$ and pheromone values $\tau(r, s) \geq 0$ of all incident edges $(r, s) \in E$. The pheromone values $\tau(r, s)$ model the amount of pheromones deposited by ants having previously passed the edges, and they are updated by local and global rules. In particular when one iteration of the ACS is finished, i.e. each ant has completed a solution, the pheromone values of the so-far best solution's edges are increased to intensify the search near this solution during the next iterations. For more general information on ant colony optimization, see [4], [7].

A. A Graph-Representation of the PFLP

The PFLP is not originally a graph-problem. However, to apply the idea of ant colony optimization, a graph representation should be defined on which we can imagine the ants walking, hereby creating candidate solutions.

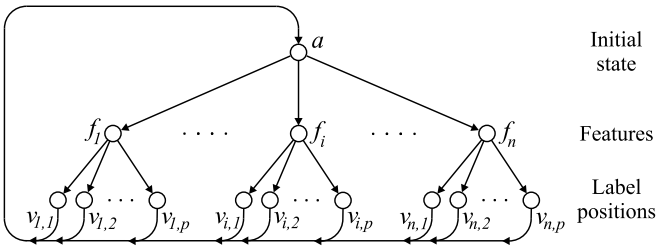


Fig. 4. A graph-representation of the PFLP for applying the ant colony system.

Note that to keep things simple, we neglect from now on labels fixed and label positions dismissed during preprocessing. It is straight-forward to take them into account in a real implementation.

Figure 4 shows the graph-representation of the PFLP. Starting from node a, an ant creates a solution by an iterated two-step process: From a it needs to decide which feature – represented by nodes $f_i, \ldots, f_n$ – to go to. We call this step *feature selection*. Having moved to the selected feature-node f_i, the ant has to decide to which of its label positions – represented by nodes $v_{i,1}, \ldots, v_{i,p}$ – to move on. This step implies the actual labeling of feature i at the corresponding position.

Then, the ant moves to node a again, and the two steps are repeated until all nodes $f_1, \ldots, f_n$ have been reached once, thus all features are labeled and the solution is complete.

The second step, where the labels are actually placed, is the more crucial one. There, we apply an ACS-typical state transition rule that uses a heuristic function $\eta(f_i, v_{i,j})$ and pheromone values $\tau(f_i, v_{i,j})$ associated to the edges $(f_i, v_{i,j})$, as Sect. III-C describes. Feature selection is done in a simpler heuristic way as explained in Sect. III-B.

Figure 5 shows a pseudo-code for the complete ACS. First, all pheromone values τ for edges between feature-nodes and nodes representing label positions are initialized with a value $\tau_0 = 1/(n \cdot g(x^0))$, where x^0 is an initial solution created by a simple greedy heuristic: Features are processed in random order, and each label is assigned to the locally best possible position.

Next, m ants are initialized. Each ant k owns a set F^k of yet unprocessed features and a vector x^k in which the ant's solution will be stored. All ants build solutions in parallel by the already mentioned two-step process, communicating only indirectly via pheromone updates.

At the end of an iteration, when all ants have completed a solution, each solution is locally improved and evaluated. The best solution x^{ib} of the iteration is determined, the globally best solution x^{gb} eventually updated, and a global pheromone update is performed. The whole process is repeated until the improvement of the globally best solution during the last γ iterations falls below a threshold ε.

More details of the algorithm are described in the following subsections.

ALGORITHM ACS–FOR–PFLP:

create initial solution x^0 by a simple greedy heuristic
$x^{\mathrm{gb}} \leftarrow x^0$ (globally best solution found so far)
$\tau_0 \leftarrow 1/(n \cdot g(x^0))$
$\forall i = 1, \ldots, n, \forall j = 1, \ldots, p: \ \tau(f_i, v_{i,j}) \leftarrow \tau_0$

REPEAT

  $\forall$ ants $k = 1, \ldots, m$:
    $F^k = \{1, \ldots, n\}$

  REPEAT n times (until solutions are completed):

    REPEAT $\forall$ ants $k = 1, \ldots, m$:

      $i \leftarrow$ select a feature from F^k
      $F^k \leftarrow F^k \setminus \{i\}$

      $\forall j = 1, \ldots, p$: compute $\varphi(f_i, v_{i,j})$

      WITH probability q_0:

        $b \leftarrow \arg\max_{j=1,\ldots,p} \varphi(f_i, v_{i,j})$

      ELSE

        choose b randomly from $(1, \ldots, p)$
          with probabilities $(\varphi(f_i, v_{i,1}), \ldots, \varphi(f_i, v_{i,p}))$

      $x_i^k \leftarrow b$
      $\tau(f_i, v_{i,b}) \leftarrow (1 - \rho) \cdot \tau(f_i, v_{i,b}) + \rho \cdot \tau_0$

  $\forall$ ants $k = 1, \ldots, m$:
    locally improve x^k
    evaluate x^k by computing $g(x^k)$

  $x^{\mathrm{ib}} \leftarrow \arg\min_{x=x^1,\ldots,x^m} g(x)$ (best solution of iteration)
  IF $g(x^{\mathrm{ib}}) < g(x^{\mathrm{gb}})$ **THEN** $x^{\mathrm{gb}} \leftarrow x^{\mathrm{ib}}$

  $\forall i = 1, \ldots, n, \forall j = 1, \ldots, p:$
    $\tau(f_i, v_{i,j}) \leftarrow (1 - \alpha) \cdot \tau(f_i, v_{i,j})$
  $\forall i = 1, \ldots, n:$
    $\tau(f_i, v_{i,x_i^{\mathrm{ib}}}) \leftarrow \tau(f_i, v_{i,x_i^{\mathrm{ib}}}) + \alpha / g(x^{\mathrm{ib}})$

UNTIL improvement of $g(x^{\mathrm{gb}})$ during the last γ iterations $< \varepsilon$

Fig. 5. The main algorithm of the ACS for the PFLP.

B. Feature Selection

Feature selection must ensure that no feature is selected twice by the same ant in the same iteration of the ACS. Note that this is similar in the ACS for the TSP, where each node may only be reached once in a feasible tour.

Furthermore, feature selection should be stochastic and ensure that features are processed in different order in general. Processing always the same features first would bias the optimization and in general lead to poor local optimal solutions.

A simple approach would therefore be to always make a uniform random choice among all not yet visited nodes from $f_1, \ldots, f_n$, thus, to process all features in pure random order. However, it is generally hard to avoid overlaps when labeling features in random order: While the first features can usually be labeled easily without much danger of conflicts, it becomes hard and often impossible to avoid overlaps when final gaps need to be processed, i.e. features completely surrounded by already previously fixed labels of other features.

It proved much more efficient to take care on spatial relationships of features by always trying to select as next feature one lying close to a previously processed feature instead of

one from somewhere in the middle of a yet untouched region. This is accomplished by the following queue-based heuristic.

Let $F^k = \{1, \ldots, n\}$ be the set initially containing all features. For each ant, a queue is furthermore maintained, which is initially empty. A feature is always selected by the following actions: If the queue is not empty, one feature i is dequeued; otherwise, a feature i is picked randomly from F^k and removed there. A *neighborhood set* $H_i \subseteq F^k$ of yet unprocessed features lying "close" to i is determined, and these features are put into the queue and removed from F^k. i is finally returned as selected feature.

Different definitions can be used for the neighborhood set H_i of a feature i:

(1) H_i consists of the h nearest neighbors of feature i in F^k with respect to the Euclidean distance. If $|F^k| < h$, then $H_i = F^k$. The features from H_i are queued in order of increasing distance.

(2) H_i consists of those features from F^k that may stay in conflict with feature i. These features can be efficiently identified via the conflict table's reference lists $P_{i,j}$, $j = 1, \ldots, p$. The features of H_i can be put either in random order or again in order of increasing Euclidean distance into the queue.

Practical experiments have shown slight advantages of neighborhood definition (1), providing a suitable value is chosen for h. A detailed study on this topic can be found in [16].

C. State Transition Rule for Feature-Nodes f_i

When an ant has selected a feature to process, i.e. it has moved to a certain feature node f_i, each edge to a possible successor node $v_{i,j}$, $j = 1, \ldots, p$, gets assigned a probability

$$\varphi(f_i, v_{i,j}) = \frac{\tau(f_i, v_{i,j}) \cdot \eta(f_i, v_{i,j})^\beta}{\sum_{k=1}^{p} \tau(f_i, v_{i,k}) \cdot \eta(f_i, v_{i,k})^\beta}\,. \qquad (2)$$

$\tau(f_i, v_{i,j})$ is the pheromone value, and $\eta(f_i, v_{i,j})$ a heuristic value representing the local attractiveness of node $v_{i,j}$. β controls the relative importance of the pheromone versus the heuristic value.

The decision over which edge the ant moves – and therefore to which position the label of feature i is assigned – is made by the following *pseudo-random-proportional rule* according to [8]:

- With probability $q_0 \in [0, 1)$ the edge $(f_i, v_{i,j})$ with maximum $\varphi(f_i, v_{i,j})$ is chosen;
- otherwise a random decision is made in which each edge $j = 1, \ldots, p$ is chosen with probability $\varphi(f_i, v_{i,j})$.

Parameter q_0 controls the relative importance of exploitation of already collected knowledge versus (biased) exploration of new possibilities.

The heuristic value $\eta(f_i, v_{i,j})$ is a function composed of three criteria:

- the number $\Gamma(i, j)$ of actual overlaps with other labels or static features that would be newly introduced when placing label i at position j in the current situation,
- the conflict value $c_{i,j}$ of the label position, and
- the general desirability of the label position, expressed by its index j.

Since $\Gamma(i, j), c_{i,j}, j \geq 0$ and smaller values represent more appealing positions, we combine them in the following way:

$$\eta(f_i, v_{i,j}) = \frac{w_\Gamma}{\Gamma(i, j) + k_\Gamma} + \frac{w_c}{c_{i,j} + k_c} + \frac{w_d}{j + k_d} \qquad (3)$$

$w_\Gamma, w_c, w_d \geq 0$ are weights controlling the general influence of the corresponding criterion, $k_\Gamma, k_c, k_d \geq 0$ determine how strongly the function distinguishes between good and bad values for each criterion.

D. Global Pheromone Update

At the end of each iteration, pheromone evaporation takes place on all edges $(f_i, v_{i,j})$, $i = 1, \ldots, n$, $j = 1, \ldots, p$:

$$\tau(f_i, v_{i,j}) \leftarrow (1 - \alpha) \cdot \tau(f_i, v_{i,j}) \qquad (4)$$

$0 < \alpha < 1$ is the pheromone decay parameter. Evaporation decreases the intensity of all pheromones as time goes on and reduces the danger of premature convergence to poor local optima.

Then, the best ant of the iteration deposits additional pheromones on the edges representing its solution x^{ib}:

$$\tau(f_i, v_{i,x_i^{\mathrm{ib}}}) \leftarrow \tau(f_i, v_{i,x_i^{\mathrm{ib}}}) + \frac{\alpha}{g(x^{\mathrm{ib}})}, \quad \forall i = 1, \ldots, n \qquad (5)$$

Note that this is slightly different to the ACS for the TSP in [8], where the edges of the *globally* best solution are reinforced. We have observed here slight advantages when using the iteration's best solution instead of the globally best one, see [16] for more details.

E. Local Pheromone Update

Every time an ant moves over an edge $(f_i, v_{i,j})$, the following local pheromone update rule is applied to the associated pheromone value:

$$\tau(f_i, v_{i,j}) \leftarrow (1 - \rho) \cdot \tau(f_i, v_{i,j}) + \rho \cdot \tau_0 \qquad (6)$$

The pheromone value is slightly modified towards the initial value τ_0 in order to change the attractivity of the used edges for the other ants building solutions in parallel. In this way, exploration is emphasized and pheromone information can be used more efficiently, since ants are searching solutions in a broader neighborhood of the best previous solution. Parameter $0 < \rho < 1$ controls the strength of local pheromone update.

F. Local Improvement

Each created solution is locally improved before its evaluation by processing all features in random order. Each label is checked if there exists a more desirable position that would result in no conflict in the current solution. If this is the case, the label is reassigned to the best position found.

G. Masking

Verner et al. [17] proposed *masking* in their genetic algorithm for PFLP. This technique preserves good subsets of spatially related alleles when applying recombination or mutation and focuses the variation to more critical regions in the surrounding of overlaps. We adopted this idea for the ACS as follows.

For each feature $i = 1, \ldots, n$, let D_i be the set of its d nearest neighboring features under the Euclidean distance metric. In a given solution x, we say a feature is *critical* if it is either self involved in a conflict or contained in the nearest-neighbor set D_j of any other feature $j \in \{1, \ldots, i-1, i+1, \ldots, n\}$ staying in a conflict.

At iterations $t \equiv 0 \pmod{\zeta}$, starting with $t = 0$, the ACS performs normally as described so far. However, at each other iteration $t \not\equiv 0 \pmod{\zeta}$, an ant's solution x' from the preceding iteration $t-1$ serves as basis, and variation takes only place at the features rated critical in the sense defined before. Thus all uncritical labels from x' are simply copied to x and not processed by the ant. Only local improvement considers finally again the complete solution and may also change inherited label positions.

Parameter d therefore controls "how far away" a feature must lie from any conflict to be considered uncritical, parameter ζ controls, how often ants are restricted to vary critical features only.

IV. EMPIRICAL COMPARISON

We empirically compare the ACS to a re-implementation of the simulated annealing algorithm (SA) of Christensen et al. [2] and a hybrid genetic algorithm (HGA). HGA is basically that one described by Raidl [15] enhanced by applying masking during mutation according to [17]. It uses a steady-state replacement scheme in which in each iteration one new solution is created by means of binary tournament selection, uniform crossover, masking-mutation, and local improvement as described in Sect. III-F. Such a new solution always replaces the worst solution in the population with one exception: To assure a minimum diversity, duplicates are always discarded.

For the comparison, standard test problem instances with the following characteristics were adopted from [3]: The number of features ranges from 50 to 1000, and they are randomly distributed on a rectangular area of 792×612 units. Each label has size 40×7 and must be placed on one of $p = 8$ possible positions according to Fig. 2. The weight of the position penalty term in the objective function is always $w_{\text{pos}} = 1$.

For the ACS, robust parameters settings that work well for several kinds of problem instances were found by extensive preliminary tests documented in [16]: $m = 6$, $q_0 = 0.7$, $\beta = 1.4$, $\alpha = 0.05$, $\rho = 0.25$, $h = 20$, $d = 20$, and $\zeta = 24$. The heuristic function $\eta(f_i, v_{i,j})$ used the weights $w_\Gamma = w_d = 1$, $w_c = 0$ (thus, the conflict values $c_{i,j}$ were not considered in these experiments) and offsets $k_\Gamma = k_d = 4$. The ACS terminated when the improvement of x^{gb} had dropped below $\varepsilon = 0.1\%$ during the last $\gamma = 600$ iterations.

SA was run with exactly those parameter settings suggested in [2]. HGA used a population size of 100, a mutation probability of 3% per (unmasked) gene, and each run was terminated when the improvement of the best solution during the last 20,000 iterations had dropped below 0.1%.

Table I shows obtained results. All values are average values over 15 runs per instance in case of HGA and ACS and 10 runs per instance in case of SA. Printed are the finally best solutions' objective values $g(x)$, their standard deviations $\sigma(g(x))$, the numbers of finally remaining labels with conflicts $conf(x)$, and the numbers of evaluated solutions $evals$.

Most of the time, HGA and ACS found better solutions than SA. While HGA exhibits slight advantages over ACS on smaller instances with up to $n = 450$ features, the ACS is superior on the harder, denser cases. Statistical t-tests reveal that the differences in $g(x)$ from ACS and HGA for $n \geq 500$ are significant at a 0.1% error-level. It is furthermore remarkable that ACS created almost always solutions with fewer conflicting labels than HGA.

Regarding the number of needed evaluations, ACS is superior in any case. However, this does not mean that ACS is the fastest. For HGA and ACS, average CPU-times t_{CPU} measured in seconds on a Celeron/450MHz PC are also printed in Table I. In particular for the larger instances, ACS needed significantly more time due to its higher computational effort for generating one candidate solution. CPU-times are not printed for SA since those runs were performed on a different machine. Nevertheless, we could clearly see that SA is generally the fastest, despite its high numbers of evaluated solutions. The reason is that SA creates a new solution from a previous one by just changing one label's position. In this way, an incremental evaluation is possible, and the computational effort is minimal.

Figure 6 shows part of a solution to a real geographical problem instance. The complete instance has $n = 1501$ features and could be solved by the ACS without overlaps and a final objective value of $g(x) = 215.75$ in 35.6 seconds.

V. CONCLUSIONS

The main features of the proposed ACS for point feature labeling are: the deterministic problem reduction during preprocessing, the heuristic feature selection strategy which considers spatial relationships of features, the local improvement of created solutions, and masking which effectively focuses the optimization to critical regions in the surrounding of conflicting labels.

Results show that the ACS is able to identify high-quality solutions. In particular for hard, dense instances, these solutions are usually significantly better than those obtained by simulated annealing and a state-of-the-art hybrid genetic algorithm for point feature labeling. The main reasons for the high performance of the ACS are the strong local heuristics and the good balance between exploration and exploitation.

Future work should try to reduce the rather large number of strategy parameters which must be set appropriately in order to provide a good balance between exploration and exploitation. Probably, self-adaption mechanisms can help here.

TABLE I
RESULTS OF SIMULATED ANNEALING (SA), THE HYBRID GA (HGA), AND THE ANT COLONY SYSTEM (ACS).

n	SA			HGA					ACS				
	$g(x)$	$conf(x)$	$evals$	$g(x)$	$\sigma(g(x))$	$conf(x)$	$evals$	$t_{\mathrm{CPU}}\ [s]$	$g(x)$	$\sigma(g(x))$	$conf(x)$	$evals$	$t_{\mathrm{CPU}}\ [s]$
50	0.5	0.0	41,251	0.4	<0.01	0.0	20,014	4.9	0.4	<0.01	0.0	1,802	<0.1
100	1.5	0.0	82,798	1.4	<0.01	0.0	20,012	4.1	1.4	<0.01	0.0	1,374	<0.1
150	4.2	0.0	126,411	3.7	<0.01	0.0	20,242	4.6	3.8	0.10	0.0	2,875	0.3
200	6.8	2.0	169,959	4.6	0.03	0.0	20,397	6.2	4.7	0.09	0.0	2,986	0.6
250	17.7	8.0	214,107	9.4	0.10	0.0	20,544	7.9	9.6	0.20	0.0	3,549	1.1
300	18.5	0.0	261,368	18.2	0.19	1.0	20,773	11.7	18.4	0.16	1.0	4,414	3.6
350	36.8	10.3	305,757	22.0	0.25	0.1	20,662	14.7	22.2	0.32	0.0	4,666	3.3
400	35.8	0.6	352,770	34.6	0.55	1.5	21,324	19.3	35.0	0.47	0.7	6,534	6.7
450	47.7	4.3	398,063	39.5	0.45	1.0	21,019	22.2	39.6	0.40	1.0	8,418	11.7
500	74.4	21.1	440,866	57.7	0.84	2.5	21,601	28.0	55.3	0.81	1.1	9,170	15.0
550	89.5	27.6	486,704	77.9	1.35	12.1	22,978	33.8	75.8	1.18	10.2	14,541	57.2
600	116.9	40.1	529,178	101.8	1.56	19.8	25,626	44.3	97.6	0.87	17.2	16,542	73.0
650	161.6	69.1	574,111	143.3	1.76	37.3	28,995	58.4	137.0	1.99	35.8	22,850	178.8
700	177.2	72.1	618,253	165.0	1.84	40.1	30,532	68.9	157.9	1.50	37.8	21,246	167.4
750	199.9	74.9	663,544	180.5	2.22	36.3	29,031	73.0	172.7	1.63	33.9	23,388	201.1
800	260.5	119.9	704,693	245.4	3.08	81.3	45,934	135.4	230.7	1.57	79.7	30,017	408.3
850	290.6	133.4	748,523	275.9	2.37	90.3	45,395	147.4	258.1	1.59	84.4	31,515	471.3
900	337.4	164.8	791,061	323.3	2.65	125.3	48,409	172.9	299.2	2.14	123.5	32,790	583.3
950	366.5	182.1	834,695	375.2	3.80	148.5	52,182	201.2	345.4	2.95	146.0	34,666	671.5
1000	437.9	251.3	875,286	444.7	4.39	209.7	57,931	241.3	408.6	2.44	208.6	35,772	791.0

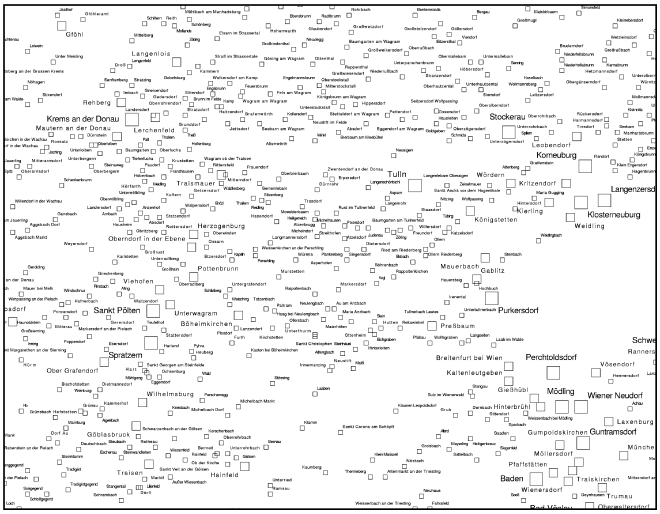


Fig. 6. Part of a solution to a geographical problem instance with $n = 1501$ features. ACS was running 35.6 seconds.

References

[1] R. Beckers, J. L. Deneubourg, and S. Goss. Trails and U-turns in the selection of the shortest path by the ant Lasius Niger. *Journal of Theoretical Biology*, 159:397–415, 1992.

[2] J. Christensen, J. Marks, and S. Shieber. Placing text labels on maps and diagrams. In P. S. Heckbert, editor, *Graphic Gems IV*, pages 497–504. Academic Press, 1994.

[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] D. Corne, M. Dorigo, and F. Glover. *New Ideas in Optimisation*. McGraw-Hill, 1999.

[5] R. G. Cromley. A spatial allocation analysis of the point annotation problem. In *Proceedings of the 2nd International Symposium on Spatial Data Handling*, pages 38–49, 1986.

[6] J. Doerschler and H. Freeman. A rule-based system for dense-map name placement. *Communications of the ACM*, 35(1):68–79, 1992.

[7] M. Dorigo, G. Di Caro, and L. M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172, 1999.

[8] M. Dorigo and L. M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[9] S. Edmondson, J. Christensen, J. Marks, and S. Shieber. A general cartographic labeling algorithm. *Cartographica*, 33(4):13–23, 1997.

[10] M. Formann and F. Wagner. A packing problem with applications to lettering of maps. In *Proceedings of the 7th Annual Symposium on Computational Geometry*, pages 281–288, 1991.

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

[12] E. Imhof. Positioning names on maps. *The American Cartographer*, 2(2):128–144, 1975.

[13] G. W. Klau and P. Mutzel. Optimal labelling of point features in the slider model. In D.-Z. Du, P. Eades, V. Estivill-Castro, X. Lin, and A. Sharma, editors, *Proceedings of the 6th Annual International Computing and Combinatorics Conference*, volume 1858 of *LNCS*, pages 340–350. Springer, 2000.

[14] J. Marks and S. Shieber. The computational complexity of cartographic label placement. Technical Report TR-05-91, Harvard University, Center for Research in Computing Technology, Cambridge, MA, 1991.

[15] G. R. Raidl. A genetic algorithm for labeling point features. In *Proceedings of the Interntational Conference on Imaging Science, Systems and Technology*, pages 189–196, 1998.

[16] M. Schreyer. Ein Genetischer Algorithmus und ein Ant Colony System für das Point Feature Labeling Problem. Master's thesis, Vienna University of Technology, Vienna, Austria, 2001.

[17] O. V. Verner, R. L. Wainwright, and D. A. Schoenefeld. Placing text labels on maps and diagrams using genetic algorithms with masking. *INFORMS Journal on Computing*, 9(3):266–275, 1997.

[18] P. Yoeli. The logic of automated map lettering. *The Cartographic Journal*, 9(2):99–108, 1972.

[19] S. Zoraster. The solution of large 0–1 integer programming problems encountered in automated cartography. *Operations Research*, 38(5):752–759, 1990.