Chapter 13

A CONSTRUCTIVE GENETIC APPROACH TO POINT-FEATURE CARTOGRAPHIC LABEL PLACEMENT

Missae Yamamoto[1] and Luiz A.N. Lorena[2]

INPE – Instituto Nacional de Pesquisas Espaciais
Caixa Postal 515
12.245-970 São José dos Campos – SP, Brazil

[1] missae@dpi.inpe.br
[2] lorena@lac.inpe.br

Abstract: The cartographic label placement is an important task in automated cartography and Geographical Information Systems (GIS). Positioning the texts requires that overlap among texts be avoided, that cartographic conventions and preference be obeyed. So, the label placement belongs to a problem area of difficult solution. A variety of methods have been proposed to generate quality labeling, with a wide range of results. In this work, two methods are presented, a Constructive Genetic Algorithm (CGA) and an initial exact method for small instances. The CGA application produced quality-labeling placements for printed maps, and the exact method is used to confirm the superiority of these results.

Key words: Constructive genetic algorithm, point-feature cartographic label placement, genetic algorithm and exact algorithm.

13.1 INTRODUCTION

Cartographic label placement refers to the text insertion process in maps and is one of the most challenging problems in geoprocessing and automated cartography [12]. Positioning the texts requires that overlap among texts be avoided, that cartographic conventions and preferences be obeyed, that unambiguous association be achieved between each text and its corresponding feature and that a high level of harmony and quality be achieved. In this article we are concerned with the placement of labels for

point features, approaching the problem from a combinatorial optimization viewpoint.

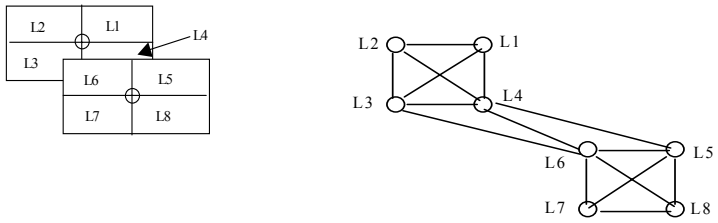

Figure 13.1. Two points – potential label positions and corresponding conflict graph

The Point-Feature Cartographic Label Placement (PFCLP) problem considers potential label positions for each point feature, which will be considered as candidates. Figure 13.1 shows two points with a set of four potential label positions and the corresponding conflict graph, where vertices correspond to labels and edges to possible overlapping in labels. The PFCLP considers the *placement of all labels* searching for the *large subset of labels with no conflict*. A related but different problem appears when label selection is permitted and a subset of points are not labeled (cannot be labeled without conflicts). In that case the problem searches the maximum independent set of vertices on the conflict graph [10].

Several heuristics and metaheuristics have been used to approximately solve the PFCLP problem. They are partially reviewed by Christensen et al. [1]. The algorithms included a version of Zoraster's integer programming algorithm (Lagrangean relaxation) [14], a version of Hirsch's continuous gradient-descent algorithm [3], a discrete gradient-descent algorithm, and a stochastic optimization algorithm using simulated annealing. A Genetic Algorithm (GA) with mask is described in [11] and [13] presented a Tabu Search algorithm showing better results in label placement quality than all these other methods.

This work describes the application of a Constructive Genetic Algorithm (CGA) to PFCLP. The CGA was proposed by Lorena and Furtado [4] and applied to timetabling [9] and Gate Matrix Layout Problems [7, 8]. Basically it differs from other GAs for evaluating schemata directly. It also has a number of new features compared to a traditional GA. These include a population of dynamic size composed of schemata and structures, and the possibility of using heuristics in the fitness function definitions.

A simple tree search algorithm is also proposed to validate the CGA computational results on small-scale instances.

The work is organized as follows. Section 13.2 presents a pseudo-code for the CGA and describes aspects of modeling for schema and structure representations and the consideration of the PFCLP as a bi-objective optimization problem. Section 13.3 describes some CGA operators, namely, selection, recombination and mutation. Section 13.4 shows computational results using instances formed by standard sets of randomly generated points suggested in the literature. Section 13.5 presents the exact algorithm with some computational results showed in the section 13.6.

13.2 CGA MODELING FOR PFCLP

The CGA for PFCLP is modeled as a bi-objective optimization problem indirectly solved by an evolutionary process. We first describe the structure and schemata representation.

Structure and schema representation: The CGA operates as an iterative procedure on a population or pool of individuals. The individuals represent an encoding of the problem into a form that is analogous to the chromosomes of biological systems. Each chromosome is made up of a string of genes, whose values are called alleles. In PFCLP, each individual or chromosome represents a label distribution configuration for a given point set, where each allele in the chromosome corresponds to a point feature label and may take on one of the potential label position or the symbol #, indicating that the point is temporarily out of the problem. So, if we use a chromosome (individual) with length 10, s_k = (1, 4, #, #, 1, 3, 2, 1, #, 2) is a possible schema, where the numbers 1, 2, 3, 4, refer to the potential label positions for each point feature (four potential positions) and the symbol # represents that the corresponding point is temporarily not being considered. The structures represent feasible solutions to PFCLP. All labels are positioned and the symbol # is not present in the individual, for example, s_k = (1, 4, 2, 1, 1, 3, 2, 1, 4, 2) is a possible structure.

So, the CGA evaluate structures and parts of it, called schemata. In this work we use the word individual as a generic term to cover both structure and schema.

Fitness functions: Let X be the space of all schemata and structures s_k = (label or #, label or #, …, label or #), that can be created using the alphabet {label, #}, where label can be any corresponding point label (in Figure 13.1: L1, …, L8), and # is an indetermination typical of schemata.

The PFCLP is modeled using two fitness functions. Function f returns the number of conflict free labels in s_k and g returns the corresponding

number after an heuristic application in s_k. The variation $(g(s_k) - f(s_k))$ reflects the local search improvement and individuals with few # labels have higher $g(s_k)$ values (schemata have lower $g(s_k)$ values than structures).

In our CGA for the PFCLP the variation $(g(s_k) - f(s_k))$ plays two roles:

- *Interval minimization*: we would like to search for a $s_k \in X$ that minimizes $g(s_k) - f(s_k)$, since the response to the evolutionary search is given on a very adapted individuals that have a large number of conflict free labels.
- *G maximization*: we would like to search for a $s_k \in X$ that maximizes $g(s_k)$, since we need to ensure feasibility. A individual that has very few labels assigned (it is a schema in which most points are labeled #) will not be a feasible solution to the PFCLP. This objective can be viewed as encouraging the process to move from schemata to structures (feasible solutions).

Hence our CGA implicitly considers the following bi-objective optimization problem (BOP):

$$\begin{array}{ll} \text{Minimize} & g(s) - f(s) \\ \text{Maximize} & g(s) \\ \text{Subject to} & g(s) \geq f(s), \quad \forall s \in X. \end{array}$$

The BOP defined above is not directly considered as the set X is not completely known. Instead we consider an *evolution process* to attain the objectives (*interval minimisation* and *g maximization*) of the BOP.

At the beginning of the process, two *expected values* are given to these objectives:

- for the *g maximization* objective we use a value $g_{max} \geq max_{s \in X} g(s)$ that is an upper bound on the objective value,
- for the *interval minimization* objective we use a value dg_{max}, obtained from g_{max} using a real number $0 < d \leq 1$.

The evolution process proceeds using an adaptive rejection threshold, which considers both objectives described before. Given a parameter $\alpha \geq 0$, let g_{max} be the total number of labels, then an individual s_k is discarded from the population if:

$$g(s_k) - f(s_k) \geq d.g_{\max} - \alpha.d[g_{\max} - g(s_k)] \qquad (13.1)$$

The right-hand side of expression (13.1) is the threshold for individual removal from the population and is composed of a expected value dg_{max} associated with the interval minimization and the measure $[g_{max} - g(s_k)]$, which is the difference between g_{max} and $g(s_k)$ evaluations. For $\alpha = 0$

equation (13.1) is equivalent to comparing the interval length associated with s_k against the expected length dg_{max}. When a $\alpha > 0$, individuals containing a high number of # labels (i.e. schemata) have a higher probability of being discarded as, in general, they have higher differences $[g_{max} - g(s_k)]$ since g_{max} is fixed and $g(s_k)$ is smaller for schemata than for solutions.

The population is controlled by an evolution parameter (time) α , receiving small increments from generation to generation. The corresponding population is denoted by P_α , and have a number of schemata and structures, ranked by the following expression (rearranging equation (13.1)):

$$\delta(s_k) = \frac{d \cdot g_{max} - [g(s_k) - f(s_k)]}{d[g_{max} - g(s_k)]} , \qquad (13.2)$$

where $g_{max} \neq g(s_k)$ (in general, otherwise an optimal solution was found, i. e., all labels positioned without conflicts), considering one label for each point of the configuration.

The size of population is then dynamically controlled by α , and can be emptied during the evolution process. At the time they are created, structures and/or schemata s_{new} receive a rank value $\delta(s_{new})$, which is compared with the current evolution parameter α. If $\alpha < \delta(s_{new})$, the new individual is accepted. At each generation this survival test is also applied to the individuals in the current population. Expression (13.2) shows that better schemata or structures have high ranks, reflected by small variations $(g(s_k) - f(s_k))$ and/or greater $g(s_k)$ values. In this sense better individuals are those trained by the heuristic and have a large number of conflict free labels.

The algorithm implemented in this work is showed as follows.

CGA { Constructive Genetic Algorithm }
  Given g_{max} *, d and* $\alpha = 0$*;*
  Initialize P_α *;*
  for all $s_k \in P_\alpha$ *compute* $g(s_k)$*,* $f(s_k)$*,* $\delta(s_k)$*;*
  while (not stop condition) do
    If $\alpha < \delta(s_k)$*, add* s_k *to* $P_{\alpha+1}$*, for all* $s_k \in P_\alpha$ *;*
    Define the selection of s_{base} *and* s_{guide} *;*
    while (number of recombinations) do
      Select s_{base} *and* s_{guide} *from* P_α *;*
      Recombine s_{base} *and* $s_{guide} \rightarrow s_{new}$ *;*
      Apply local search heuristic (mutation) to s_{new} *;*
      Compute $g(s_{new})$*,* $f(s_{new})$*,* $\delta(s_{new})$*;*

If $\alpha < \delta(s_{new})$, *add* s_{new} *to* $P_{\alpha+1}$;
end_while
Do s_{base} *mutation and keep the best solution;*
Update α ;
end_while

The best $g(s_k)$ is kept in the process, because it can be a good potential solution. The initial population, selection, recombination, mutation, the α updating and the heuristics used to calculate $g(s_k)$ and the local search heuristic, are all detailed in the following.

13.3 CGA OPERATORS

The *initial population* is composed exclusively of schemata, considering that for each schema, a proportion of random positions receive a label number. The remaining positions receive labels #. Along the generations, the population increases by addition of new offspring generated out of the combination of two schemata.

For *selection*, the structures and schemata in population P_α are maintained in ascending order by the key: $[1 + [(g(s_k) - f(s_k)) / g(s_k)]] / \eta(s_k)$, where $\eta(s_k)$ is the number of points labeled in s_k . The best-ranked individuals (schemata and/or structures) appear in first order positions.

Two structures and/or schemata are selected for recombination. The first is called the base (s_{base}) and is randomly selected out of the first positions in P_α, and in general it is a good structure or a good schema. The second structure or schema is called the guide (s_{guide}) and is randomly selected out of the total population. The objective of the s_{guide} selection is the conduction of a guided modification on s_{base}.

In the *recombination* operation, the current labels in corresponding positions are compared. Let s_{new} be the new structure or schema (offspring) after recombination. The structure or schema s_{new} is obtained by applying a recombination with mask based in Verner et al. [11]:

M_{base} = *mask of* s_{base} *(0 = conflict, 1 = without conflict, 2 = #)*
M_{guide} = *mask of* s_{guide} *(0 = conflict, 1 = without conflict, 2 = #)*
U = 0 or 1 (randomly generated)
Repeat for each position (j) in structure or schema representation:
If $M_{base}(j) = 1$ *then* $s_{new}(j) \leftarrow s_{base}(j)$
If $M_{base}(j) \neq 1$ *and* $M_{guide}(j) = 1$ *then* $s_{new}(j) \leftarrow s_{guide}(j)$
If $M_{base}(j) \neq 1$ *and* $M_{guide}(j) \neq 1$ *and* $U = 0$

$$\text{then } s_{new}(j) \leftarrow s_{guide}(j)$$
$$\text{If } M_{base}(j) \neq 1 \text{ and } M_{guide}(j) \neq 1 \text{ and } U = 1$$
$$\text{then } s_{new}(j) \leftarrow s_{base}(j)$$

The *mutation* operation applies a simple local search heuristic to s_{new} or to s_{base}. If s_{base} is a schema, it is initially transformed in structure changing #s by labels. The labels are positioned searching for a small number of extra conflicts. The new individuals s_{new} and these new structures obtained directly from s_{base}, are then improved by a *local search heuristic* that tries to change each conflicted label by a conflict free position. Computational tests showed that 5 replications reached good results with reasonable processing times, then the local search heuristic is recursively applied five times to improve structure quality. The best solution is kept in the process, but the mutants are not inserted into the new population.

Considering that the well-adapted individuals need to be preserved for recombination, the *evolution parameter* α is started with zero, and Oliveira and Lorena [7] suggest to increase it with step proportional to actual population size $|P_\alpha|$, following this formula:

$$\alpha = \alpha + k \cdot |P_\alpha| \cdot \frac{\delta_{top} - \delta_{bot}}{Gr} + l \qquad (13.3)$$

where, k is a proportionality constant, l is the minimum increment allowed, Gr is the remaining number of generations, and $(\delta_{top}-\delta_{bot})$ is the actual range of values of δ. The adaptive increment of α is then affected by the search history (population size, best and worst δ's, etc). Thus, once the CGA achieves very good regions and does not get to improve the best rank, the parameter α goes eliminating the individuals until the population is emptied.

To calculated the $g(s_k)$, a *recursive smallest first* (RSF) heuristic is used and drives the evolution process to a trained population. The RSF is very simple (other sophisticated heuristics can be tried). It takes the subset of labeled points in s_k (points without #s) and their original potential label conflicts, i. e., a subgraph of the conflict graph. The vertices of the subgraph are then considered in non-decreasing order of degrees. The corresponding point receives a label with no conflict and the subgraph is updated eliminating this label (vertex) and their incident vertices and edges. The process is then repeated until no more labels to place.

The *local search heuristic* (mutation) produces smaller changes to individuals than the RSF and is also used to calculate the $g(s_k)$ on the initial population. It allows the initial individuals to survive in the next generation, since they will have high ranks. The heuristic looks points without #s in

natural order at each position in structure or schemata representation and changes the label if the corresponding number of conflicts results to be smaller.

13.4 CGA COMPUTATIONAL RESULTS

Christensen et al. [1], Verner et al. [11] and Yamamoto et al. [13] compared several algorithms using standard sets of randomly generated points: grid size of 792 by 612 units, fixed size label of 30 by 7 units and page size of 11 by 8.5 inch.

In order to compare the CGA algorithm with previous works, the standard sets of randomly generated points and the same conditions as described by [1] are used, following the same assumptions as [11]. The set of instances has the following characteristics (all the instances used in this paper are available at *www.lac.inpe.br/~lorena/instancias.html*):

a) Number of the points: N = 100, 250, 500, 750, 1000;
b) Configurations: For each problem size, 25 different configurations with random placement of point feature using different seeds;
c) Penalties: No penalty was attributed for labels that extended beyond the boundary of the region;
d) 4 potential label positions were considered;
e) Cartographic preferences were not taken into account;
f) No point selection was allowed (i.e., no points are removed even if avoiding superposition is inevitable);

The parameters used for CGA are presented in Table 13.1. These values change in accordance with the instance and are determined executing a preliminary battery of computational tests with different parameters values until acceptable results were found. The computational results are presented in Table 13.2. The results, for each problem size, present the average percentage of labels placed without conflict for the 25 trials.

Table 13.1. CGA parameters

N	**100**	**250**	**500**	**750**	**1000**
Initial population size	30	75	150	200	300
k	0.01	0.01	0.01	0.01	0.01
l	0.00001	0.00001	0.00001	0.00001	0.00001
Offspring	30	30	30	30	30
Number of generations	25	25	25	25	25
d	0.2	0.2	0.2	0.2	0.2
Number of # (initial population)	1	2	10	30	70
% of sub-population Base	15%	15%	15%	15%	15%

Table 13.2. Computational results

Algorithm	**100**	**250**	**500**	**750**	**1000**
CGA_{best}	100.00	100.00	99.6	97.1	90.7
$CGA_{average}$	100.00	100.00	99.6	96.8	90.4
Tabu search [13]	100.00	100.00	99.2	96.8	90.00
GA with masking [11]	100.00	99.98	98.79	95.99	88.96
GA [11]	100.00	98.40	92.59	82.38	65.70
Simulated Annealing [1]	100.00	99.90	98.30	92.30	82.09
Zoraster [14]	100.00	99.79	96.21	79.78	53.06
Hirsch [3]	100.00	99.58	95.70	82.04	60.24
3-Opt Gradient Descent [1]	100.00	99.76	97.34	89.44	77.83
2-Opt Gradient Descent [1]	100.00	99.36	95.62	85.60	73.37
Gradient Descent [1]	98.64	95.47	86.46	72.40	58.29
Greedy [1]	95.12	88.82	75.15	58.57	43.41

Regarding the optimization algorithms of the literature, the CGA showed superior results in quality of label placement. Table 13.2 shows the percentage of labels placed without conflict for 100, 250, 500, 750 and 1000 points, considering different algorithms of the literature. The $CGA_{average}$ reports the average result for 6 trials and CGA_{best} is the best result. The lines show the percentage of labels placed without conflict by the optimization algorithms tested on [1] (greedy-depth first, gradient descent, 2-opt gradient descent, 3-opt gradient descent, Hirsch, Zoraster and simulated annealing), [11] (GA without masking and GA with masking), [13] (Tabu search) and the CGA.

Figure 13.2 shows a label placement for 1000 points after using our CGA algorithm.

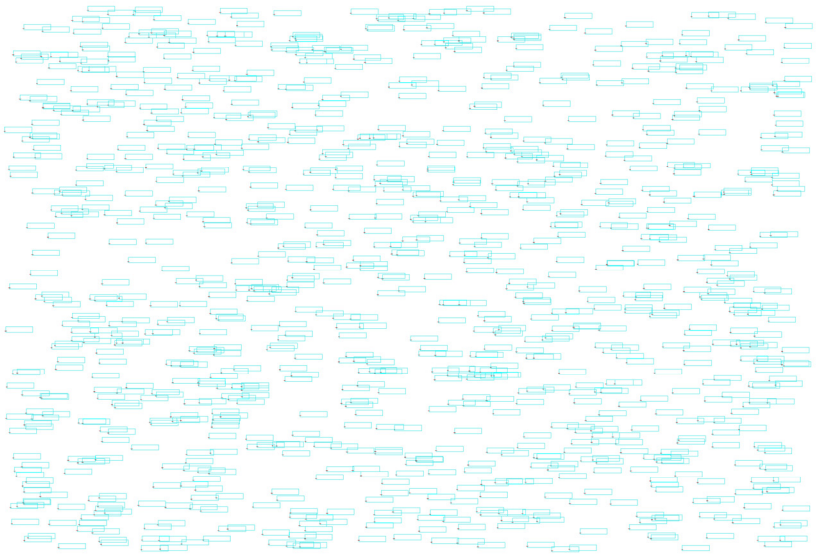

Figure 13.2. Final results after CGA application for 1000 random points (overlap = 88).

Table 13.3 compares the Tabu search of [13], CGA_{best} and $CGA_{average}$ running times (seconds) to obtain the best solutions (using a Pentium II computer). The Tabu search approach is faster than CGA.

Table 13.3. Computational times to reach the best solutions

Algorithm	**100**	**250**	**500**	**750**	**1000**
CGA_{best}	0	0.6	21.5	228.9	1227.2
$CGA_{average}$	0	0.6	21.5	195.9	981.8
Tabu search [13]	0	0	1.3	76.0	352.9

13.5 EXACT ALGORITHM

The label placement is a combinatorial optimization problem of difficult solution and Marks and Shieber [6] showed that the point features label placement problem is NP-hard. The PFCLP defined in this paper have the same characteristics and is also a difficult problem. This section describes an

initial tree search exact algorithm to validate the CGA and Tabu search results presented and compared in last section.

The exact algorithm to solve PFCLP problem follows the Loukakis and Tsouros [5] algorithm for maximum independent set problems (see also Dowsland [2]), adapting the pruning bounds and branching decisions. It searches the tree in a depth first procedure.

The algorithm implemented in this work is showed as follows.

Exact algorithm

Given:

n number of points

$S := \{\}$ set of active label positions (solution)

V set of all potential label positions

M number of labels without conflicts *(CGA estimative)*

$P = \{P_1, P_2, ..., P_n\}$ *set of* points in ascending order of vertices degrees

While $|S| \neq n$ do

Choose from V a label of point P_i and add it to S

Compute *nc (number of labels with conflicts in S)*

Compute *nsc (number of labels without conflicts in S)*

If $(nsc > M)$ then $M := nsc$

Verify *future (number of points unlabeled that have candidate label positions without conflict with active labels of S)*

If $(nsc + future \leq M)$ *or* $(nc > n\text{-}M)$ execute pruning

end_while

The point ordination, labels choice, future verification, pruning and branching, are detailed in the following.

The *point ordination* refers to the sequence that the points are labeled. This sequence is set up in ascending order of vertices degrees, where vertices are the potential label positions of all points, and vertex degree refers to the number of conflicts of the potential label position.

The *labels choice* is doing in ascending order of the point potential label positions cost. The algorithm places each label position for points in a prescribed order. If, as the algorithm proceeds, a point cannot be labeled (either because the number of conflicts exceeded an established boundary, or the sum of the number of labels positioned without conflicts and the number of labels without conflicts that can be positioned in future is smaller than the CGA estimative of number of labels without conflicts), the algorithm returns to the most recently labeled point and considers the next available position

(known as *pruning the tree)*. The algorithm continues in this way until an acceptable labeling is identified or until the entire search space has been exhausted.

For the exact algorithm, pruning in higher tree levels becomes a very attractive technique, as it reduces the number of the configurations to be analyzed, taking to a lower processing time.

In the *future* verification, the algorithm do a foresight, verifying if at least one potential label position, of the points unlabeled, do not have conflicts with active labels of the labeled points.

This algorithm is in a very preliminary fashion and can be improved in some ways, as for example, in pre-processing the initial conflict graph. If a mathematical programming formulation was available, upper bounds (Lagrangean for example) could be useful.

13.6 EXACT ALGORITHM RESULTS

The computational tests presented in Table 13.2 show that for the standard set of randomly generated points (grid size of 792 by 612 units, fixed size label of 30 by 7 units and page size of 11 by 8.5 inch) the difficult instances have 750 and 1000 points. It appears that positioning all labels will be impossible for these instances, particularly for the 1000 points. Then in order to approximately get the 1000 points configuration difficulty, we generate 8 different configurations with random placement of 25 point features using different seeds. The same region size and the same paper size suggested for Christensen et al. [1] is used, but the label sizes increased of 8.5%. The rest of conditions were remaining: no penalty was attributed for labels that extended beyond the boundary of the region, 4 potential label positions were considered, cartographic preferences were not taken into account and no point selection was allowed. The instances used in this paper are also available at *www.lac.inpe.br/~lorena/instancias.html*.

The exact algorithm, implemented in C++ language, was applied to these instances and the results (average over 8 trials) are recorded in Table 13.4, as follows:

Table 13.4. Results from Exact Algorithm

Configuration	Time (sec)	Overlapping labels	Not overlapping labels	Results (%)
1	681	2	23	92
2	876	3	22	88
3	864	3	22	88
4	1495	7	18	72
5	749	3	22	88
6	572	4	21	84
7	1726	7	18	72
8	2061	4	21	84
Results (average)	1128	4.125	20.875	83.5

Where:

a) Configuration: configuration number.
b) Time (sec.): processing time to get exact solution in a SUN – SPARC20 workstation.
c) Overlapping labels: labels placed with conflicts.
d) Not overlapping labels: labels placed without conflict.
e) Results (%): the percentage of labels placed without conflict.
f) Results (average): average of time, overlapping labels, not overlapping labels and results (%).

The CGA and TS (Tabu search of Yamamoto et al. [13]) results are compared with the exact solution, using this set of points.

The parameters used for TS are (see Yamamoto et al. [13]):

a) Tabu list size = 7 + INT(0.25 * number of labels that overlap).
b) Candidate list size = 1 + INT(0.05 * number of labels that overlap).
c) Number of iterations for recalculation = 50.

The parameters used for CGA are:

a) Initial population size: 500.
b) K: 0.00005.
c) *l:* 0.0001.
d) Offspring: 500.
e) Number of generations: 100.
f) d: 0.9
g) Number of # (initial population): 13.
h) % of sub-population Base: 15%.

Both TS and CGA algorithms showed results equal or very close to the exact solution for the 8 configurations. The CGA approach resulted to be better than TS. Table 13.5 shows the labels placed without conflict and

percentage of labels placed without conflict for the exact algorithm, TS and CGA. The $CGA_{average}$ reports the average result for 6 trials and CGA_{best} is the best result.

Table 13.5. Comparison of TS, CGA and Exact Algorithm (labels placed without conflict)

	Exact algorithm		**Tabu search**		**$CGA_{average}$**		**CGA_{best}**	
Config	Res	%	Res	%	Res	%	Res	%
1	23	92	22	88	23	92	23	92
2	22	88	22	88	22	88	22	88
3	22	88	21	84	21	84	21	84
4	18	72	18	72	18	72	18	72
5	22	88	19	76	20.5	82	22	88
6	21	84	21	84	20	80	20	80
7	18	72	17	68	16	64	16	64
8	21	84	21	84	21	84	21	84
Average	20.875	83.5	20.125	80.5	20.1875	80.75	20.375	81.5

Where:

a) Config.: configuration number.
b) Res: labels placed without conflict.
c) %: percentage of labels placed without conflict.
d) Results (average): average of not overlapping labels and percentage of labels placed without conflict, for exact algorithm, TS and CGA.

The CGA and TS algorithms were tested on a Pentium II computer, and exact algorithm on a SUN – SPARC20 workstation, and therefore processing times are not comparables, but the workstation was necessary to accelerate the exact algorithm computer times.

13.7 CONCLUSION

This work has proposed and evaluated a CGA applied to the PFCLP problem. By using a standard set of randomly generated points and the same conditions described by [1], [11] and [13], the CGA showed better results in label placement quality than other methods published in the literature.

The exact algorithm proposed in this work labels all points even if some of them present conflicts. The algorithm tries to reach the exact solution in a reasonable time, pruning the tree when the known beforehand estimative (provided from CGA) is better.

The CGA and TS found results very close to exact solutions, and the CGA has better results in label placement quality than TS. The low density (in edges) of the conflict graph shows that some improvement can be done in

CGA computer times if the graph is partitioned and solved by a computer-distributed system.

The CGA can be recommended to solve the automatic cartographic label placement problem for point features.

ACKNOWLEDGEMENTS

The first author acknowledges Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) for financial support. The second author acknowledges Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq (process 300837/89-5) for partial financial research support.

The comments and suggestions of two anonymous referees were very useful and appreciated.

REFERENCES

1. Chistensen, J.; Marks, J.; Shieber, S. *An empirical study of algorithms for point-feature label placement.* ACM Transactions on Graphics, 14 (3): 203-232, 1995.
2. Dowsland, K. A. *An exact algorithm for the pallet loading problem.* European Journal of Operational Research, 31(1): 78-84, 1987.
3. Hirsch, S. A. *An algorithm for automatic name placement around point data*. American Cartographer, 9(1): 5-17, 1982.
4. Lorena, L.A.N.; Furtado J.C. *Constructive genetic algorithm for clustering problems*. Evolutionary Computation. 9(3): 309-327, 2001.
5. Loukakis, E. and Tsouros, C. *Determining the number of internal stability of a graph.* International Journal of Computer Mathematics. 11: 207 – 220, 1982.
6. Marks, J.; Shieber, S. *The Computational Complexity of Cartographic Label Placement.* TR-05-91, Center for Research in Computing Technology, Harvard University, 1991.
7. Oliveira A.C.M.; Lorena L.A.N. *A Constructive Genetic Algorithm for Gate Matrix Layout Problems*. IEEE Transactions on Computer-Aided Designed of Integrated Circuits and Systems, 21 (8): 969-974, 2002.
8. Oliveira, A.C.M.; Lorena, L.A.N. *2-Opt Population Training for Minimization of Open Stack Problem.* In Advances in Artificial Intelligence, G. Bittencourt and G. L. Ramalho (Eds). Springer Lecture Notes in Artificial Intelligence 2507: 313-323, 2002.
9. Ribeiro Filho, G. and Lorena, L. A. N. "A constructive evolutionary approach to school timetabling," In Applications of Evolutionary

Computing, Boers, E.J.W., Gottlieb, J., Lanzi, P.L., Smith, R.E., Cagnoni, S., Hart, E., Raidl, G.R., Tijink, H., (Eds.). Springer Lecture Notes in Computer Science vol. 2037, pp. 130-139 - 2001

10. Strijk, T.; Verweij, B.; Aardal, K. *Algorithms for Maximum Independent Set Applied to Map Labeling. September*, Technical Report UU-CS-2000-22, Department of Computer Science, Utrecht University, 2000. 42p. Available at *ftp://ftp.cs.uu.nl/pub/RUU/CS/techreps/CS-2000/2000-22.pdf*

11. Verner, O. V.; Wainwright, R. L.; Schoenefeld, D. A. *Placing text labels on maps and diagrams using genetic algorithms with masking*. INFORMS J. on Computing, 9: 266-275, 1997.

12. Wolff, A.; Strijk, T. *The Map Labeling Bibliography*. http://i11www.ilkd.uni-karlsruhe.de/map-labeling/bibliography/ , 1996.

13. Yamamoto, M., Camara, G. and Lorena, L. A. N. *Tabu search heuristic for point-feature cartographic label placement.* GeoInformatica. Kluwer Academic Publisher, Netherlands, 6(1): 77-90, 2002.

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