VII Simpósio Brasileiro de Geoinformática, Campos do Jordão, Brasil, 20-23 novembro 2005, INPE, p. 122-135.

Fast Point-Feature Label Placement Algorithm for Real Time Screen Maps

Missae Yamamoto, Gilberto Camara, Luiz Antonio Nogueira Lorena

National Institute of Space Research - INPE, São José dos Campos, SP, Brazil

São José dos Campos – SP – Brazil

{missae, gilberto}@dpi.inpe.br, lorena@lac.inpe.br

Abstract. *The generation of good label placement arrangements is a frequent problem when producing maps. The objective of a good label placement is to display the geographic position of the features with their matching label in a clear and harmonious fashion, following accepted cartographic conventions. In this work, we propose the fast algorithm for label placement (FALP) for generation of real time screen maps. FALP is a cost-effective choice, with good quality performance and excellent runtime performance.*

1. Introduction

Point label placement refers to insertion of text in maps and is a challenging problem in geoinformatics and automated cartography (Wolff and Strijk 1996). Text positioning requires avoiding overlaps and adherence to cartographic conventions and preferences. There should be unambiguous association between each text and its matching feature. Overall, good labeling needs harmony and quality in the resulting maps. The motivation for research in automated label placement includes:

- Label placement is a fundamental part of producing good maps and essential item to communicate spatial information;
- Placing text manually in maps is a laborious procedure;
- A good label placing algorithm brings a substantial increase in map productivity;

Label placement would be much simpler if labels could be pre-computed for each layer at once and their position stored beforehand. Unfortunately, this is not possible. Map production frequently requires information from more than one layer. It is the last step in map production. After the user chooses the output layers, the scale and the limits of the map, object labels need to be placed effectively.

Screen maps are created as answers to spatial queries on geographical information systems (GIS), either in desktop or Web applications. To allow for a smooth uninterrupted response, the maps must be produced "on the stroke of a key". Otherwise, the user gets impatient. Screen map production needs to balance the quality of label placement with the processing time. Label placement techniques that are of extremely good quality but need much processing time are not useful for screen maps.

The most common label placement problem is the point feature label placement (PFLP) problem. Given a set of places associated to point locations, the PFLP consists in placing the names of these places in the map with quality and efficiency. This work

describes the Fast Algorithm for Label-Placement (FALP) to solve the PFLP. Our results show the FALP has good performance in label placement processing time and presents a good label placement quality for a screen map. Given its relative simplicity of implementation and efficient performance, we believe that FALP method is a good solution to the PFLP problem for screen maps.

In Section 2 of the paper, we review the literature on label placement. In Section 3, we present the conflict graph, a structure that represents all conflicts between candidate labels. Section 4 we introduce the FALP algorithm. Section 5 shows computational results using instances formed by standard sets of randomly generated points suggested in the literature.

2. Review of the point label placement problem

There are three different label-placement problems: labeling of point features (cities, schools, hospital, mountain peaks …), line features (rivers, roads, …), and area features (countries, states, oceans, …). In this article we focus on *placing labels for point features* using combinatorial optimization. We will use three key notions, defined below:

- *Label positions for each point feature.* Given a point, a label can be placed on one of four positions relative to it (see Figure 1). Therefore, a *label position* is an ordered pair (*point location, relative position*). In Figure 1, each box marks a label position.
- *Cartographic preference.* Usually, we prefer to place labels at the upper right corner of the point, and this preference decreases counterclockwise from this position. The value inside each box in Figure 1 matches the order of cartographical preference for placing a label. Lower values mark more desirable positions.
- *Objective function*. The objective function (*f*) measures the quality of the label placement. The quality of labeling depends on the number of overlaps between labels and the cartographic preference for label placement.

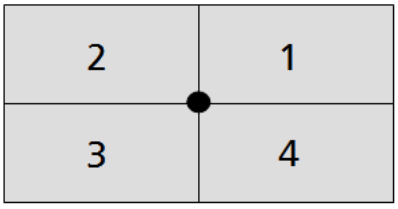

Figure 1. A set of label positions and their cartographic preference (best = 1; worse = 4)

We take *npos* as the number of label positions and take *np* as the total number of point features. There are $npos^{np}$ possible arrangements, a number which increases exponentially. Since the set of possible solutions is finite, theoretically we could select the best solution by enumeration. As the number of points increases this becomes unfeasible, because of the combinatorial explosion of possible solutions. Marks and

Scheiber (1991) have shown the point feature label placement (PFLP) problem is NP-hard.

For screen maps, we need algorithms that seek a compromise solution in cost-benefit, with good quality and a short response time. In our algorithm, we have considered a limited set of four possible label positions. Our approach can be extended to account for a larger number of positions. However, a larger set of the label positions would result in a problem with larger number of possible solutions. In spite of the apparent better looking results, the growth in problem complexity results in a much higher computational effort. We consider that a set of four positions to be an acceptable compromise in terms of cost-benefit analysis.

Several heuristics and metaheuristics have been used for the PFLP problem. For an early review of methods, see Christensen et al. (1995), which includes Zoraster's integer programming algorithm (Lagrangean relaxation) (Zoraster 1990) and Hirsch's continuous gradient-descent algorithm (Hirsch 1982). Relevant recent proposals include solutions based on *simulated annealing* (Christensen et al. 1995), *genetic algorithms* (GA) (Verner et al. 1997) and *tabu search* (Yamamoto et al. 2002). The most recent result is the *constructive genetic approach* (Yamamoto and Lorena 2005), that has better results in label placement quality than all previous methods.

In this paper, we will consider the PFLP as the problem of *placing labels in all points.* We will try to find out the *largest subset of labels with no conflicts* with good quality at acceptable runtime. Our algorithm considers that a typical generation of screen maps consists of three steps:

- The user requests an overview of the area of interest. The software displays the area, with a suitable choice of labels depicted at that scale.
- The user requests a zoom in an area for detailed analysis. The software displays the area, and labels invisible at the largest scale are made visible at this scale.
- The user may zoom for further detail or pan for visualization of neighboring areas. In the latter case, more labels will be made visible and the software will try to display them.

Based on this conception, we consider that label placement on screen maps needs a preparation stage, where the software will build a list of labels which will be visible at different scales. This pre-processing phase is outside the scope of this paper. The proposed algorithm will run at one particular scale, where the visible labels are known in advance. Therefore, our algorithm should be embedded in a more general visualization software. In the proposed method, the basic data structure is the conflict graph, described in the next section.

3. The conflict graph

The conflict graph is a structure that describes conflicts due to label overlap. Each node of the conflict graph is a label position. Recall from section 2 that we consider a label position as a pair (*point location, relative position*). The edges of the graph link conflicting label positions. Figure 2 shows two points with their label positions and the

corresponding conflict graph. A similar idea was proposed by Strijk et al (2000). However, their algorithm for point label placement based on the conflict graph (diversified neighborhood search) solves as different label placement problem. Their label placement algorithm allows label selection for conflicting label position. Our problem is *placing labels in all points*. Furthermore, their algorithm does not use the same refinement steps as ours.

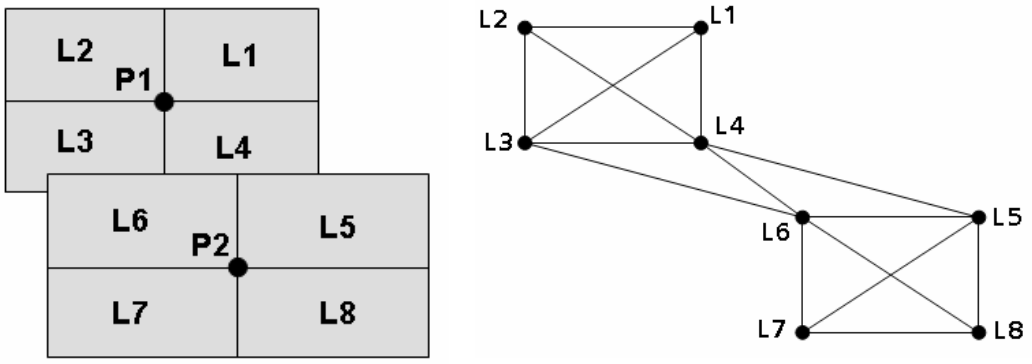

Figure 2. Two points, their label positions and the corresponding conflict graph

The conflict graph is a graph G = (N, E), where N is a list of nodes and E is a list of edge. Two nodes are adjacent if both nodes are connected by an edge. Each node represents a label position, and keeps track of the number of incident edges.

A *conflict* arises both from the choice between four label positions for a point and from the overlap between label positions of different points. In Figure 2, label L_2 has three edges due to the conflicts between it and the three other potential positions for point P_1. Label L_4 has five edges, three arising from the conflict between other label positions for point P_1, and two arising from the conflict with label positions for point P_2.

The *degree* of a node is the number of incident edges in that node. The degree is a measure of label conflicts. The higher the degree of a node, the more difficult it is to place the associated label on the map. The conflict graph can be mapped to an adjacency matrix, where rows and columns represent nodes (label positions), and each entry is either 1 for connected nodes or 0 otherwise. Figure 3 shows the adjacency matrix of the graph shown in Figure 2. In the adjacency matrix, the degree of each node is the sum of values in a line or in a column.

We compute the conflict graph in a pre-processing phase. Our suggestion is to build one conflict graph for each scale, regardless of the zoom and pan area. Recall that our conception of a screen map is that, for a given scale, all labels to be displayed are known. In this way, panning or zooming at the same scale will not need a rebuild on the conflict graph. Once built, the conflict graph provides quick information on all label placement conflicts.

Our algorithm allows labels to have different sizes. This is possible since we test for overlapping between label boxes when building the conflict graph. The adjacency matrix indicates if there is conflict between the labels. Figure 2 shows an example of building an adjacent matrix from labels of different size.

	L1	L2	L3	L4	L5	L6	L7	L8
L1	0	1	1	1	0	0	0	0
L2	1	0	1	1	0	0	0	0
L3	1	1	0	1	0	1	0	0
L4	1	1	1	0	1	1	0	0
L5	0	0	0	1	0	1	1	1
L6	0	0	1	1	1	0	1	1
L7	0	0	0	0	1	1	0	1
L8	0	0	0	0	1	1	1	0

Figure 3. Adjacency matrix of Figure 2

4. A fast algorithm for label placement

This section describes the fast algorithm for label placement (FALP). The *first step* of the algorithm produces the set S_1 of all nonconflicting label positions. The *second step* deals with the set S_2 of label positions with conflicts. For each location in S_2, we choose the label position that creates the smallest number of additional conflicts. The *third and final step* is a local search to improve the results. The algorithm reprocesses all the label positions selected in steps one and two. It changes a label position if the overlap between it and its neighbors can be further reduced.

The aim of the algorithm is producing a map with the smallest possible number of conflicting labels. Given the set of all labels L and the set of conflicting labels L_c, we use the following objective function (*f)*:

$$f = \#\{L_c\},\ L_c \subset L$$

The main challenge of label placement is avoiding getting caught in local minimums of the objective function and thus not being able to reach the global maximum. Conceptually, the first and second steps of the FALP try to reach the global maximum. The third step is a set of local adjustments that improves the method's performance. By making local changes, we further direct the algorithm towards the global maximum. Should the third step be performed alone, there would be a large chance the algorithm would get caught in local minimums. The FALP algorithm works as follows:

Step 0. Create the conflict graph (done off-line).

Step 1. Apply maximum nonconflict labeling algorithm to get the set S_1 (label positions without conflict).

Step 2. Take the set S_2 to be all points not contained in S_1. For each point in S_2, choose a label position with minimum conflict.

Step 3. Take the solution S to be $S_1 \cup S_2$. Calculate the value of the objective function *f* for S. If there are no conflicts, *exit*. Otherwise, make S* = S and repeat the steps below *t* times:

- Apply local search to all points in S* to produce a new potential solution S*.
- Calculate the value of *f* for S*. If $f(S^*) < f(S)$, take S = S*.

In our tests, a value of t=5 has proved to be enough for good results. We describe the maximum nonconflict labeling algorithm (step 1) below.

4.1. The maximum nonconflict labeling algorithm

Given a conflict graph, the *maximum nonconflict labeling algorithm* produces a set of points and their matching label positions without conflicts. This algorithm is an improved version of a similar idea from (Strijk et al. 2000). The algorithm builds an adjacency matrix from the conflict graph, as pictured in Figure 3. Each line of the matrix will be one label position. The algorithm orders the label positions in ascending order of degree. The degree of each label position is the number of conflicts for that label. If two or more label positions have the same degree, they are ordered according to their corresponding points. The label position whose corresponding point has less available label positions left in the matrix is ranked higher.

At each step, the algorithm takes the label with smallest degree. It removes this label from the matrix and places it in the solution set. For each chosen label, it cuts out all label positions which are in conflict with it. The process is then repeated until the matrix is empty. The algorithm works as follows:

Step 0. The nonconflict set S_1 is empty. The active node set L* is equal to the full node set N.

Step 1. If the active node set L* is empty, exit with S_1 as the result. Otherwise, do steps 2 to 5.

Step 2. Calculate degrees of all nodes in L*.

Step 3. Select l_{min}, the node of smallest degree on L*. Place it in the nonconflict set S_1.

Step 4. Remove l_{min} and all nodes adjacent to it from L*.

Step 5. Go to step 1.

As an example, we consider an arrangement of six points, where each point has four label positions, given in Table 1.

Table 1 – Example arrangement

Point	Label position	Label size (characters)
P1	L01, L02, L03, L04	10
P2	L05, L06, L07, L08	7
P3	L09, L10, L11, L12	6
P4	L13, L14, L15, L16	9
P5	L17, L18, L19, L20	15
P6	L21, L22, L23, L24	13

	L01	L02	L03	L04	L05	L06	L07	L08	L09	L10	L11	L12	L13	L14	L15	L16	L17	L18	L19	L20	L21	L22	L23	L24
L01	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
L02	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
L03	1	1	0	1	0	1	1	0	0	1	0	0	0	1	0	0	1	1	0	0	0	0	0	0
L04	1	1	1	0	1	1	1	1	1	1	0	0	1	1	0	0	1	0	0	0	0	0	0	0
L05	1	0	0	1	0	1	1	1	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0
L06	1	1	0	1	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
L07	0	0	1	1	1	1	0	1	0	1	1	0	1	1	0	0	1	1	1	1	0	0	0	0
L08	0	0	0	1	1	1	1	0	1	1	1	1	1	0	0	0	1	0	0	1	0	0	0	0
L09	0	0	0	1	1	0	0	1	0	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0
L10	0	0	1	1	1	1	1	1	1	0	1	1	1	1	0	0	1	1	0	0	0	0	0	0
L11	0	0	0	0	0	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	0	0	0
L12	0	0	0	0	0	0	0	1	1	1	1	0	1	0	0	1	1	0	0	1	1	0	0	0
L13	0	0	0	1	0	0	1	1	1	1	1	1	0	1	1	1	1	0	0	1	0	0	0	0
L14	0	0	1	1	0	0	1	0	0	1	1	0	1	0	1	1	1	1	1	1	0	0	0	0
L15	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	1	0	0	1	1	1	1	0	0
L16	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	1	1	0	0	0
L17	0	0	1	0	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0	0
L18	0	0	1	0	0	1	1	0	0	1	1	0	0	1	0	0	1	0	1	1	0	0	0	0
L19	0	0	0	0	0	0	1	0	0	0	1	0	0	1	1	0	1	1	0	1	1	1	0	0
L20	0	0	0	0	0	0	1	1	0	0	1	1	1	1	1	1	1	1	1	0	1	0	0	0
L21	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	1	1	0	1	1	1
L22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	1	1
L23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1
L24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0

Figure 4 – Adjacency matrix for example (iteration 1)

Iteration 1 $L^* = \{L01, L02, L03, ..., L24\}$

select L24 $S_1 = \{(P6, L24)\}$ (smallest degree label from L^*)

We cut out L21, L22, L23 and L24 from L^*

	L01	L02	L03	L04	L05	L06	L07	L08	L09	L10	L11	L12	L13	L14	L15	L16	L17	L18	L19	L20
L01	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
L02	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
L03	1	1	0	1	0	1	1	0	0	1	0	0	0	1	0	0	1	1	0	0
L04	1	1	1	0	1	1	1	1	1	1	0	0	1	1	0	0	1	0	0	0
L05	1	0	0	1	0	1	1	1	1	1	0	0	0	0	0	0	1	0	0	0
L06	1	1	0	1	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0
L07	0	0	1	1	1	1	0	1	0	1	1	0	1	1	0	0	1	1	1	1
L08	0	0	0	1	1	1	1	0	1	1	1	1	1	0	0	0	1	0	0	1
L09	0	0	0	1	1	0	0	1	0	1	1	1	1	0	0	0	1	0	0	0
L10	0	0	1	1	1	1	1	1	1	0	1	1	1	1	0	0	1	1	0	0
L11	0	0	0	0	0	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1
L12	0	0	0	0	0	0	0	1	1	1	1	0	1	0	0	1	1	0	0	1
L13	0	0	0	1	0	0	1	1	1	1	1	1	0	1	1	1	1	0	0	1
L14	0	0	1	1	0	0	1	0	0	1	1	0	1	0	1	1	1	1	1	1
L15	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	1	0	0	1	1
L16	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	1
L17	0	0	1	0	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1
L18	0	0	1	0	0	1	1	0	0	1	1	0	0	1	0	0	1	0	1	1
L19	0	0	0	0	0	0	1	0	0	0	1	0	0	1	1	0	1	1	0	1
L20	0	0	0	0	0	0	1	1	0	0	1	1	1	1	1	1	1	1	1	0

Figure 5 – Adjacency matrix for example (iteration 2)

Iteration 2 L* = {L01, L02, L03, ... , L20}

select L02 S_1 = {(P6, L24), (P1, L02)} (smallest degree labels from L*)

We cut out L01, L02, L03, L04 and L06 from L*

	L05	L07	L08	L09	L10	L11	L12	L13	L14	L15	L16	L17	L18	L19	L20
L05	0	1	1	1	1	0	0	0	0	0	0	1	0	0	0
L07	1	0	1	0	1	1	0	1	1	0	0	1	1	1	1
L08	1	1	0	1	1	1	1	1	0	0	0	1	0	0	1
L09	1	0	1	0	1	1	1	1	0	0	0	1	0	0	0
L10	1	1	1	1	0	1	1	1	1	0	0	1	1	0	0
L11	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1
L12	0	0	1	1	1	1	0	1	0	0	1	1	0	0	1
L13	0	1	1	1	1	1	1	0	1	1	1	1	0	0	1
L14	0	1	0	0	1	1	0	1	0	1	1	1	1	1	1
L15	0	0	0	0	0	1	0	1	1	0	1	0	0	1	1
L16	0	0	0	0	0	1	1	1	1	1	0	0	0	0	1
L17	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1
L18	0	1	0	0	1	1	0	0	1	0	0	1	0	1	1
L19	0	1	0	0	0	1	0	0	1	1	0	1	1	0	1
L20	0	1	1	0	0	1	1	1	1	1	1	1	1	1	0

Figure 6 – Adjacency matrix for example (iteration 3)

Iteration 3 L* = {L05, L07, L08, ... , L20}

select L05 S_1 = {(P6, L24), (P1, L02), (P2/L05)}

We cut out L05, L07, L08, L09, L10 and L17 from L*

	L11	L12	L13	L14	L15	L16	L18	L19	L20
L11	0	1	1	1	1	1	1	1	1
L12	1	0	1	0	0	1	0	0	1
L13	1	1	0	1	1	1	0	0	1
L14	1	0	1	0	1	1	1	1	1
L15	1	0	1	1	0	1	0	1	1
L16	1	1	1	1	1	0	0	0	1
L18	1	0	0	1	0	0	0	1	1
L19	1	0	0	1	1	0	1	0	1
L20	1	1	1	1	1	1	1	1	0

Figure 7 – Adjacency matrix for example (iteration 4)

Iteration 4 L* = {L11, L12, L13, L14, L15, L16, L18, L19, L20}

select L12 S_1= {(P6, L24), (P1, L02), (P2/L05), (P3/L12)}

We cut out L11, L12, L13, L16 and L20 from L*

P3 has label positions {L11, L12} and P5 has label positions {L18, L19, L20}, therefore we choose L12 and not L18.

	L14	L15	L18	L19
L14	0	1	1	1
L15	1	0	0	1
L18	1	0	0	1
L19	1	1	1	0

Figure 8 – Adjacency matrix for example (iteration 5)

Iteration 5 L* = {L14, L15, L18, L19}

select L15 S_1 = {(P6, L24), (P1, L02), (P2/L05), (P3/L12), (P4/L15)}

We cut out L14, L15 and L19 from L*

	L18
L18	0

Figure 9 – Adjacency matrix for example (iteration 6)

Iteration 6 L* = {L18}

select L18 S_1= {(P6, L24), (P1, L02), (P2/L05), (P3/L12), (P4/L15), (P5/L18)}

We cut out L18 from L*

Iteration 7 L* = { }

In the above example the maximum non-conflict set was S_1 = {(P6, L24), (P1, L02), (P2, L05), (P3, L12), (P4, L15), (P5, L18)}, and all six points have labels without conflicts.

4.2. Processing of labels with conflicts

The second step of the FALP processes all the labels, since the *maximum nonconflict labeling algorithm* cannot solve all conflicts. We take as input the set $S = S_1 \cup S_2$. The set S_1 is the set of pairs (point, label position) without overlapping produced by the first step of FALP. The set S_2 contains all points that do not have a label assigned to them. The procedure takes the set S_2 and chooses the best label position for each point in the set. The label position is chosen as to have the least conflicts. The algorithm works as follows:

Step 0. Take S* = S, where $S = S_1 \cup S_2$,

Step 1. If S_2 is empty, exit with S* as the solution. Otherwise, continue.

Step 2. For each point p_i in S_2, do step 3.

Step 3. For each label position l_k of the point p_i, do steps 4 and 5.

Step 4. Calculate g_k, the number of all active labels that overlap with l_k.

Step 5. Take the label with the smallest g_k to be the best label position for point p_i. Update S* with this pair (point, label position).

Step 6. Go to step 2.

The output of this algorithm is the set S* where all points have been assigned a label position. Some of these label positions may still be in conflict with other, and this requires a further refinement step, described below. As an example, we consider an arrangement of six points, where each point has four label positions, given in Table 1. For each point p_i the candidate list will be composed of the pair (p_i, g_k) .

Step 0 S* = {(P1, L02), (P2, L05), (P3, 0), (P4, L14), (P5, 0), (P6, L23)}

S_2 = {(P3,0), (P5, 0)}

Step 1 Process the set S_2.

Step 2.1 Point P3 has candidate labels L = {(L09, 1), (L10, 2), (L11, 1), (L12,0)}

Choose L12 as the best label

Update S* = {(P1, L02), (P2, L05), (P3, L12), (P4, L14), (P5, 0), (P6, L23)}

Step 2.2 Point P5 has candidate labels L= {(L17, 3), (L18, 1), (L19, 1), (L20, 2)}

Choose L18 as the best candidate

Update S* = {(P1, L02), (P2, L05), (P3, L12), (P4, L14), (P5, L18), (P6, L23)}

In the above example the solution was S = {(P1, L02), (P2, L05), (P3, L12), (P4, L14), (P5, L18), (P6, L23)}, and there is 2 labels with overlaps: L14 and L18. This overlapping can be improved using the algorithm of the next section.

4.3. Local search algorithm

The third and final step of the FALP is a local search procedure that examines each point, comparing its label position to that of its neighbors. It changes a label position if this brings a reduction of conflicts in the local region. The method is repeated until no further improvements are possible or up to a maximum number of predefined iterations. In our tests, we found that five iterations are enough to produce good maps. The local search algorithm takes as input the complete solution S, where all points have a designated label position. It performs a set of local adjustments to improve the solution:

Step 0. Take S* = S.

Step 1. Repeat T times steps 2 to 6.

Step 2. For each point p_i in S* and for each label position l_k of the point p_i, do steps 3 and 4.

Step 3. Calculate g_k, the number of all active labels that overlap with l_k.

Step 4. If g_k is zero, go to step 2. Otherwise, continue.

Step 5. Take the label with the smallest g_k to be the best label position for point p_i. Update S* with this pair (point, label position). Go to step 2.

The result of this algorithm is a new solution set S*. In each iteration, this algorithm will not change labels already assigned with no conflicts. It will try to update only those with conflicts. However, an update done in a previous step may create conflicts in the next iteration. For this reason, this local search algorithm will only work in situations close to the global maximum. In our tests, we found that taking T=5 is enough for good results. As an example, we consider an arrangement of six points, where each point has four label positions, as given in Table 1. For each point p_i the candidate list will be composed of the pair (p_i, g_k).

Iteration 1	S* = {(P1, L02), (P2, L05), (P3, L12), (P4, L16), (P5, L18), (P6, L22)}
Point P1	List = {(L01, 1), (L02, 0)} (L02 is the best candidate)
	S* = {(P1, L02), (P2, L05), (P3, L12), (P4, L16), (P5, L18), (P6, L22)}
Point P2	List = {(L05, 0)} (L05 is the best candidate)
	S* = {(P1, L02), (P2, L05), (P3, L12), (P4, L16), (P5, L18), (P6, L22)}
Point P3	List = {(L09, 1), (L10, 2), (L11, 2), (L12,1)} (L09 is the best candidate)
	S* = {(P1, L02), (P2, L05), (P3, L09), (P4, L16), (P5, L18), (P6, L22)}
Point P4	List = {(L13, 1), (L14, 1), (L15, 0)} (L15 is the best candidate)
	S* = {(P1, L02), (P2, L05), (P3, L09), (P4, L15), (P5, L18), (P6, L22)}
Point P5	List = {(L17, 2), (L18, 0)} (L18 is the best candidate)
	S* = {(P1, L02), (P2, L05), (P3, L09), (P4, L15), (P5, L18), (P6, L22)}
Point P6	List = {(L21, 1), (L22, 1), (L23, 0)} (L23 is the best candidate)
	S* = {(P1, L02), (P2, L05), (P3, L09), (P4, L15), (P5, L18), (P6, L23)}

Iteration 2 S* = {(P1, L02), (P2, L05), (P3, L09), (P4, L15), (P5, L18), (P6, L23)}

Point P1 List = {(L01, 1), (L02, 0)} (L02 is the best candidate)

S* = {(P1, L02), (P2, L05), (P3, L12), (P4, L16), (P5, L18), (P6, L22)}

Point P2 List = {(L05, 1), (L06, 1), (L07, 1), (L08, 1)} (L05 is the best candidate)

S* = {(P1, L02), (P2, L05), (P3, L12), (P4, L16), (P5, L18), (P6, L22)}

Point P3 List = {(L09, 1), (L10, 2), (L11, 2), (L12,0)} (L12 is the best candidate)

S* = {(P1, L02), (P2, L05), (P3, L12), (P4, L16), (P5, L18), (P6, L22)}

Point P4 List = {(L13, 1), (L14, 1), (L15, 0)} (L15 is the best candidate)

S* = {(P1, L02), (P2, L05), (P3, L12), (P4, L15), (P5, L18), (P6, L22)}

Point P5 List = {(L17, 2), (L18, 0)} (L18 is the best candidate)

S* = {(P1, L02), (P2, L05), (P3, L12), (P4, L15), (P5, L18), (P6, L22)}

Point P6 List = {(L21, 2), (L22, 1), (L23, 0)} (L23 is the best candidate)

S* = {(P1, L02), (P2, L05), (P3, L12), (P4, L15), (P5, L18), (P6, L23)}

In the above example the solution is S = {(P1, L02), (P2, L05), (P3, L12), (P4, L15), (P5, L18), (P6, L23)}, and all six points have labels without conflicts.

5. Results

Christensen (1995), Verner (1997), Yamamoto et al (2002) and Yamamoto et al. (2005) 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. To compare the FALP algorithm with previous works, the standard sets of randomly generated points and the same conditions as described by Christensen et al. (1995) are used, following the same assumptions as Verner et al. (1997). 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*):

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

Compared with other point label placement algorithms, the FALP showed good results in quality of label placement and excellent results in processing time. The computational results are presented in Table 1.

Table 1. Percentage of labels without conflict for different number of points

Algorithm	100	250	500	750	1000
CGA	100.00	100.00	99.6	97.1	90.7
FALP	100.00	100.00	99.5	96.7	90.12
Tabu search	100.00	100.00	99.2	96.8	90.00
GA with masking	100.00	99.98	98.79	95.99	88.96
GA	100.00	98.40	92.59	82.38	65.70
Simulated Annealing	100.00	99.90	98.30	92.30	82.09
Zoraster	100.00	99.79	96.21	79.78	53.06
Hirsch	100.00	99.58	95.70	82.04	60.24
3-Opt Gradient Descent	100.00	99.76	97.34	89.44	77.83
2-Opt Gradient Descent	100.00	99.36	95.62	85.60	73.37
Gradient Descent	98.64	95.47	86.46	72.40	58.29
Greedy	95.12	88.82	75.15	58.57	43.41

Table 1 shows the percentage of labels placed without conflict for 100, 250, 500, 750 and 1000 points, considering different algorithms of the literature. The results, for each problem size, present the average percentage of labels placed without conflict for the 25 trials. The results show the results of the optimization algorithms tested by Christensen et al. (1995) (greedy-depth first, gradient descent, 2-opt gradient descent, 3-opt gradient descent, Hirsch, Zoraster and simulated annealing), Verner et al. (1997) (GA without masking and GA with masking), Yamamoto et al. (2002) (Tabu search), Yamamoto and Lorena (2005) (Constructive genetic algorithm - CGA) and the FALP.

Table 2 compares computational times in seconds (Pentium II computer). It compares the FALP to the Tabu search of Yamamoto et al. (2002) and the CGA of Yamamoto and Lorena (2005). FALP is much faster than CGA and Tabu search.

Table 2. Computational times to reach the best solutions for different number of points

Algorithm	100	250	500	750	1000
FALP	0	0	1	2.8	5.9
CGA	0	0.6	21.5	228.9	1227.2
Tabu search	0	0	1.3	76.0	352.9

6. Conclusion

This work has proposed and evaluated a fast algorithm for point label placement (FALP) in maps. Compared with other algorithms, the FALP showed good results in label placement quality and excellent performance. The FALP can be recommended to solve the automatic cartographic label placement problem for point features in real time screen maps, due to the quality of its label placement and to its runtime performance.

One of the interesting results of the FALP is the comparison with other optimization methods, such as simulated annealing, genetic algorithms and tabu search. Algorithms based on these methods are adaptation of known optimization methods to the label placement problem. By contrast, the FALP uses simple heuristics that arise from the nature of the label placement problem. This follows from the general rule that problem-based techniques perform better than general-purpose techniques in optimization problem. The FALP is a further example that it pays to consider the specific character of a GIScience problem before trying to solve it with general-purpose methods.

References

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

Hirsch, S. A. (1982). "An algorithm for automatic name placement around point data." American Cartographer **9**(1): 5-17.

Marks, J. and S. Shieber (1991). The Computational Complexity of Cartographic Label Placement, Center for Research in Computing Technology, Harvard University.

Strijk, T., B. Verweij and K. Aardal (2000). Algorithms for Maximum Independent Set Applied to Map Labeling, Department of Computer Science, Utrecht University.

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

Wolff, A. and T. Strijk. (1996). "The Map Labeling Bibliography." from http://i11www.ilkd.uni-karlsruhe.de/map-labeling/bibliography/

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

Yamamoto, M. and L. A. N. Lorena (2005). A Constructive Genetic Approach to Point-Feature Cartographic Label Placement. Metaheuristics: Progress as Real Problem Solvers. T. N. Ibaraki, Koji; Yagiura, Mutsunori (Eds.) Springer. **32**.

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