

Discrete Optimization

POPMUSIC for the point feature label placement problem

Adriana C.F. Alvim^a, Éric D. Taillard^{b,*}

^a Department of Applied Informatics, Federal University of the State of Rio de Janeiro, Avenida Pasteur 458, Rio de Janeiro, RJ 22.290-240, Brazil

^b Department of Industrial Systems, University of Applied Sciences of Western Switzerland, Route de Cheseaux 1, Case postale, CH-1401 Yverdon, Switzerland

Received 29 August 2006; accepted 3 October 2007

Available online 10 October 2007

Abstract

Point feature label placement is the problem of placing text labels adjacent to point features on a map so as to maximize legibility. The goal is to choose positions for the labels that do not give rise to label overlaps and that minimize obscuration of features. A practical goal is to minimize the number of overlaps while considering cartographic preferences. This article proposes a new heuristic for solving the point feature label placement problem based on the application of the POPMUSIC frame. Computational experiments show that the proposed heuristic outperformed other recent metaheuristics approaches in the literature. Experiments with problem instances involving up to 10 million points show that the computational time of the proposed heuristic increases almost linearly with the problem size. New problem instances based on real data with more than 13,000 labels are proposed.

© 2007 Elsevier B.V. All rights reserved.

Keywords: Metaheuristics; POPMUSIC; Map labeling; Large-scale optimization; Tabu search

1. Introduction

Automated label placement is a problem of fundamental importance in cartography, where text labels must be placed on maps while avoiding overlaps with cartographic symbols and other labels. Interactive creation of maps increases the importance of this problem. This task must be performed with limited computational effort, typically less than one second. Applications in cartography require different label placement tasks.

First, the object to be labeled may have several different dimensions:

- Dimension 0, labeling point features (such as cities and mountain peaks)
- Dimension 1, labeling line (segment) features (such as rivers and roads) and

- Dimension 2, labeling area features (such as countries and oceans)

Then, overlapping labels may be accepted or not. If two or more labels cannot overlap, two different problems can be defined: In the *label number maximization problem* [10], certain features (and their labels) are allowed to be deleted and the objective is to place as many labels as possible with no overlaps. This problem is equivalent to finding a maximum vertex independent set in a conflict graph [13,17] where each node is a candidate label and there is an edge between two nodes whenever there is a conflict between the corresponding labels. In the *label size maximization problem*, the objective is to determine the maximum scale factor for the label size and a corresponding labeling without overlaps.

If all features must be labeled and scaling is not allowed, overlaps must be permitted. In that case, there are two objectives: to minimize the number of overlaps, which is called the *label overlap minimization problem* by [10] and to minimize the number of labels obstructed by at least

* Corresponding author.

E-mail addresses: adriana@unriotec.br (A.C.F. Alvim), eric.taillard@heig-vd.ch (É.D. Taillard).

one other label [3] which is called *maximum number of conflict free labels problem* by [12].

Concerning the position of the labels, there are two models. In the first one, an explicit enumerated set is considered for potential label positions (discrete model). In the second model, an infinite number of possible label positions may be used. This continuous model is also called the slider model.

Finally, for all labeling problems, it is possible to assign a weight (penalty) for each label position and to use as alternate objective the minimization of the sum of the penalties.

Construction of good labeling, regardless of the features being labeled, leads to combinatorial optimization problems that are generally NP-hard [7,9,11]. Exact algorithms are able to solve problems with just a few hundred points to label [5,10,13,22]. Therefore, heuristic algorithms must be designed for dealing with larger problems or for getting approximate solutions with low computational effort.

Although the methodology proposed in this paper can be applied to various labeling problems, we are going to illustrate this methodology on the NP-Hard point-feature label placement problem (PFLP) with *label overlap minimization*. The size of the labels is fixed, the potential positions of labels are discrete and all points must be labeled. Therefore, the first objective considered is to minimize the number of overlaps. Cartographic preferences can also be considered as an alternate objective.

Christensen et al. [4] presented a good review on the PFLP. The authors developed a local search technique based on a discrete form of gradient descent and a simulated annealing based algorithm. Verner et al. [16] proposed a heuristic based on genetic algorithms (GA). More recent works, considering that all point features must be labeled, include a tabu search [19], a constructive genetic approach [21] and a fast algorithm for label placement [20]. Wagner et al. [17] proposed a two phase algorithm for the *label number maximization problem*. An extensive Map-Labeling bibliography is maintained by Wolff [18].

This work, outlined in Alvim and Taillard [1], is based on the preliminary work of Burri and Taillard [2,14] which investigates the evaluation of the POPMUSIC methodology for the PFLP. POPMUSIC is a *general* optimization method especially designed for optimizing large instances of combinatorial problems and can be seen as a large scale neighborhood search (see [15]). The basic idea of POPMUSIC is to locally optimize sub-parts of a solution, once a solution of the problem is available. These local optimizations are repeated until no improvements are found. The local optimizations are performed by a new implementation of the tabu search proposed by [19].

Although the results presented in the present article are restricted to this labeling problem, this methodology can be used to deal with other labeling problems due to the ability of the tabu search used as local optimizer in POPMUSIC.

The paper is organized as follows: Section 2 introduces the PFLP problem. Then, the adaptation of POPMUSIC

for the PFLP is presented in Section 3. This section starts by describing the general POPMUSIC frame (Section 3.1). A practical implementation based on POPMUSIC requires the design of few components specific to the problem being solved. The first one is the way an initial solution is obtained (Section 3.2.1), then the way subproblems are built (Section 3.2.2) and finally a procedure for optimizing the subproblems (Section 3.2.3). The last is based on a tabu search proposed by [19]. The tabu search is presented in more detail than in [19] (and is certainly slightly different), making the present article self-contained. Computational results and new instances are presented in Sections 4 and 5. Concluding remarks are made in the last section.

2. The point feature label placement problem

In this paper, we consider a set of n points that have to be labeled. Each point has p candidate label positions of identical size, identified by the integers $1, \dots, np$. The position of the label associated to point x , $x = (1, 2, \dots, n)$, is given by variable y_x that can take p different values: $y_x \in \{(x - 1) * p + 1, (x - 1) * p + 2, (x - 1) * p + 3, \dots, x * p\}$.

Fig. 1 shows the possible label positions for a point feature when $p = 4$. Each box corresponds to a region in which the label may be placed. According to cartographic standards, there is a preference (or, more precisely, a penalty) for each possible label position, lower values indicating better positions: top right (position 1), top left (position 2), bottom left (position 3) and bottom right (position 4). An arbitrary weight $w(y_x) < 1$ is associated with each label position y_x . In this paper, we have considered problem instances with $p \in \{2, 4, 8\}$ label positions for each point x and respective weights $w(y_x) = ((y_x - 1) \bmod p) * 0.0001$. We are also given an overlap symmetrical $np \times np$ matrix A where $a_{ij} = 1.0 + w(j)$ if label i overlaps with label j , $a_{ij} = w(i)$ for $i = j$ and $a_{ij} = 0$ otherwise. A solution S is a list of n labels (y_1, y_2, \dots, y_n) . For a given solution S , the cost measure which counts the number of point features labeled with one or more overlaps is expressed by $f(S) = \sum_{i=1}^n \min\{1, \sum_{j \in \{1, \dots, n\} \setminus i} a_{y_i y_j}\}$; and the function that counts the number of overlaps and takes the cartographic preferences into account is expressed by $\bar{c}(S) = \sum_{i=1}^n \sum_{j=1}^n (a_{y_i y_j})$. For the special case where $w(y_x) = 0$, for $x = 1, \dots, n$, we note by $c(S)$ the function which simply

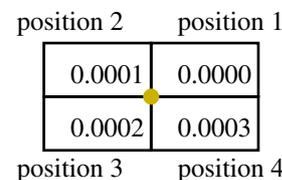


Fig. 1. A point feature (circle) with four potential label positions (boxes). A weight (penalty) $\{0, 0.0001, 0.0002, 0.0003\}$ is associated with each label position, lower values indicating best positions according to cartographic standards.

counts the number of overlaps without considering cartographic preferences. In this work we will use both $\bar{c}(S)$ and $c(S)$ as objective functions to minimize. Let us note that the objective function $c(S)/2$ (which counts the number of pairwise overlaps) is also used in practice when cartographic preferences are not taken into account.

Fig. 2 illustrates an example with three points ($n = 3$), each one having $p = 4$ candidate label positions ($y_1 \in \{1, 2, 3, 4\}$, $y_2 \in \{5, 6, 7, 8\}$, $y_3 \in \{9, 10, 11, 12\}$). Given solution $S = (4, 6, 9)$ we have: $f(S) = \min(1, (a_{4,6}) + (a_{4,9})) + \min(1, (a_{6,4}) + (a_{6,9})) + \min(1, (a_{9,4}) + (a_{9,6})) = \min(1, 1.0001) + \min(1, 2.0003) + \min(1, 1.0001) = 3$, $\bar{c}(S) = (a_{4,4}) + (a_{4,6}) + (a_{4,9}) + (a_{6,4}) + (a_{6,6}) + (a_{6,9}) + (a_{9,4}) + (a_{9,6}) + (a_{9,9}) = 0.0003 + 1.0001 + 0 + 1.0003 + 0.0001 + 1.0 + 0 + 1.0001 + 0 = 4.0009$ and $c(S) = 4$.

2.1. Reducing the size of problem instances

For the label number maximization problem in which omitting labels is allowed, Wagner et al. [17] have proposed rules for reducing the size of problem instances. For our problem, the following two rules of the first phase are applicable:

Rule L1: If candidate label x_i of point feature x has no conflicts, then assign label x_i to point feature x and eliminate all other candidates of x .

Rule L2: If candidate label x_i of point feature x is only in conflict with some label y_k of point feature y , and y has a candidate label $y_j (j \neq k)$ that is only in conflict with label $x_l (l \neq i)$ of point feature x , then assign labels x_i and y_j , respectively, to point features x and y , and eliminate all other candidates of x and y .

When assigning a label to its feature and removing all other candidates of that feature, the labels that are in conflict with those labels removed have their list of conflicting labels changed. For this reason, these two rules are applied while changes occur. The new reduced problem instance is composed of all point features, with their corresponding labels, that have not been fixed by this procedure.

Let us mention that: (i) Wagner et al. [17] have proposed another rule that cannot be applied for the problem considered in this article; (ii) after applying this procedure, the partial solution composed by the labeled point features has no conflicts; (iii) if there is a solution for the label over-

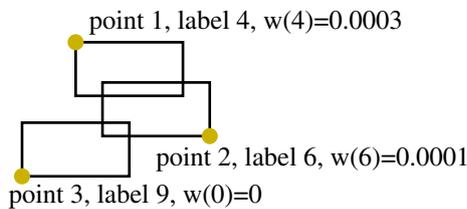


Fig. 2. Example with $n = 3$ and $S = (4, 6, 9)$. In solution S there are three labels with one or more overlaps ($f(S) = 3$) and four overlaps ($c(S) = 4$): label 4 with label 6, label 6 with labels 4 and 9, and label 9 with label 6. If cartographic preferences are considered, the cost is $\bar{c}(S) = 4.0009$.

lap minimization problem for the original instance with t overlaps, then there is a solution with t overlaps after fixing labels in this way; (iv) when cartographic preferences have to be taken into account, the optimum of the reduced problem is not necessarily the optimum of the original one. So, problem size reduction is not systematically used in our computational results.

3. POPMUSIC for PFLP

In this section, we review the basic ideas of POPMUSIC introduced by Taillard and Voss [15] and we present our adaptation of the POPMUSIC frame for the PFLP.

3.1. General POPMUSIC frame

The basic POPMUSIC frame can be summarized as follows:

```

procedure POPMUSIC( $r, S$ );
1  Solution  $S$  composed of  $q$  parts  $s_1, \dots, s_q$ ;
2   $O \leftarrow \emptyset$ ;
3  while  $O \neq \{s_1, \dots, s_q\}$  repeat
4    Select  $s_i \notin O$ ;
5    Create a subproblem  $R_i$  composed of the  $r$  closest parts from  $s_i$ ;
6    Optimize  $R_i$ ;
7    if  $R_i$  has been improved then update  $S$ ,  $O \leftarrow O \setminus R_i$ ;
8    else  $O \leftarrow O \cup \{s_i\}$ ;
end POPMUSIC.
    
```

Let us suppose that a solution S can be represented as a set of parts s_1, \dots, s_q . Let us also suppose that a distance measure can be defined between two parts. The central idea of POPMUSIC is to select a part s_i , called seed part, and a number $r < q$ of the closest parts from the seed part s_i to form a subproblem called R_i . If parts and subproblems are defined in an appropriate way, an improvement in the solution of the whole problem can be found for every improvement in the subproblem.

Once the parts of a solution have been defined, POPMUSIC tries to improve the solution by locally improving a subproblem composed of the seed part and a few parts that mostly interact with the chosen seed part. The number of parts considered for optimizing the subproblem so defined is given by r , the unique parameter of the method (supposing that the parameters of the optimization procedure are not chosen by the user).

To avoid generating twice the same subproblem, a set O of parts is stored. O contains the seed parts that have been used to define a subproblem already treated without success. Once O contains all the parts of the complete solution, then all subproblems have been examined without success and the process stops. If subproblem R_i has been successfully improved, a number of parts from s_1, \dots, s_q have been changed and further improvements may be found in the neighborhood of these parts. In this case, all the parts used

for building R_i are removed from O before continuing the process.

There are few possible variants for a POPMUSIC frame. A faster one uses $O \leftarrow O \cup R_i$ instead of $O \leftarrow O \cup \{s_i\}$ in line 8. A slower variant uses $O \leftarrow \emptyset$ instead of $O \leftarrow O \setminus R_i$ in line 7.

3.2. POPMUSIC adaptation for the PFLP

Like other metaheuristics, the general POPMUSIC frame presented above does not define several points precisely. These points depend on the problem being solved and their definitions must be completed by the designer of a POPMUSIC-based method.

The main points that are left free in the POPMUSIC frame are:

- How a first *initial solution* is obtained;
- What is the definition of the *parts* of a solution (and hence, what type of subproblem a set of several parts is defining);
- What is the *Selection* procedure of a part in O ;
- How the measure of the *distance* between parts is defined;
- What procedure is used as subproblem *optimizer*.

The choices we have made in our POPMUSIC implementation for the PFLP are the following: First, the initial solution is obtained with a fast constructive procedure. Then, each of the n points defines a part s_1, \dots, s_n , and consequently each part is composed of p potential labels. After that, the choice of the next *seed* part is arbitrary (index order). To define the notion of distance between two parts, we (implicitly) build an undirected graph with n vertices, one per part. In this graph, we put an edge between vertices x and y whenever there is at least a label of point x that can overlap with a label of point y . The distance between part x and y is the number of edges of the shortest path between x and y in this graphical representation. Starting from a seed part s_i , we construct subproblem R_i by including in R_i all neighbors of s_i , then all neighbors of the neighbors of s_i , and so on. Finally, we use procedure `Tabu_PFLP`, presented in Section 3.2.3, as subproblem optimizer. Let us now present all these components in detail.

3.2.1. Building an initial solution

In order to get an initial solution for the problem as quickly as possible, we used the first two steps of procedure FALP [20,13]. The remaining steps are not used since their complexity, $O((np)^2)$, is too high. The initial solution S is built in the following way: First, all label positions are sorted by increasing number of conflicts with other label positions. Then, the label positions are readily inserted into partial solution S in this order, provided that their corresponding point has not already been labeled and that the label position is not in conflict with other labels already in S . Each time a label is selected, all label positions (of

points not yet labeled) are removed from the list and the remaining label positions are re-sorted. Our implementation maintains a collection of overlap lists $Overlaps[k]$, $k = 1, \dots, np$, where $Overlaps[k]$ points to all labels that overlap with k . We suppose that the weight of label k can be computed in constant time as a function of its identifier $(1, \dots, np)$.

The basic steps of procedure FALP are presented in Fig. 3. In this figure, Function `Point(k)` returns the point of label k . Line 1 initializes a priority queue L that stores all candidate labels that can still be used to complete solution S (priority= $|Overlaps[i]|$, the number of potential conflicts of label i , $i = 1, \dots, np$). At each iteration (lines 3–13), the label position with the lowest number of potential conflicts is inserted into partial solution S (line 5). This label corresponds to a label without conflict for partial solution S . The inner loop (lines 6–12) eliminates all labels $v \in L$ that overlap with label u that has just been inserted into solution S (line 7). Indeed, label v cannot be used anymore without creating a conflict. Since label v is removed, the number of conflicts of labels overlapping with v must have decreased (line 9). The loop ends when there are no more labels in L , meaning that no labels can be added to partial solution S without creating an overlap.

The second step is performed in the next loop (lines 14–17) which completes solution S by inserting, for each point x not yet labeled ($S[x] = \emptyset$), the label which least increases the number of overlaps in solution S . Line 15 considers each of the p candidate label positions for current point x and chooses the one which has the least conflicts with labels already in solution S . Line 16 updates partial solution S .

Complexity of FALP. Creating the priority queue L (line 1) grows quasi-linearly with total number of labels ($O(np \log np)$). The selection of label u in line 3 costs $O(1)$ and removing u from L (line 4) costs $O(\log np)$ and is repeated np times. Let d be the maximum number of conflicts for a given label. There are d labels at most to consider in line 6, and for each one there are at most d labels in conflict (line 8). Lines 7 and 10 make $\log np$ moves each, and decrementing priority (line 9) costs $O(1)$. Therefore loop 2–13 takes $O(npd^2 \log np)$ time. The second step considers np labels (lines 14–15). For each one there are at most d labels in conflict to consider. For this step, the total number of moves is $O(np d)$. Therefore the overall complexity of this implementation of FALP is $O(npd^2 \log np)$.

3.2.2. Subproblem creation

As presented above, each of the n points to label is one part. An undirected graph with n nodes is implicitly built by putting an edge of length 1 between vertices x and y whenever there is at least one label of point x that can overlap with at least one label of point y . The length of the shortest path between two points x and y is used to define the distance between x and y . Note that x and y may belong to different connected components. In this case, x and y are completely independent of each other and the

```

procedure Two_Steps_FALP( $n, p, Overlaps$ );
  /* first step */
  1 Put all labels  $i = 1, \dots, np$  in  $L$  with priority  $|Overlaps[i]|$ ;
  2 while  $L \neq \emptyset$  do
  3   Choose the label  $u \in L$  with lowest priority (number of conflicts);
  4   Remove  $u$  from  $L$ ;
  5    $S[Point(u)] \leftarrow u$ ;
  6   forall labels  $v \in Overlaps[u]$  and  $v \in L$  do
  7     Remove  $v$  from  $L$ ;
  8     forall labels  $w \in Overlaps[v]$  and  $w \in L$  do
  9       Decrement the priority (number of conflicts) of  $w \in L$ ;
 10      Sort  $L$ ;
 11    end forall;
 12  end forall;
 13 end while;
  /* second step */
 14 forall point  $x \in \{1, \dots, n\}$  and ( $S[x] = \emptyset$ ) do
 15   Choose label  $k \in \{(x-1) * p + 1, \dots, x * p\}$ 
      which minimizes the number of conflicts in solution  $S$ ;
 16    $S[x] \leftarrow k$ ;
 17 end forall;
 18 return  $S$ ;
end Two_Steps_FALP.

```

Fig. 3. Pseudo-code of the first two steps of procedure FALP [20].

problem itself can be broken down into several independent problems, one for each connected component.

The pseudo-code for the procedure used to create subproblem R_i is presented in Fig. 4. As input parameters, Procedure SubProblemRi takes r , the ideal size of the

subproblem we aim to create, the number p of candidate positions, the seed part s_i , a collection of overlap lists $Overlaps[k]$ and control array $Border[k]$. Let us define the following functions: $Point(l)$ returns the point of label l ; $Empty(Q)$ returns true if list Q is empty and false

```

procedure SubProblemRi( $r, p, s_i, Overlaps, Border$ );
  1  $Q \leftarrow s_i$ ;
  2  $R_i \leftarrow \emptyset$ ;
  3 while  $|R_i| < r$  and not  $Empty(Q)$  repeat
  4    $x \leftarrow First(Q)$ ;
  5    $R_i \leftarrow R_i \cup \{x\}$ ;
  6   forall  $k \in \{(x-1) * p + 1, (x-1) * p + 2, \dots, x * p\}$  do
  7     forall  $j \in Overlaps[k]$  do
  8       if  $Point(j) \notin R_i$  and  $Point(j) \notin Q$  do
  9          $Insert(Q, Point(j))$ ;
 10      end if
 11    end forall
 12  end forall
 13 end while
 14 while not  $Empty(Q)$  repeat
 15    $x \leftarrow First(Q)$ ;
 16    $R_i \leftarrow R_i \cup x$ ;
 17    $Border[x] \leftarrow .true.$ ;
 18 end while
 20 return  $R_i, Border$ ;
end SubProblemRi.

```

Fig. 4. Pseudo-code of the procedure used to create a subproblem.

otherwise; $\text{First}(Q)$ returns the first element of Q and $\text{Insert}(Q,i)$ inserts element i at the end of Q . Queue Q is used to store the points that could be further included in R_i . Q is initialized with seed part s_i in line 1. When we assign a value to parameter r we cannot guarantee that there will be exactly r neighbors related to the seed part. In case a connected component contains less than r points, the size of a subproblem can be smaller than r . Conversely, there are situations for which R_i will be greater than r . These situations are explained later. However, in this case, the optimization procedure is allowed to change the position of r labels at most.

The loop in lines 3–13 is repeated while the size of the subproblem is lower than r and there are still points that can interact with R_i (Q not empty). Lines 4 and 5 initialize current part x and subproblem R_i with x . For each label position k associated with the current point x being inserted into subproblem R_i (Line 6), all its pertinent neighbors are included in Q (Line 9), *i.e.* neither those already in subproblem R_i nor in queue Q . When Loop 3–13 ends, there might still be parts (*points*) in queue Q , meaning that the last part included in R_i has neighbors which are not in R_i . To guarantee that an improvement of the solution of the subproblem implies in an improvement in the whole solution, we include these parts in R_i but we also store these parts in an auxiliary array *Border* to indicate that they are *border* parts and that they cannot have their label position changed. In this situation, the number of parts of subproblem R_i will be strictly greater than r . Meanwhile, as most r parts can have their label changed by the optimization procedure, the other parts are *Border* parts used only to calculate the cost of the solution being optimized. Loop 14–18 updates array *Border*. Procedure SubProblemRi returns subproblem R_i (line 19) and array *Border*.

Complexity. There are r parts to consider in line 3, for each one there are p candidate labels (line 6) and d labels

in conflict (line 7), therefore Loop 3–13 and Loop 14–18 takes $O(rpd)$ time.

Fig. 5 illustrates the creation of a subproblem with $r = 25$. The seed part (with its $p = 4$ possible label positions) is identified by 0. The neighbors of the seed part are numbered 1 (points that can be in conflict with the seed part). The neighbors of parts numbered 1 are numbered 2 and so on. The parts not numbered in the figure are marked by (x). For this figure, we have $r = 25$ points whose label can be moved (those with numbers 0, 1, 2, 3 and a subset of those with number 4) and $|R_i| = 32$ points that are taken into consideration in the subproblem. The last is composed of all objects with numbers 0, 1, 2, 3, 4 and a subset of those with number 5. In this figure, the 7 points belonging to the border are darker, contain dashed lines and have numbers 4 or 5. The remaining objects with numbers 5 or larger, or not numbered, are completely ignored during subproblem optimizations.

3.2.3. Subproblem optimization with tabu search

In order to optimize subproblems, we used a new implementation of the tabu search procedure proposed by [19]. Since [19] does not provide a detailed pseudo code of their algorithm nor discuss their implementation, we present our implementation precisely. The last is slightly different from [19]. The parameter settings are different, as well as the initial solution.

Neighborhood. Let $S = (y_1, y_2, \dots, y_n)$ be a solution to a problem. Let $L_S(x)$ be the label of point x in solution S and let $\Delta_S(y_x) = \sum_{j=1}^n (a_{y_x, y_j})$ be the cost of label y_x in solution S . Solution S' is a neighbor of S whenever only one label differs between S and S' . Move $x(i \rightarrow k)$ is defined by changing the label of point x from i to k . The solution S' resulting from applying this move to solution S is characterized by $L_{S'}(x) = k$, $L_{S'}(\ell) = L_S(\ell) \forall \ell \neq x$. The cost of a move is approximated by $\Delta_S(k) - \Delta_S(i)$.

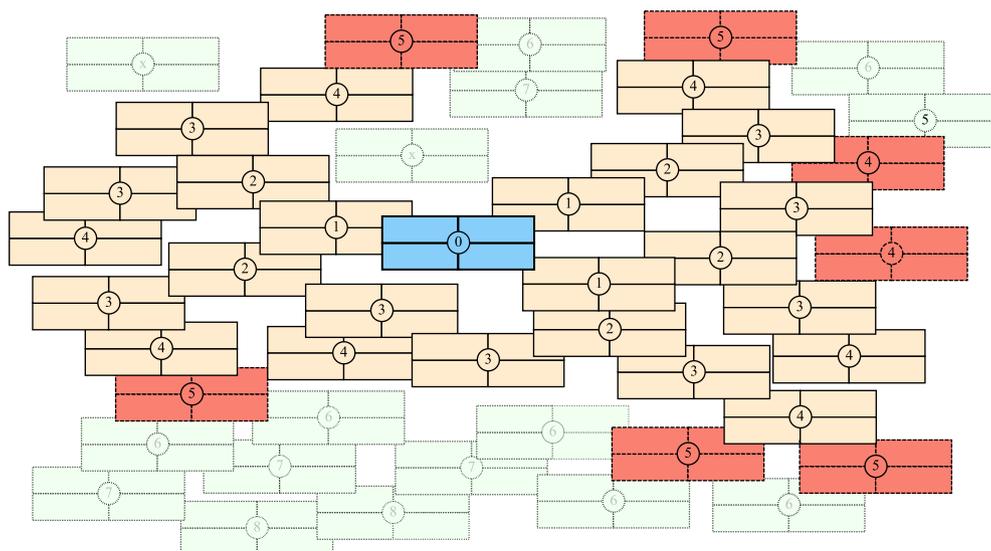


Fig. 5. Illustration of the creation of a subproblem in POPMUSIC.

Tabu status and candidate list. Whenever a move $x(i \rightarrow k)$ is performed, we forbid all moves that would change the label of point x for the duration of parameter *tenure* iterations. To reduce the neighborhood size, we

store a list of candidate labels (of size *candidateListSize*) with the highest values of $\Delta_S(y_x)$. At each tabu iteration, we scan the candidate list (in non-increasing order of $\Delta_S(i)$) and choose the allowed move $x(i \rightarrow k)$ with the

```

procedure Tabu_PFLP( $n, p, Overlaps, S = (y_1, y_2, \dots, y_n), maxTabuIt,$ 
   $candidateBaseFactor, minCandidateLSize, reductionFactor$ 
   $minTabuTSize, TabuFactor, GrowingFactor, Border$ );
1   $iterations \leftarrow 0;$ 
2   $TabuList[i] \leftarrow -1, i = 1, \dots, n;$ 
3  Calculate  $\Delta_S(k) \forall k = \{1, \dots, (np)\};$ 
4   $CandidateList \leftarrow (y_{k_1}, y_{k_2}, \dots, y_{k_{candidateListSize}}) \subseteq S$ 
   where  $\Delta_S(y_{k_1}) \geq \Delta_S(y_{k_2}) \geq \dots \geq \Delta_S(y_{k_{candidateListSize}});$ 
5   $bestSolution \leftarrow S;$   $candidateFactor \leftarrow candidateBaseFactor;$ 
6  while ( $iterations < maxTabuIt$ ) and ( $c(S) > 0$ ) do
7     $candidateListSize, candidateFactor, tenure \leftarrow Actualize\_Tabu\_CandidateL(m,$ 
    $iterations, c(S), candidateListSize, candidateBaseFactor,$ 
    $candidateFactor, minCandidateLSize, reductionFactor,$ 
    $minTabuTSize, tabuFactor, tenure);$ 
8     $kRetained \leftarrow \infty;$   $iRetained \leftarrow \infty;$   $minDelta \leftarrow \infty;$ 
9    forall  $y_i \in CandidateList$  and not  $Border[i]$  do
10     forall  $k \in \{(i-1) * p + 1, (i-1) * p + 2, \dots, i * p\} \setminus y_i$  do
11        $authorized \leftarrow (TabuList[i] < iterations);$ 
12       if ( $\Delta_S(k) < minDelta$ ) then
13         if not ( $authorized$ ) then
14            $S' \leftarrow S;$ 
15            $L_{S'}(i) \leftarrow k;$ 
16            $aspired \leftarrow (\bar{c}(S') < bestCost);$ 
17         end if;
18         if ( $authorized$  or  $aspired$ ) then
19            $kRetained \leftarrow k;$ 
20            $iRetained \leftarrow i;$ 
21            $minDelta \leftarrow \Delta_S(k);$ 
22         end if;
23       end if;
24     end forall;
25   end forall;
26    $iterations \leftarrow iterations + 1;$ 
27   if ( $kRetained = \infty$ ) then
28      $candidateListSize, candidateFactor \leftarrow Actualize\_CandidateL(c(S),$ 
    $candidateListSize, candidateBaseFactor, candidateFactor,$ 
    $minCandidateLSize, growingFactor);$ 
29   else
30      $TabuList[iRetained] \leftarrow TabuList[iRetained] + tenure;$ 
31     Update  $\Delta_S(k) \forall k = \{1, \dots, (np)\};$ 
32     Reorder  $CandidateList;$ 
33      $L_S(iRetained) \leftarrow kRetained;$ 
34     if  $\bar{c}(S) < \bar{c}(bestSolution)$  then
35        $bestSolution \leftarrow S;$ 
36     end if;
37   end if;
38 end while;
39 return  $bestSolution;$ 
end Tabu_PFLP.

```

Fig. 6. Pseudo-code of tabu search based local search procedure for PFLP.

smallest value of $\Delta_S(k)$. We also consider the classical aspiration criterion which allows a tabu move to be selected if it improves the best solution found so far. The tabu search procedure stops either when a solution with no overlaps is found or when a total of `MaxTabuIt` tabu search iterations have been performed.

Dynamic modification of parameters. The size of the candidate list has to depend on the number $c(S)$ of overlaps of the current solution S . The larger $c(S)$ is, the larger *candidateListSize* should be. Similarly, the tabu tenure has to depend on $c(S)$. We choose to adapt these parameters dynamically with $c(S)$. The tabu search procedure uses the following parameters:

- $tenure \leftarrow minTabuTSize + (tabuFactor * c(S))$ where *minTabuTSize* is the minimal size of the tabu list and *tabuFactor* is a factor, weighted by the number of overlaps;
- $candidateListSize \leftarrow \min(n, minCandidateLSize + candidateFactor * c(S))$ where *minCandidateLSize* is the minimal size of the candidate list and *candidateFactor* is a factor weighted by the number of overlaps;
- *maxTabuIt*, the maximum number of iterations performed by tabu search.

After some iterations, the number of overlapping labels decreases. So, after m consecutive iterations, we recalculate parameters *tenure* and *candidateListSize*. If all points are tabu, we multiply the size of *candidateFactor* by *growingFactor* which, as a consequence, increases the size of the candidate list. Further, at each iteration, this factor is divided by *reductionFactor* until it is equal to the original factor *candidateBaseFactor*. Fig. 6 summarizes procedure `Tabu_PFLP`.

Time complexity of our implementation. Let us suppose that label density (called d) does not vary with problem size (n), meaning that the average number of overlaps of each label is a constant, regardless of the number of labels in the problem instance. Procedure `Tabu_PFLP` returns the best solution found and takes as input problem dimension

n , number p of candidate positions, a collection of overlap lists, the initial solution S , seven tabu search parameters and the optional Boolean vector *Border* indicating whether a label can be moved or not. Initializations are performed in lines 1–5. The computation of Δ_S for each label (line 3) costs $O(npd)$ and the creation of an ordered list of size np (line 4) costs $O(np \log np)$. Iterations (lines 6–38) are performed until a solution with no overlaps is found or a total of *maxTabuIt* tabu search iterations have been performed. The update of the parameters (line 7), detailed in Fig. 7, costs $O(1)$. At each iteration, the algorithm determines the best move (loop 9–25). Scanning the candidate list (lines 9–10) costs $O(np)$. Verifying whether the point is tabu (line 11) is done in constant time and verifying whether the move being examined would produce a solution with lower $c(S)$ (lines 14–16) has cost $O(d)$. In line 27, we verify whether or not all the moves are tabu. In this case, we actualize parameter *candidateListSize* (Line 28; this step is specified precisely in Fig. 8). The move selected is applied to the current solution (lines 30–33). Updating and reordering the candidate list (lines 31–32) costs $O(nd)$. Therefore, the initialization step costs $O(npd + np \log np)$ and each tabu iteration costs $O(npd)$.

3.3. Putting all components together

The pseudo-code of our POPMUSIC approach for PFLP is given in Fig. 9. Procedure `Pop_PFLP` takes as input problem dimension n , number p of candidate positions, parameter r (the size of the subproblem), a collection of overlap lists *Overlaps[k]*, and tabu search parameters. It returns the best solution found. Initializations are performed in lines 1–3. A seed part not in O is selected in line 5 and the corresponding subproblem is constructed in lines 6–7 (detailed in Fig. 4). In line 8, we apply the tabu search based heuristic (described in the previous subsection) to the current subsolution S_i of the subproblem being examined. If the number of overlaps of S_i decreases, then we update the solution of the original problem S (line 10) and remove from O all the parts used to build

```

procedure Actualize_Tabu_CandidateL( $m, iterations, c(S),$ 
   $candidateListSize, candidateBaseFactor, candidateFactor,$ 
   $minCandidateLSize, reductionFactor, minTabuTSize,$ 
   $tabuFactor, tenure$ );
1 if  $candidateFactor > candidateBaseFactor$  then
2    $candidateFactor \leftarrow candidateFactor / reductionFactor$ ;
3 if ( $iterations \bmod m = 0$ ) then
4    $candidateListSize \leftarrow \min(n, minCandidateLSize +$ 
      $candidateFactor * c(S))$ ;
5    $tenure \leftarrow minTabuTSize + (tabuFactor * c(S))$ ;
6 end if;
7 return  $candidateListSize, candidateFactor, tenure$ ;
end Actualize_Tabu_CandidateL.

```

Fig. 7. Pseudo-code of procedure used to actualize parameters *candidateListSize* and *tenure*.

```

procedure Actualize_CandidateL( $c(S)$ ,  $candidateListSize$ ,
     $candidateBaseFactor$ ,  $candidateFactor$ ,  $minCandidateLSize$ ,
     $growingFactor$ );
1 if  $candidateListSize < n$  then
2      $candidateFactor \leftarrow candidateFactor * growingFactor$ ;
3 end if
4  $candidateListSize \leftarrow \min(n, minCandidateLSize +$ 
     $candidateFactor * c(S))$ ;
5 return  $candidateListSize, candidateFactor$ ;
end Actualize_CandidateL.

```

Fig. 8. Pseudo-code of procedure used to actualize parameter $candidateListSize$.

```

procedure Pop_PFLP( $n, p, r, Overlaps[k], t_1, \dots, t_6$ );
1  $S = (s_1, \dots, s_n) \leftarrow Two\_Steps\_FALP(n, p, Overlaps)$ ;
2  $O \leftarrow \emptyset$ ;
3  $Border[x] \leftarrow .false., x = 1, \dots, n$ ;
4 while  $O \neq \{s_1, \dots, s_n\}$  and  $c(S) > 0$  repeat
5     Select  $s_i \notin O$ ;
6      $R_i, Border \leftarrow SubProblemRi(r, p, s_i, Overlaps, Border)$ ;
7     Construct sub-solution  $S_i$  corresponding to  $S$  and  $R_i$ ;
8      $S'_i \leftarrow Tabu\_PFLP(|R_i|, p, Overlaps, S_i, (r * 10), t_1, \dots, t_6, Border)$ ;
9     if  $\bar{c}(S'_i) < \bar{c}(S_i)$  then
10        Update the position of the  $r$  points of  $S'_i$  in the whole solution  $S$ ;
11         $O \leftarrow O \setminus R_i$ ;
12    else
13         $O \leftarrow O \cup \{s_i\}$ ;
14    end if
15    forall  $x \in R_i$  do  $Border[x] = .false.;$ 
16 end while
17 return  $S$ ;
end Pop_PFLP.

```

Fig. 9. Pseudo-code of POPMUSIC-based procedure for PFLP.

R_i (line 11). Otherwise (line 13) we include the current seed part in O . Array $Border$ is initialized in line 3 and actualized in lines 6 and 15. The main loop (lines 4–16) ends either when a solution without overlaps is found or when there is no seed part available to create a subproblem.

The overall complexity of this procedure cannot be deduced in the worst case since, *a priori*, the number z of times Loop 4–16 is repeated is unknown. It is shown experimentally in Section 6 that z is increasing almost linearly with n , for uniformly generated instances.

However, the best case complexity is: $\Omega(npd^2 \log(np) + nrp \log(rp))$. Indeed, the initial solution has to be created ($O(npd^2 \log np)$) and at least $n \leq z$ subproblem of size r have to be optimized with a tabu search performing $10r$ iterations. Since the initial solution is built only once, the part of the algorithm requiring the most computational effort is the optimization of subproblems with tabu search. As we have shown in Section 3.2.3, time complexity of tabu

search is $O(npd + np \log np)$ for the initialization step and $O(npd)$ for each tabu iteration. To get the complexity of the call to the tabu search optimization procedure (line 8) we substitute n by r and multiply the iteration step by $10r$, the maximum number of tabu iterations performed. We then get a complexity of $O(rp d + rp \log rp + r^2 p d) \equiv O(rp \log rp + r^2 p d)$. Since the loop of lines 4–16 is done at least n times and the initialization step costs $O(npd^2 \log np)$, the overall best complexity is $\Omega(npd^2 \log np + nrp \log rp + nr^2 p d)$. So, Pop_PFLP complexity grows quasi-linearly with the number of labels and quadratically with r and d .

4. Computational results

This section presents numerical results obtained from the application of our POPMUSIC-based algorithm (Pop_PFLP) and our implementation of the basic tabu search presented in this paper (Tabu_PFLP).

4.1. Environment

All computational experiments were performed on a Pentium M Centrino 1.6 GHz with 512 MB of RAM memory. Algorithms `Pop_PFLP` and `Tabu_PFLP` were coded in C and compiled with `g++ version 4.1.2` using the optimization flag `-O4`. All running times reported in this paper for our implementations are CPU seconds measured with `getrusage` function up to the *end* of the computation. Running times do not include the cost of reading the input data.

4.2. Parameter setting

In all our computational experiments, we have always used the following tabu search parameter values (see Section 3.2.3 for their definition):

- $minTabuTSize = 9$
- $tabuFactor = 0.5$
- $minCandidateLSize = 18$
- $candidateBaseFactor = 0.73$
- $growingFactor = 15$
- $reductionFactor = 1.3$
- $m = 50$.

These values originate from [2]. When invoked from `Pop_PFLP` for optimizing subproblems of size r , parameter $maxTabuIt$ (maximum number of tabu iterations) is set to $10 * r$, as previously mentioned.

We provide results for `Pop_PFLP` for various subproblem sizes, namely: $r = 10$ (`Pop(10)`), $r = 30$ (`Pop(30)`) and $r = 70$ (`Pop(70)`). We have also tried an “ascending” version called `Pop(asc)`. In this version we have started with $r = 10$ and, each time the list of seed elements is revisited, we increment the value of r by 20 until $r = 70$. For this version we used $O \leftarrow \emptyset$ in line 10 of Fig. 9 instead of $O \leftarrow O \setminus R_i$ (implying a higher complexity). Method `Tabu_PFLP` was tested with different settings for parameter $maxTabuIt$. We try $maxTabuIt = 50 * n$ (`Tabu(50n)`), $maxTabuIt = 100 * n$ (`Tabu(100n)`) and $maxTabuIt = 500 * n$ (`Tabu(500n)`). For each of these seven algorithms we investigate two variants using objective functions: minimize $c(S)$ or minimize $\bar{c}(S)$. Regardless of the objective

function used to guide the searches, for a given solution S we can always compute the number $c(S)$ of overlaps, the % of labels placed without overlap and the cost $\bar{c}(S)$ of the solution if preferences were taken into account.

4.3. Comparison with existing methods

To compare our algorithms with other implementations, we first considered the set of test problems introduced by L.A.N. Lorena and available from [8]. There are twenty five instances for each value of the number of points $n \in \{25, 100, 250, 500, 750, 1000\}$ with four potential label positions for each point. We have considered only instances with $n \geq 250$ in our computational results, since the other instances are very easy to solve.

Ribeiro and Lorena [12] report lower bounds on the number of pairwise overlaps $\binom{c(S)}{2}$ for Lorena’s instances [8]. Table 1 shows, for each n , the best lower bound extracted from [12], and upper bound ub , gap $(100 \cdot \frac{ub-lb}{lb})$ and time in seconds found, respectively, by `LagClus` [12] and by `pop(70)`. We see in this table that our results are very close to optimal solutions. So, the challenge for these instances is either to prove optimality (which is done for $n = 500$) or to obtain excellent solutions in moderate computational time (which is the main purpose of Table 2).

To investigate the effectiveness of `POPMUSIC` strategy, we compared the results obtained by algorithms `Pop(10)`, `Pop(30)`, `Pop(70)`, `Pop(asc)`, `Tabu(50n)`, `Tabu(100n)` and `Tabu(500n)` with those obtained by other approaches reported in the literature: `FALP` [20], the Genetic Heuristic of Yamamoto et al. [21], the Tabu Search Heuristic of Yamamoto et al. [19] and a recent Lagrangean relaxation with cluster by Ribeiro and Lorena [12] were developed while the present article was under revision. Let us mention that the numerical results provided in the first three references do not consider cartographic preferences. So, we only provide results for the variants of our algorithms when they use $c(S)$ as objective.

Table 2 shows for each problem dimension $n \in \{250, 500, 750, 1000\}$: Algorithm name, percentage of labels placed without conflicts and CPU time in seconds (for our methods, this is the total computational time up to termination of the program). The results were obtained by averaging over 25 instances. The size of the problem

Table 1
Best lower bounds on the number of pairwise overlaps $\binom{c(S)}{2}$

n	best lb	LagClus			pop(70)		
		ub	Gap (%)	Time [second] ^c	ub	Gap (%)	Time [second]
250	0.00 ^a	0.00	0.00	0.12	0.00	0.0	0.00
500	0.84 ^a	0.84	0.00	0.4	0.84	0.0	0.04
750	8.09 ^a	8.96	10.75	53.84	8.92	10.26	0.79
1000	31.23 ^b	44.80	43.45	3842.84	38.76	24.11	3.22

^a Extracted from Table 5 [12].

^b Extracted from Table 6 [12].

^c Pentium IV 2.66 GHz.

Table 2
Comparison with other approaches

	minimize $c(S)$							
	$n = 250$		$n = 500$		$n = 750$		$n = 1000$	
	%	Time	%	Time	%	Time	%	Time
Pop(10)	100.00	0.00	99.67	0.01	97.46	0.11	91.94	0.30
Pop(30)	100.00	0.00	99.67	0.02	97.72	0.23	92.54	0.96
Pop(70)	100.00	0.00	99.67	0.04	97.73	0.79	92.58	3.22
Pop(asc)	100.00	0.00	99.67	0.01	97.72	0.41	92.68	2.57
Tabu(50n)	100.00	0.00	99.57	0.07	97.53	0.48	91.54	1.29
Tabu(100n)	100.00	0.00	99.57	0.14	97.54	0.96	91.54	2.58
Tabu(500n)	100.00	0.00	99.57	0.70	97.55	4.78	91.59	12.86
FALP [20]	100.00	0.00 ^a	99.50	0.10 ^a	96.70	0.28 ^a	90.12	0.59 ^a
CGA(best) [21]	100.00	0.06 ^b	99.60	2.15 ^b	97.10	22.89 ^b	90.70	122.72 ^b
CGA(average) [21]	100.00	0.06 ^b	99.60	2.15 ^b	96.80	19.59 ^b	90.40	98.18 ^b
Tabu [19]	100.00	0.62 ^c	99.26	2.53 ^c	96.76	5.44 ^c	90.00	26.2 ^c
LagClus [12]	100.00	0.12 ^d	99.67	0.4 ^d	97.65	53.84 ^d	91.42	3842.84 ^d

^a Time to reach the best solutions on a Pentium II? divided by 10 [6].

^b Time to reach the best solutions on a Pentium III divided by 10 [6].

^c Time to reach the best solutions on a Sun Sparc 20 divided by 45 [6].

^d Time on a Pentium IV 2.66 GHz processor with 512 MB of RAM memory.

instances was not reduced. Since the numerical results of other authors were obtained using different computers, we have divided their computational times with appropriated factors [6] so that the values indicated in Table 2 should be directly comparable. The results for FALP (Fast Point-Feature Label Placement Algorithm) were extracted from [20], Tables 1 and 2. The authors report average results over 25 instances. The results for CGA(best) and CGA(average) were extracted from [21], Tables 2 and 3. CGA(best) refers to the best result over six trials and CGA(average) reports average results obtained by the constructive genetic approach. The results for Tabu (Tabu search implementation of [19]) were extracted from [19], Table 2. The reported time for these three methods is the time to reach the best solution (and not total computational time). The results for LagClus were extracted from [12], Table 7. Best results are highlighted.

Comparing the results obtained by our Tabu_PFLP implementation with those reported by the Tabu Search Heuristic of Yamamoto et al. [19], we note that the computational effort of Tabu [19] is of the same order of magnitude as those of our tabu search running for 500n iterations. Note however that Yamamoto et al. [21] only provide the computational time to reach the best solution. We also observed that our Tabu_PFLP implementation provides solutions of better quality than those reported by Tabu [19], which might be explained as we used different initial solutions and a better tuning of the parameters. Therefore, the computational times are difficult to compare, but without a doubt the computational effort required for our methods is significantly lower than those for CGA and LagClus.

From Table 2, one can conclude that, for the set of instances available at [8], the fastest POPMUSIC variant (Pop(10)) is faster than the best methods previously published while providing solutions of higher quality. Indeed,

Algorithm Pop(10) found better solutions in less time than FALP [20], CGA(best) [21] and Tabu [19]. Concerning the new method LagClus [12], Pop(30) provides better solutions with computational efforts several orders of magnitude lower.

The best POPMUSIC variant seems to be Pop(asc) but it requires computational efforts higher than FALP for $n \in \{750, 1000\}$. However, the solution quality is much better (the gap to optimality is divided by 2 or more). Our first conclusion is that embedding the basic tabu search proposed by [19] in a POPMUSIC frame, which is very simple to implement, is an excellent choice when dealing with large instances of point feature label placement problems.

4.4. Computational results for problem instances of Lorena [8]

Since other authors do not provide results considering alternate objectives, we report next on more complete computational results for our new implementations only.

Table 3 summarizes the results we obtained, averaged over the 25 instances for each problem size $n \in \{500, 750, 1000\}$ and for two objective functions (minimize $c(S)$ and minimize $\bar{c}(S)$). Results are provided both with and without reduction of problem size. Let us mention that for $n = 250$ all our algorithms find solutions without overlap (so with $f(S) = c(S) = 0$) in less than 0.01 seconds and with $\bar{c}(S) = 0.03$. Table 3 provides the following information: algorithm name, percentage of labels placed without conflict, number of labels placed without conflict, number of overlaps, cost of the solution if we consider cartographic preferences, and finally CPU time (in seconds). For each n we also provide the number $f(S_{\text{FALP}})$ of labels placed without conflict and the number $c(S_{\text{FALP}})$ of overlaps in the initial solution. When problem size reduction was used, we

Table 3
Experimental results for variants of algorithms Pop_PFLP and Tabu_PFLP on PFLP instances

	Without cartographic preferences					With cartographic preferences				
	%	$f(S)$	$c(S)$	$\bar{c}(S)$	Time	%	$f(S)$	$c(S)$	$\bar{c}(S)$	Time
Without reduction										
$n = 500, f(S_{\text{FALP}}) = 2.32(=99.54\%), c(S_{\text{FALP}}) = 2.72$										
Pop(10)	99.67	1.64	1.68	1.73	0.01	99.66	1.68	1.68	1.73	0.01
Pop(30)	99.67	1.64	1.68	1.73	0.02	99.65	1.76	1.76	1.81	0.02
Pop(70)	99.67	1.64	1.68	1.73	0.04	99.63	1.84	1.84	1.89	0.05
Pop(asc)	99.67	1.64	1.68	1.73	0.01	99.66	1.68	1.68	1.73	0.03
Tabu(50n)	99.57	2.16	2.32	2.37	0.07	99.55	2.24	2.32	2.37	0.12
Tabu(100n)	99.57	2.16	2.32	2.37	0.14	99.55	2.24	2.32	2.37	0.24
Tabu(500n)	99.57	2.16	2.32	2.37	0.70	99.57	2.16	2.24	2.29	1.14
$n = 750, f(S_{\text{FALP}}) = 24.56(=96.72\%), c(S_{\text{FALP}}) = 28.24$										
Pop(10)	97.46	19.04	20.08	20.17	0.08	97.44	19.20	19.84	19.92	0.11
Pop(30)	97.72	17.12	17.92	18.01	0.23	97.50	18.76	19.52	19.59	0.34
Pop(70)	97.73	17.04	17.84	17.93	0.79	97.25	20.60	21.52	21.59	1.39
Pop(asc)	97.72	17.12	17.92	18.01	0.41	97.57	18.24	18.80	18.87	2.03
Tabu(50n)	97.53	18.52	18.88	18.97	0.48	96.84	23.72	25.28	25.36	0.69
Tabu(100n)	97.54	18.44	18.80	18.89	0.96	96.86	23.52	25.04	25.12	1.38
Tabu(500n)	97.55	18.36	18.72	18.81	4.78	96.89	23.36	24.40	24.48	6.85
$n = 1000, f(S_{\text{FALP}}) = 94.76(=90.52\%), c(S_{\text{FALP}}) = 116.80$										
Pop(10)	91.94	80.60	86.64	86.78	0.30	91.76	82.36	85.36	85.47	0.40
Pop(30)	92.54	74.56	78.64	78.78	0.96	91.85	81.48	85.76	85.86	1.40
Pop(70)	92.58	74.20	77.52	77.66	3.22	91.22	87.84	94.80	94.90	5.29
Pop(asc)	92.68	73.20	77.84	77.98	2.59	92.05	79.52	82.24	82.34	8.18
Tabu(50n)	91.54	84.64	87.28	87.42	1.29	90.53	94.68	104.32	104.46	1.76
Tabu(100n)	91.54	84.60	87.04	87.18	2.58	90.53	94.68	104.32	104.46	3.51
Tabu(500n)	91.59	84.12	86.56	86.70	12.86	90.53	94.68	104.32	104.46	17.63
With reduction										
$n = 500(343.40\text{points reduced}), f(S_{\text{FALP}}) = 2.52(=99.50\%), c(S_{\text{FALP}}) = 2.80$										
Pop(10)	99.67	1.64	1.68	1.73	0.00	99.66	1.68	1.68	1.73	0.01
Pop(30)	99.67	1.64	1.68	1.73	0.01	99.66	1.68	1.68	1.73	0.01
Pop(70)	99.67	1.64	1.68	1.73	0.03	99.66	1.68	1.68	1.73	0.03
Pop(asc)	99.67	1.64	1.68	1.73	0.01	99.66	1.68	1.68	1.73	0.01
Tabu(50n)	99.58	2.08	2.16	2.21	0.02	99.54	2.32	2.40	2.45	0.02
Tabu(100n)	99.58	2.08	2.16	2.21	0.04	99.54	2.32	2.40	2.45	0.05
Tabu(500n)	99.58	2.08	2.16	2.21	0.17	99.54	2.32	2.40	2.45	0.23
$n = 750(346.28\text{points reduced}), f(S_{\text{FALP}}) = 24.84(=96.69\%), c(S_{\text{FALP}}) = 28.56$										
Pop(10)	97.58	18.16	19.28	19.37	0.08	97.51	18.68	19.28	19.37	0.07
Pop(30)	97.72	17.12	17.92	18.01	0.15	97.50	18.72	19.28	19.36	0.20
Pop(70)	97.72	17.08	17.92	18.01	0.36	97.48	18.88	19.52	19.60	0.47
Pop(asc)	97.71	17.20	18.08	18.17	0.20	97.58	18.12	18.56	18.64	0.40
Tabu(50n)	97.44	19.20	19.44	19.53	0.20	96.84	23.72	24.88	24.97	0.28
Tabu(100n)	97.45	19.12	19.36	19.45	0.39	96.84	23.68	24.80	24.89	0.56
Tabu(500n)	97.49	18.84	19.04	19.13	1.95	96.89	23.32	24.16	24.25	2.71
$n = 1000(274.92\text{points reduced}), f(S_{\text{FALP}}) = 97.12(=90.29\%), c(S_{\text{FALP}}) = 121.36$										
Pop(10)	91.83	81.72	87.44	87.58	0.28	91.75	82.52	86.00	86.12	0.33
Pop(30)	92.48	75.20	78.80	78.94	0.81	91.88	81.24	85.76	85.88	1.08
Pop(70)	92.54	74.56	77.68	77.82	2.36	90.97	90.32	96.16	96.28	3.13
Pop(asc)	92.59	74.08	77.68	77.82	1.93	92.05	79.48	82.48	82.60	3.72
Tabu(50n)	91.45	85.52	88.08	88.22	0.83	90.28	97.20	106.88	107.01	1.12
Tabu(100n)	91.47	85.32	87.92	88.06	1.65	90.28	97.20	106.80	106.93	2.24
Tabu(500n)	91.52	84.84	87.52	87.66	8.20	90.28	97.20	106.80	106.93	11.30

Left: using number of conflict $c(S)$ as the objective to minimize. Right: using cartographic preferences $\bar{c}(S)$ as the objective to minimize.

also provide the number of points that have been suppressed by the reduction. Again, all results have been averaged over 25 instances.

Comparing the two different approaches Pop_PFLP and Tabu_PFLP we notice that all eight POPMUSIC-based vari-

ants systematically find solutions at least as good as the best tabu search variant (Tabu(500n)), with the exception of variant Pop(10) with $n = 750$, objective function $c(S)$ while taking much shorter CPU times. Therefore, the POPMUSIC approach is fundamental to finding good

solutions. Variant Pop(asc) provides the best solutions in terms of quality.

If we consider individual results for each of the 25 instances and for the four objectives considered in Table 3, there is no case where $c(S) \neq \text{int}(\bar{c}(S))$. So, $\text{int}(\bar{c}(S))$ expresses the number of overlaps for the set of problems available at [8]. This is due to two facts. First, we have chosen very low values for cartographic preferences. Second, for this set of instances, good solutions have a very low number of overlaps. We hoped that adding very low cartographic preferences could help guide the tabu search process toward good solutions. Indeed, when there are several solutions in the neighborhood having exactly the same number of overlaps (a frequently occurring situation), taking cartographic preferences could guide the tabu search. Unfortunately, these hopes were soon dashed. Apparently, solving the problem by considering cartographic preferences (minimize $\bar{c}(S)$) is more difficult than without it (minimize $c(S)$). In all cases, using $\bar{c}(S)$ as an objective function produces a solution quality ($f(S)$) no better than the solution quality obtained using $c(S)$ as an objective function.

This explains why increasing the subproblem size from $r = 30$ to $r = 70$ in Pop_PFLP seems to be counter-productive when minimizing $\bar{c}(S)$: the computational time is higher and the solutions worse (for $n \in \{500, 750, 1000\}$). This means that our tabu search optimization procedure cannot find good solutions to problems with a moderately large number of labels while taking cartographic preferences into account.

4.5. Algorithm behavior when changing objective function

Fig. 10 (and, respectively, Fig. 11) illustrates the average behavior of our algorithms with objective functions minimize $c(S)$ (respectively, minimize $\bar{c}(S)$) for the 25 problem

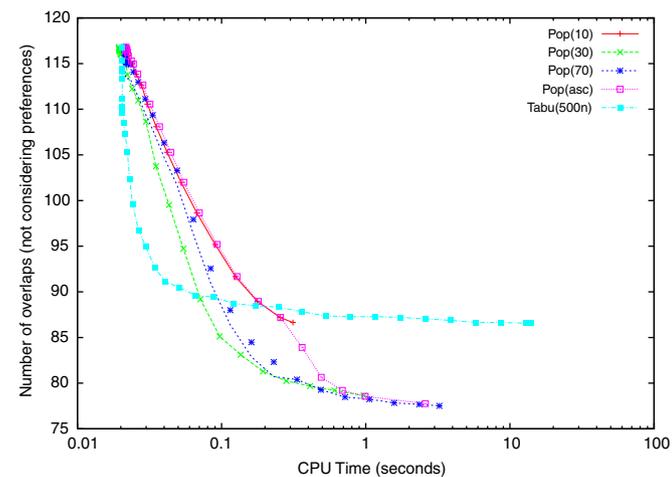


Fig. 10. Number of overlaps (minimize $c(S)$) as a function of time. Average results for 25 instances with $n = 1000$. Comparison of algorithms Pop(10), Pop(30), Pop(70), Pop(asc) and Tabu(500n). Logarithmic scale for horizontal axis.

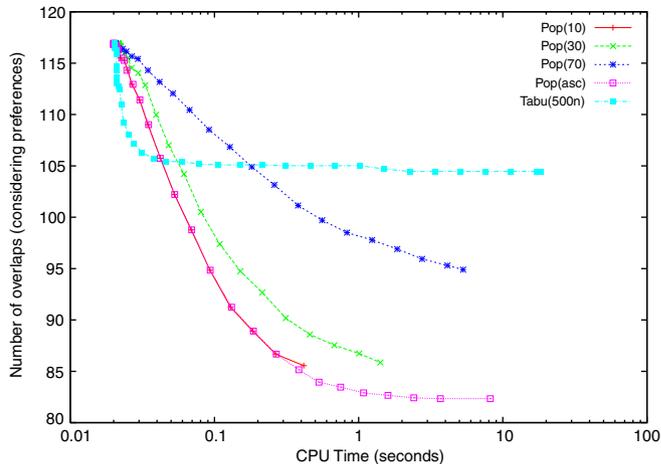


Fig. 11. Number of overlaps (minimize $\bar{c}(S)$) as a function of time. Average results over 25 instances with $n = 1000$. Comparison of algorithms Pop(10), Pop(30), Pop(70), Pop(asc) and Tabu(500n). Logarithmic scale for horizontal axis.

instances with $n = 1000$. In these figures, we plot the evolution of the number of overlaps of the best solution found so far as a function of computational time. As each POPMUSIC run might stop for different computational times, we end the plots as soon as one of the 25 runs stops.

These figures show that the tabu search is very aggressive. Initially, it finds better solutions faster than the four variants of Pop_PFLP methods. This is because the tabu search is implemented using a candidate list and, for these instances, there are several regions that do not need to be improved. So, the tabu search does not try any moves involving labels in these regions while Pop_PFLP launches an optimization process in these regions.

We can also observe that our implementation of tabu search is relatively ineffective for optimizing the number of overlaps in the long term while taking the cartographic preferences into account. This is true even for very small problem instances with $n = 70$, since the improvements are relatively moderate for Pop(70) compared to faster Pop_PFLP variants. So, an effort should be undertaken to design a good optimization process minimizing $\bar{c}(S)$.

After relatively few iterations, Tabu(500n) reaches a local optima and is unable to escape. Conversely, Pop_PFLP variants make small but constant improvements, cross Tabu(500n) and continue to improve the solutions.

Fig. 10 suggests that one could profit from tabu search aggressiveness and start Pop_PFLP with a solution obtained by a short tabu search. This strategy is shown in Fig. 12, also for problem instances with $n = 1000$ (without preferences). We call Tabu(50n)+Pop(70) the version that first runs Tabu_PFLP with MaxTabuIt = 50n and then Pop_PFLP with $r = 70$.

In this figure, we see that Pop_PFLP is indeed able to easily improve the solution produced by Tabu(50n). However, the final solution is not as good as the standard

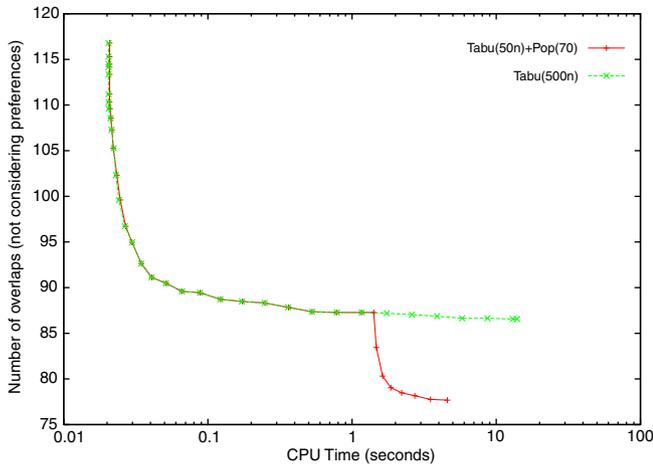


Fig. 12. Number of overlaps (minimize $c(S)$) as a function of time. Comparison of algorithms Tabu(50n) + Pop(70) and Tabu(500n). Results averaged over 25 instances with $n = 1000$.

Pop(70) starting with the FALP solution while exacting higher computational effort, mainly due to the time spent to obtain the Tabu(50n) solution.

5. New problem instances

Since a true geographical map contains several thousand points to label which are not uniformly distributed (as the problem instances considered above), we propose new map labeling problem instances. The positions of the points to label corresponds to 13,206 nodes of the road network in Switzerland. Twenty instances were generated by varying the size of the labels. Five different label surfaces have been chosen and for each surface, four different heights H and

lengths L . Fig. 13 shows the best solution found for the problem instance with $H = 4$ and $L = 12$ (units are hectometers).

Computational results for Pop_PFLP(10) are given in Table 4. For each problem instance we provide: the height H of the label, the length L of the label, the number p of candidate positions, the maximal degree $\max d$ of a node in the graph of incompatibilities, the number of point cover for which all p label positions cover another point, the number of point features for the reduced problem, the number $c(S_{\text{FALP}})$ of conflicts for the initial solution generated by the first two steps of FALP, the percentage of labels placed without conflicts (when using $c(S)$ as the objective to minimize), the number $f(S)$ of labels in conflict (when using $c(S)$ as the objective to minimize), the number $c(S)$ of conflicts, the objective value $\bar{c}(S)$ when cartographic preferences are taken into account, the computational time (in seconds) of Pop_PFLP(10), either if it uses $c(S)$ or $\bar{c}(S)$ as the objective to minimize.

In Table 4, we see that the computational time of Pop_PFLP(10) remains moderate for large problem instances. The computing time increases with the difficulty of the problem, even if the number of labels is constant. For optimizing problem instances of a size around 10, our implementation of the tabu search seems to be able to take cartographic preferences into account. Indeed, $\bar{c}(S)$ values are generally lower than $c(S)$ values. So, this means that adding cartographic preferences guides the tabu search into better regions than a pure Boolean criterion (whether or not the label is in conflict). As previously mentioned, for larger problems, the tabu search is not valuably guided with cartographic preferences. So, in Table 5, we compare the performances of Tabu(50n) and Pop(asc) by considering the versions of our algorithms that use $c(S)$

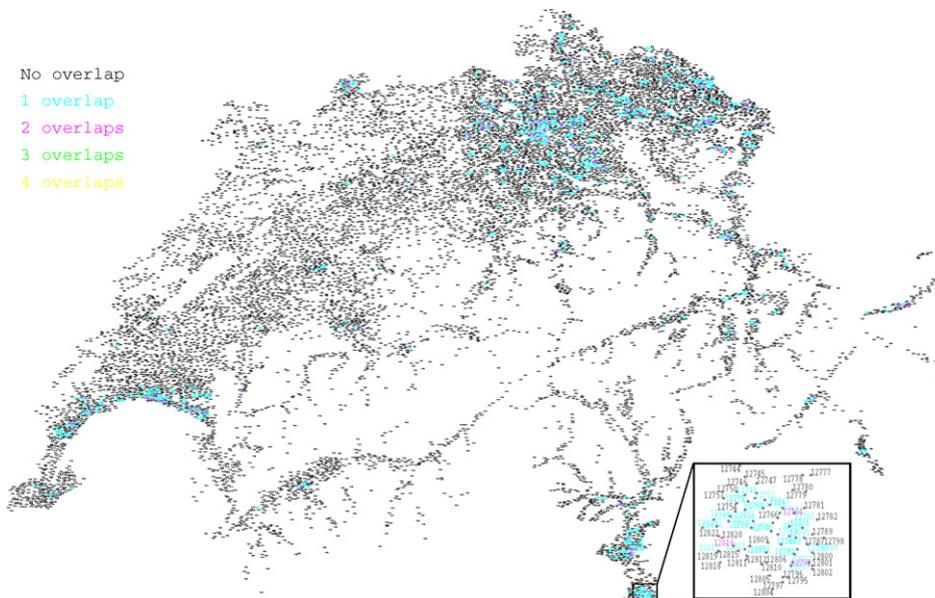


Fig. 13. Identifiers of 13,206 nodes of the road network in Switzerland. Black identifiers do not overlap with other identifiers.

Table 4
Experimental results for Pop_PFLP(10) on new PFLP instances

H	L	p	$\max d$	Cover	#Points	$c(S_{\text{FALP}})$	%	$f(S)$	$c(S)$	$\bar{c}(S)$	Time (c)	Time (\bar{c})
2	24	4	27	17	6624	1490	92.55	984	1092	1073.38	2.64	3.45
		8	57	0	7664	542	97.22	367	382	385.87	4.88	7.24
3	16	4	30	50	7628	2292	88.73	1488	1708	1709.50	3.68	4.88
		8	71	0	8536	1442	92.70	964	1056	1041.97	9.66	13.26
4	12	4	38	94	7939	2756	86.69	1758	2084	2047.57	4.23	5.47
		8	75	0	8634	1424	93.06	917	1002	970.18	9.92	13.89
6	8	4	41	149	8422	3294	84.42	2057	2448	2467.60	4.91	6.21
		8	86	0	9019	1746	91.54	1117	1234	1204.30	11.74	16.11
2	32	4	28	41	8077	3098	84.73	2017	2304	2305.59	4.42	5.73
		8	60	0	8982	1424	92.51	989	1034	1026.28	10.16	14.34
4	16	4	41	199	9468	5524	75.49	3237	4092	4091.83	6.90	9.04
		8	79	0	9893	3240	84.96	1986	2290	2306.77	18.06	24.81
8	8	4	48	338	9939	6616	71.16	3808	5130	5089.94	7.96	10.40
		8	100	0	10,265	4210	81.35	2463	2974	2937.04	21.75	29.09
16	4	4	42	204	8422	3294	84.42	2057	2448	2467.60	4.91	6.18
		8	88	0	9019	1746	91.54	1117	1234	1204.30	11.70	16.09
3	24	4	38	171	9615	5690	74.28	3396	4412	4337.85	6.82	8.68
		8	77	0	10,211	4260	81.07	2500	2972	2974.71	21.07	27.65
4	18	4	42	274	10,007	6972	69.96	3967	5250	5247.95	8.12	10.31
		8	80	0	10,423	4226	80.61	2561	3090	3025.07	21.52	30.21
6	12	4	48	406	10,497	7980	66.66	4403	6080	6026.08	9.14	11.68
		8	100	0	10,776	5118	77.46	2976	3700	3609.35	25.29	34.31
8	9	4	52	446	10,517	8386	65.10	4609	6504	6442.12	9.57	12.06
		8	114	0	10,781	5834	75.17	3279	4194	4151.34	27.71	36.82
2	42	4	34	88	9401	5652	74.15	3414	4296	4313.79	6.75	8.62
		8	69	0	10,010	3046	84.48	2050	2226	2198.86	17.12	23.13
3	28	4	41	359	10,238	7776	67.12	4342	5972	5930.03	8.72	10.94
		8	85	0	10,769	5892	74.58	3357	4188	4155.15	26.79	34.26
4	21	4	43	405	10,613	9188	62.86	4905	6996	7002.15	9.76	12.08
		8	84	0	10,975	6236	73.29	3527	4452	4357.49	28.51	37.59
6	14	4	51	551	11,019	10,626	58.87	5432	8156	8008.33	11.18	14.28
		8	107	0	11,245	7282	70.04	3956	5142	5091.87	31.73	41.83
2	48	4	37	126	9898	7356	68.17	4204	5496	5469.95	8.22	10.35
		8	73	0	10,475	4236	79.52	2705	3018	2969.13	21.25	28.66
3	32	4	43	360	10,711	9932	60.44	5224	7648	7574.22	10.30	13.00
		8	96	0	11,127	7792	68.13	4209	5542	5479.55	32.42	42.35
4	24	4	46	519	11,023	11,608	56.55	5738	8814	8792.38	11.56	14.54
		8	93	0	11,313	7972	67.86	4244	5650	5558.00	33.88	44.69
6	16	4	57	774	11,363	13,262	51.98	6342	10,294	10152.59	12.82	15.90
		8	117	0	11,564	9504	63.66	4799	6770	6700.37	37.69	48.90

Values for $\%$, $f(S)$ and $c(S)$ are obtained by using $c(S)$ as the objective. Values for $\bar{c}(S)$ are obtained by using $\bar{c}(S)$ as the objective to minimize. It can be noted that FALP produces solutions that are between 29% and 43% (average: 36%) worse than Pop_PFLP(10) in terms of $c(S)$ objective.

as the objective. Let us mention that the values of objectives for longer tabu search runs are almost identical, while taking up more computational time.

In Table 5 we see that Tabu(50n) requires two to three times more computational efforts than Pop(asc) and always produces inferior quality solutions. The number of labels in conflict with Tabu(50n) ranges from 1.5% to 13.3% higher than with Pop(asc) while both the number of conflicts $c(S)$ and the objective value with cartographic preferences $\bar{c}(S)$ ranges from 7% to 26% higher. Excepting the first two instances, the comparison of Tabu(50n) with Pop_PFLP(10) reveals that the latter method produces better results in 40 instances with up to 130 times less computational effort. We also note that instances with $p = 8$ are easier than instances with $p = 4$. In fact, in all cases, column cover equal to zero for $p = 8$, this explains why solution quality is better for $p = 8$ (Tables

4 and 5). Finally, let us call attention to the fact the label density is very high in populated areas. This can be seen in Fig. 13 and explains why the % of labels without conflict decreases rapidly as label size increases.

6. Evaluating the practical complexity of our approach

Since the problem instances treated up to now have very different structures, a new class of instances was generated in order to evaluate the practical time complexity of our approach. To generate problems that are homogeneous, regardless of their size, we used the following process: labels are of equal size (12×4). The points are distributed uniformly on a square of size $D\sqrt{n} \times D\sqrt{n}$, where $D = 10$ is a constant. For $p = 2$ and for various n between 33 and 10^7 one instance was generated and solved with Pop(10), Pop(70) and Pop(asc) without reduction. In addition,

Table 5
Experimental results for variants of algorithms Pop(asc) and Tabu(50n) on new PFLP instances

H	L	p	Pop_PFLP(asc)					Tabu(50n)				
			%	f(S)	c(S)	$\bar{c}(S)$	Time	%	f(S)	c(S)	$\bar{c}(S)$	Time
2	24	4	93.10	911	982	983.57	62.03	92.20	1030	1076	1077.57	92.68
		8	97.61	315	318	320.19	82.24	97.13	379	394	396.17	107.29
3	16	4	89.18	1429	1608	1609.75	96.30	88.29	1546	1724	1725.75	205.19
		8	93.43	867	912	914.45	168.12	92.54	985	1056	1058.42	363.41
4	12	4	87.12	1701	1956	1957.84	145.47	86.12	1833	2092	2093.83	276.59
		8	94.09	780	832	834.67	184.05	92.87	941	1014	1016.63	348.56
6	8	4	84.87	1998	2346	2347.89	167.87	83.80	2140	2554	2555.89	399.60
		8	92.38	1006	1082	1084.84	220.27	91.40	1136	1216	1218.83	454.16
2	32	4	85.26	1947	2158	2159.87	162.18	84.25	2080	2310	2311.88	322.56
		8	93.47	863	872	874.79	175.62	92.44	999	1062	1064.74	416.74
4	16	4	75.92	3180	3936	3938.23	339.30	74.31	3392	4248	4250.27	849.92
		8	85.85	1868	2086	2089.55	486.72	84.56	2039	2238	2241.51	1043.46
8	8	4	71.76	3729	4964	4966.39	593.19	69.23	4063	5634	5636.44	1225.38
		8	82.23	2347	2744	2747.91	720.32	80.54	2570	3020	3023.88	1431.32
16	4	4	75.15	3282	4098	4100.24	335.84	73.21	3538	4466	4468.26	939.22
		8	85.39	1930	2160	2163.59	517.12	83.95	2119	2376	2379.59	1119.59
3	24	4	74.81	3327	4220	4222.23	373.63	73.00	3565	4582	4584.29	924.09
		8	81.84	2398	2776	2779.61	604.55	80.57	2566	2932	2935.59	1390.86
4	18	4	70.33	3918	5068	5070.43	442.18	67.47	4296	5862	5864.49	1245.52
		8	81.29	2471	2846	2849.99	708.57	79.84	2662	3082	3085.98	1489.45
6	12	4	67.11	4344	5890	5892.55	635.40	63.56	4812	6848	6850.67	1702.11
		8	78.45	2846	3422	3426.30	1017.04	76.58	3093	3736	3740.33	1890.08
8	9	4	65.62	4540	6294	6296.61	784.77	62.01	5017	7364	7366.70	1969.58
		8	76.00	3169	3924	3928.33	972.40	74.36	3386	4308	4312.39	2219.83
2	42	4	74.54	3362	4134	4136.24	370.31	73.19	3540	4448	4450.28	889.66
		8	85.70	1889	1974	1977.61	576.91	84.55	2040	2128	2131.56	999.98
3	28	4	67.34	4313	5826	5828.52	431.65	64.44	4696	6736	6738.61	1587.63
		8	75.09	3289	4000	4004.17	1034.79	73.63	3482	4282	4286.16	2155.98
4	21	4	63.03	4882	6840	6842.71	618.81	61.95	5025	8428	8430.71	2221.33
		8	74.20	3407	4192	4196.59	1137.65	72.57	3623	4510	4514.60	2322.39
6	14	4	59.21	5387	7906	7908.89	670.03	58.23	5516	9924	9926.99	3030.45
		8	70.82	3854	4880	4884.94	1660.72	68.35	4180	5546	5551.02	2852.81
2	48	4	68.82	4118	5268	5270.42	479.02	67.07	4349	5730	5732.49	1229.80
		8	80.70	2549	2778	2782.05	889.74	79.22	2744	2956	2960.01	1445.53
3	32	4	60.87	5167	7388	7390.81	659.05	60.02	5280	9248	9250.83	2543.14
		8	69.07	4085	5250	5254.72	1810.27	66.96	4363	5732	5736.79	2976.49
4	24	4	56.75	5711	8576	8579.03	625.10	56.10	5797	10,770	10773.06	3342.41
		8	68.89	4108	5344	5349.20	1901.56	65.53	4552	6120	6125.23	3128.55
6	16	4	52.23	6309	10,066	10069.19	762.60	51.89	6354	12,498	12501.32	4327.42
		8	64.52	4686	6438	6443.68	1899.16	58.98	5417	8166	8171.85	4961.51

All values were obtained by running these algorithms using $c(S)$ as the objective (also for the $\bar{c}(S)$ column).

several of these instances were solved with Pop(10) considering $p = 4$ and $p = 8$. A computer with 6 GB RAM (Pentium Xeon 3.2 GHz) was used to avoid swapping when solving large instances. The speed of this computer is similar to the Pentium M Centrino used in the previous sections. To give an idea of the difficulty of the instances, solutions found with all POPMUSIC variants have about 32% (respectively: 63% and 90%) of labels that can be placed without overlap for $p = 2$ (respectively: $p = 4$ and $p = 8$). These proportions remain sensibly the same, whatever the problem size, meaning that the quality of the solutions does not degrade as problem size increases. So, problem instances with $p = 2$ appears to be more difficult to solve than instances with $p = 4$ or $p = 8$. This just demonstrates the fact that the same set of points is easier to label when the number of possible label positions increases.

Fig. 14 shows the evolution of POPMUSIC computational time as a function of n . The time does not include the production of the initial solution. This time is only a small % of the total computational time. Fig. 14 shows that POPMUSIC computational time grows almost linearly with n as predicted. Computational time increases less than linearly with p due to the decrease in problem difficulty.

Exponential regressions, conducted on problem sizes $n \geq 1000$ ($n \geq 3162$ for Pop(asc)), provide expected computational time modeled as $c_1 \cdot n^{c_2}$, where c_2 is a constant value that can be estimated between 1.02 (for Pop(10), $p = 8$) and 1.12 (for Pop(asc), $p = 2$). In Fig. 14, the lines are extrapolated computational time while measures are represented by geometrical patterns. Purposely, we have generated only one problem instance for each n and executed only one run for each instance. This allows putting

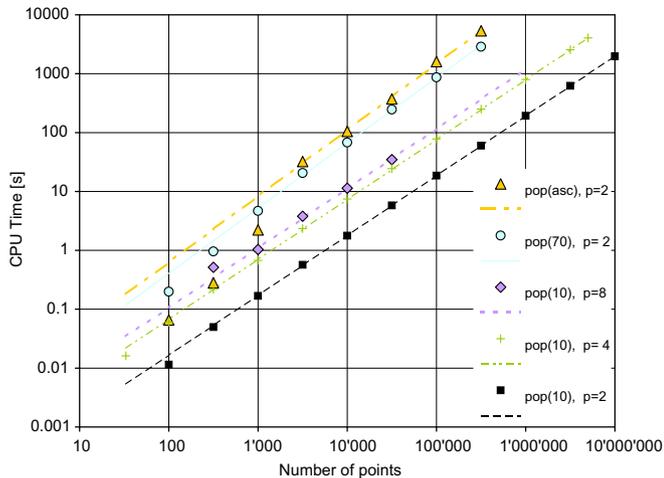


Fig. 14. CPU time as a function of the number of points (n) for various POPMUSIC variants and various p . Geometrical patterns are measures. Lines are exponential regressions.

in evidence the variations of computational effort. We see in Fig. 14 that the variations are negligible as soon as the size of the problem instance is over a few thousand. For $\text{Pop}(\text{asc})$, the increase in computational time is almost quadratic for small problem instances.

7. Conclusions

This article exploits the POPMUSIC frame within the context of point feature label placement. The results obtained show that POPMUSIC is very efficient for this problem, since it is much faster than the best methods previously published while providing higher-quality solutions. POPMUSIC requires an initial solution as input. The latter can be obtained with a fast constructive method. Another feature that POPMUSIC requires is a basic optimization process that is able to produce a good solution to problem instances of small size. The use of a tabu search previously proposed by other authors was generally convenient for this purpose. Once both of these features are available, the implementation of a POPMUSIC-based algorithm is very easy.

Typically, the time complexity of POPMUSIC grows almost linearly with problem size. So, POPMUSIC is ideal for dealing with large problem instances such as those originating from real life. Therefore, we have proposed a set of instances with more than 13,000 points to label.

We have discovered that the tabu search we have embedded in our POPMUSIC algorithm is not able to deal efficiently with cartographic preferences, even for problem instances of very moderate size. Therefore, a good optimization procedure for labeling with cartographic preferences should be developed in the future.

Acknowledgements

The work of Adriana C.F. Alvim was partially funded by Fundação Carlos Chagas Filho de Amparo à Pesquisa

do Estado do Rio de Janeiro (FAPERJ), Grant E-26/171.215/2006. É. Taillard was partially supported by the strategic research funds of the University of Applied Sciences of Western Switzerland, Grant RCSO/TIC-17912-PAL. We also acknowledge the referees for their comments that have allowed us to improve the quality of the paper.

References

- [1] A.C.F. Alvim, É.D. Taillard, POPMUSIC for the point feature label placement, in: Extended Abstracts of the VI Metaheuristics International Conference, Vienna, 2005, pp. 39–44.
- [2] G. Burri, Éric. Taillard, Problème du placement géographique de labels, in: Rapport de travail de diplôme, École d'ingénieurs du canton de Vaud, 2003.
- [3] J. Christensen, J. Marks, S. Shieber, Placing text labels on maps and diagrams, in: Paul Heckbert (Ed.), Graphics Gems IV. Academic Press, Cambridge.
- [4] J. Christensen, J. Marks, S. Shieber, An empirical study of algorithms for point-feature label placement, ACM Transactions on Graphics 14(1995) 203–232.
- [5] R.G. Cromley, A spatial allocation analysis of the point annotation problem, in: Proceedings of the 2nd International Symposium on Theory of Computing 1986, pp. 151–158.
- [6] J.J. Dongarra, Performance of various computers using standard linear equations software, Linpack Benchmark Report, University of Tennessee Computer Science Technical Report, 2005, CS-89-85.
- [7] M., Formann, F. Wagner, A packing problem with applications to lettering of maps, in: Proceedings of the 7th Annual ACM symposium on Computational Geometry (SoCG 1991), 1991, pp. 281–288.
- [8] L.A.N. Lorena, Problem Instances, Map Labeling. <<http://www.la-c.inpe.br/lorena/instancias.html>> (Last visited on 29.12.2006).
- [9] T. Kato, H. Imai, The NP-completeness of the character placement problem of 2 or 3 degrees of freedom, in: Record of Joint Conference of Electrical and Electronic Engineers, 1138, Kyushu, Japan, 1988.
- [10] Gunnar W. Klau, A Combinatorial Approach to Orthogonal Placement Problems, Ph.D. Thesis, Saarbrueck University, Shaker Verlag, Aachen, Germany, 2002.
- [11] J. Marks, S. Shieber, The Computational Complexity of Cartographic Label Placement. Technical Report [TR-05-91], Center for Research in Computing Technology, Harvard University, USA, 1991.
- [12] G.M. Ribeiro, L.A.N. Lorena, Lagrangean relaxation with clusters for point-feature cartographic label placement problems, Computers an Operations Research, Available online, 2006.
- [13] T.W. Strijk, A.M. Verweij, K.I. Aardal, Algorithms for Maximum Independent Set Applied to Map Labelling. Technical report UU-CS-2000-22, Institute of Information and Computing Sciences, Utrecht University, 2000.
- [14] Éric. Taillard, G. Burri, POPMUSIC pour le placement de légende sur de plans, in: Francoro IV, 2004, pp. 95–97.
- [15] Éric. Taillard, S. Voss, POPMUSIC: Partial optimization metaheuristic under special intensification conditions, in: C. Ribeiro, P. Hansen (Eds.), Essays and Surveys in Metaheuristics. Kluwer Academic Publishers, Boston, USA, pp. 613–629.
- [16] O.V. Verner, R.L. Wainwright, D.A. Schoenefeld, Placing tex labels on maps and diagrams using genetic algorithms with masking, INFORMS Journal on Computing 9(1997) 266–275.
- [17] F. Wagner, A. Wolff, V. Kapoor, T. Strijk, Three rules suffice for good label placement, Algorithmica 30(2001) 334–349.
- [18] A. Wolff, The Map-Labeling Bibliography. <<http://i11www.iti.uni-karlsruhe.de/awolff/map-labeling/bibliography/>> (Last visited on 22.2.2007).
- [19] M. Yamamoto, G. Camara, L.A.N. Lorena, Tabu search heuristic for point-feature cartographic label placement, GeoInformatica 6(2002) 77–90.

- [20] M. Yamamoto, G. Camara, L.A.N. Lorena, Fast point-feature label placement algorithm for real time screen maps, in: Presented at VII Brazilian Symposium on GeoInformatics (GEOINFO 2005), Campos do Jordão, Brazil, November, 2005.
- [21] M. Yamamoto, L.A.N. Lorena, A constructive genetic approach to point-feature cartographic label placement, in: Presented at The Fifth Metaheuristics International Conference 2003 (MIC 2003), Kyoto, Japan, August, 2003.
- [22] S. Zoraster, The solution of large 0–1 integer programming problems encountered in automated cartography, *Operations Research* 38 (5)(1990) 752–759.