Algorithms for Map Labeling

Sergei Bezrukov

1 The problem

Assume that we are given a set of points on a 2-dimensional plane along with a rectangle associated with each point. The points represent some objects on a map, such that cities or light houses, etc. The labels usually represent some text assigned to a point, such that name of the city, altitude of a hill, period and parameters of a light house flash. We model a label by a rectangle and throughout the paper assume that a label is nothing else as a rectangle of a given size.

Furthermore, it is usually assumed that the labels can appear on a specific positions around the point object. Usually, there is a number of discrete positions which have weights according to a preference of putting a label at that position. The goal is to place as much labels as possible so that the labels do not overlap.

2 Preliminaries

We will be dealing with the following label positioning models:

2-position model: In this model, a label of each point can be placed in one of two given positions. Examples of the label placement are shown in Fig. 1.

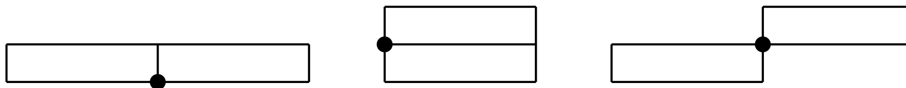

Figure 1: Examples of label placement in the 2-position model

4-position model: In this model a label can be placed in one of the 4 positions around the point object shown in Fig. 2.

For each of these models, the positions may get some weight according to the label placement preferences. The weights may be just integer numbers in the range [0..4]. The smaller numbers correspond to higher priorities. The value 4 of the weight corresponds to a not shown label.

We will be using the *intersection graph* defined below. The vertex set of the intersection graph is formed by the point labels. Thus, if N points have to be labeled, this graph has $2N$ vertices for

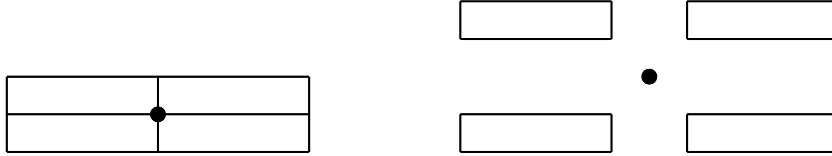

Figure 2: Label placement in the 4-position model

the 2-position model and $4N$ vertices for the 4-position model. Two vertices of the intersection graphs are adjacent if the corresponding labels overlap.

Since for our application about 400 points have to be labeled, the intersection graph can be constructed by using a trivial approach: for each label L scan all other labels and check whether they intersect with L. This results in an $O(N^2)$ running time, which for the considered range of N is acceptable.

3 Label placement for the 2-position model

For this model the placement algorithms are particularly fast and simple. We associate a Boolean variable x_i to each point p_i and arbitrarily assign the literals x_i and $\overline{x_i}$ with its label positions. Now, if two label placements corresponding to the literals, say x_i and $\overline{x_j}$ overlap, we form a clause

$$\overline{(x_i \wedge \overline{x_j})} = \overline{x_i} \vee x_j.$$

This gives us a set of clauses $C_1, \ldots, C_m$. Joining them together by using the logical AND operation results in a conjunctive normal form (CNF) $C = C_1 \wedge C_2 \wedge \cdots \wedge C_m$.

The problem of maximization of the number of displayed labels is now equivalent to the problem of finding a truth assignment to the boolean variables $\{x_i\}$ that satisfies the maximum number of clauses in the CNF C. A clause is called satisfied if its truth value in the assignment is TRUE. This problem is known as a MAX-2-SAT problem. The decision version of this problem (that is to check whether or not it is possible to display all the point labels) is polynomially solvable. The maximization version is, however, NP-hard. Nevertheless, there exist fast polynomial approximation algorithms for the MAX-2-SAT problem, see [1] for a survey. Below we describe the approximation algorithm of Johnson.

The algorithm uses a greedy approach and at each step selects a literal that occurs in the maximum number of clauses. If the literal corresponds to the Boolean negation of some variable x_i, then set x_i =FALSE, otherwise set x_i =TRUE. The clauses satisfied by this literal are then deleted from the CNF and the algorithm stops when all clauses are satisfied or all variables have been assigned values.

Input: A set of clauses $C = \{C_1, \ldots, C_m\}$.
Output: Truth assignment U.

//For a literal l, denote by $u(l)$ the corresponding variable

Set $S = \emptyset$, Left $= C$, $V = \{$ set of variables$\}$
do

Find l with $u(l) \in V$, which is in max. number of clauses in Left.
Resolve the ties arbitrarily.
Let $\{C_{l_1}, \ldots, C_{l_k}\}$ be the clauses in which l occurs.
Update $S = S \cup \{C_{l_1}, \ldots, C_{l_k}\}$ and Left = Left $\setminus \{C_{l_1}, \ldots, C_{l_k}\}$.
Assign the truth value to $u(l)$ as mentioned above.
Update $V = V \setminus \{u(l)\}$.

until no literal l with $u(l) \in V$ is contained in any clause of Left.
if $V \neq \emptyset$ **then for** all $u \in V$ **do** $u =$ TRUE
return $U =$ set of truth values $\{u(l)\}$.

In our case, when each clause consists of exactly 2 literals, this algorithm provides approximation rate 2/3. If some other map objects would prevent placing one of the two point labels, then some of the clauses in the CNF C would consist of one variable. In this case the approximation rate of the Johnson algorithm is 1/2.

4 Label placement for the 4-position model

We apply the following gradient-descent approach, which would also work for the 2-placement model. A general algorithm description can be found in a survey [2], we extend this description by providing more implementation details.

We assume that one is given n points $p_1, \ldots, p_n$ to be labeled, each by one of 4 rectangles l_{i_j} ($1 \leq i \leq n$, $1 \leq j \leq 4$) from the set $r_1, \ldots, r_{4n}$ (see Fig. 1) of all point labels.

Input: A set of points $\{(x, y)\}$ to be labeled.
Output: Label assignment.

For each point, place its label randomly in any of the available candidate positions
(alternatively, put it into the highest preferable position)
Compute the initial label placement cost.
do

Set $m = 10 * n$; $i_{min} = 0$
for each point p_i **do**

move its label to each of the alternative positions
for each position, calculate the change in the placement cost
Let delta$_i$ be the minimum cost of repositioning the i-the label
if (delta$_i < m$) **then** $m =$delta$_i$ and $i_{min} = i$

Reposition the label of $p_{i_{min}}$
Recompute the new placement cost

while no cost improvement occurs

4.1 Initialization

The initial label placement cost can be evaluated during the construction of the intersection graph. We assume that the set of rectangles (bounding boxes for labels) are ordered.

We assign a cost to each point p_i according to its label position and possible intersection of this label with other labels. The label placement cost c_i is a number in the range [0..4]. If the label of p_i intersects another label, we additionally add 5 to c_i. This way we achieve the property that positioning of a label so that it intersects another label leads to a larger placement cost.

We maintain a binary characteristic vector $\mathbf{v} = (v_1, \ldots, v_{4n})$ of a placement (initialized to $(0, \ldots, 0)$), where an entry is one iff the corresponding rectangle (label position) is actually used in the placement.

Input: A set of rectangles $\{r_1, \ldots, r_{4n}\}$.
Output: Characteristic vector $\mathbf{v}$ and cost c of the placement.

Set $c = 0$ and $\mathbf{v} = (0, \ldots, 0)$.
for $i = 0$; $i < n$; i++
    select (randomly) one of 5 possible positions $j \in [0..4]$ for a label of p_i
    update the corresponding entry of $\mathbf{v}$
    update $c = c + c_i$
    for $j = 0$; $j < n$; j++
        if $(r_i \cap r_j) \neq \emptyset$
            form an edge (i, j) in the intersection graph
            if $(v_i \neq 0$ && $v_j \neq 0)$
                update $c = c + 5$

Since we will need to access the neighbors of a vertex in the intersection graph, it is beneficial to represent this graph in the adjacency list form.

4.2 Moving a label and recomputing the replacement cost

First, we remove a label and recompute the cost change by visiting all labels that it intersects (if any). Next, we make a loop over all four label positions of p_i, for each of them compute the resulting change in the cost placement and take the minimum.

Input: A set of rectangles $\{r_1, \ldots, r_{4n}\}$ and index i of repositioning label
Output: A new label position i' and change delta$_i$ of the replacement cost $i \mapsto i'$.

Set delta$_i = -c_i$ and $i' = 0$
for every neighbor r_j of r_i in the intersection graph **do**
    if $(v_j \neq 0)$ **then** delta$_i$ = delta$_i - 10$
Set min$= 10 * n$
for every label position k of p_i **do**
    $d = c_k$
    for every neighbor r_j of r_k in the intersection graph **do**
        if $(v_j \neq 0)$ **then** $d = d + 10$
    if ($d <$ min) **then** min $= d$ and $i' = k$
Update delta$_i$ = delta$_i$ + min

4.3 Updating the placement and its cost

At this step the index i_{min} and a new position i'_{min} of the point label is used to actually modify the label placement. The corresponding label is moved to a new place, followed by updating the characteristic vector $\mathbf{v}$ and the placement cost c.

Input: Indices i_{min} and i'_{min} and the cost difference delta$_i$
Output: Update of the characteristic vector $\mathbf{v}$ and the placement cost c.

for every neighbor r_j of $r_{i_{min}}$ in the intersection graph **do**
    if $(v_j \neq 0)$ **then** $c = c - 10$
Update $c = c - c_{i_{min}} + c_{i'_{min}}$
Update $v_{i_{min}} = 0$ and $v_{i'_{min}} = 1$
for every neighbor r_j of $r_{i'_{min}}$ in the intersection graph **do**
    if $(v_j \neq 0)$ **then** $c = c + 10$

The entire gradient algorithm is over if after a run over all points no improvement caused by replacement of a single label occurs. In our terms, we do the loop while delta$_i < 0$.

4.4 Local optimization

One can add a local optimization at each step of the algorithm. The optimization consists of checking each not labeled point and trying to find an admissible position for it. This feature allocates up to 20 labels which were originally not shown (out of 150 objects to be labeled).

Optionally, one can do even a deeper optimization: for each position of a not shown label l by trying to reallocate the labels of those points whose current labels intersect l. In the implementation we do this optimization at the very end of the procedure to save the running time. In our tests this optimization allocated 1-3 labels (out of 150 objects to be labeled).

4.5 Not overlapping objects

In many practical applications some objects on the map must not be overlapped by labels. We call such objects forbidden objects. To extend the algorithm to take such objects into consideration, just a few changes are necessary.

First of all, assuming there are m forbidden objects and each of them is a rectangle, we extend the characteristic vector of the placement by adding to it m entries. These entries will be initially set to 1 and won't be modified in the coarse of the algorithm.

Next, building the intersection graph we additionally check for intersection the i-th label ($i = 0, \ldots, 4n-1$) and the j-th forbidden object ($j = 0, \ldots, m-1$). If they do intersect, we add the number $4n + j$ into the adjacency list of the i-th label. Note that the adjacency graph will still have only $4n$ lists (one for each label), since we never check the neighbors of a forbidden object. With these additions, no further modifications of the algorithm are required.

5 Simulated annealing method

Simulated annealing is a well-known method for solving combinatorial optimization problems [5, 6]. The method consists of the following general steps.

1. For each point object, place its label randomly in one of the admissible positions for that object
2. Initialize a "temperature" T to an initial high value
3. Repeat until the rate of improvement falls below a given threshold
   a. Decrease T according to the annealing schedule
   b. Pick a point randomly and move its label to a new position randomly chosen from the set of admissible positions
   c. Compute the change of the cost function ΔC
   d. If the new positioning is worse, accept it with the probability $\exp(-\Delta C/T)$

The method strongly depends on the annealing schedule, that is the way how and when to decrease the temperature T. As experiments show, this method requires several dozens of thousand iterations before producing a solution of appropriate quality. Therefore, fast recomputing of the cost function is very desirable.

We use the same as above data structure (the intersection graph) and placement costs in the implementation of this method. So, all one needs to specify is the steps 2 and 3a of the algorithm.

5.1 Initialization of the Temperature

On the first 50 passes through the annealing schedule (that is, on the first 50 random choices of a label to reposition without modifying the temperature), we make the algorithm accept any non-improving move with probability 1/3. At the same time the sum of the non-improving Δ changes of the cost function is encountered during the 50 passes is recorded and used to compute an average value for non-improving $\overline{\Delta}$. The program then sets up T so that the probability of acceptance of the worse placement is 1/3 for the average value of $\overline{\Delta}$:

$$T = \overline{\Delta}/\ln 3.$$

From then on, T is reduced according to the schedule given below.

5.2 The Annealing Schedule

Given an initial value for the temperature T, the most commonly used annealing schedule is to compute new values of T from the old values as follows:

$$T_{new} = \lambda \cdot T_{old}, \qquad 0 < \lambda < 1.$$

Assuming the process does not terminate earlier due to some other criteria, we let 50 reductions of temperature with $\lambda = 0.9$. For each value of T, the program makes a number of random perturbations of the solution which is proportional to the number of label placement positions.

Usually this proportionality factor is set to 10. Finally, if four successive reductions of the system temperature do not produce a better solution, the annealing process is stopped without finishing the normal 50 reductions.

5.3 The entire Algorithm

1. **for** $i = 0$; $i < n$; $i{+}{+}$ **do**
   Generate an integer random number in the range [1..5]
   and place the label for the i-th point in the corresponding position
2. Initialize $\overline{\Delta} = 0$ and $b = 0$
   for $i = 0$; $i < 50$; $i{+}{+}$ **do**
      Generate an integer random number r in the range $[1..n]$
      Generate an integer random number t in the range [1..5]
      Compute the change of the cost Δ of placing the label of
         point r in position t
      if $(\Delta < 0)$ **then** accept the change
      else
         $\overline{\Delta} = \overline{\Delta} + \Delta$ and $b{+}{+}$
         Generate a random real number p in the range [0..1]
         if $(p \leq 1/3)$ **then** accept the change
   if $(b > 0)$ **then** $\overline{\Delta} = \overline{\Delta}/b$
   Set $T = \overline{\Delta}/\ln 3$
3. **if** $(T > 0)$ **then for** $i = 0$; $i < 50$; $i{+}{+}$ **do**
   $T = T * 0.9$
   for $j = 0$; $j < 50$; $j{+}{+}$ **do**
      Generate an integer random number r in the range $[1..n]$
      Generate an integer random number t in the range [1..5]
      Compute the change of the cost Δ of placing the label of
         point r in position t
      if $(\Delta < 0)$ **then** accept the change
      else
         Generate a random real number p in the range [0..1]
         if $(p \leq \exp(-\Delta/T))$ **then** accept the change

5.4 Hill climbing

This method is a special case of the annealing schedule when the temperature is always zero. In this case changes in label placement are made randomly and are only kept if they give no deterioration to the solution. If a problem is not too complicated, this method performs surprisingly well. The rate of improvement is very high at the start of the algorithm.

6 Experiments with algorithms

The Gradient Descent method and Simulating Annealing method were implemented in C++ programming language. Up to 150 objects were randomly placed. The size of each object

was chosen 5×5 pixels. The label of each object was of the form *label* x, where x is the object number. Hence, the labels had the same height but different widths in the experiments. However, the label and object class constructors in the implementation allow to set up different sizes for each label as well as different size for each object. Every other object was set up as a forbidden one.

With the settings describe above both methods provide comparable results. The advantages of the first method is its simplicity, speed, and ability to work with various types of data (labels/objects) without any adjustments. The only disadvantage is that it always chooses one of the best ways to relabel the objects at each step and, hence, is unable to get out of a local optimum.

The simulating algorithm has a chance to get out of a local optimum due to some stochastic featured in its implementation. However, this method required a fine tuning for real data. The number of iterations for each temperature and the number of times to decrease the temperature (the annealing schedule) strictly depends on the data. It requires many experiments to work out the good settings for the data to work.

Both methods running time is about 1 second. Most of this time is required to redisplay the label positioning after the placement is computed. Both methods decrease the initial cost from the range of 800-900 (for 150 random objects) for the initial random placement down to $\approx$100-150. Around 70% of labels in all tests were placed on their preferred positions, in average 8 labels were not shown.

7 Genetic Algorithms

This approach also brings up an element of randomness to the optimization process and gives it a chance to escape from a local minimum. The approach emulates the Darwin's Evolution Process, which briefly consists of the following.

Assume there is a population of some species whose (useful) properties are defined by genes. There is, however, not known what exactly genes and their combinations influence on the desired properties. During the population lifetime, some of the species can produce siblings and, thus, a new population is created. In this process, the set of sibling's genes is combined from the ones of their parents. It is also assumed that there is a way to evaluate the siblings with respect to the desired properties. Then, from the set of parents and their siblings, one leaves only those whose desired properties are stronger, to make the population size finite and bounded.

The siblings can also be selected by combining the genes of their parents. There are various approaches to combine the genes. One of the most widely used one is called crossover. In this approach a pair of parents produces just two siblings. Modeling the genes sequence (whole length is the same for all species) by a one-dimensional vector, one chooses randomly a set of genes S. One of the siblings gets then the genes from the set S from one parent and the remaining ones from the other parent. The other sibling gets the complementary set of genes. In some models only a few genes are allowed to be modified in both siblings. This is provided by designing a suitable mask and called focusing.

Additionally, some parent genes in both siblings can be randomly modified with a small probability. This process is called muting. After several populations, the set of species will be

stronger in the desired property.

This is just a vague idea. Translated to the problem of labeling the point sets, the population consists of several label placements (usually several hundreds). Each label placement has to be stored in computer memory and evaluated against a fixed cost function. One step of iteration procedure (which results in a new population in the above description) consists in choosing two label placements from the set and applying to them the crossover and muting operations. Out of 4 obtained labelings produced this way, one chooses only 2 which have smallest cost, and replaces the original two labelings in the population with the obtained ones. The genes in our problem are simulated by the positions of the labels for each point, so the number of genes is equal to the number of points.

The step above leads to a new population (of the same size) with the property that the average size of the placement cost tends to be smaller. After several thousands of evolutions (iterations) one chooses a labeling that has a minimum cost from the pool.

The method is very sensible to the process parameters. Crossover is just one of the techniques used. Its implementation also varies and there are many approaches known in the literature [3]. There are also various techniques to muting, and the evolution steps are usually followed by a local optimization around each point, which essentially influences on the running time and the quality of the final placement. There are also several approaches to obtaining an initial population with a sufficient diversity.

The genetic algorithms provide a very high quality map placement. However, as the comparison results in [3] show, their quality is comparable with the one provided by the simulation annealing algorithms. The running time is much higher and usually for the conversion of the method one needs to do several millions intersection tests. These algorithms require a very precise tuning to the considered placement model and cautious choosing of the cost function.

References

[1] R. Battiti and M. Protasi, Approximate Algorithms and Heuristics for MAX-SAT, in *Handbook of Combinatorial Optimization*, vol. 1 (1998), 77–148, Kluwer Academic Publishers.

[2] J. Christensen, J. Marks, and S. Shieber, An empirical study of algorithms for point-feature label placement, *ACM Transactions on Graphics*, vol. 14, no. 3 (1995), 203–232.

[3] S. van Dijk, D. Thierens, and M. de Berg, Robust genetic algorithms for high quality map labeling, *Technical Report* UU-CS-1998-41, Department of Computer Science, Utrecht University, 1998.

[4] M. Formann and F. Wagner, A Packing Problem with Applications to Lettering of Maps, in *Proceedings of the 7th ACM Symposium on Computational Geometry*, ACM Press (1991) 281–288.

[5] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by Simulated Annealing, *Science* vol. 220 (1983), 671–680.

[6] S. Zoraster, Practical results using simulated annealing for point feature label placement, *Cartography and GIS* vol. 24, no. 4 (1997), 228–238.