JOURNAL OF SPATIAL INFORMATION SCIENCE
Number 2 (2011), pp. 3–28
doi:10.5311/JOSIS.2011.2.4

RESEARCH ARTICLE

Optimizing map labeling of point features based on an onion peeling approach

Wan D. Bae[1], Shayma Alkobaisi[2], Sada Narayanappa[3], Petr Vojtěchovský[4], and Kye Y. Bae[5]

[1]Department of Mathematics, Statistics, and Computer Science, University of Wisconsin-Stout, Menomonie, WI 54751, USA
[2]Faculty of Information Technology, United Arab Emirates University, Al-Ain, UAE
[3]Jeppesen Inc, Englewood, CO 80112, USA
[4]Department of Mathematics, University of Denver, Denver, CO 80208, USA
[5]DigiPen Institute of Technology, Redmond, WA 98052, USA

Received: August 2, 2010; returned: November 11, 2010; revised: December 30, 2010; accepted: March 17, 2011.

Abstract: Map labeling of point features is the problem of placing text labels to corresponding point features on a map in a way that minimizes overlaps while satisfying basic rules for the quality. This is a critical problem in the application of cartography and geographical information systems (GIS). In this paper we study the fundamental issues related to map labeling of point features and develop a new genetic algorithm to solve this problem. We adopt a method called convex onion peeling and utilize it in our proposed convex onion peeling genetic algorithm (COPGA) to efficiently manage map labels of point features. The proposed algorithm takes advantage of a convex onion peeling structure to achieve better map label initialization and to enhance the evolutionary process. The performance of the proposed algorithm was evaluated through extensive experiments on both synthetic and real datasets. In experiments with an implementation of our algorithm using *OpenMap*, the results show that our genetic algorithm, based on convex onion peeling, is an efficient, robust, and extensible algorithm for automated map labeling of point features.

Keywords: cartography, GIS, computational geometry, automated map labeling, onion peeling, genetic algorithm, simulated annealing, hill climbing

1 Introduction

Placing text labels with corresponding features, i.e., points, lines, or polygons, is a critical task in graphics, cartography, and geographic information systems (GIS). Labels are essential components in a map for delivering information to users. Hence, map labeling needs to avoid label-label and label-point overlaps so that users can easily identify the corresponding label associated with each feature.

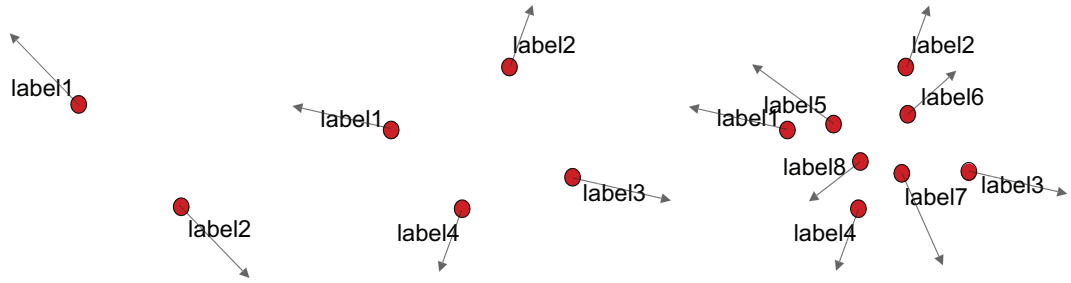
Map labeling makes up a large proportion of map production. Although many solutions have been proposed for automated map labeling, placing labels is still performed manually in many applications. This is because the human brain is better suited to managing the conflicting requirements of map labeling (legibility, ambiguity, aesthetic aspects, and avoiding overlaps) than any automated techniques available today. However, manual resolution of label conflicts is time-consuming and expensive. Hence, developing efficient methods for automated map labeling is a central problem in many applications of cartography and GIS. Automatic map labeling requires a combination of methods from various research areas such as cartography, geosciences, computational geometry, and computer science. Solutions to this problem can also be beneficial to other application domains including graph diagrams, architectural drawings, and medical image analysis.

Map features to be labeled could be points (e.g., cities), lines (e.g., streets), or polygons (e.g., countries). The combinatorial aspect of map labeling should consider all these features in order to provide a proper map labeling method. As a result, solutions to the general map labeling problem are complex. This paper focuses on point features (which are also referred to as sites) by simplifying the general map labeling problem to map labeling of point features. The problem becomes more complex and challenging when the map has background objects. In our paper, we only consider points (sites) and corresponding labels as main constraints.

Due to the high degree of freedom in placing each label, map labeling of point features is a complex combinatorial problem where a search space and a cost function need to be defined. It has been proven that finding the optimal solution to automated map labeling is NP-hard [16, 22]. It is therefore reasonable to resort to approximate solutions based on heuristics. There are three main components to the point-feature map labeling problem: 1) the solving methods, for example, hill climbing [18], simulated annealing [10] or a genetic algorithm [18]; 2) the constraints (completeness), the way label positions are computed; and 3) the quality of initial positioning and repositioning. Our paper focuses on a solution to computing the initial position and repositioning based on a genetic algorithm.

Several studies based on genetic algorithms [18, 26] have discussed map labeling of point features and have shown promising results. However, little research has been done on developing good structures in order to solve the map labeling problem. In this paper, we develop a structure that can efficiently manage map labels to create better initial positions and improve the performance of the evolutionary process in genetic algorithms.

Our approach is based on a simple observation in which labels tend to be placed away from each other in order to avoid conflicts. Suppose that we have two labels, *label1* and *label2*, which need to be placed near to the corresponding points (sites). In Figure 1a, we place *label1* at a left-upper position of its corresponding point, then a right-lower position of the other point might be one of the possible positions for *label2*. Similarly, Figure 1b might be a solution to map labeling placement for four points. The idea is to group the points into layers and start placing labels to corresponding points towards the outer direction of each



(a) Two points (b) Four points (c) Eight points

Figure 1: A heuristic approach to map labeling

layer while keeping the labels away from the other labels and points. If we have a small number of points, achieving this goal seems simple. However, this is not the case in more complicated real world cases. Figure 1c shows an example of eight points where additional techniques need to be considered for placing the labels.

In this paper, we define the search space and a cost function for map labeling of point features. Our cost function is based on two important principles of map labeling: overlap avoidance and unambiguity. To minimize this cost function, we consider the following important aspects of a genetic algorithm:

- Initialization: generating a good initial population (positions of labels) can result in a better solution;
- Selection: exploring constraints vs. non-constraints in the search space of label positions during the evolutionary process; and
- Global optimization: managing the local optimal trap (no improvement after a certain number of evolutions), from which any genetic algorithm may suffer [11].

We propose a genetic algorithm that utilizes convex onion peeling structure [9] to manage point feature labels. The convex onion peeling structure produces better initial map label populations (fewer number of conflicts in the initially generated population) in our experiments, regardless of map labeling algorithms used: hill climbing [18], simulated annealing [10], a genetic algorithm [18] and our proposed genetic algorithm. Hence, convex onion peeling can be also applied independently for improving other existing map labeling algorithms. Adopting the convex onion peeling technique also enhances the evolutionary process of the genetic algorithm, which results in faster convergence to the optimal solution for map labeling of point features. Our proposed algorithm is empirically evaluated, compared to three well-known map labeling algorithms through extensive empirical experiments. The results illustrate that our approach is a robust and efficient solution for map labeling of point features.

The remainder of this paper is organized as follows: related work is discussed in Section 2 and the problem is formally defined in Section 3. The search space and our cost function for the map labeling problem are also defined in Section 3. In Section 4 we present the convex onion peeling structure and our proposed algorithm. The proposed algorithm is experimentally evaluated in Section 5. Finally, Section 6 concludes the paper.

2 Related work

Placing text labels to point features is a well-studied problem and many researchers have proposed various algorithms to solve this problem. While detecting conflicts is the primary goal of a labeling algorithm, it is also beneficial to utilize cartographic knowledge to improve the overall readability. General cartographic principles for map label placement have been defined in [19] and are listed below:

- Legibility: Labels must have legible font sizes and be positioned in a way that is easily read.
- Unambiguity: Each label must clearly identify a single feature and not interfere with any other label or feature. Label positions to the right of a point feature are preferred to those on the left. Labels above point features are preferred to those below.
- Overlap avoidance: A label should not overlap with any other label or feature.
- Aesthetics: Labels should not be overly clustered or distract from map features.

In our paper, we provide a solution to map labeling of point features that minimizes overlaps while satisfying basic rules such as unambiguity, legibility, and aesthetics for the clarity of a map.

Popular solutions to the map labeling problem using a rule-based approach were proposed in [1, 2, 13]. The authors in [31] represented the map labeling problem as follows: each site is represented as a point in the plane and the task is to find a square width w for a set of n squares such that every point is a corner of exactly one square and all squares are pairwise disjoint. Their definition disallows any overlaps between labels. This version of the problem was shown to be NP-hard in [16] and an $O(n \log n)$ running time approximation algorithm was proposed. The approximation algorithm finds a valid labeling of at least half of the optimal size.

Another type of map labeling called boundary labeling was presented in [7]. The problem was to label point sites with large labels placed around the map and connected to corresponding sites through polygonal lines that do not intersect. This problem is a combination of label-placement and graph drawing problems. The authors provided different versions of their algorithms and compared the running time of each.

The hill climbing algorithm [18] is a simple local optimization algorithm. The algorithm begins with an initial, random placement of labels. It then searches through the adjacent new solutions generated randomly from the current solution and selects the optimal solution. The algorithm continues searching for new solutions until no improvement is found from the current solution.

In [10] a heuristic method to the optimization of the map labeling was proposed based on simulated annealing [20]. The quality of the proposed labeling method was quantified using a metric that calculates a scoring function based on the number of conflicts of labels with sites and labels with labels. The scoring function also takes into consideration the label position preference and adds a penalty to the score whenever a label is placed in a non-preferred position. The main advantages of simulated annealing over others are shown to be the simplicity of implementation and the generality of the method that can be applied to any feature (point, line, and area). Another advantage of simulated annealing is that it may escape from local minima.

Genetic algorithms are strategies for function optimization based on genetics and Darwin's theory of evolution. A genetic algorithm works in parallel on a population of can-

didate solutions from the search space [14]. There are many variations; however, most of them suffer from the same difficulty: the local minimum trap [11].

A genetic algorithm for labeling point features was proposed in [26]. Their approach is based on three main phases. The preprocessing phase in which a conflict table is created for the initial positioning of the labels to reduce the search space. Second, the population generation is performed based on a selection, recombination, and mutation evaluation. Finally, each newly generated solution is locally improved before its evaluation. The authors compared their solution to the simulated annealing approach and have shown that their genetic algorithm provided slightly better solutions than simulated annealing. However, it required more CPU time.

Another genetic algorithm for automated map labeling of point feature was proposed in [18]. The authors compared their approach to previously proposed algorithms based on hill climbing, simulated annealing and random algorithms and showed by experimental results that their proposed algorithm outperformed the other approaches in terms of number of conflicts.

A good introduction to genetic algorithms can be found in [30], where the authors apply genetic algorithms to solve different GIS problems including the point-feature map labeling problem. The authors defined the "fitness" measurement as the number of non-conflicting labels, which in their search space includes four fixed locations with each label placed initially randomly. Through experimental results, the authors demonstrated that their proposed algorithm performs as well as simulated annealing. It was also tested on a map of US cities, demonstrating how their proposed algorithm can be modified to take into consideration "soft" constraints (such as top-left preferred position of a label) and deleting labels whenever the map is too crowded.

A comprehensive survey of map labeling algorithms can be found in [11]. The paper discussed the NP-hard nature of the map labeling problem and the exponential time complexity of heuristic solutions. It also proposed two algorithms: one based on discrete gradient descent and the other based on simulated annealing that were empirically evaluated along with other previously proposed solutions. Extensive experimental results showed that none of the map labeling algorithms outperforms the others in all cases. For example, their experiments showed that simulated annealing algorithms should be favored over the other alternatives when solution quality is important.

A method based on the "slider model" for map labeling was presented in [28] in which a label is placed in any position that touches the site to be labeled. Their proposed algorithm offers more flexibility in the search space as opposed to several fixed locations defined for a label position. However, the authors consider line segments (e.g., state boundary) as obstacles that cannot be intersected by any labels. Given n number of points to be labeled and m number of line segments to be avoided, the main contribution is a preprocessing step algorithm that runs in $O((n+m)\log(n+m))$. The authors also extended their solution to support labels of different font sizes.

In [15], the authors provided a label placement approach that can be applied to general cartographic maps that include point, line, and area features. In their presented framework, the authors identified three subtasks for their map labeling algorithm: candidate-position generation; position evaluation (a score that indicates the quality of a label); and finally position selection to choose a label position from a set based on the overall quality of the labeling.

An evolutionary algorithm that supports different user objectives was presented in [8]. Their approach provided a set of solutions based on three main criteria which are: maximizing the font size of the labels; maximizing the clarity of bindings (telling which label belongs to which feature on the map); and minimizing the number of conflicts. The authors illustrated the result of their methods on two maps.

A heuristic approach has been developed recently for finding good solutions to point-feature labeling [3]. The computational complexity of this method grows quasi-linearly with the number of labels and quadratically with the label density.

The authors of [12] developed a rule-based system using Prolog to place names on different types of maps such as road maps and county maps. Their system's rules depend on the spatial relationship between the feature to be labeled and the name (label) to be placed. Thus, the authors have developed a spatial database that their system accesses in order to find non-conflicting positions of the labels. The main drawback of their proposed solution to map labeling is that it is based on exhaustive searching and run time might be unacceptably long. This was solved by either subdividing the problem into subproblems (each consisting of a set of labels to be placed), or by a preprocessing step that sorts the labels in some predefined order.

In [24, 25] the authors solved the dynamic map labeling problem in which maps are affected by continuous zooming (change of state) and panning (change of region of interest) operations. They proposed a two phase solution to the problem: the preprocessing phase and the interaction phase. In the first phase, the authors construct the reactive graph, a data structure that is used to detect conflicts. In the interaction phase, the reactive graph is queried for conflicts and the subset of labels that can be placed without conflicts are chosen in a greedy approach. This work was the predecessor for recent research on dynamic map labeling that was presented in [6]. A framework for dynamic labeling that allows for fast interactive display of labels was proposed. This was achieved by performing all the selection and placement decisions in a preprocessing phase.

Similarly, the authors in [32] presented a solution to real-time map labeling using the slider method on continuous search space to label points and line features. Their proposed method aimed to provide an online solution for mobile devices with small screens. They start with arbitrary positions of labels defined by a fixed four positions search space. Then, the best possible position for a label is identified based on normal cartographic preference where candidate positions are reduced to avoid conflicts. The authors' proposed method was demonstrated through a case study. However, it was not compared to other previously proposed models in terms of efficiency (both visually and computationally).

Our previous work [4] proposed an efficient solution to map labeling of point-features. We adopted convex onion peeling structure in our proposed genetic algorithm to efficiently manage point-features of a map. This paper extends our previous work and provides a comprehensive study of convex onion peeling genetic algorithm. We present a thorough explanation of the use of the genetic algorithm for map labeling of point features. To demonstrate the overall quality of map labeling generated by our algorithm, the algorithm was implemented on top of a real map application.

3 Problem definition: Map labeling of point features

The point-feature map labeling problem consists of a set S of n sites in the plane $S = \{s_1, s_2, ..., s_n\}$ and a set $L = \{l_1, l_2, ..., l_n\}$ corresponding to the sites. Our objective is to place each l_i near the corresponding s_i within a search space while minimizing a cost function.

A solution is a subset $S' \subseteq S$ of sites and a function λ which maps every site $s_i \in S'$ to a label $\lambda(s) \in L$, such that no label conflicts with another label or with a site. A number of labeled sites, i.e., the cardinality of S', is the number of the labels that are successfully placed in a map. An optimal labeling is a solution having the maximum size among all labeling solutions. A complete labeling is a solution where every site receives a label without any conflicts.

This problem can be thought of as a combinatorial optimization problem where a search space and a cost function need to be defined. As mentioned in Section 2, detecting conflicts is not the only goal of a labeling algorithm; utilizing cartographic conventions is also important to improve the overall readability. Hence, we develop methods to minimize overlaps while satisfying other basic cartographic rules for the map quality.

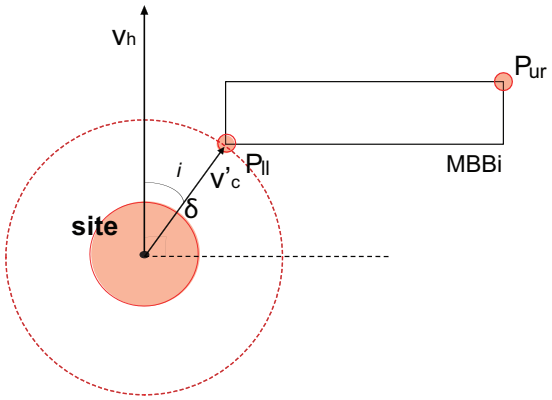
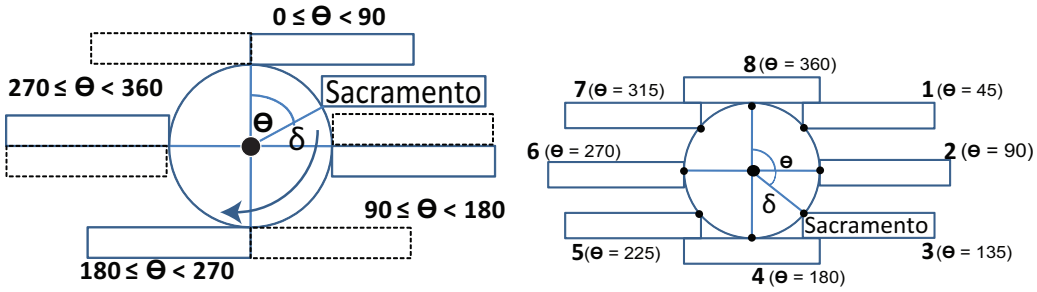


Figure 2: Notations for labels

3.1 Search space

We first discuss basic notations and structures in placing a label to a site that are used in our paper. The minimum bounding box (MBB) [17] of label l_i is the smallest box that encloses text label l_i, and is denoted by MBB_i. MBB_i is represented by two corner points, P_{ll} (lower-left corner) and P_{ur} (upper-right corner). Let δ be the distance between the center of site s_i and the closest point on MBB_i of the label l_i as shown in Figure 2. In this paper, we use a fixed value of δ for all labels. The upright vector v_h from the site and the vector $v_c^{'}$ connecting the site to the closest point on MBB_i create an angle Θ_i.

Next we define the search space for labeling positions and present our cost function. A label candidate is one of many possibilities to place a label for a certain site. We define two different models of the search space: 1) continuous space (use an infinite number of candi-



(a) Continuous search space (b) Discrete search space

Figure 3: Search space for label positions

dates); and 2) discrete space (use a discrete number of label candidates). For the discrete number of label positions we use eight positions in this paper so as to give fair comparisons to previously proposed algorithms. The algorithm could perform better if we increase the number of label candidates. Figure 3a and b show examples of a continuous and a discrete search space, respectively. The two models are used in our proposed algorithm as follows:

- Initialization: The continuous space model is used for generating initial label positions for all sites. For each label l_i, any value between 0 and 360 is assigned as its Θ_i.
- Selection: The discrete model is used for possible positions in the mutation step of the evolutionary process.
- Global optimization: The continuous space model is used for the inverse of the search space in the inversion step of the evolutionary process.

3.2 Cost function

It is only possible to approximate solutions when the problem admits a cost function—that is, a measure of how far the solution is from an optimal solution. The usual scenario is to count the labels that conflict with each other or with sites. Much of the previous work considered two of the main principles of map labeling [19], overlap avoidance and unambiguity, for solving map labeling of point features. For a cost function, there must be some trade-off between the two principles, and the solution might be varied depending on the applications. In our paper, we focus on overlap avoidance while we balance these two principles. Our approach is to provide a solution to map labeling of point features in a way that minimizes overlaps while satisfying basic rules for the map clarity.

The task of the evaluation step for labeling is to detect all conflicts and rate the label candidates accordingly. The evaluation process has to detect the following categories: 1) conflicts of a label candidate with other labels (label-label); 2) conflicts of a label candidate with sites other than the associated site (label-site); and 3) aesthetic preference and tradition. The outcome of this evaluation can be used in a cost function to assign a particular value to each label candidate describing its overall suitability.

Let C be a set of n label costs, $C = \{C_1, C_2, ..., C_n\}$, where C_i is the cost of l_i's position. The parameters for a label cost are defined as follows:

- c_i^l: the number of conflicts of l_i with $\forall l_j \in L \setminus l_i$;
- c_i^s: the number of conflicts of l_i with $\forall s_j \in S \setminus s_i$; and
- p_i: a penalty for preferred location. Positions in the 1^{st} quadrant ($0 \leq \Theta \leq 90$) are considered as preferred positions and the penalty values of these positions are set to 0 and all the rest of the positions have penalty equal to 1.

C_i is then calculated as follows: $C_i = a_1 \cdot c_i^l + a_2 \cdot c_i^s + a_3 \cdot p_i$, where a_1, a_2 and a_3 are constant factors (see below). The cost function for the map labeling is $\sum_{i=1}^{n} C_i$. Hence, we define a cost function $F(S)$ for a given set of sites S as follows:

$$F(S) \quad = \quad \sum_{i=1}^{n} (a_1 \cdot c_i^l + a_2 \cdot c_i^s + a_3 \cdot p_i)$$

We give more value to label conflicts (1.0 and 1.0 are used for the values of a_1 and a_2, respectively) as opposed to unambiguity (0.1 is used for the value of a_3) in our experiments. The main reason for this setup is that we want to reduce number of conflicts quickly by relaxing label positions in the initialization step and the evolutionary process. Then we try to keep a low number of conflicts while we seek a better solution (a greater number of preferred positions).

Of course, different weight values for the penalty on non-preferred positions could be used. This could improve the quality of maps but it might require greater numbers of evolutions. Since the same cost function is applied in all our experiments, the comparisons should be fair to all the algorithms.

To simplify the discussion, we made the following assumptions: 1) sites are all point features; 2) all labels are equally important; 3) text sizes of all labels are the same, although our implementation can support different sizes for label texts.

4 Convex onion peeling genetic algorithm (COPGA)

In this section, we present the convex onion peeling genetic algorithm (COPGA) that adopts a genetic algorithm with the search space and cost function defined in Section 3. COPGA provides a solution to the basic elements of the genetic algorithm by utilizing the convex onion peeling structure.

The COPGA algorithm consists of three main steps: convex onion peeling construction, initial population generation, and evolution. The following is an outline of these steps:

1. Convex onion peeling (COP) construction: Construct the convex onion peeling structure of the sites in a given map (line 2 in Algorithm 1). An example of COP structure is shown in Figure 4.

2. Initial population generation: Generate a population of candidate solutions, where each solution is a vector of labeling positions. Each candidate solution in a genetic algorithm is represented as an individual (or chromosome) of a population [14]. Let $P(t)$ be the population of individuals at generation t, where t is the generation index and $t = 0, 1, 2, ...$ Then the initial population is $P(0)$. $P(0)$ is evaluated using the cost function, and the best candidate solution's cost is returned as the cost of the current population C_{P_t} (lines 3–5 in Algorithm 1).

3. Evolutionary process: This step applies if the evaluation result of $P(0)$ does not satisfy the terminal condition. In the evolutionary process, an offspring population is generated by means of selection and search operators. The main search operators are cross over and mutation [8, 23]. The following describes the evolutionary process:

   3.1 Generate an offspring population $P(t+1)$ from the parent population $P(t)$ using selection methods and search operators (cross over and mutation). There are intermediate populations generated during the evolutionary process: P^1, P^m, P^2, and P^3. If no improvement is made for some number of evolutions, we inverse the search space for mutation to get out of the local optimal trap (lines 6–14 in Algorithm 1).

   3.2 A number of offspring or parents survive this natural selection, and the rest are discarded. The surviving solutions become the new parents for the next generation. This selection process is based on the evaluation of the cost function and the cost C_{P_t} is calculated (lines 15–16 in Algorithm 1).

   3.3 Evaluate the offspring population using the cost function. A comparison of each solution's cost is made using the algorithm's cost function (line 17 in Algorithm 1).

   3.4 Repeat these steps until the termination condition has been satisfied.

The detail of each step of COPGA will be discussed in the following subsections.

Algorithm 1 COPGA(S, L); a set of sites, a set of labels

1: $N \leftarrow 100$; $t \leftarrow 0$ {population size; # of evolutions}
2: $COP \leftarrow$ constructCOP(S)
3: $P(0) \leftarrow$ initializeSites(COP, L, N)
4: $P(t) \leftarrow P(0)$
5: $C_{P_t} \leftarrow$ evaluate $P(t)$
6: **while** termination condition not satisfied **do**
7: $P^1 \leftarrow$ selectForCrossOver(P_t)
8: $P^m \leftarrow$ selectForMating(P^1)
9: $P^2 \leftarrow$ crossover(P^m)
10: **if** local optimal trap condition **then**
11: $P^3 \leftarrow$ mutate(P^2) with inversion
12: **else**
13: $P^3 \leftarrow$ mutate(P^2) without inversion
14: **end if**
15: $P(t+1) \leftarrow$ selectReplacement(P^3, $P(t)$)
16: $P(t) \leftarrow P(t+1)$
17: $C_{P_t} \leftarrow$ evaluate $P(t)$
18: $t \leftarrow t+1$
19: **end while**

4.1 Convex onion peeling construction

The convex onion peeling of a set of points is the organization of these points into a sequence of interpolating convex polygons (a structure that consists of a sequence of nested convex hulls) [9]. This structure on a set S of n points is obtained by the following procedure: compute the convex hull of S, and let S' be the set of points remaining in the interior of the hull. Compute the convex hull of S' and recursively repeat this process until no more points remain. One ends up with a sequence of nested convex hulls, called the convex onion peeling of S. This structure can be obtained in $O(n \log n)$ time [9]. The convex onion peeling technique has been widely used in many application domains, e.g., image processing, pattern recognition, photo image analysis [21], and the study of Earth's atmosphere [27].

COPGA first constructs a sequence of nested convex hulls for the sites (S) in a map (*constructCOP(S)* in line 2 of Algorithm 1). We refer to this structure as convex onion peeling (COP) of the sites. Sites in a map are divided into several groups (nested layers) using the convex onion peeling technique. Figure 4 illustrates an example of constructing a map's COP, where the COP consists of sequence of convex hulls (layers), COP_{map}= {*layer1*, *layer2*, *layer3*}, and the layers are as follows: *layer1* = $\{S_1, S_2, S_3, S_4, S_5, S_6\}$, *layer2* = $\{S_7, S_8, S_9, S_{10}\}$, and *layer3* = $\{S_{11}, S_{12}, S_{13}\}$.

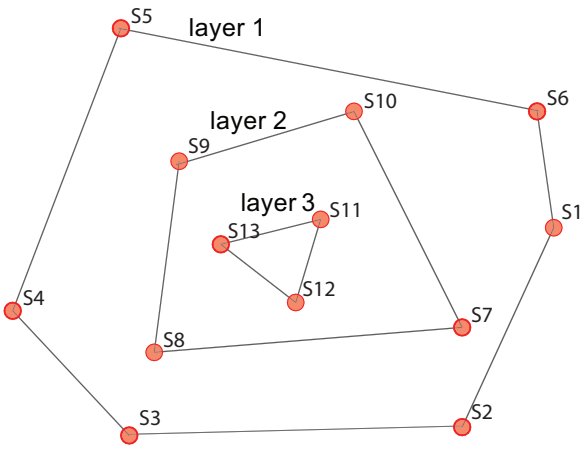


Figure 4: Convex onion peeling construction

4.2 Initial population generation

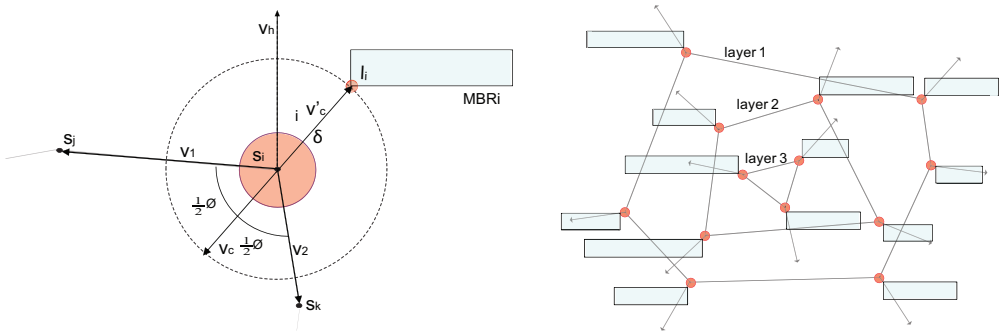
Two well-known initialization techniques are commonly used in many of the proposed algorithms for the map labeling problem of point features: preferred position (upper-right position) and random position. A new initialization technique is used in COPGA and its performance is compared to those of the two existing techniques in Section 5.

COPGA calculates the initial position of each site's label based on the convex onion peeling. Figure 5a shows how to calculate a label's initial position. Each site is associated with

three vectors; v_1 and v_2 are related to the neighbor sites, s_j and s_k, in the layer and v_c is the half of the inner angle ϕ. Then the opposite vector $v_c^{'}$ and the vertical vector of the site v_h determines the angle Θ_i as discussed in Section 3.1. A label's initial position is determined based on the value of Θ_i; an example of the initialization result is shown in Figure 5b.

The convex onion peeling provides an efficient geometric structure for placing labels to reduce the possibility of conflicts by placing labels outside of each layer. It also allows us to solve the problem in a divide and conquer manner; it places the labels to the sites of each layer and combines all label positions for an initial solution to the whole map labeling problem. Convex onion peeling initialization is compared to two other initialization techniques, preferred position and random position. Our experimental results show that onion peeling initialization always results in less conflicts than preferred position and random position.

Each individual solution is a vector of labeling positions corresponding to the sites. The cost of the initial population $P(0)$ is calculated and this is assigned to the cost of the current population C_{P_t}. If C_{P_t} does not satisfy the termination condition, the evolutionary process of COPGA starts (line 5–6 in Algorithm 1).



(a) Calculate a label position (b) Generate initial population

Figure 5: Label positions using convex onion peeling

4.3 Evolutionary process

In the evolutionary process, the new generation $P(t+1)$ is obtained from the population $P(t)$ by means of the following steps: 1) selection for cross over; 2) cross over; 3) mutation and inversion; 4) selection for replacement and survival; 5) evaluation.

4.3.1 Selection for cross-over

The individuals of population $P(t)$ are selected for cross-over according to their costs. Selected individuals represent an intermediate population P^1 (line 7 in Algorithm 1). Then based on a given probability (cross-over probability $p_c = 0.5$), a certain proportion of individuals from P^1 enter the mating pool for cross-over. This mating pool represents another intermediate population P^m (line 8 in Algorithm 1).

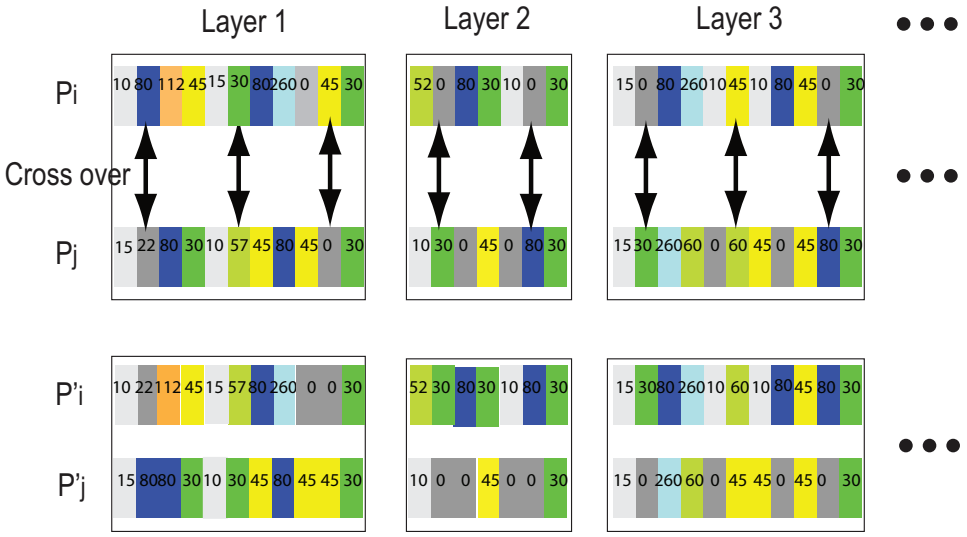


Figure 6: Example of a cross over (a number on each individual represents the value of Θ for each label)

4.3.2 Cross-over

The cross-over operator [14, 23] is used to create new individuals by combining the genetic information of two parents. The individuals from P^m are mated using the cross-over operator (line 9 in Algorithm 1). Cross-over on the pair for mating (two solutions of map labeling) is conducted based on each layer of the convex onion peeling. Our algorithm does not allow cross-over across the layers. Sites are chosen with a probability from each layer of one individual (one solution of map labeling). The corresponding sites in another individual are chosen accordingly. Then two new individuals are created by switching the label positions of the selected sites from the two existing individuals. New individuals from cross-over on P^m are included in P^2. Figure 6 shows an example of the cross-over in the COPGA algorithm. Let P_i and P_j be the chosen two parents (two chosen individuals). Sites in each layer of the onion peeling of the two parents are selected based on the proportion. From P_i and P_j, the algorithm selects the second, sixth, and 10th sites in *layer1*, the second and sixth sites in *layer*2, and second, sixth, and 10th sites in *layer*3. Two new individuals, $P_i^{'}$ and $P_j^{'}$, are generated by exchanging the labels' positions of these chosen sites. For example, the second site in *layer1* of P_i and the second site in *layer1* of P_j have 80 and 22 as the Θ values of their label positions, respectively, before the cross-over. The cross-over process switches these two positions (Θ values). As a result, the Θ value of the second site's label in *layer1* of the newly generated individual solution $P_i^{'}$ is set to 22. Similarly, the Θ value of the second site's label in *layer1* in another individual solution $P_j^{'}$ is set to 80.

4.3.3 Mutation

The mutation operator [14, 23] generates new individuals by variations (labels' position changes) of a single individual with the probability of mutation $p_m = 0.1$. Let P^3 be the population obtained by applying mutation and, possibly, inversion to P^2 (line 10–14 in Algorithm 1). The selected label's position is replaced with a new position.

Considering all eight possible locations shown in Figure 3b, this change could cause more conflicts because most labels tend to be outside of the convex hull. Hence we modify the search space by removing the angle ϕ resulting in only positions outside of the convex hull. No significant effect on the freedom of the search space is found since the convex hull always creates outer angles greater than $180°$. Hence there are always at least four possible label positions available. Figure 7 shows an example for the search space by mutation. In this example, the positions 4, 5, and 6 are removed from the candidates of the positions. We also investigate another approach, a greedy approach, for the selection of mutation. All conflicted labels are retrieved and random selection for mutation is conducted on these conflicted labels.

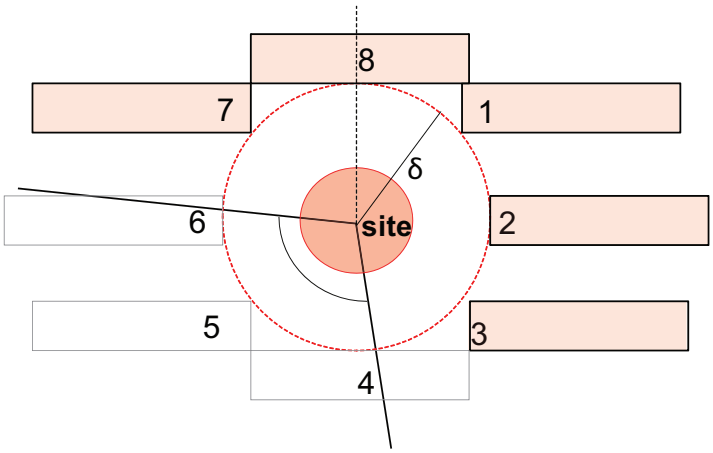
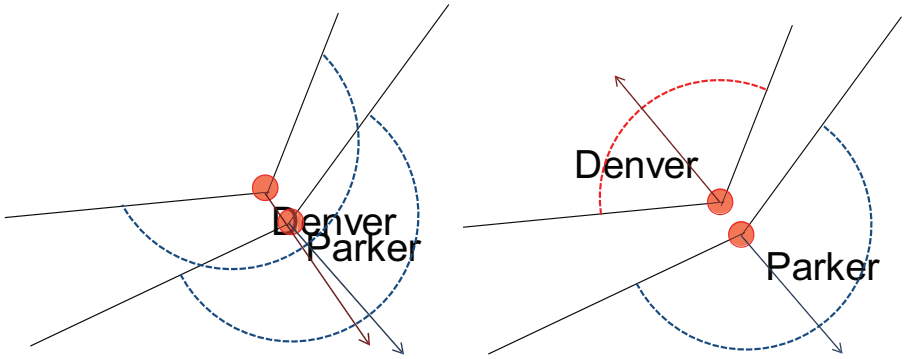


Figure 7: Mutation

4.3.4 Inversion

Convex onion peeling does not work well when the sites in the adjacent layers are too close, resulting in labels tending to be shifted in similar directions. The methods for initialization, recombination, and mutation are all based on convex onion peeling structure. Hence it may not be easy to resolve this problem. We relax this restriction when the algorithm reaches a possible local optimal trap and no better solution is obtained after a certain number of evolutions ($0.3 *$ number of initial conflicts used in our experiments). In that case, we configure the label positions so that the search space for the conflicting labels is changed to the inner space. Figure 8 shows an example of the unsolved problem and our solution by using the inverse of the search space.



(a) Unsolved conflicts (b) Inverse search space

Figure 8: Unsolved local conflict and inverse space

4.3.5 Selection for Replacement and Survival

The selection for replacement operator is used to obtain which individuals from $P(t)$ and their offspring will effectively enter the new generation $P(t+1)$. The new generation $P(t+1)$ contains the individuals of the population P^3, and may include other selected individuals. COPGA keeps the subset of the individuals from $P(t)$ that have not been selected for cross-over as part of $P(t+1)$.

4.3.6 Evaluation

COPGA then computes the quality (the cost) of each individual of the current population and assigns the cost of the best individual to the cost of the current population C_{P_t}. The evolutionary process continues until C_{P_t} satisfies the termination condition.

5 Performance evaluation

In this section, we evaluate the performance of the convex onion peeling genetic algorithm (COPGA). We first show the results of our initialization using convex onion peeling on three well-known existing algorithms and then present the results of COPGA compared to a previously proposed genetic algorithm (GA) [18].

5.1 Datasets and experimental methodology

In our experiments, we considered both synthetically generated maps and real maps. The number of sites in the synthetic datasets were varied between 40 and 160, and the sites' locations were distributed uniformly and independently. For each size of the synthetic datasets, we randomly generated 100 maps and conducted 100 trials for each experiment. Then the average values were reported. Our real datasets were obtained and extracted from USGS [29]. Table 1 presents the details of the real datasets. Each dataset was scaled

to fit into a given boundary of the map with a size of 650 x 650 pixels. For each dataset, we conducted 100 trials for each experiment and reported the average values.

Real data	Region	Boundary (latitude, longitude)		# Sites
		Lower-left	Upper-right	
Dataset 1	Englewood, CO	(26.00, -122.00)	(48.00, -75.00)	73
Dataset 2	New Jersey, NY	(20.00, -110.00)	(48.00, -43.10)	116
Dataset 3	North America & Europe	(20.00, -100.00)	(68.00, -3.10)	161

Table 1: Real datasets

Although our algorithm supports different font styles and sizes of map labels, we only present the results with the following setup for sites and labels due to space considerations: 1) sites were represented by circles with radius equal to three pixels; 2) the font style was set to "courier," the font size was set to 10, and a random length between six and 15 characters were used for labels; 3) $\delta = 3$ pixels was used for the distance between the center of a site and the closest point on the corresponding label's MBB. Similar qualitative and quantitative trends were observed in all other experiments.

To compare the effect of initialization, we implemented three existing map labeling algorithms, hill climbing (HC) [18], simulated annealing (SA) [10] and a genetic algorithm (GA) in [18] by applying the three initialization methods. The common performance metrics for the map labeling problem are the values of the cost function $F(S)$, the CPU time, and the number of evolutions. However, different algorithms use various implementation characteristics and parameters. Both CPU time and reduction rate were used for the termination condition of HC. For SA, we set the initial temperature to 2.5 and the temperature drop rate was 0.2. These values are the same values used for the SA algorithm in [10]. Normal temperature drop and sudden temperature drop were set to five times the number of sites and one times the number of sites, respectively. As described in [10], these values were chosen primarily to provide reasonable execution times and the parameters of annealing have a relatively minor affect on the performance of the algorithm. To make comparisons to SA, we also set parameters of the GA and COPGA according to the descriptions in [10], which were determined by preliminary tests and found to be robust and well suited for GA and COPGA. We applied inversion after a certain number of evolutions failed to find a new best solution (0.3 times the number of initial conflicts was used in our experiments). Table 2 summarizes the parameters used for each map labeling algorithm in our experiments.

5.2 Results of initialization

First, we present the results of our proposed map initialization method using convex onion peeling compared with two other initialization methods, preferred position (default) and random position.

We conducted the three initialization methods on the synthetic and real datasets. Figure 9 shows the average costs after initialization of synthetic datasets computed using $F(S)$ in equation 1. On average, convex onion peeling initialization resulted in 26.04% and 15.22% reduction over default and random initialization, respectively. In addition, the initialization using onion peeling produced less conflicts than the default and random initialization methods in all cases.

Algorithms	Parameters	
HC	Termination	1) Given CPU time; 2) given reduction rate
SA	Initial temperature	2.5
	Temperature drop rate	0.2
	Normal (sudden) temperature drop	5 times number of sites (1 times number of sites)
GA	Population	100
	Cross-over	Uniform cross-over with $P_c = 0.5$
	Mutation	Random replacement with $P_m = 0.1$ on conflicted labels
	Termination	Cost $= 0$ or 1000 evolutions
COPGA	Population	100
	Cross-over	onion peeling cross-over with $P_c = 0.5$
	Mutation	Onion peeling random replacement with $P_m = 0.1$ on conflicted labels
	Termination	Cost $= 0$ or 1000 evolutions
	Inversion	# Evolutions ($0.3 \times$ # initial conflicts) without finding a new best solution

Table 2: Implementation characteristics and parameters

Figure 10 illustrates the performance of default, random, and convex onion peeling initializations using the synthetic datasets. Figure 10a and b show the comparisons of HC using the three initialization methods. In Figure 10a, we plotted the average conflict costs of HC after 10 seconds of CPU time along with the initial costs. For example, when the number of sites is 110, the final costs with default, random, and convex onion peeling are 6.5, 5, and 2, respectively. The overall cost reduction rates using default, random, and convex onion peeling initializations were 75%, 81%, and 84%, respectively. Figure 10b shows the CPU time to reach 90% conflict reduction rate of HC with different initializations. On average, HC using convex onion peeling required 25.72% and 15.96% less CPU time to obtain 90% cost reduction compared to HC using default and random positions, respectively. The results also show that the larger the dataset is, the greater the advantage in using convex onion peeling over the other methods.

Figure 10c and d show the average final conflict costs and the CPU time of SA using the three initializations. SA using convex onion peeling resulted in 39.31% less conflict cost than SA using default position and 27.71% less final cost than SA using random position on average. The CPU time required for termination was also compared. When the number of sites was 120, the required CPU time was 25, 23, and 19 seconds for default, random and convex onion peeling, respectively. The results show that SA with convex onion peeling required 13.51% and 11.03% less CPU time than SA with default and random initializations. Similar qualitative and quantitative trends were observed in the results of GA using these three initializations. In Figure 10e and f, the average final conflict costs and CPU times of GA with different initializations were plotted. Figure 10f shows that GA with convex onion peeling required 28.20% and 19.70% less CPU time than GA with default and random initializations, respectively.

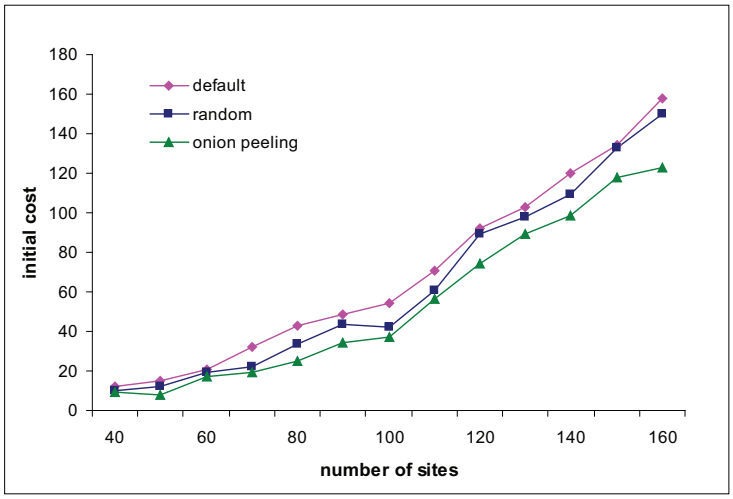


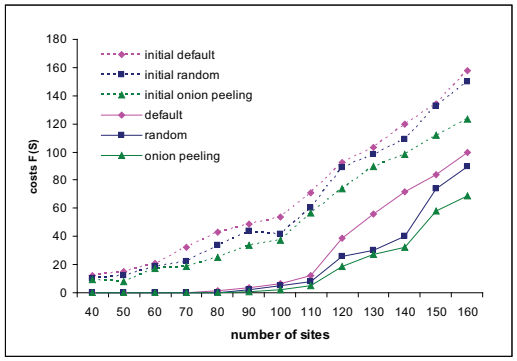
Figure 9: Map initialization

Real data	Initialization Method	Initial cost	HC		SA	GA
			Cost	CPU	Final cost	Final cost
Dataset 1	default	47	0 (100%)	4212	2.5(95%)	0 (100%)
	random	43	0 (100%)	2371	2 (95%)	1 (98%)
	onion peeling	30	0 (100%)	2121	1 (97%)	0.5 (98%)
Dataset 2	default	171	39 (77%)	15226	25 (85%)	21 (88%)
	random	157	33 (79%)	13195	21.5 (86%)	20 (87%)
	onion peeling	135	26 (81%)	12574	20 (85%)	17 (87%)
Dataset 3	default	486	231 (52%)	95005	131 (73%)	125 (74%)
	random	402	201 (50%)	72431	128 (68%)	116 71%)
	onion peeling	346	153 (56%)	60710	121 (65%)	109 (68%)

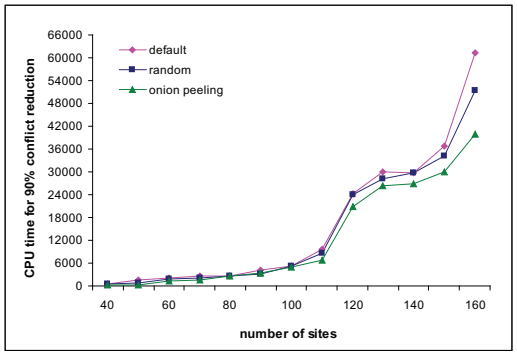
Table 3: Real data result of HC, SA, and GA: numbers in ( ) are reduction rate

Table 3 shows the comparisons of the three initialization methods using the real datasets. For HC, we show the cost after 30 seconds and the CPU time to achieve a 80% conflict reduction rate using the three initialization. For SA and GA, the final costs and CPU time are reported. Overall, the results of the real datasets are similar to those of the synthetic datasets despite the skewed distribution of the real datasets.

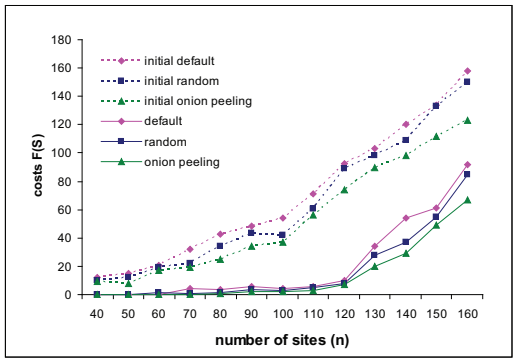
All the results show that better map initialization can improve the performance of map labeling algorithms. The results illustrate that the performance improvement with GA using convex onion peeling was the best among the three algorithms. This motivated us to implement COPGAnot only in the initialization step, but also in the evolutionary process.



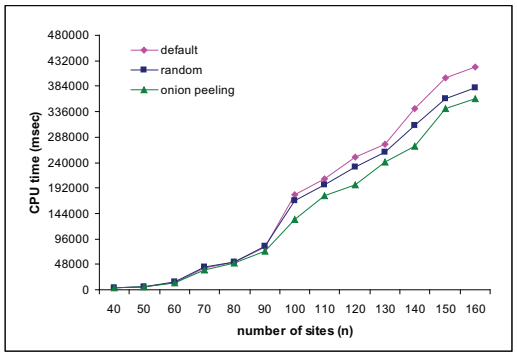
(a) HC: costs after 10000 msec



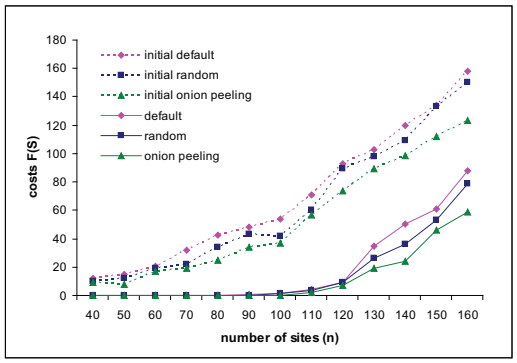
(b) HC: CPU time for 90% reduction



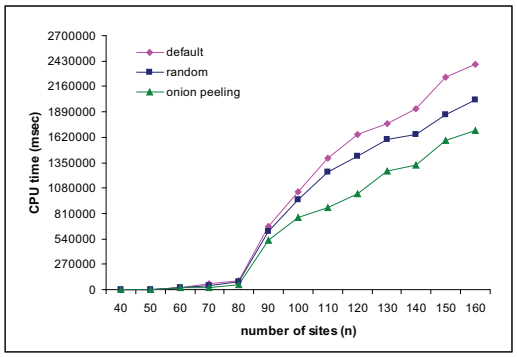
(c) SA: final costs for termination



(d) SA: CPU time for termination



(e) GA: final costs for termination



(f) GA: CPU time for termination

Figure 10: Synthetic data results of HC, SA, and GA

5.3 Results of the evolutionary process

The results of the three different initialization methods in the previous section showed that onion peeling initialization method performed better than the two other methods. In this section, we present the result of COPGA , our proposed algorithm, compared to GA proposed in [18]. The implementation characteristics of these two algorithms are listed in Table 2. COPGA used convex onion peeling initialization and GA used random initialization.

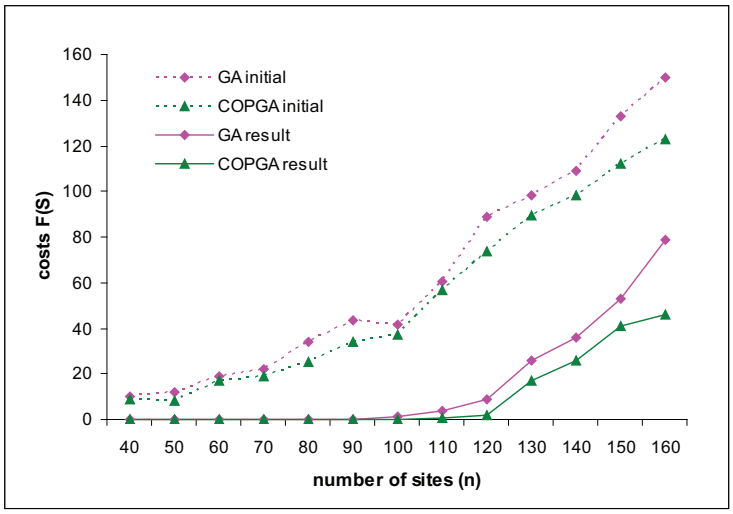


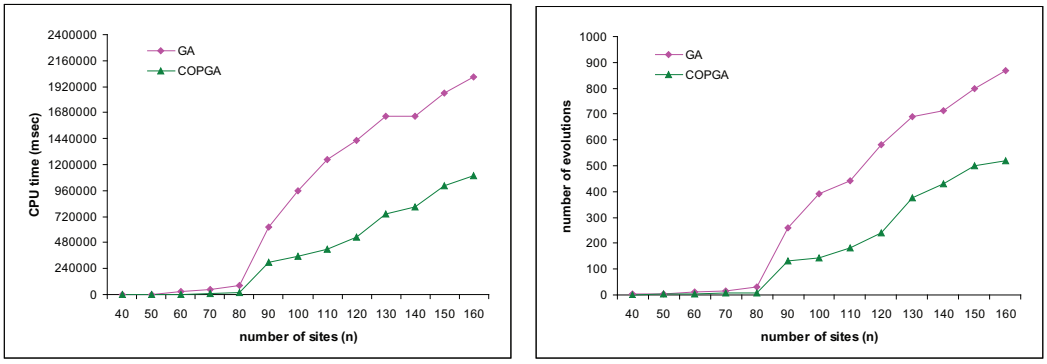
Figure 11: Cost comparison of COPGAand GAwith synthetic datasets, weight $a_3 = 0.1$

In Figure 11, we plotted the average final conflict costs of the algorithms along with their initial costs. The average initial cost of COPGA was 15.56% less than that of GA. For the maps with a number of sites between 40 and 80, both COPGA and GA resulted in no conflicts in the final solutions. However, COPGA outperformed GA for the maps with the sizes of sites between 90 and 160, resulting in 61.29% less conflict costs on average.

Figure 12a shows the CPU time required for termination. The performance improvement of COPGA over GA was between 52.67% and 93.85%. On average, COPGA required 64.77% less CPU time than GA. In addition, the number of evolutions of COPGA to terminate was 50.38% less than that of GA on average as shown in Figure 12b. Up to a certain number of labels (map label density), the differences between COPGA and GA were minimal because both could solve the conflicts while having preferred positions. The results clearly showed that COPGA outperformed GA in terms of CPU time and number of evolutions required. However, if the density increases, then the problem becomes harder as both algorithms required many more evolutions to resolve the conflicts.

Table 4 illustrates the performance of COPGA and GA using the real datasets. The initial conflict costs, final costs, and the reduction rates are presented. We also show the number of evolutions of COPGA and GA for termination. COPGA resulted in 20.00% less cost for the map with 116 sites and 21.55% less cost for the map with 161 sites. The numbers of evolutions for the termination of COPGA were 47.76%, 55.00% and 68.16% less than those

of GA for datasets 1, 2, 3, respectively. The results show that the larger the dataset is, the better the performance of COPGA over GA is.



(a) CPU time ($a_3 = 0.1$) (b) Number of evolutions ($a_3 = 0.1$)

Figure 12: Comparison of COPGAand GAwith synthetic datasets

Real data	COPGA			GA		
	Initial cost	Result (reduction)	# Evol.	Initial cost	Result (reduction)	# Evol.
Dataset 1	30	0 (100%)	22	43	1 (98%)	42
Dataset 2	135	16 (88%)	243	157	20 (87%)	540
Dataset 3	346	91 (74%)	312	402	116 (71%)	980

Table 4: Comparisons of COPGAand GA: cost, CPU time, and number of evolutions with real datasets

5.4 COPGA implementation on OpenMap

To illustrate the overall quality of map labeling generated by COPGA we implemented COPGA on top of a real map application called *OpenMap* [5]. The latest version of OpenMap (OpenMap 4.6.5 released in 2009) was used in our experiments.

OpenMap is a Java Beans based toolkit for building applications and applets that require geographic information. Using OpenMap components, developers can access data from legacy applications, in-place, in a distributed setting. At its core, OpenMap is a set of Java Swing components that understand geographic coordinates. These components help users show map data, and handle user input events to manipulate that data. OpenMap provides the means for users to see and manipulate geospatial information. It has been used in many applications, e.g., marine navigation software, transport modeling software, and integration map service for environmental studies.

OpenMap includes an implementation of map labeling of point features which is called the "decluttering" algorithm [5]. The following is the summary of this algorithm: OpenMap has an option to declutter labels; if the decluttering option is not chosen, then the

labels are placed in their default position (position 2, $\Theta = 45$ in Figure 3b). Open Map decluttering method works by constructing a bit map of pixels that reflects the current projection extents (screen size of map) and all bits are initialized to empty. Decluttering occurs during the preparation of label objects. The label bounds (length, width that is computed based on the label, font, etc.) are used to find an empty region on the pixel map that is within the specified distance from the site. The search is done in a predefined order around the site, increasing the distance; if a location is found, then that part of the bits in the matrix is marked as taken. If there is no non-overlapping location, then the label is dropped (removed) from the map. The labels are chosen to be placed on the map in the same order as they are read from the data source. Therefore, high priority labels must appear before the low priority labels to ensure that they appear on the screen. In the decluttering algorithm of OpenMap, the conflict computation ignores the conflicts of labels with the sites; also the proximity of the labels to other sites is ignored. The OpenMap deconflicting algorithm considers 9 positions including the possibility of placing the label centered at the site. Our algorithm specifically avoids this position. In addition, the OpenMap deconflicting algorithm does not consider label to site conflicts.

We evaluated COPGA using real datasets of city names for two different regions. Table 5 shows the properties of our two real datasets, projection center (longitude, latitude), the size of the region in X, the size of the region in Y, the total number of cities (sites), and the final costs of the COPGA algorithm. Figure 13a and b illustrated the results of the COPGA algorithm using OpenMap. The final result of COPGA on the two real datasets were zero conflicts for the dataset 4 and five conflicts for the dataset 5.

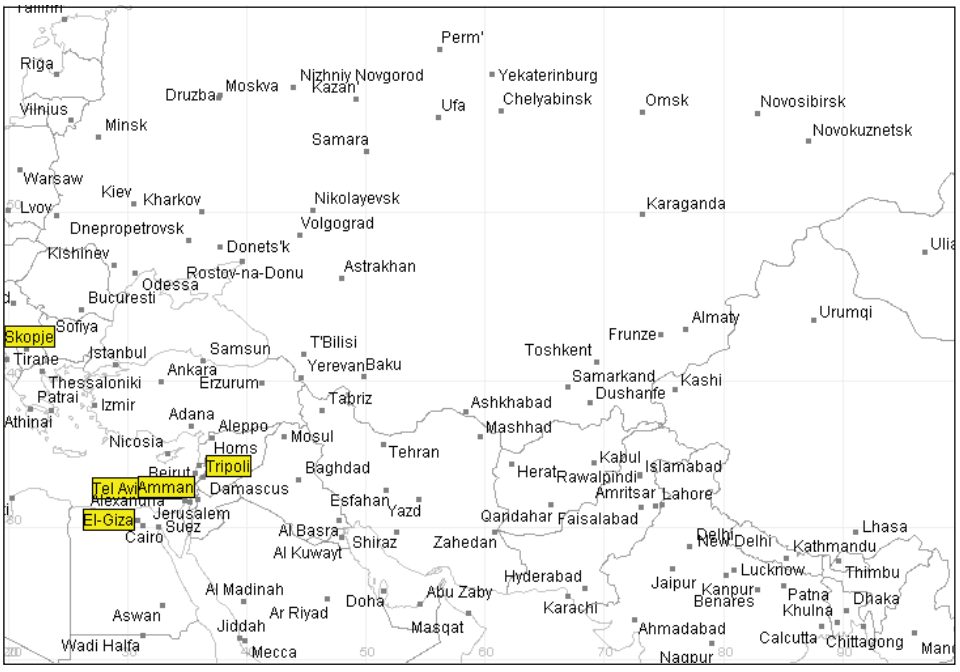
Real data	Projection center (latitude, longitude)	Boundary size		Scale	# Sites	Resulting conflicts
		X	Y			
Dataset 4	(-98.42, 34.15)	727	503	32000000	66	0
Dataset 5	(59.51, 43.00)	727	503	40000000	117	5

Table 5: Results of COPGA using OpenMap with real datasets

6 Conclusions

Map labeling of point features is proven to be of interest in many applications. Due to the difficulty of the problem, different heuristic solutions have been proposed in the literature. We proposed a new solution, the convex onion peeling genetic algorithm (COPGA) that utilizes the convex onion peeling structure in population initialization and the evolution processes of a genetic algorithm. Experimental results using hill climbing [18], simulated annealing [10], a genetic algorithm (GA) [18] and COPGA on the synthetic and real datasets showed that the convex onion peeling initialization results in less conflicts compared to the random and the preferred-position initializations regardless of the algorithm used. Compared to a previously proposed genetic algorithm, COPGA clearly outperformed the other algorithm with respect to the running CPU time and the number of evolutions to reach a terminal condition with reduced conflicts in all cases. We implemented COPGA as well as the other map labeling algorithms using OpenMap and demonstrated the output of running COPGA on real datasets.

(a) Result of labeling real dataset 4 (66 sites)



(b) Result of labeling real dataset 5 (117 sites)

Figure 13: Visualization of COPGA results using OpenMap

For future work, we plan to investigate the effect of clustering labels. A natural extension to convex onion peeling is to perform a density-based spatial clustering algorithm on the sites, and then apply onion peeling on the created clusters independently. This approach could improve map labeling quality regarding cartographic aspects. Also, we will study the effect of applying the inversion operator at the initialization step. From the results of the real datasets in Figure 13, we observed that some labels were not placed at the preferred positions although no conflict was detected. We could apply an heuristic adjustment step at the end of evolutionary process of COPGA, which encourages upper-right positions of labels (label positions in the first quadrant).

References

[1] AHN, J., AND FREEMAN, H. A program for automatic name placement. *Cartographica 21*, 2/3 (1984), 101–109. doi:10.3138/0646-Q262-6636-3681.

[2] AHN, J., AND FREEMAN, H. A comparison of simple mathematical approaches to the placement of spot symbols. *Cartographica 24*, 3 (1987), 46–63. doi:10.3138/GH04-24H7-63T8-167V.

[3] ALVIM, A., AND TAILLARD, E. POPMUSIC for the point feature label placement. *European Journal of Operational Research 192* (2009), 396–413. doi:10.1016/j.ejor.2007.10.002.

[4] BAE, W. D., ALKOBAISI, S., VOJTECHOVSKY, P., NARAYANAPPA, S., AND Y., B. K. Convex onion peeling algorithm: An efficient solution to map labeling of point-features. In *Proc. 25th ACM Symposium on Applied Computing (SAC)* (2010), pp. 892–899.

[5] BBN TECHNOLOGIES. Openmap. http://www.openmap.org, July 2010. Last checked: 19 April 2011.

[6] BEEN, K., AND YAP, C. Dynamic map labeling. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 773–780. doi:10.1109/TVCG.2006.136.

[7] BEKOS, M. A., KAUFMANN, M., SYMVONIS, A., AND WOLFF, A. Boundary labeling: Models and efficient algorithms for rectangular maps. *Computational Geometry 36*, 3 (2007), 215–236. doi:doi:10.1016/j.comgeo.2006.05.003.

[8] BRADSTREET, L., BARONE, L., AND WHILE, L. Map-labelling with a multi-objective evolutionary algorithm. In *Proc. International Conference on Genetic and Evolutionary Computation* (2005), pp. 1937–1944. doi:doi:10.1145/1068009.1068335.

[9] CHAZELLE, B. On the convex layers of a planar set. *IEEE Transactions on Information Theory 31*, 4 (1985), 509–517. doi:10.1109/TIT.1985.1057060.

[10] CHRISTENSEN, J., MARKS, J., AND SHIEBER, S. *Placing Text Labels on Maps and Diagrams*. Academic Press, Cambridge, MA, 1994.

[11] CHRISTENSEN, J., MARKS, J., AND SHIEBER, S. An empirical study of algorithms for point-feature label placement. *ACM Transactions on Graphics 14*, 3 (1995), 203–232. doi:10.1145/212332.212334.

[12] COOK, A. C., AND JONES, C. B. A Prolog rule-based system for cartographic name placement. *Computer Graphics Forum 9*, 2 (1990), 109–126. doi:10.1111/j.1467-8659.1990.tb00384.x.

[13] DOERSCHLER, J., AND FREEMAN, H. A rule-based system for dense-map name placement. *Communications of ACM 35*, 1 (1992), 68–79. doi:10.1145/129617.129620.

[14] DUMITRESCU, D., LAZZERINI, B., JAIN, L., AND DUMITRESCU, A. *Evolutionary Computation*. CRC Press, Boca Raton, FL, 2000.

[15] EDMONDSON, S., MARKS, J., CHRISTENSEN, J., AND SHIEBER, S. A general cartographic labeling algorithm. *Cartographica 33*, 4 (1996), 13–23. doi:10.3138/U3N2-6363-130N-H870.

[16] FORMANN, M., AND WAGNER, F. A packing problem with applications to lettering of maps. In *Proc. 7th Annual Symposium on Computational Geometry* (1991), pp. 281–288.

[17] GUTTMAN, A. R-trees: A dynamic index structure for spatial searching. In *Proceedings of ACM SIGMOD* (1984), pp. 45–57. doi:10.1145/602259.602266.

[18] HONG, F., KAIJUN, L., AND ZUXUN, Z. An efficient and robust genetic algorithm approach for automatic map labeling. In *Proc. International Cartographic Conference (ICC'05)* (2005).

[19] IMHOF, E. Positioning names on maps. *The American Cartographer 2*, 2 (1975), 128–144. doi:10.1559/152304075784313304.

[20] KIRKPATRICK, S., GELATT, C. D., JR., AND VECCHI, M. P. Optimization by simulated annealing. *Science 220*, 4598 (1983), 671–680. doi:10.1126/science.220.4598.671.

[21] MANZHOS, S., AND LOOCK, H. Photogragment image analysis using the onion-peeling algorithm. *Computer Physics Communications 154* (2003), 76–87. doi:10.1016/S0010-4655(03)00277-7.

[22] MARKS, J., AND SHIEBER, S. The computational complexity of cartographic label placement. Tech. Rep. TR-05-91, Harvard University, 1991.

[23] MICHALEWICZ, Z. *Gentic Algorithms + Data Structures = Evolution Programs*. Springer, Berlin, 1992.

[24] PETZOLD, I., GRÖGER, G., AND PLÜMER, L. Fast screen map labeling—data-structures and algorithms. In *Proc. International Cartographic Conference (ICC'03)* (2003), pp. 288–298.

[25] PETZOLD, I., PLÜMER, L., AND HEBER, M. Label placement for dynamically generated screen maps. In *Proc. International Cartographic Conferences (ICC'99)* (1999), pp. 893–903.

[26] RAIDL, G. R. A genetic algorithm for labeling point features. In *Proc International Conference on Imaging Science* (1998), pp. 189–196.

[27] RUSSELL, J. M. The halogen occultation experiment (HALOE). http://haloe.gats-inc.com, 2005. Last checked: 11 April 2011.

[28] STRIJK, T., AND KREVELD, M. V. Practical extensions of point labeling in the slider model. *Geoinformatica 6*, 2 (2002), 181–197. doi:10.1023/A:1015202410664.

[29] USGS (UNITED STATED GEOLOGICAL SURVEY). CITIESX020—U.S. National Atlas Cities and Towns. http://coastalmap.marine.usgs.gov/GISdata/basemaps/usa/cities/citiesx020.htm, 2004. Last checked: 19 April 2011.

[30] VAN DIJK, S., THIERENS, D., AND DE BERG, M. Using genetic algorithms for solving hard problems in GIS. *Geoinformatica 6*, 4 (2002), 381–413. doi:10.1023/A:1020809627892.

[31] WAGNER, F., AND WOLFF, A. A practical map labeling algorithm. *Computational Geometry: Theory and Applications 7* (1997), 387–404. doi:10.1016/S0925-7721(96)00007-7.

[32] ZHANG, Q., AND HARRIE, L. Real-time map labelling for mobile applications. *Computers, Environment, and Urban Systems 30*, 6 (2006), 773–783. doi:10.1016/j.compenvurbsys.2006.02.004.