INFORMS Journal on Computing, Vol. 9, No. 3, Summer 1997, pp. 266-275.

Placing Text Labels on Maps and Diagrams using Genetic Algorithms with Masking

Oleg V. Verner *oleg@elvis.msk.su*

Roger L. Wainwright *rogerw@penguin.mcs.utulsa.edu*

Dale A. Schoenefeld *dschoen@euler.mcs.utulsa.edu*

Mathematical and Computer Sciences Department
The University of Tulsa
600 South College Avenue
Tulsa, OK 74104-3189, USA

Abstract

Cartographic label placement is one of the most time-consuming tasks in the production of high quality maps and other high quality graphical displays. It is essential that text labels used to identify various features and objects be placed in a clear and unobscured manner. In this paper we are concerned with the placement of labels for point features. Specifically, the point feature label placement (PFLP) problem is the problem of placing text labels to point features on a map, graph or diagram in such a manner so as to maximize legibility. The PFLP problem has been shown to be NP-hard. We propose a heuristic method for the PFLP problem based on genetic algorithms (GA), an adaptive robust search and optimization technique based on the principles of natural genetics and survival of the fittest. In particular we emphasize the notion of masking to preserve optimal subsequences in chromosomes and prevent their disruption during crossover and mutation. We ran our algorithms on randomly placed point features in a region, and on datasets from various regions of the USA map with great success. Our GA implementation with masking solved each of the test cases extremely well, and proved to be an excellent heuristic for solving the PFLP problem. Furthermore, our GA with masking performed significantly better than other PFLP algorithms from the literature.

Text labels used to identify various features and objects are a fundamental part of producing good graphs and maps. It is essential that text labels be placed in a clear and unobscured manner.

Typically in maps there are three types of features that need to be labeled: point features such as cities, linear features such as boundaries, rivers, roads, and area features such as lakes, airports, parks, states and counties. Cook and Jones [5] claim that the placement of labels by cartographers typically takes up to one half of the total time required for generating high-quality maps.

In this paper we are only concerned with the placement of labels for point features. Labeling point features is the essential part of producing quality maps. Specifically, the point feature label placement (PFLP) problem is the problem of placing text labels to point features on a map, graph or diagram in such a manner so as to maximize legibility. There are three issues to consider when placing text on point features. The first issue is the number and extent in which labels overlap each other. The second concern is the number of times a label may overlap another point feature on the map (but, perhaps not the label for that point). The third issue concerns a preference assignment to the placement of a label for a point as long as no other labels and points are obscured. The third issue is less significant than the first two. The notion of labeling quality has been studied by many researchers. Figure 1 shows the traditional set of eight standard label positions for a point feature. Each rectangle indicates a region in which a label may be placed. The value inside each rectangle indicates the traditional accepted order of preference for placing a label. Lower values indicate more desirable (or more aesthetically pleasing) positions. A continuous label placement model is possible if one is allowed to place the label anywhere in a much larger rectangle around the point. It is also possible to specify a circle around the point feature such that the label must be placed entirely within the circle. In all instances labels are placed in a horizontal position. In our research, we allow for varying length labels around a point feature depending on the length of the labeled text for that point.

In many applications it may be impossible to place text labels on all points such that there are no overlapping labels. This may be the case in highly congested areas of a map. A variation of the PFLP problem allows for the deletion of points and their associated labels in order to produce unobscured label placement. This variation is called the PFLP *point selection problem.* Of course, the object is to minimize the number of deleted points and associated labels and still produce an unobscured label placement for the remaining points. Marks and Shieber [15] provide an excellent proof that the PFLP problem is NP-hard. Clearly from Figure 1, the number of possible label placements for n point features is 8^n, which is exponential, not polynomial in n.

A wide variety of techniques have been developed over the years for automating the PFLP problem as best as possible. Yoeli [21] developed a simple deterministic greedy algorithm in the early 1970's for the PFLP problem. Jones [13] developed a nondeterministic algorithm based on recursive backtracking. Zoraster [22] developed an algorithm based on integer programming. Other algorithms for the PFLP problem are presented by Hirsch [12], and Ahn and Freeman [1]. Christensen *et al.* [3, 4] developed a simulated annealing algorithm for the PFLP problem. Their research will be discussed later in the paper. A comprehensive bibliography of the PFLP problem along with a more extensive review of previous research on the PFLP problem is given by Christensen *et al.* [4] and Marks and Shieber [15]. In this paper we present a genetic algorithm (GA) for solving the PFLP problem. The authors are not aware of any other GA implementation for the PFLP problem. The rest of the paper is described as follows. In Section 1 an overview of genetic algorithms is presented along with our GA implementation for the PFLP problem. A discussion of masking in genetic algorithms is also presented in this section. Results of our experiments and our conclusions are presented in Section 2.

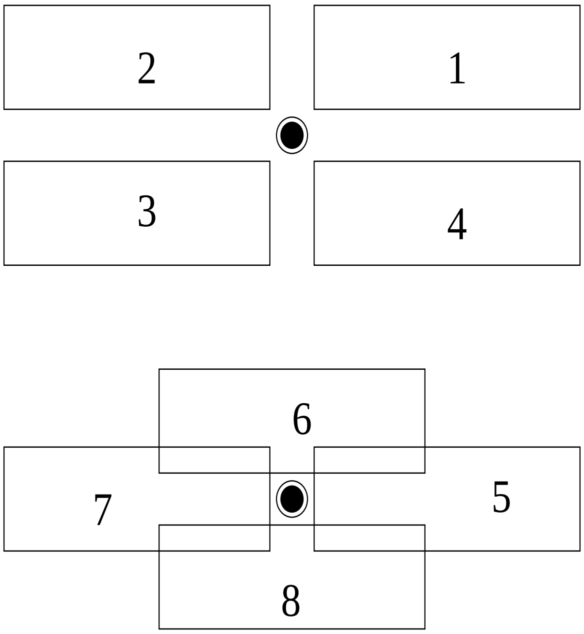

Figure 1: Set of possible label positions and their associated desirability (1 is desirable, 8 is undesirable)

1 Genetic Algorithms

The genetic algorithm (GA) is an adaptive robust search and optimization technique based on the principles of natural genetics and survival of the fittest. Genetic algorithms use the laws of natural selection and genetics to guide a nondeterministic search. By simulating natural evolution in this way, a GA can effectively search the problem domain and easily solve complex problems. Furthermore, by emulating biological selection and reproduction techniques, a GA can perform the search in a general, representation-independent manner.

The genetic algorithm operates as an iterative procedure on a fixed size population or pool of candidate solutions. The candidate solutions represent an encoding of the problem into a form that is analogous to the chromosomes of biological systems. Each chromosome represents a possible solution for a given fitness function. Associated with each chromosome is a fitness value, which is found by evaluating the chromosome with the fitness function. It is the fitness of a chromosome that determines its ability to survive and produce offspring. Each chromosome is made up of a string of genes (whose values are called alleles). The chromosome is typically represented in the GA as a string of bits. However, integers, floating point numbers and characters can easily be used. For the interested reader, Davis, Goldberg and others provide an excellent in-depth study of genetic algorithms [9, 11, 17]. Furthermore, several researchers have investigated the benefits of solving various combinatorial optimization problems using genetic algorithms [6, 10, 16, 18, 19, 20]. It is assumed that the reader is familiar with the fundamental concepts of genetic algorithms.

1.1 Masking

Corcoran and Wainwright [7] present several techniques to preserve optimal subsequences in chromosomes and prevent their disruption during crossover and mutation. These techniques include *Sliding Window*, *Reduction*, *Masking*, and *Relative Bonding Strength*. In this paper we will use the *masking* concept to preserve the more highly fit genes within a chromosome. A general overview of masking is described below independent of any specific problem.

The most natural way to preserve individual genes in a chromosome is to provide a bit mask the same length as the chromosome. Each bit in the mask indicates whether the corresponding gene is highly fit or not. For example, suppose that in the chromosome ABCDEFGH, genes A, B, D, and F are considered highly fit. The associated bit mask,

```
A B C D E F G H
1 1 0 1 0 1 0 0
```

makes this fact explicit by indicating the highly fit genes with '1' bits.

One minor problem with masks is that traditional crossover operators must be adapted to use the information it provides. This generally requires a simple modification. Several suggestions are given below:

- For many crossover functions a postprocessing step could be added to ensure the masked genes survive. The crossover could create children as it normally does, and the postprocessing operation could undo any damage created. This is especially useful in single and multipoint crossover techniques.

- The uniform order crossover (UOX) operator of Davis [9] uses randomly generated bit masks. These could easily be replaced by the user defined bit mask from each parent chromosome. An element of randomness could be preserved by performing a logical OR between the random and user defined masks. Corcoran *et al.* [7] define this operator as UOX/MASK. If it is necessary to maintain a balance between set and unset bits, then the random number generator could be adaptive based on the number of set bits in the user mask. When the user mask is sparse, the random mask could have a large number of bits set at random. That is, the number of bits set in the random mask is inversely proportional to the number of set bits in the user mask.

- For order based crossovers, the crossover operator may need to be rewritten in order to take advantage of the user defined bit mask.

The mask concept is not limited to just bit masks. Integers or real values could be used just as easily. For example, the fitness for a gene can be represented as a real number or a normalized fraction [0..1]. A mapping function could be used to adaptively discretize real valued masks and transform them into integer or bit masks. Non-bit values provide more information, and may be more useful in discriminating between distinct schemata or to distinguish groups of related genes.

1.2 A Genetic Algorithm for the PFLP Problem with Masking

In our research we did not consider the PFLP point selection problem. That is, none of the point features were deleted. Given a dataset with n point features, we used a chromosome with length n, where each allele in the chromosome corresponds to a point feature and may take on one of the values from 1..8 depicting which label position is used for each point feature. Note this is not an order based problem where the chromosome is a permutation of the values 1 to n. The position in the chromosome of each point feature (allele) is fixed and the order does not matter. We used two types of datasets, random and contrived. We used a 128 point randomly generated dataset called

R128, and a 500 point randomly generated dataset called R500, and four non-random datasets consisting of 44, 69, 94, and 128 point features denoted by USA44, USA69, USA94, and USA128, respectively. The four non-random datasets came from various regions of the USA map, and can be found in the miles.dat dataset in Knuth [14]. In the R128 and R500 datasets each text label was assigned a random length. In the four USA datasets each text label varies in length depending on the name of the city it represents. This aids in the label placement process, and makes it more realistic. In order to compare our algorithm with other PFLP algorithms, we also ran our GA implementation on a group of standard datasets from the literature with 100, 250, 500, 750 and 1000 point features. These datasets are described later in the paper.

Each chromosome represents a unique placement of all of the text labels for a given dataset. The choice of a good fitness function is extremely critical in order to evolve the population of chromosomes to a solution with non-overlapping labels. We considered the following factors in our fitness function.

1. The number of overlapping labels.

2. The total area of the labels that overlap. It could be that there are several labels that overlap, but each overlap could be very small, and perhaps not very noticeable. This factor addresses this issue.

3. Calculate the sum of the Euclidean distances between the center of the label for a given point feature and the center of the labels for the m nearest point features. We define this to be the *distance factor* of a point feature. In our tests, we used $m = 4$. We considered *distance factors* for only those point features that had an overlapping label.

4. The average label value over all of the point features according to the numeric assignment given to each label placement in Figure 1. This factor attempts to honor the priority of label placements.

We found that some of the above factors were more significant than others in driving the population to good solutions. Specifically, the objective function that we used is given below:

Fitness(Chromosome) =
a * (Number of overlapping labels)
+ b * (Total area of the overlapping labels)

- c * (Sum of the *distance factors* for points with overlapping labels)

+ d * (Average label value (from Figure 1)).

We used Real values for a,b,c,d in the range from [0..10]. Notice each part of the fitness function can be ignored by using a zero value for any of a,b,c, and d.

There are several genetic algorithm packages that have been made available to researchers over the past several years. We implemented our genetic algorithm for the PFLP problem using LibGA [8]. In all of our test cases the initial population was generated randomly. In all cases the crossover operator was *uniform order crossover*, the crossover rate was 0.9, the mutation operator was *simple*, and the mutation rate was 0.1. The *simple* mutation operator randomly selects an allele, then replaces the allele with a random integer in the range 1..8. The population size for all of the USA datasets, R128 and R500 was 200. Uniform order crossover is traditionally used with an order-based GA [9]. Even though our PFLP GA implementation is not an order-based problem we found the uniform crossover worked very well for us.

In each of our test datasets we ran our GA with and without the masking option. In Section 2.1 we discussed, in general, the user defined bit mask for an unspecified problem. We now define a specific user defined bit mask for the PFLP problem. When the masking option is used, we define a bit string mask for a given chromosome as follows. Note, in the following discussion we use the term, city, interchangeably with point feature. This is because in all of our test cases we are concerned with placing text labels on a map to identify cities.

1. Determine every allele (city) in the given chromosome whose label overlaps with a label from another city.

2. For each city found in Step 1, determine the set of four closest cities according to Euclidean distance. (The four closest cities to each of the cities in the dataset was determined *a priori*).

3. The mask bit for each city generated from either Step 1 or Step 2 is set to zero. All other alleles are set to one.

In this way, cities with overlapping labels or cities near other cities with overlapping labels will be allowed to exchange information in mutation and crossover and perhaps correct the situation. All other cities have labels that do not conflict, nor are they near a city with an overlapping label problem.

The following example explains in detail the uniform crossover operation with masking for the PFLP problem: Consider two parent chromosomes, *P1* and *P2*, with associated masks, *M1*, and *M2*, representing a 20 city text labeling problem. Let *U* be the randomly generated bit string used for uniform crossover, and let *C1* and *C2* be the children chromosomes that result from applying uniform crossover. The uniform crossover operation with masking works as follows:

First apply the uniform crossover to the parent chromosomes to obtain two intermediate children *c1* and *c2* as shown below.

```
P1:  37281426837516385346
P2:  27164583552743188214
 U:  01100110010010101101
c1:  27264423532713385316
c2:  37181586857546188244
```

Next, the masking option is applied. There are four cases for each allele:

```
if M1=1 and M2=0, then C1=P1 and C2=P1
if M1=0 and M2=1, then C1=P2 and C2=P2
if M1=1 and M2=1, then C1=P1 and C2=P2
if M1=0 and M2=0, then C1=c1 and C2=c2
```

In other words, when only one of the two mask bits is set, then the resulting allele for both children comes from the parent associated with the set mask bit. If both mask bits are set, the children alleles are the same as the parents. In the case when both mask bits are zero then the results come from the uniform crossover. The rest of this example is shown below.

```
M1:  10010110000010001011
M2:  01100001001100001010
C1:  37184423532713385346
C2:  37181423852716188216
```

Note, when masking is in effect only alleles that are associates with a mask value of zero are candidates for mutation.

2 Results and Conclusions

2.1 GA Fitness Factors and Masking

Table 1 depicts the results from the USA44 dataset. Eighteen different combinations of factors for a, b, and c in the fitness function were used. For example, the first row represents results when the values for a, b, and c in the fitness function are 1, 0, and 0, respectively. The d factor in the fitness function was not used in our analysis. The various combination of values for a, b, and c were arrived at by numerous trial and error testing to determine the best combinations to use. In the first 10 cases in Table 1 the masking option was not used. In the last eight cases masking was used. This is indicated by 0 (masking not used) or 1 (masking used) in the Mask column. In each of the 18 cases for various values of a, b, and c (with and without masking), we ran 20 executions using different random number seeds each time. This translates to 360 different test runs per dataset. The Results column indicates how many times out of a possible 20 executions that a solution to the PFLP problem was found using the associated parameters. We define a solution as any label placement for all of the points such that no labels overlap. Figure 2 depicts a random initial label placement from a chromosome in the initial population for the USA44 dataset. Compare this to Figure 3 which depicts a solution for the USA44 dataset using our GA implementation for the PFLP problem.

Table 2, Table 3, and Table 4 depict the results from the USA69, USA94, and USA128 datasets, respectively. It is clear from each of the USA datasets (Tables 1 through 4) that masking improved the results significantly compared to not masking. Compare the mask = 0 section versus the mask = 1 section in each of the Tables 1 through 4. The c factor in the fitness function corresponds to the distance factors for points with overlapping labels. In order to have the factors a, b, and c contribute somewhat equally to the fitness function, a rather small c value was necessary. We determined 0.0001 was a good factor to use. Notice in the fitness function with respect to distance factors that larger distance factors are preferred over smaller distance factors. Since we are trying to minimize the fitness function the c contribution must be subtracted. It is clear from the results that better performance occurred when $c = 0.0001$ was used versus $c = 0$. This is clearly evident in Tables 1 through 3. However, in Table 4 the effect of a zero verses a non-zero c value is inconclusive since all of the unmasked results were very poor. As expected, the number of overlapping labels is a significant factor in the fitness function. This corresponds to a positive value for a in Tables 1 through 4. The two entries in each of these tables where $a = 0$ produced poor solutions compared

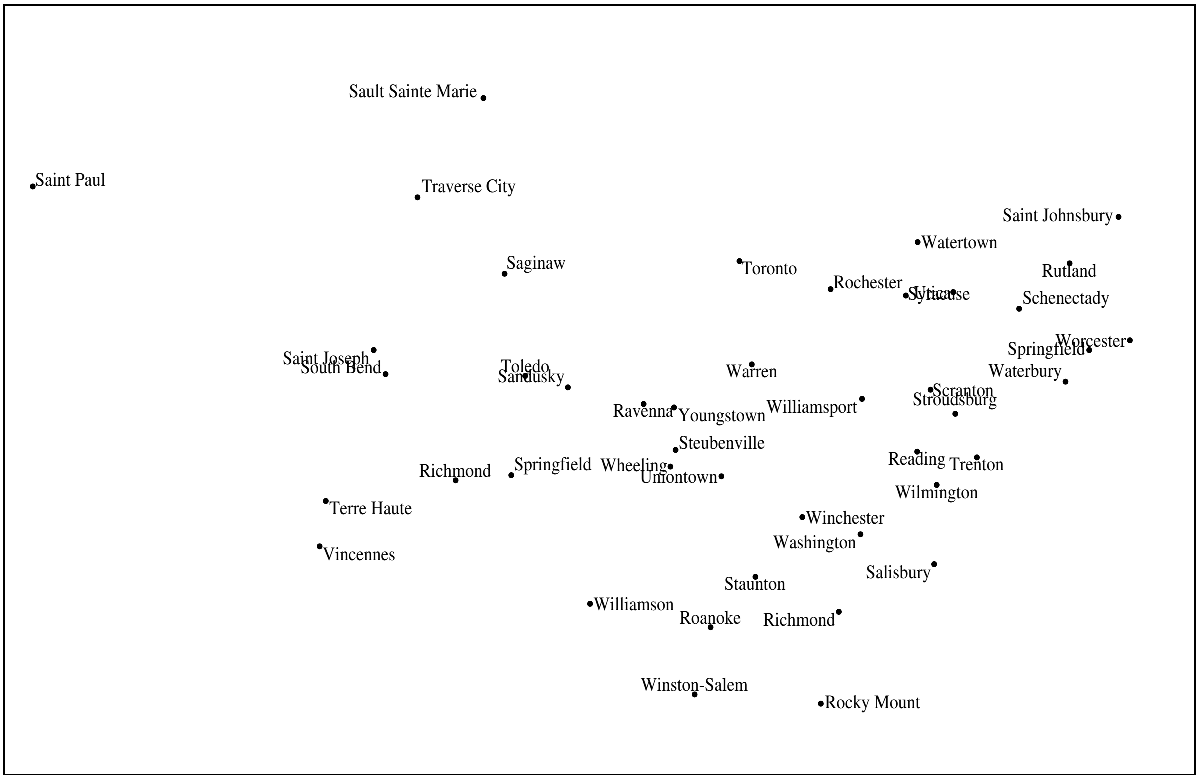

Figure 2: A Example Random Label Placement for the USA44 Dataset. (11) Overlapping Labels

to the others. The effect of the total area of the overlapping labels corresponds to a positive value for b in Tables 1 through 4. It is inconclusive if this factor has any effect on the fitness function or not. In some cases a positive b value appeared to be counter productive in producing good solutions. In summary, from Table 1 through Table 4 representing the USA datasets, it appears the best set of parameters to use are $a = 1$, $b = 0$, $c = 0.0001$, and especially mask = 1.

Table 5 and Table 6 depict the results from the random datasets, R128 and R500, respectively. The results shown in Table 5 for the R128 dataset are no different than the results obtained from the USA datasets shown in Table 1 through Table 4. That is, masking is clearly superior to nonmasking, a positive value for a is better than a zero value, and a positive c value of 0.0001 is better than a value of zero.

Dataset R500 proved to be a very interesting problem. The results are shown in Table 6. In this example a c value of .000001 was used to keep the factor in line with the weight of the other parameters. Notice that a solution was never found in each of the 20 executions of the parameter sets when masking was not used. When masking was used as part of the parameters a solution was obtained anywhere from three to 10 times out of 20 executions. In order to determine the

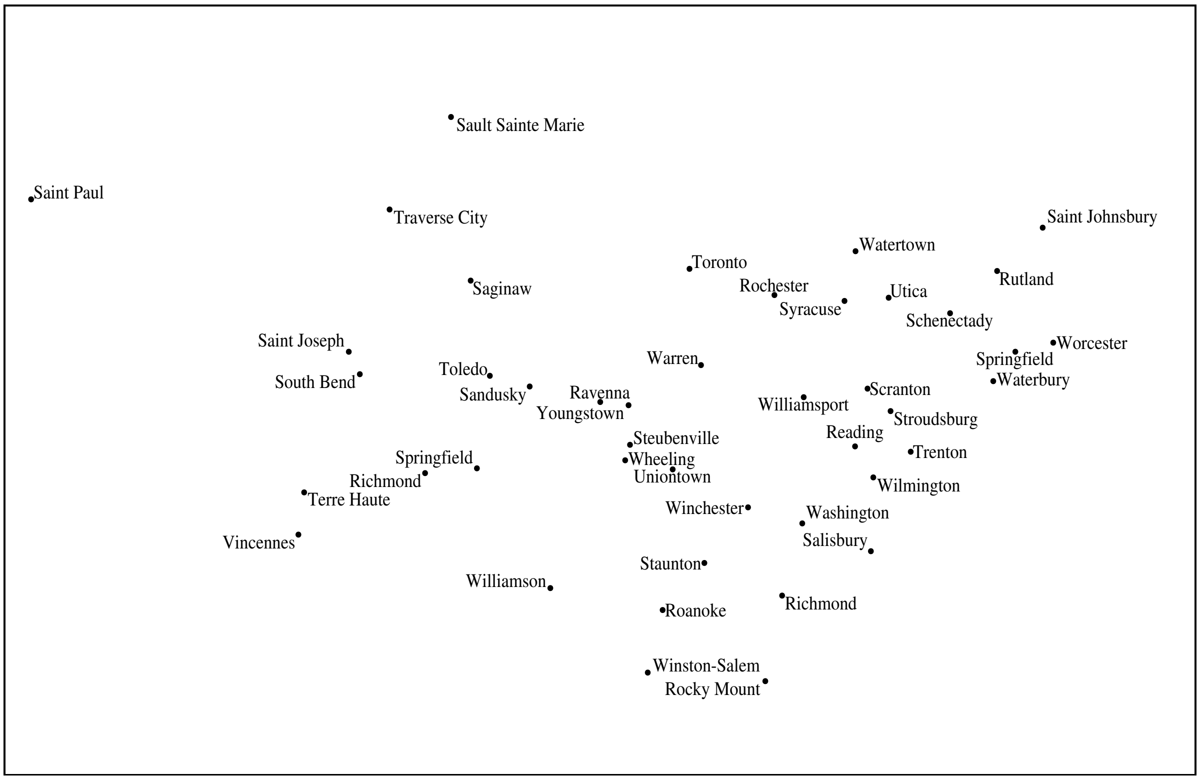

Figure 3: A Solution Label Placement for the USA44 Dataset. No Overlapping Labels

quality of the solutions we included a column in Table 6 denoted as *Avg. Num. Overlaps.* This column indicates the average number of overlapping labels that occurred in the best solution over all 20 trials. For example, in the first row the average number of overlapping labels at the time of the solution was 18.45. This represents very poor solutions. Contrast this with the row where the *a, b, c* and *mask* values are 1, 0, .00001 and 1, respectively and the number of overlapping labels at the time of the solution was 0.50. This means a total of 10 overlaps occurred in the 20 cases, and since 10 cases were solutions, this means the remaining 10 non-solution cases had exactly one overlapping label each. Hence, in this case the GA produced a solution in 10 of the cases, and in each of the remaining 10 cases a near optimal solution of only one label overlap was produced. In Table 6 we also included the average generation number where the best solution first occurred, and the average number of generations required for convergence. Note the best solution seems to occur shortly before convergence. The R500 dataset shows the significance of masking with the GA.

Figure 4 depicts a random initial label placement from a chromosome in the initial population for the R500 dataset. Compare this to Figure 5 which depicts a solution for the R500 dataset using our GA implementation for the PFLP problem.

2.2 GA Compared to other PFLP Algorithms

Christensen *et al.* [4] developed a simulated annealing algorithm for the PFLP problem. They compared their simulated annealing PFLP algorithm to several other PFLP algorithms found in the literature. Their results showed that their simulated annealing algorithm was superior compared to all of the other PFLP algorithms they tested over all of the test datasets. The other PFLP algorithms that Christensen tested include a Greedy Depth-First algorithm, a Discrete Gradient Descent algorithm, a 2-opt Gradient Descent algorithm, a 3-opt Gradient Descent algorithm, an algorithm developed by Hirsch [12], an algorithm developed by Zoraster [22], and a Random Placement algorithm which serves as a worst case bound. Christensen *et al.* [2] provides a complete discussion of the implementation details of all of these algorithms. Christensen *et al.* [4] compared all of these algorithms using a *Standard* set of randomly generated datasets with the following parameters: inside a grid size of 792 by 612, n point features were randomly placed, each with a fixed size label of 30 by 7 units.

In order to compare our GA PFLP algorithm with the results from the algorithms presented by Christensen *et al.* [4], we ran our GA PFLP algorithm on the same Standard set of datasets for n = 100, 250, 500, 750 and 1000. We call these datasets S100, S250, S500, S750 and S1000, respectively. We simulated the same conditions as described by Christensen. That is, our GA algorithm was run with a different random placement of the point features 25 different times for each of the five Standard datasets. The percent of labels placed without conflict (averaged over the 25 trials) resulting from our GA PFLP algorithm is recorded in Table 7 along with the results from the other PFLP algorithms as reported by Christensen *et al.* [4]. We ran our GA algorithm with and without masking. In every case, the a, b, c parameters were kept constant at 1, 0, .0001, respectively. The population size that we used for S100, S250, S500, S750 and S1000 was 200, 250, 400, 400, and 500, respectively. The execution time for a single trial for the GA algorithm with masking for S100, S250, S500, S750 and S1000 was 6, 49, 414, 1637, and 7256 seconds, respectively on a Sun Sparc 10 workstation.

The results in Table 7 clearly show our GA algorithm with masking performed significantly better than all of the other PFLP algorithms on all of the datasets we tested. This includes the simulated annealing PFLP algorithm recently developed by Christensen *et al.* [4].

2.3 Summary

In summary, we developed a GA implementation for the PFLP problem. We introduced a fitness function based on the number of overlapping labels, total area of the overlapping labels, a distance factor from the point features with overlapping labels, and the average label value for the chromosome. We experimentally determined the weight that each of the above factors should contribute to the total fitness value. We also introduced the concept of masking as it applies to the PFLP problem. In every test case, the GA with masking significantly outperformed the GA without masking. This occurred on very simple problems to very difficult problems. There is no guarantee the masking option will always outperform the non-masking option. Indeed it may be possible that for certain datasets the masking option may prove to be a hindrance to good performance. The best approach is to always run both the masking and non-masking option. The GA implementation with masking proved to be an excellent heuristic for solving the PFLP problem; it solved every one of our test cases extremely well. Clearly it is the masking option that makes the GA work well. Without implementing the masking option, the GA PFLP algorithm would be a rather poor PFLP algorithm, not only in results (see Table 7) but in CPU time as well. On a SUN Sparc 10 workstation, the GA implementation without masking took about 6 hours per trial for the $n =$ 1000 case compared to about 2 hours for the GA implementation with masking. Finally, our GA with masking algorithm was superior in performance to all of the other PFLP algorithms from the literature that we tested.

Acknowledgements

This research has been partially supported by OCAST Grant AR2-004. The authors gratefully acknowledge Jon Christensen and Stuart Shieber for providing additional results from their paper [4] so we could compare our GA algorithm to other PFLP algorithms. The authors also acknowledge the support of Sun Microsystems, Inc. Oleg Verner is a Senior Analyst at the Moscow State Institute of Electronic Engineering, Russia (MIET), and with the ELVIS-PLUS Company, Moscow. This research was conducted while Dr. Verner was a visiting professor at The University of Tulsa.

References

[1] J. AHN and H. FREEMAN, 1984. A program for Automatic Name Placement, *Cartographica,*

Table 1: Results from 20 GA Executions for USA44

a	b	c	Mask	Results
1	0	0	0	13/20
0	1	0	0	6/20
1	1	0	0	16/20
10	1	0	0	12/20
1	10	0	0	13/20
1	0	.0001	0	13/20
10	0	.0001	0	14/20
1	1	.0001	0	13/20
10	1	.0001	0	15/20
1	10	.0001	0	13/20
1	0	0	1	19/20
0	1	0	1	18/20
1	1	0	1	20/20
1	0	.0001	1	20/20
10	0	.0001	1	20/20
1	1	.0001	1	20/20
10	1	.0001	1	20/20
1	10	.0001	1	20/20

Table 2: Results from 20 GA Executions for USA69

a	b	c	Mask	Results
1	0	0	0	7/20
0	1	0	0	1/20
1	1	0	0	8/20
10	1	0	0	13/20
1	10	0	0	7/20
1	0	.0001	0	15/20
10	0	.0001	0	13/20
1	1	.0001	0	14/20
10	1	.0001	0	12/20
1	10	.0001	0	10/20
1	0	0	1	20/20
0	1	0	1	18/20
1	1	0	1	20/20
1	0	.0001	1	20/20
10	0	.0001	1	20/20
1	1	.0001	1	20/20
10	1	.0001	1	20/20
1	10	.0001	1	20/20

Table 3: Results from 20 GA Executions for USA94

a	b	c	Mask	Results
1	0	0	0	7/20
0	1	0	0	1/20
1	1	0	0	6/20
10	1	0	0	12/20
1	10	0	0	4/20
1	0	.0001	0	11/20
10	0	.0001	0	9/20
1	1	.0001	0	10/20
10	1	.0001	0	10/20
1	10	.0001	0	6/20
1	0	0	1	17/20
0	1	0	1	18/20
1	1	0	1	20/20
1	0	.0001	1	20/20
10	0	.0001	1	20/20
1	1	.0001	1	19/20
10	1	.0001	1	20/20
1	10	.0001	1	20/20

Table 4: Results from 20 GA Executions for USA128

a	b	c	Mask	Results
1	0	0	0	6/20
0	1	0	0	5/20
1	1	0	0	8/20
10	1	0	0	9/20
1	10	0	0	5/20
1	0	.0001	0	0/20
10	0	.0001	0	8/20
1	1	.0001	0	1/20
10	1	.0001	0	9/20
1	10	.0001	0	3/20
1	0	0	1	19/20
0	1	0	1	19/20
1	1	0	1	20/20
1	0	.0001	1	20/20
10	0	.0001	1	20/20
1	1	.0001	1	20/20
10	1	.0001	1	20/20
1	10	.0001	1	20/20

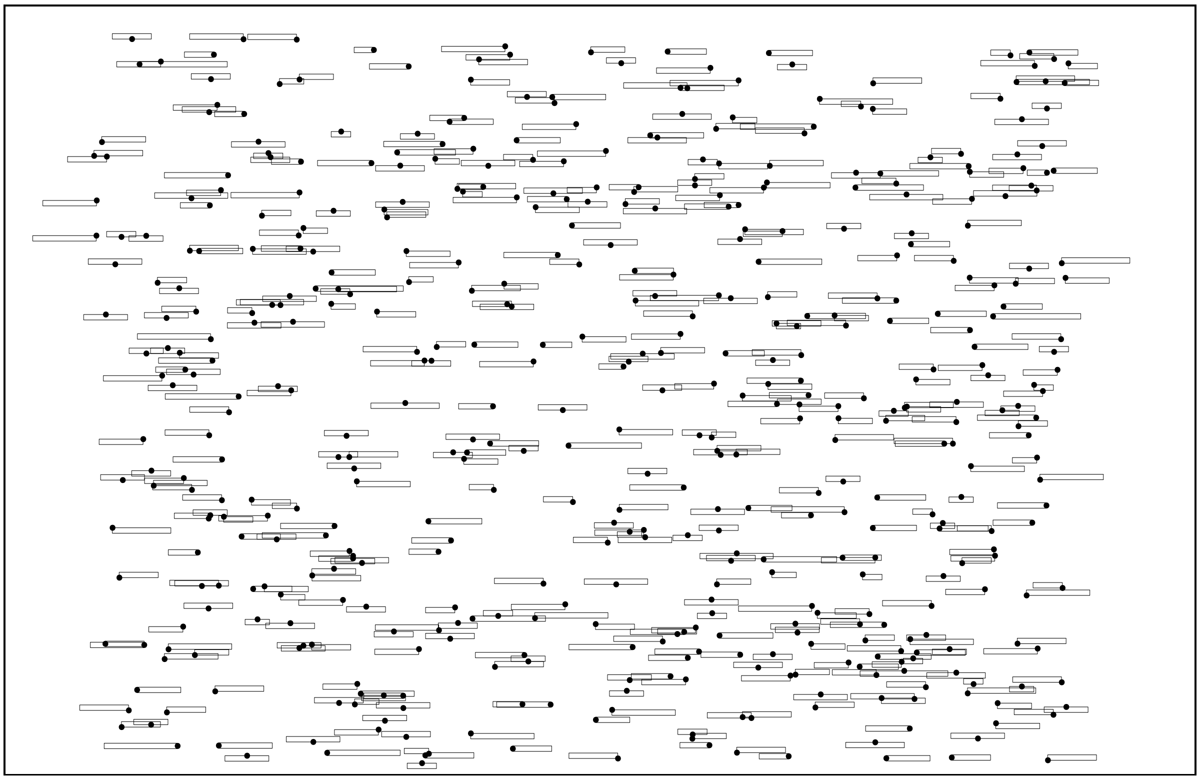

Figure 4: A Example Random Label Placement for the R500 Dataset. (130) Overlapping Labels

21, (2&3), pp. 101-109.

[2] J. CHRISTENSEN, J. MARKS and S. SHIEBER, 1992. Labeling Point Features on Maps and Diagrams, Center for Research in Computing Technology, Harvard University, TR-25-92.

[3] J. CHRISTENSEN, J. MARKS and S. SHIEBER, 1994. Placing Text Labels on Maps and Diagrams, in *Graphics Gems*, P. S. Heckberk, (ed.), AP Professional, pp. 497-505.

[4] J. CHRISTENSEN, J. MARKS and S. SHIEBER, 1995. For Point Feature Label Placement, *ACM Transactions on Graphics, 14*, (3), pp. 203-232.

[5] A. C. COOK and C. B. JONES, 1990. A Prolog Rule-based System for Cartographic Name Placement, *Computer Graphics, 9*, pp. 109-126.

[6] A. L. CORCORAN and R. L. WAINWRIGHT, 1992. A Genetic Algorithm for Packing in Three Dimensions, *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing*, ACM Press, pp. 1021-1030.

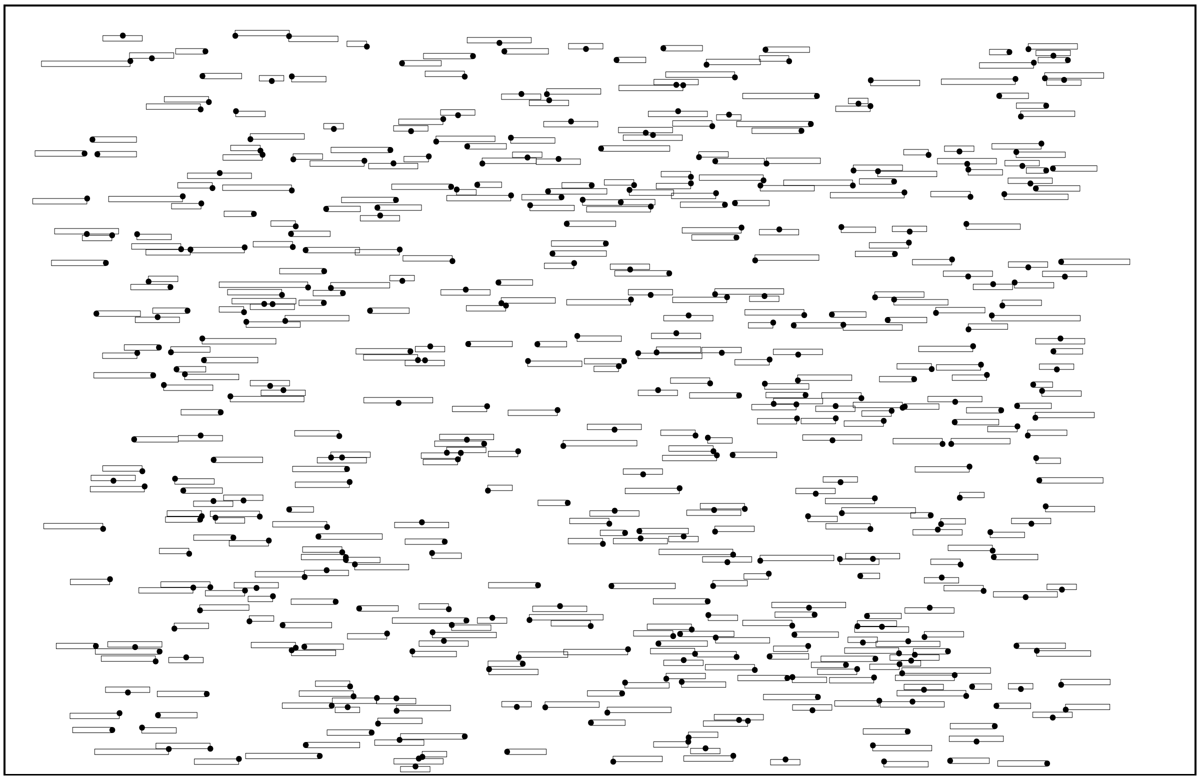

Figure 5: A Solution Label Placement for the R500 Dataset. No Overlapping Labels

[7] A. L. CORCORAN and R. L. WAINWRIGHT, 1996. Reducing Disruption of Superior Building Blocks in Genetic Algorithms, *Proceedings of the 1996 ACM/SIGAPP Symposium on Applied Computing*, ACM Press, pp. 269-276.

[8] A. L. CORCORAN and R. L. WAINWRIGHT, 1995. Using LibGA to Develop Genetic Algorithms for Solving Combinatorial Optimization Problems, in *Practical Handbook of Genetic Algorithms, Applications Volume I*, L. Chambers, (ed.), CRC Press, pp.143-172.

[9] L. DAVIS, (ed.), 1991. *Handbook of Genetic Algorithms*, Van Nostrand Reinhold.

[10] K. A. DE JONG and W. M. SPEARS, 1989. Using Genetic Algorithms to Solve NP-Complete Problems, in *Proceedings of the Third International Conference on Genetic Algorithms*, J. D. Schaffer, (ed.), Morgan Kaufmann, pp. 124-132.

[11] D. E. GOLDBERG, 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley.

[12] S. HIRSCH, 1982. An Algorithm for Automatic Name Placement Around Point Data, *The American Cartographer, 9*, (1), pp. 5-17.

Table 5: Results from 20 GA Executions for R128

a	b	c	Mask	Results
1	0	0	0	1/20
0	1	0	0	0/20
1	1	0	0	3/20
10	1	0	0	2/20
1	10	0	0	0/20
1	0	.0001	0	7/20
10	0	.0001	0	2/20
1	1	.0001	0	4/20
10	1	.0001	0	8/20
1	10	.0001	0	0/20
1	0	0	1	18/20
0	1	0	1	17/20
1	1	0	1	20/20
1	0	.0001	1	20/20
10	0	.0001	1	20/20
1	1	.0001	1	20/20
10	1	.0001	1	20/20
1	10	.0001	1	20/20

[13] C. JONES, 1989. Cartographic Name Placement with Prolog, *IEEE Computer Graphics Applications, 9*, (5), pp. 36-47.

[14] D. E. KNUTH, 1993. *The Stanford GraphBase: A Platform for Combinatorial Optimization*, Addison-Wesley.

[15] J. MARKS and S. SHIEBER, 1993. The Computational Complexity of Cartographic Labeled Placement, Harvard University Center for Research in Computing Technology technical Report TR-05-91.

[16] H. MÜHLENBEIN, 1989. Parallel Genetic Algorithms, Population Genetics and Combinatorial Optimization, in *Proceedings of the Third International Conference on Genetic Algorithms.* J. D. Schaffer, (ed.), Morgan Kaufmann, pp. 416-421.

[17] G. RAWLING, (ed.), 1991. *Foundations of Genetic Algorithms*, Morgan Kaufmann.

[18] T. STARKWEATHER, D. WHITLEY, and K. MATHIAS, 1991. Optimization Using Distributed Genetic Algorithms, in *Parallel Problem Solving from Nature*, H. Schwefel and R. Maenner, (eds.), Springer Verlag, Berlin Germany.

[19] R. TANESE, 1989. Distributed Genetic Algorithms, in *Proceedings of the Third International Conference on Genetic Algorithms*, J. D. Schaffer, (ed.), Morgan Kaufmann, pp. 434-439.

Table 6: Results from 20 GA Executions for R500

a	b	c	Mask	Results	Avg. Num. Overlaps	Avg. Gen. Best	Avg. Gen. Converge
1	0	0	0	0/20	18.45	88.3	97.9
0	1	0	0	0/20	26.05	218.3	226.3
1	1	0	0	0/20	14.15	108.1	116.5
10	1	0	0	0/20	15.00	108.5	113.0
1	10	0	0	0/20	19.90	125.6	134.2
1	0	.00001	0	0/20	16.60	125.1	126.3
10	0	.00001	0	0/20	20.80	98.9	99.6
1	1	.00001	0	0/20	12.15	148.3	150.3
10	1	.00001	0	0/20	13.65	138.8	140.6
1	10	.00001	0	0/20	18.15	160.5	163.5
1	0	0	1	7/20	0.80	23.7	28.9
0	1	0	1	6/20	1.15	31.3	38.0
1	1	0	1	8/20	0.60	28.6	37.4
1	0	.00001	1	10/20	0.50	34.3	40.7
10	0	.00001	1	7/20	0.75	35.6	39.6
1	1	.00001	1	7/20	0.75	42.2	47.1
10	1	.00001	1	3/20	0.85	34.6	41.5
1	10	.00001	1	8/20	0.75	41.3	45.9

[20] D. WHITLEY, T. STARKWEATHER, and D. FUQUAY, 1989. Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator, in *Proceedings of the Third International Conference on Genetic Algorithms.* J. D. Schaffer, (ed.), Morgan Kaufmann, pp. 133-140.

[21] P. YOELI, 1972. The Logic of Automated Map Lettering, *The Cartographic Journal, 9*, (2), pp. 99-108.

Table 7: Results from several PFLP Algorithms using the Standard datasets. Entries show the percent of labels placed without conflict (averaged over 25 runs)

Algorithm	S100	S250	S500	S750	S1000
GA with Masking	100.00	99.98	98.79	95.99	88.96
GA without Masking	100.00	98.40	92.59	82.38	65.70
Simulated Annealing	100.00	99.90	98.30	92.30	82.09
Zoraster	100.00	99.79	96.21	79.78	53.06
Hirsch	100.00	99.58	95.70	82.04	60.24
3-Opt Gradient Descent	100.00	99.76	97.34	89.44	77.83
2-Opt Gradient Descent	100.00	99.36	95.62	85.60	73.37
Gradient Descent	98.64	95.47	86.46	72.40	58.29
Greedy-Depth First	95.12	88.82	75.15	58.57	43.41
Random Placement	84.56	65.63	44.06	29.06	19.53

[22] S. ZORASTER, 1986. Integer Programming Applied to the Map Placement Problem, *Cartographia, 23*, (3), pp. 16-27.