Problem-Independent Schema Synthesis for Genetic Algorithms

Yong-Hyuk Kim, Yung-Keun Kwon, and Byung-Ro Moon

School of Computer Science & Engineering, Seoul National University
Shilim-dong, Kwanak-gu, Seoul, 151-742 Korea
{yhdfly, kwon, moon}@soar.snu.ac.kr

Abstract. As a preprocessing for genetic algorithms, static reordering helps genetic algorithms effectively create and preserve high-quality schemata, and consequently improves the performance of genetic algorithms. In this paper, we propose a static reordering method independent of problem-specific knowledge. One of the novel features of our reordering method is that it is applicable to any problem with no information about the problem. The proposed method constructs a weighted complete graph from the gene distances calculated from solutions with relatively high fitnesses, transforms them into a gene-interaction graph, and finds a gene rearrangement. Extensive experimental results showed significant improvement for a number of applications.

1 Introduction

By the schema theorem, Holland showed that highly fit schemata of short defining lengths and low orders have high probabilities of survival in the traditional genetic framework [19]. High-quality schemata with the above features are called *building blocks*. Building blocks are gene groups with high contribution to the fitnesses that have mutually strong interactions. The performance of a genetic algorithm highly depends on the survival environment and reproducibility of building blocks.

The survival probability of a gene group through crossovers is strongly affected by the positions of genes in the chromosome. Schemata consisting of widely scattered specific positions have poor survival probabilities through crossovers due to their long defining lengths. Thus, the strategy of locating genes significantly affects the performance of genetic algorithms. Inversion is a genetic operator devised for changing the loci of genes dynamically [3]. The efforts to exploit the loci of genes dynamically are called *linkage learning* [18]. Messy genetic algorithm is an example that implicitly pursues dynamic gene repositioning [16].

It has been observed that the performance of genetic algorithms on problems with locus-based encoding can be improved by statically reordering the indices of the genes. The technique of static reordering for genetic algorithms was first suggested in [6] [10], whose basic idea is to reassign the loci of genes in chromosomal representation to help genetic algorithms effectively preserve good schemata. A good reordering also leads to better creation of high-quality schemata than in the original ordering. A number of studies on static reordering of gene positions in locus-based encodings showed performance improvement [6] [8] [10] [29].

However, previous reorderings depend on the specific information of their application [6] [8] [10]. Hence, a new heuristic has to be derived for the reordering of each new problem. In this paper, we describe a static reordering method which is free from problem-specific knowledge. The method requires locus-based encodings for chromosomal representation. We perform experiments on three representative combinatorial optimization problems that are NP-hard [15]: graph bisection, linear arrangement, and traveling salesman problem. Our experiments showed notable improvement when compared against the cases without reordering. In this paper, we use rearrangement, reordering, and preprocessing interchangeably.

The remainder of this paper is organized as follows. In Section 2, we summarize three testbed problems. The genetic framework that we used in this work is described in Section 3. In Section 4, we describe the problem-independent schema preprocessing for general purpose. We present experimental results in Section 5. Finally, we make our conclusions in Section 6.

2 Preliminaries

2.1 Graph Bisection

Let $G = (V, E)$ be an unweighted undirected graph, where V is the set of n vertices and E is the set of e edges. A bisection $\{C_1, C_2\}$ of the graph G satisfies $C_1, C_2 \subset V$, $C_1 \cup C_2 = V$, $C_1 \cap C_2 = \phi$, and $||C_1| - |C_2|| \leq 1$. The cut size of $\{C_1, C_2\}$ is $|\{(v, w) \in E : v \in C_1, w \in C_2\}|$. The graph bisection problem is the problem of finding a bisection with the minimum cut size. The problem has been extensively studied in the past [10] [25] [4] [23]. It is known to be NP-hard [15].

2.2 Linear Arrangement

Let $G = (V, E)$ be an unweighted undirected graph. The linear arrangement problem is the problem of finding a permutation $\sigma : V \to V$ of vertices with the minimum value of $\sum_{(u,v)\in E} |\sigma(u) - \sigma(v)|$. There have been a number of studies for the problem [1] [12] [37]. It is also NP-hard [15].

2.3 Traveling Salesman Problem (TSP)

Let $G = (V, E)$ be a complete graph with weights on the edges. A Hamiltonian cycle of G is a cycle that visits every vertex of the graph exactly once. The traveling salesman problem (TSP) is the problem of finding a Hamiltonian cycle with the minimum weight. TSP is well known to be NP-hard [15]. It has been extensively studied in the past due to its wide applications as well as for its complexity. Genetic algorithms have been applied to TSP with varying degrees of success [24] [32] [21] [38].

```
Preprocess;
Create an initial population;
repeat {
    choose parent1 and parent2 from population;
    offspring = crossover(parent1, parent2);
    local-improvement(offspring);
    replacement(population, offspring);
} until (stopping condition);
return the best solution;
```

Fig. 1. The framework of our hybrid genetic algorithm

3 A Hybrid Genetic Algorithm

A hybrid genetic algorithm is a genetic algorithm (GA) combined with a local improvement heuristic. Some people call it a memetic genetic algorithm [34] [11] [27] [28] [26] [5]. The general framework of hybrid steady-state genetic algorithm is used in our GA as shown in Figure 1. In the following, we describe each part of the GA that we used for this work.

– *Locus-based encoding*: Each solution in the population is represented by a chromosome. A binary encoding is used for the graph bisection problem. A gene has a value '0' or '1' depending on the side that the corresponding vertex belongs to. We use a permutation encoding for the linear arrangement problem. Each gene corresponds to a vertex in the graph and its value means the position in the arrangement. We also use a permutation encoding for the TSP. A gene corresponding a vertex v represents another vertex following vertex v in the Hamiltonian cycle. These encodings, where each gene location has an explicit meaning, are called locus-based encoding. It is necessary to use the locus-based encoding since the preprocessing heuristic presented in Section 4 is applicable only to locus-based encodings.
– *Selection and crossover*: To select two parents, we use a proportional selection scheme where the probability for the best solution to be chosen is four times higher than that for the worst solution. A crossover operator creates a new offspring by combining parts of parents. In the graph bisection problem, we use five-point crossover. After the crossover, an offspring may not satisfy the balance. It selects a random point on the chromosome and changes the required number of 1's to 0's (or 0's to 1's) from that point on. In the permutation encoding, we use the partially matched crossover [17]. There is no duplicated gene value in the offspring and it need not be repaired in case of the linear arrangement problem. However, since it may consist of more than one mutually disconnected subcycle, it may not be a proper Hamiltonian cycle in case of TSP. To resolve this problem, we used the repair algorithm introduced in [8].
– *Local improvement*: Hybrid genetic algorithms have been considered natural in solving a difficult problem to get desirable performance since genetic

1. Generate M solutions with relatively high fitness;
2. Compute distance for each gene pair;
3. Make a gene-interaction graph;
4. Find a gene arrangement;

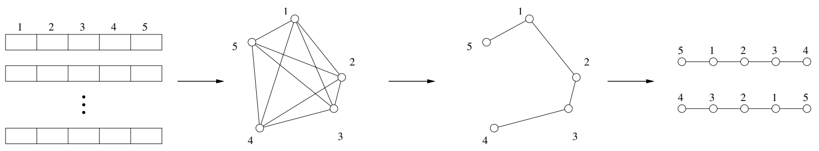

Fig. 2. The structure of problem-independent schema preprocessing

algorithms are not so good at fine tuning near local optima. In this study, we use one of the most basic local improvement heuristic, 2-Opt, which has 2-exchange as its neighbor structure. It is applied to the offspring after crossover in the GA.

– *Replacement and stop condition*: After generating an offspring, the GA replaces the worse of the two parents with the offspring. It is called *preselection* replacement. It stops after a fixed number of generations.

4 Problem-Independent Gene Rearrangement

As mentioned in Section 3, we use locus-based encodings for GA and rearrange the genes. Figure 2 shows the framework of the proposed schema preprocessing. It does not depend on any problem-specific knowledge.

– *Generating high quality solutions*: First, it generates M solutions with relatively high fitness. In this study, we generated 100 solutions using 2-Opt heuristics.
– *Computing the distance for each gene pair*: From the generated solution set, it computes the gene distance between each pair of genes according to its encoding type. Figure 3 describes how to measure the distance $D(g_i, g_j)$ between two genes g_i and g_j. In the figure, $f_l(g_i)$ means the value of gene g_i in the l^{th} solution. As explained in Section 3, binary encoding is used for graph bipartition problem. Sequential permutation encoding and cyclic permutation encoding are used for linear arrangement problem and TSP, respectively. Thus, we get a weighted complete graph with vertices and edges corresponding to genes and gene distances, respectively.
– *Making a gene-interaction graph*: It transforms the obtained weighted graph into unweighted sparse graph called *gene-interaction graph*. We assume that the edge weights in the weighted graph has the Gaussian distribution. To get the gene-interaction graph, it chooses only the heavy-weighted edges with 95% confidence level.

Binary encoding

$$D(g_i, g_j) = \frac{1}{M}\sum_{l=1}^{M} I(f_l(g_i) \neq f_l(g_j))$$

Sequential permutation encoding

$$D(g_i, g_j) = \frac{1}{M}\sum_{l=1}^{M} |f_l(g_i) - f_l(g_j)|$$

Cyclic permutation encoding

$$D(g_i, g_j) = \frac{1}{M}\sum_{l=1}^{M} argmin_k(g_i = f_l^k(g_j) \ or \ g_j = f_l^k(g_i))$$

Fig. 3. Gene-distance measure in locus-based encoding

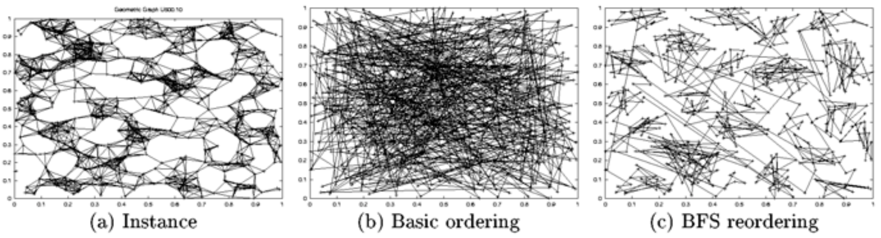
(a) Instance (b) Basic ordering (c) BFS reordering

Fig. 4. Reordering in graph bisection: instance U500.10

– *Finding a gene arrangement*: From the gene-interaction graph, it performs gene rearrangement. Given the set of genes $\{g_1, g_2, \ldots, g_n\}$, a gene rearrangement $\{g_{\sigma(1)}, g_{\sigma(2)}, \cdots, g_{\sigma(n)}\}$ is represented by a bijective map $\sigma : \{1, 2, \ldots, n\} \rightarrow \{1, 2, \ldots, n\}$. Gene v_i is the j^{th} gene in the gene rearrangement if $\sigma(j) = i$. In general, the objective of gene rearrangement is to preserve the clustering structure of the gene-interaction graph. In this paper, we use three general graph-search methods: BFS, DFS, and Max-Adjacency [2]. BFS and DFS reordering performs a breadth first search and a depth first search, respectively, on the input graph starting at a random vertex. The order in which the vertices are visited by the BFS or DFS is used to reorder the vertices. In Max-Adjacency reordering [31], starting at a random vertex, the vertex with the most edges incident to previously ordered vertices is iteratively added to the ordering.

5 Experimental Results

5.1 Graph Bisection

We tested our approach on a total of 21 graphs which consist of three groups of graphs: random graphs (G$n.d$), random geometric graphs (U$n.d$), and caterpillar graphs (cat.n and rcat.n). They have been used in a number of other studies

Table 1. Experimental results in graph bisection problem

Graph	Basic ordering	BFS reordering	DFS reordering	Max-Adj reordering
G500.2.5	52.48	52.38	51.50	51.94
G500.05	220.96	220.58	221.12	220.84
G500.10	630.32	629.88	630.34	629.64
G500.20	1751.48	1749.12	1750.14	1748.84
G1000.2.5	101.08	99.96	101.64	100.02
G1000.05	455.28	454.72	456.20	454.90
G1000.10	1374.04	1374.62	1372.88	1372.18
U500.05	8.66	5.88	4.66	4.72
U500.10	34.24	28.06	26.74	26.38
U500.20	178.28	178.12	178.18	178.36
U500.40	412.00	412.00	412.00	412.00
U1000.05	26.18	15.00	16.24	12.80
U1000.10	72.26	56.82	44.98	48.44
U1000.20	239.50	231.62	228.16	230.32
U1000.40	737.00	737.00	737.00	737.00
cat.352	3.56	1.04	1.28	1.76
cat.702	7.64	8.60	4.36	5.60
cat.1052	12.32	15.84	9.76	8.40
rcat.134	1.00	1.00	1.00	1.00
rcat.554	2.48	1.92	1.52	1.32
rcat.994	3.64	2.44	2.48	2.68

Average over 100 runs.

[20] [7] [10] [4]. Table 1 shows the experimental results. In the table, "BFS," "DFS," and "Max-Adj" represent the gene preprocessing methods. We should note again that this preprocessing is performed on the gene-interaction graphs which are independent of problems differently from previous static reordering methods such as [6] and [10]. We arrange the genes randomly in "Basic ordering." The GAs have the same framework except the preprocessing. Preprocessed GAs significantly outperformed the GA in which genes are arranged randomly. In particular, the preprocessing showed larger performance improvement on geometric graphs and caterpillar graphs. We should also note that we can get improved solutions by using a stronger local optimization heuristic than 2-Opt. Here, we fixed the local optimization with 2-Opt since our major concern is the effect of the suggested reordering method.

To get visual insight into the reordering, we drew a problem instance and the preprocessed order in Figure 4. Figure 4(a) shows the original graph. Figure 4(b) and 4(c) were acquired by drawing segments between all consecutive vertices in each ordering. Of course, randomly ordered genes do not reflect any relation between most pair of vertices. We can observe that BFS helps highly related genes to stay close in the chromosome.

Table 2. Experimental results in linear arrangement problem

Graph	Basic ordering	BFS reordering	DFS reordering	Max-Adj reordering
U500.05	162852.40	158473.18	159004.50	158562.42
U500.10	317022.28	306477.68	308955.18	307840.62
U1000.05	658743.66	643714.92	647381.98	643003.50
U1000.10	1355660.12	1333089.58	1339380.60	1335310.82

Average over 100 runs.

Table 3. Experimental results in TSP

Instance	Basic ordering	BFS reordering	DFS reordering	Max-Adj reordering
lin318	42499.67	42451.82	42404.98	42406.36
pcb442	51886.53	51476.48	51510.38	51489.06
att532	28629.40	28173.80	28209.30	28199.80
rat783	9304.47	9104.72	9111.20	9095.08

Average over 100 runs.

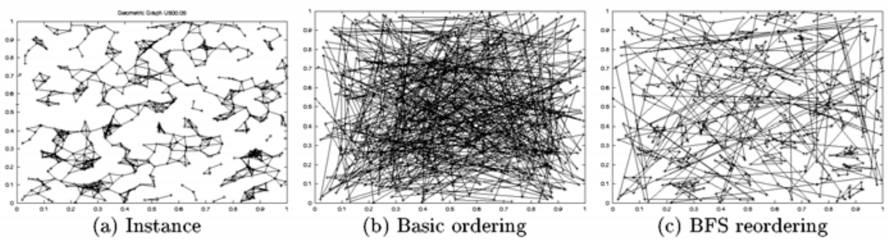

Fig. 5. Reordering in linear arrangement: instance U500.05

5.2 Linear Arrangement

To test our approach on the linear arrangement problem, we used sparse geometric graphs out of the graphs used in Table 1. Table 2 shows the experimental results. The three methods outperformed "Basic ordering" in all instances. In particular, "BFS" showed the best performance on the average. We also drew in Figure 5 the order of genes after a reordering on the linear arrangement problem. We can also observe that highly related genes stay close in the chromosome.

5.3 Traveling Salesman Problem

Table 3 shows the experimental result on four instances of TSPLIB[1]. The results are consistent with the two previous experiments. GAs preprocessed by BFS, DFS, and Max-Adj have more chances to find good solutions than the basic ordering. Figure 6 shows the drawing of the gene orders on an instance att532. We can observe that closely located cities tend to locate closely in the

[1] http://www.iwr.uniheidelberg.de/iwr/comopt/soft/TSPLIB95/TSPLIB.html

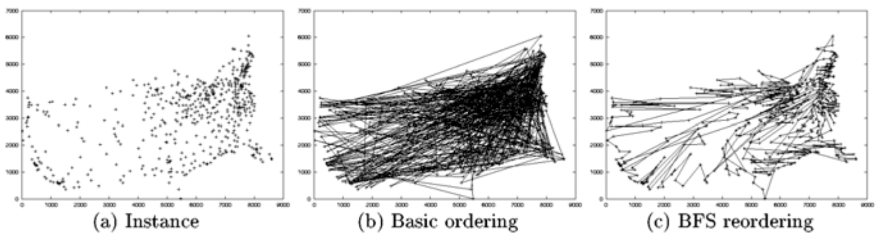

Fig. 6. Reordering in TSP: instance att532

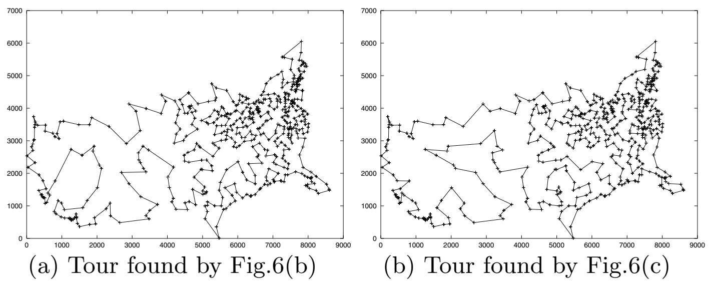

Fig. 7. TSP solutions: instance att532

chromosome with the BFS reordering. We can also observe the effect of reordering by visualizing the TSP tour. Figure 7 shows representative tours of TSP by GAs with the orderings of Figure 6. The GAs found considerably different tours according to their orderings.

6 Conclusions

In this paper, we proposed a static reordering framework of genes in locus-based encodings. It showed consistent performance improvement over genetic algorithms without reordering. One may be able to devise a better reordering, as a result of exploiting problem-specific knowledge, as far as each problem is concerned. The most notable feature of the suggested method is that it does not need any problem-specific information during the reordering process. When a new problem is given for GA, we do not have to devise a new preprocessing heuristic. The only thing we need is a measure of gene interaction for each problem. However, it may not be a big burden since most problem encodings can be classified into a number of representative encodings. Moreover, there exist useful studies on gene interactions [13] [35] [36] [14] [30] [33].

We considered only the linear encoding in this study. Although it is traditional and the most popular encoding, multi-dimensional encodings are also becoming common in the GA community [9] [22]. The proposed reordering framework has a limitation that it can be just applied to the linear encoding. Extending the reordering to multi-dimensional encodings seems to be a topic worth trying.

Acknowledgment. This work was partly supported by Optus Inc. and Brain Korea 21 Project. The RIACT at Seoul National University provided research facilities for this study.

References

1. D. Adolphson and T. Hu. Optimal linear ordering. *SIAM J. Appl. Math.*, 25(3):403–423, 1973.
2. C. Alpert and A. B. Kahng. A general framework for vertex orderings, with applications to netlist clustering. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 63–67, 1994.
3. J. Bagley. *The Behavior of Adaptive Systems Which Employ Genetic and Correlation Algorithms.* PhD thesis, University of Michigan, Ann Arbor, MI, 1967.
4. R. Battiti and A. Bertossi. Greedy, prohibition, and reactive heuristics for graph partitioning. *IEEE Trans. on Computers*, 48(4):361–385, 1999.
5. M.J. Blesa, P. Moscato, and F. Xhafa. A Memetic Algorithm for the Minimum Weighted k-Cardinality Tree Subgraph Problem. In *4th Metaheuristics International Conference*, volume 1, pages 85–90, 2001.
6. T. N. Bui and B. R. Moon. Hyperplane synthesis for genetic algorithms. In *Fifth International Conference on Genetic Algorithms*, pages 102–109, July 1993.
7. T. N. Bui and B. R. Moon. A genetic algorithm for a special class of the quadratic assignment problem. *The Quadratic Assignment and Related problems, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 16:99–116, 1994.
8. T. N. Bui and B. R. Moon. A new genetic approach for the traveling salesman problem. In *IEEE Conference on Evolutionary Computation*, pages 7–12, June 1994.
9. T. N. Bui and B. R. Moon. On multi-dimensional encoding/crossover. In *Sixth International Conference on genetic Algorithms*, pages 49–56, 1995.
10. T. N. Bui and B. R. Moon. Genetic algorithm and graph partitioning. *IEEE Trans. on Computers*, 45(7):841–855, 1996.
11. E. K. Burke, J. P. Newall, and R. F. Weare. A memetic algorithm for university exam timetabling. In *1st International Conference on the Practice and Theory of Automated Timetabling (ICPTAT'95, Napier University, Edinburgh, UK, 30th Aug – 1st Sept 1995)*, pages 496–503, 1995.
12. C. Cheng. Linear placement algorithms and applications to VLSI design. *Networks*, 17:439–464, 1987.
13. Y. Davidor. Epistasis variance: A viewpoint on ga-hardness. In *Foundations of Genetic Algorithms 3*, pages 23–35. Morgan Kaufmann, 1991.
14. Cyril Fonlupt, Denis Robilliard, and Philippe Preux. A bit-wise epistasis measure for binary search spaces. *Lecture Notes in Computer Science*, 1498:47ff., 1998.
15. M. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.
16. D. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex System*, 3:493–530, 1989.
17. D. Goldberg and R. Lingle. Alleles, loci, and the traveling salesman problem. In *First International Conference on Genetic Algorithms and Their Applications*, pages 154–159, 1985.

18. G. R. Harik and D. E. Goldberg. Learning linkage. In *Foundations of Genetic Algorithms 4*, pages 247–262. 1996.
19. J. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.
20. D. S. Johnson, C. Aragon, L. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation, Part 1, graph partitioning. *Operations Research*, 37:865–892, 1989.
21. S. Jung and B. R. Moon. Toward minimal restriction of genetic encoding and crossovers for the 2D Euclidean TSP. *IEEE Transactions on Evolutionary Computation*, 6(6), 2002.
22. A. B. Kahng and B. R. Moon. Toward more powerful recombinations. In *Sixth International Conference on genetic Algorithms*, pages 96–103, 1995.
23. Y. H. Kim and B. R. Moon. A hybrid genetic search for graph partitioning based on lock gain. In *Genetic and Evolutionary Computation Conference*, pages 167–174, 2000.
24. P. Merz and B. Freisleben. Genetic local search for the TSP: New results. In *IEEE Conference on Evolutionary Computation*, pages 159–164, 1997.
25. P. Merz and B. Freisleben. Memetic algorithms and the fitness landscape of the graph bi-partitioning problem. In *Proceedings of the 5th International Conference on Parallel Problem Solving From Nature*, 1998. *Lecture Notes in Computer Science*, 1498:765–774, Springer-Verlag.
26. P. Merz and B. Freisleben. Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE-EC*, 4(4):337, November 2000.
27. Peter Merz and Bernd Freisleben. A comparison of memetic algorithms, tabu search, and ant colonies for the quadratic assignment problem. In *Proceedings of the Congress on Evolutionary Computation*, volume 3, pages 2063–2070. IEEE Press, 6-9 1999.
28. Peter Merz and Bernd Freisleben. Fitness landscapes and memetic algorithm design. In David Corne, Marco Dorigo, and Fred Glover, editors, *New Ideas in Optimization*, pages 245–260. McGraw-Hill, 1999.
29. B. R. Moon and C. K. Kim. A two-dimensional embedding of graphs for genetic algorithms. In *International Conference on Genetic Algorithms*, pages 204–211, 1997.
30. M. Munetomo and D. Goldberg. Identifying linkage by nonlinearity check, 1998.
31. H. Nagamochi and T. Ibaraki. Computing edge-connectivity in multigraphs and capacitated graphs. *Siam J. of Disc. Math*, 5(1):54–66, Feb 1992.
32. Y. Nagata and S. Kobayashi. Edge assembly crossover: A high-power genetic algorithm for the traveling salesman problem. In *7th International Conference on Genetic Algorithms*, pages 450–457, 1997.
33. Martin Pelikan, David Goldberg, and Fernando Lobo. A survey of optimization by building and using probabilistic model. Technical Report 99018, IlliGAL, September 1999.
34. Nicholas J. Radcliffe and Patrick D. Surry. Formal memetic algorithms. In *Evolutionary Computing, AISB Workshop*, pages 1–16, 1994.
35. Colin Reeves and Christine Wright. An experimental design perspective on genetic algorithms. In *Foundations of Genetic Algorithms 3*, pages 7–22. Morgan Kaufmann, 1995.
36. Colin Reeves and Christine C. Wright. Epistasis in genetic algorithms: An experimental design perspective. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 217–224. Morgan Kaufmann, 1995.

37. Y. Saab and C. Chen. An effective solution to the linear placement problem. *VLSI Design Journal*, 2(2):117–129, 1994.
38. D. I. Seo and B. R. Moon. Voronoi quantized crossover for traveling salesman problem. In *Genetic and Evolutionary Computation Conference*, pages 544–552, 2002.