INCORPORATING HEURISTIC INFORMATION INTO GENETIC SEARCH

Jung Y. Suh
Dirk Van Gucht

Computer Science Department
Indiana University
Bloomington, Indiana 47405
(812) 335-6429
CSNET: jysuh@indiana, vgucht@indiana

Keywords: Genetic Algorithms, Heuristics, Optimization Problems, Simulated Annealing, Sliding Block Puzzle, Traveling Salesman Problem.

Abstract

Genetic Algorithms have been shown to be robust optimization algorithms for (positive) real-valued functions defined over domains of the form R^n (R denotes the real numbers). Only recently have there been attempts to apply genetic algorithms to other optimization problems, such as combinatorial optimization problems. In this paper, we identify several obstacles which need to be overcome to successfully apply genetic algorithms to such problems and indicate how integrating heuristic information related to the problem under consideration helps in overcoming these obstacles. We illustrate the validity of our approach by providing genetic algorithms for the Traveling Salesman Problem and the Sliding Block Puzzle.

1. Introduction

Suppose we have an *object space* X and a function $f : X \rightarrow R^+$ (R^+ denotes the positive real numbers) and our task is to find a global minimum (or maximum) for the function f. In this paper, we will concentrate on *genetic algorithms*, a class of adaptive algorithms invented by John Holland [8], to solve (or partially solve) this problem.

Genetic algorithms differ from more standard search algorithms (e.g., gradient descent, controlled random search, hill-climbing, simulated annealing [9] etc.) in that the search is conducted using the information of a *population* of *structures* instead of that of a single structure. The motivation for this approach is that by considering many structures as potential candidate solutions, the risk of getting trapped in a local optimum is greatly reduced.

Genetic algorithms have been applied with great success by De Jong [4] to a wide variety of functions defined over object spaces of the form R^n, i.e., each structure x consists of n real numbers $x[1] \ldots x[n]$. Only recently have there been attempts to apply genetic algorithms to other optimization problems such has the Traveling Salesman Problem (TSP) [6, 7], Bin Packing [13], Job Scheduling [2, 3]. An important observation made by Grefenstette et.al. [7] was that to successfully apply genetic algorithms to such problems, heuristic information has to be incorporated into the genetic algorithm; in particular they proposed a heuristic crossover operator and showed the dramatic improvement as compared to crossover operators which did not such heuristic information. In this paper, we continue the efforts of [7]. In Section 2 we identify several problems which need to be overcome to successfully apply genetic algorithms to optimization problems other than the standard function optimization problems. In Sections 3 and 4 we illustrate the validity of our approach by providing genetic algorithms for the Traveling Salesman Problem and the Sliding Block Puzzle [12].

2. Design Issues of Genetic Algorithms

In this section, we outline the major obstacles in the design of genetic algorithms for optimization problems other than standard function optimization problems and suggest approaches to overcome them†.

2.1. The Representation Problem

As mentioned before, genetic algorithms have almost exclusively been applied to functions defined over object spaces of the form R^n. When we want to solve other optimization problems, such as combinatorial optimization problems, simple parametric representations of the structures can no longer be used. This suggests that the first step towards successfully applying genetic algorithms to these problems is to use a natural representation for the structures of the problem at hand. In particular, we suggest that the choice of such a representation allows for the definition of recombination operators which incorporate heuristic information of the problem. We thus imply that the selection of "good" representations and recombination operators are highly correlated (for a more detailed discussion of these issues we refer to [5]).

2.2. The Selection of Appropriate Recombination Operators and the Importance of Local Improvement

The power of applying genetic algorithms to functions defined over R^n is that the standard recombination operators, crossover and mutation, make intuitive sense in this problem. In other problem domains, however, this is not usually the case. Since the recombination step is critical for the success of a genetic algorithm, it is important to carefully select an appropriate set of recombination operators for such problem domains.

Early research on genetic algorithms [4, 8] was primarily concerned with operators which guarantee that (some) structural information of the structures to which they are applied is preserved. Examples of such recombination operators are the standard crossover, mutation and inversion operators used in function optimization. Grefenstette et.al. argued that such an approach does not carry over with similar success to the traveling salesman problem (TSP). They showed that considering recombination operators (in their case, only crossover operators) that merely preserve structural information results in poorly performing genetic algorithms (i.e., not much better than random search). They discovered however that it is possible and natural to incorporate heuristic information about the TSP into the crossover

† It should be noted that De Jong [5] and Grefenstette et.al. [7] already identified some of these problems.

operator and still maintain its fundamental property, namely: preservation of structural information of the structures to which the operator is applied. This resulted in a fairly successful genetic algorithm for the TSP, but certainly not an algorithm that is competitive with other approximation algorithms for the TSP (see [10]).

We claim that it is often the case that additional improvements can be gained if one incorporates even more heuristics about the problem into the recombination step of a genetic algorithm. Often, heuristics about problems are incorporated into algorithms in the form of operators which iteratively perform *local improvements* to candidate solutions. Examples of such operators can be found in gradient descent algorithms, hill climbing algorithms, simulated annealing, etc. We will argue that it is usually straightforward, and in fact, we think, essential if a competitive genetic algorithm is desired, to incorporate such local improvement operators into the recombination step of a genetic algorithm. An additional advantage of this approach is that it suggests a natural technique of blending genetic algorithms with more standard optimization algorithms.

2.3. Avoiding Premature Convergence

One of the major difficulties with genetic algorithms (and in fact with most search algorithms) is that sometimes premature convergence, i.e. convergence to a suboptimal solution, occurs. It has been observed that this problem is closely tied to the problem of losing diversity in the population. One source of loss in diversity is the occasional appearance of a "super-individual" which in a few generations takes over the population. One way of avoiding this problem is to change the selection procedure, as was demonstrated by Baker [1]. Another source of loss of diversity results from poor performance of recombination operators in terms of sampling new structures. To overcome such problems, we claim that the recombination operators should be selected carefully so that they can offset each others vulnerabilities (for a more detailed discussion of these issues, we refer to [1, 4, 7, 8]).

3. The Traveling Salesman Problem

In this section we show that solutions to the problems raised in Section 2 enable us to develop a genetic algorithm for the TSP.

The TSP is easily stated: Given a complete graph with N nodes, find the shortest Hamiltonian tour through the graph (in this paper, we will assume Euclidean distances between nodes). For an excellent discussion on the TSP, we refer to [10].

The object space X obviously consists of all Hamiltonian tours (tours, for short) associated with the graph, and f, the function to be optimized, returns the length of a tour.

As in [7], we represent a tour by its *adjacency representation*. It turns out that this representation allows us to easily formulate and implement heuristic recombination operators. In the adjacency representation, a tour is described by a list of cities. There is an edge in the tour from city i to city j if and only if the value in ith position of the adjacency representation is j. For example, the tour shown in Figure 1 is represented as (3 1 5 2 4).

We now turn to the most critical step of the design: the selection of appropriate recombination operators. We elected to have two such operators. The first operator is a slight modification of the heuristic crossover operator introduced by Grefenstette et.al. [7]. This operator constructs an offspring from two parent tours as follows: Pick a random city as the starting point for the offspring's tour. Compare the two edges leaving the starting city in the parents and choose the shorter edge. Continue to extend the partial tour by choosing the shorter of the two edges in the parents which extend the tour. If the shorter parental edge would introduce a cycle into the partial tour, check if the other parental edge introduces a cycle. In case the second

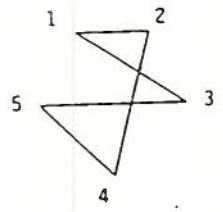


Figure 1. The tour (3 1 5 2 4).

edge does not introduce a cycle, extend the tour with this edge, otherwise extend the tour by a random edge. Continue until a complete tour is generated. It is in the selection of the shorter edges that we exploit heuristic information about the TSP; indeed, it seems likely that a good tour will contain short edges. The effect of the heuristic crossover operator is to "glue" together "good" (i.e., short) subpaths of the parent tours. (Notice also that it preserves structural information about the parent tours.) The problem with the heuristic crossover operator is that it leaves undesirable crossings of edges as illustrated in Figure 2 (see also Appendix 1). In other words, the heuristic crossover operates performs poorly when it comes down to fine-tuning candidate solutions. This motivated us to introduce a second recombination operator.

Whereas the heuristic crossover operator can be thought of as a global operator, the second recombination operator has a more local behavior and thus qualifies as a local improvement operator. It was introduced by Lin and Kernighan [11] and is called the 2-opt operator. The 2-opt operator randomly selects two edges (i_1, j_1) and (i_2, j_2) from a tour (see Figure 2) and checks if $ED(i_1, j_1) + ED(i_2, j_2) > ED(i_1, j_2) + ED(i_2, j_1)$ (ED stands for Euclidean distance). If this is the case, the tour is replaced by removing the edges (i_1, j_1) and (i_2, j_2) and replacing them with the edges (i_1, j_2) and (i_2, j_1) (see Figure 3). Actually, we use a more subtle variation of the 2-opt operator, inspired by recent work on *simulated annealing* for the TSP by Kirkpatrick et.al. [9]. In this variation there is a (small) probability (depending on a slowly decreasing temperature) that when $ED(i_1, j_1) + ED(i_2, j_2) \leq ED(i_1, j_2) + ED(i_2, j_1)$, the original tour is replaced using the previously described transformation †.

To make the description of our genetic algorithm complete, we need to describe three parameters:

a. `crossover rate`: this parameter indicates the amount of structures in the population which will undergo crossover.
b. `local improvement rate`: this parameter indicates the amount of structures in the population which will undergo 2-opt operations.
c. `2-opt rate`: if the graph under consideration has N nodes, each structure which is selected to undergo local improvement will undergo ($N \times 2 -$ `opt rate`) 2-opt operations per generation.

The algorithm was stopped when the majority of the tours in the population were identical.

We tried our algorithm on a wide variety of (euclidean)

† It should be noted, however, that the performance of the algorithm with the simple 2-opt is usually only slightly worse than an algorithm that uses the simulated annealing version.

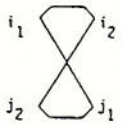


Figure 2. Tour with edges (i_1, j_1) and (i_2, j_2).

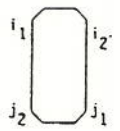


Figure 3. Tour with edges (i_1, j_2) and (i_2, j_1).

traveling salesman problems. In Figure 4, we show a selection of such problems.

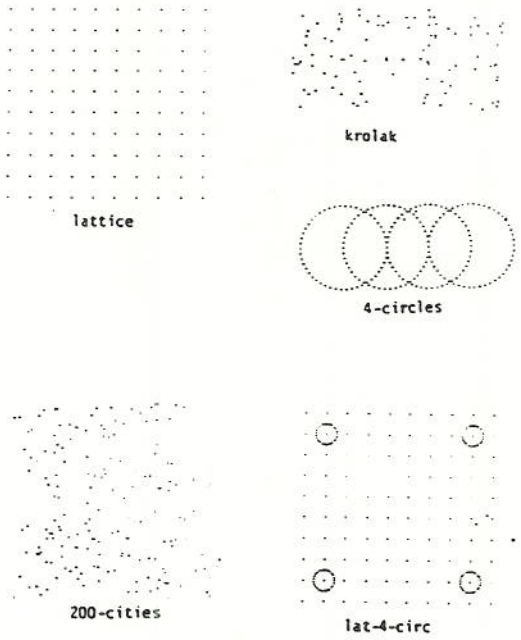


Figure 4. Five Traveling Salesman Problems

In Table 1, we show the results obtained by the algorithm of Grefenstette et.al. [7] for the following parameter settings: `initial population` = 100 randomly chosen tours, `crossover rate` = 50%, `local improvement rate` = 0%, `2-opt rate` = not applicable. In Table 2, we show the results of a genetic algorithms which uses the local improvement operator with the following parameter settings: `population size` = 100 structures, `crossover rate` = 50%, `local improvement rate` = 50%, `2-opt rate` = 0.1.

Table 1

Genetic Algorithm Without Local Improvement

TSP	Nodes	Optimum	Our Solution	Generations
krolak[10]	100	21282	25691	104
lattice	100	100	104.9	209
4-circles	200	24.67	39.0	300
lat-4-circ	200	112.56	139.2	286
200-cities	200	?	192.8	376

Table 2

Genetic Algorithm With Local Improvement

TSP	Nodes	Optimum	Our Solution	Generations
krolak	100	21282	21651	679
lattice	100	100	100	188
4-circles	200	24.5	24.5	218
lat-4-circ	200	112.56	113.3	669
200-cities	200	?	153.6	946

In Figure 5, we show the (best) tours obtained and the generation in which they were first found by the algorithm which uses local improvement. Clearly, the addition of a local improvement technique improves the performance (measured in terms of the tour length of the best tour obtained) of the algorithm dramatically. (In terms of extra resources, on average, the algorithm using local improvement required about 2.2 times more generations to obtain its best structure.) In fact, the results obtained by our algorithm are very competitive, again, in terms of the tour length of the best tour obtained, compared to results reported in the literature for other approximation algorithms for the TSP [9, 10]. (In Appendix 1, we give additional results.)

4. The Sliding Block Puzzle (SBP)

We now describe how the approach described in Section 2 can be used in the design of a genetic algorithm for a problem which is not usually thought of as a function optimization problem: the Sliding Block Puzzle [12]†.

Consider the initial board of the puzzle shown in Figure 6 and let the board shown in Figure 7 be a goal board (the empty tile is represented by the symbol 0).

1	2	3	4
5	6	7	8
9	0	10	11
12	13	14	15

Figure 6. The Initial Board of a Sliding Block Puzzle.

1	2	3	4
6	9	0	8
5	10	7	11
12	13	14	15

Figure 7. A Goal Board a Sliding Block Puzzle.

† In our implementation, we used 3×3 and 4×4 puzzles.

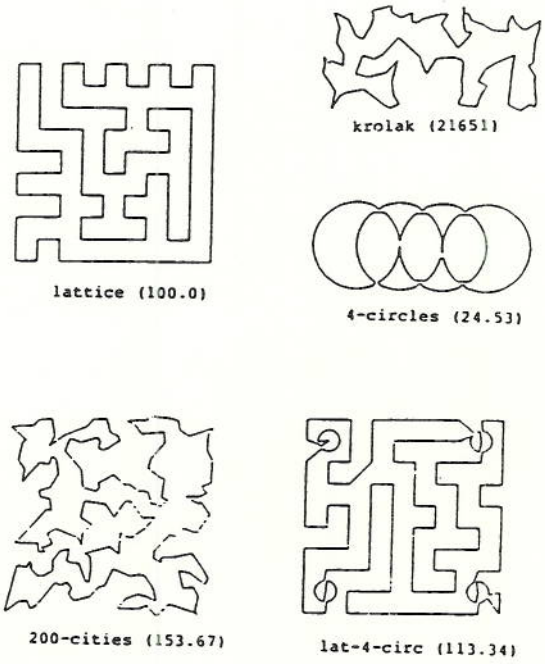


Figure 5. Best Tours Obtained by a Genetic Algorithm with Local Improvement.

The objective of the SBP is to reach the goal board starting from the intial board using a sequence of valid moves. There are four basic moves:

L: move the empty tile to the left.
U: move the empty tile upwards.
R: move the empty tile to the right.
D: move the empty tile downwards.

The only precondition required for applying a move is that it should not move the empty tile out of the board. For example, a sequence which transforms the board shown in Figure 6 into the board shown in Figure 7 is (L,U,R,D,R,U).

In order to apply genetic algorithms to the SBP, we need to formulate the problem as a function optimization problem. The object space X consists of all valid sequences of moves applicable to the initial board. Notice that the structures in X do not have a fixed length representation. Other research with genetic algorithms on object spaces with structures having variable length representations was done by Smith [14] who implemented a machine learning system (LS-1) using structures corresponding to production system programs.

In order to define f, the function to be optimized, we need to introduce some extra notation. We will denote the initial board by IB and the goal board by GB. Let $(x_1, \ldots, x_n)$ be a sequence of valid moves (i.e., an element of X), we denote by $IB(x_1, \ldots, x_n)$ the board which is obtained by applying the sequence of moves $(x_1, \ldots, x_n)$ to IB.

Consider the boards $IB(x_1, \ldots, x_n)$ and GB. For each tile (except the empty tile) in $IB(x_1, \ldots, x_n)$, compute the *Manhattan distance* between the tile's position in $IB(x_1, \ldots, x_n)$ and its position in GB. We define the $performance(x_1, \ldots, x_n)$ as the sum of all these Manhattan distances.

In our first attempt, we defined

$$f(x_1, \ldots, x_n) = performance(x_1, \ldots, x_n)$$

, but quickly discovered that a much better measure is

$$f(x_1, \ldots, x_n) = min\{performance(x_1, \ldots, x_i) | 1 \leq i \leq n\},$$

i.e., the value of a structure $(x_1, \ldots, x_n)$ is defined as the performance of the sub-sequence $(x_1, \ldots, x_i)$ whose corresponding intermediate board $IB(x_1, \ldots, x_i)$ comes closest to GB (it should be noted that computing $f(x_1, \ldots, x_n)$ can be done in $O(n)$). It should be clear that whenever $f(x_1, \ldots, x_n) = 0$, the sequence $(x_1, \ldots, x_n)$ contains a subsequence $(x_1, \ldots, x_i)$ which is a solution to the SBP. We now turn to the selection of the recombination operators.

The crossover process here is similar to that in the TSP. Suppose that two sequences of operators are given. We pick the first operator from each sequence. Apply each operator to the initial board to see which operator yields a new board closer to the goal board. Choose with high probability the operator which yields the closer board, i.e., the one with the better performance. Notice that it is here that we employ heuristic information about the Sliding Puzzle Problem, indeed, it seems likely that we should try to obtain intermediate configurations that get closer to the goal board. We do not always choose the better operator, however, because this may eventually lead to a bad sequence whose performance we can not improve as long as it starts with that particular operator. In short, we could get stuck in the local optimum. However, it is our assumption that in general it is more likely the case that selecting the better operator will contribute to constructing a good sequence. In case the two operators have the same performance, pick any of them randomly. Once the operator is chosen, it becomes the first operator of the new sequence and the board is updated accordingly. Now we pick the second operators of each sequence. Again, we will take the one with the better performance. It may however be the case that one or both of them is no longer legal, i.e., it pushes the empty tile off the edge of the board. This is possible because the operator chosen for the new sequence is not necessarily the one which preceded the current two operators. In case only one of the operators is illegal, choose the one which is legal. Otherwise, randomly generate a legal one. It becomes the second operator of the new sequence. Again update the board. This process is repeated until we reach the end of the two sequences.

The local improvement process is performed on a single structure. First randomly pick m positions ($0 \leq m \leq n$) in the sequence. For the left-most position, make the corresponding board arrangement by applying the operators in the sequence preceding this position. Randomly generate a legal operator in that position and check if this new operator is acceptable by comparing it with the old operator using the boltzman distribution test (i.e., perform simulated annealing). This test goes as follows: Accept the new operator if it yields a board closer to the goal board than the old one does. Otherwise, accept it with the probability according to a boltzman distribution. If the temperature in the boltzman distribution is high, the new operator will be accepted with high probability, even if its performance is bad. If the temperature is low, the operator is accepted with less probability. (Temperature deceases exponentially. We chose the temperature $T = T_0 \rho^{gen}$, where T_0 is a initial temperature, $0 < \rho < 1$ and gen is the number of generations). If the new operator is accepted, it replaces the old one in the sequence. If not, the old one is kept. Now proceed scanning the given sequence to the right until the second initially chosen position at which local improvement will be performed is reached, checking if, along the way, any of the operators should be updated due to the replacement of the previous

operator. If an operator has to be updated, replace it with a legal one. Repeat this process for all the initially chosen positions at which local improvement will be performed. Upon completion of this process, the whole new sequence is compared with the initial sequence and accepted according to the boltzman distribution test.

To overcome the difficulties of the non-fixed length representation of sequences, the genetic algorithm is embedded in a loop which periodically extends the length of the structures of the population under consideration. For example, we may start out with a population of sequences of length 10, apply the genetic algorithm to this population until a steady state is reached, then extend the sequences by randomly adding a fixed amount of valid moves to each sequence in the steady state population and resume the GA on the new population until a new steady state is reached. This extension process is continued until a solution or near-optimum solution is obtained.

The results of initial experiments are promising. Consistently we find solutions or near-optimum solutions using little computational time. In the case of the 3×3 sliding puzzle games, we always found a solution using the following parameter settings: `initial population` = 20 randomly chosen structures of length 10, the extension of the structures when the algorithm reached a steady state was done in chunks of length 5, `crossover rate` = 70%, `local improvement rate` = 30%. In the case of 4×4 sliding puzzle games, the boards we reached (for difficult cases) are within 4 to 5 in Manhattan distance from the goal configuration, hence the majority of tiles are in place. Unfortunately, we seem to have trouble generating exact solutions for these puzzles. In Figure 8 we show a typical case of such a "difficult" puzzle. For this example, it takes 31 moves to transform the initial board into the goal board. In Figure 9, we show a board reached by a genetic algorithm using the following parameters: `initial population size` = 50 randomly chosen structures of length 20, the extension of the structures when the algorithm reached a steady state was done in chunks of 10, `crossover rate` = 70%, `local improvement rate` = 30%. This board has a Manhattan distance of 4 from the goal board. In Appendix 2, we report additional results.

1	2	3	4
5	6	7	8
9	0	10	11
12	13	14	15

10	1	6	2
3	8	15	4
9	7	13	11
5	0	12	14

Figure 8. The Initial and Goal Board of a 4×4-SBP.

10	1	6	2
3	8	15	4
5	7	13	14
9	0	12	11

Figure 9. A Board Obtained by a Genetic Algorithm.

The fact that we only reached near-optimum solution was not totally unexpected since it is well known that genetic algorithms are excellent in finding near-optimum solutions (see the TSP above and [4]), but are usually not powerful enough to find exact solutions. However, we plan to fine-tune our current algorithm and expect, at least for the SBP, to overcome this problem.

5. Conclusion

We have identified several problems in generalizing genetic algorithms to optimization problems other than the standard function optimization problems. By addressing these problems we designed genetic algorithms for two well-know problems: the Traveling Salesman Problem, an example of a combinatorial optimization problem, and the Sliding Block Puzzle, an example of a puzzle problem studied in Artificial Intelligence. It turned out that the selection of a natural representation and the selection of heuristically motivated recombination operators is critical in the design of robust genetic algorithms for such problems. We believe that our approach is quite general and can be applied to many related problems.

Finally, it is worthwile to mention that the "operator-oriented" approach used in the SBP is easily generalized to many other problems, thus rendering genetic algorithms applicable to such problems. As an example, in the standard function optimization problem (assuming that the arguments to the function are represented in binary code), we could define the following operators:

`Set(i,1)`: set the i-th position in the bitstring 00...00 to 1

and design heuristically motivated recombination operators which work on sequences of such operations. A similar approach could be taken for the TSP.

Acknowledgements

We would like to thank the referees for their insightful comments and criticisms which helped us to improve the paper.

6. References

[1] J. Baker," Adaptive Selection Methods for Genetic Algorithms", Proc. of an Int'l Conf. on Genetic Algorithms and Their Applications, pp. 101-111 (July 1985).

[2] L. Davis, "Job Shop Scheduling with Genetic Algorithms", Proc. of an Int'l Conf. on Genetic Algorithms and Their Applications, pp. 136-140 (July 1985).

[3] L. Davis, "Applying Adaptive Algorithms to Epistatic Domains", Proc. of 9th IJCAI, pp. 162-164 (Aug 1985).

[4] K.A. De Jong, "Adaptive System Design: a Genetic Approach", *IEEE Trans. Syst., and Cyber. Vol. SMC-10(9)*, pp. 556-574 (September 1980).

[5] K.A De Jong, "Genetic Algorithms: a 10 Year Perspective", Proc. of an Int'l Conf. on Genetic Algorithms and Their Applications, pp. 169-177 (July 1985).

[6] D.E. Goldberg and R. Lingle, "Alleles, Loci, and the Traveling Salesman Problem", Proc. of an Int'l Conf. on Genetic Algorithms and Their Applications, pp. 154-159 (July 1985).

[7] J.J. Grefenstette, R. Gopal, B.J. Rosmaita and D. Van Gucht, "Genetic Algorithms for the Traveling Salesman Problem", Proc. of an Int'l Conf. on Genetic Algorithms and Their Applications, pp. 160-168 (July 1985).

[8] J. Holland, *Adaptation in Natural and Artificial Systems*, Univ. of Michigan Press, Ann Arbor (1975).

[9] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, "Optimization by Simulated Annealing", *Science Vol. 220*(4598), pp. 671-680 (May 1983).

[10] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys (Ed), *The Traveling Salesman Problem*, John Wiley & Sons Ltd (1985).

[11] S. Lin and B.W. Kernighan, "An Effective Heuristic Algorithm for the Traveling Salesman Problem", *Operations Research* 1972, pp. 498-516.

[12] N.J. Nilsson, *Principles of Artificial Intelligence*, Tioga Publishing Company Palo Alto, California (1980).

[13] D. Smith, "Bin Packing With Adaptive Search", Proc. of an Int'l Conference on Genetic Algorithms and Their Applications, pp. 202-206 (July 1985).

[14] S.F. Smith, "Flexible Learning of Problem Solving Heuristics Through Adaptive Search", Proc. of 8th IJCAI (Aug. 1983).

APPENDIX 1

In this appendix we give some additional results concerning the Traveling Salesman Problem. The following is a description of the essential parameters and some terminology used in our experiments.

- **Population Size** : The number of tours in a population at any given time.
- **Structure Length** : The tour length.
- **Crossover Rate** : The portion of population undergoing cross-over. The rest will undergo local improvement.
- **DECRATIO** : If T is the current temperature used in annealing scheme, DECRATIO * T will be the new temperature.
  ** NOTE: the initial temperature is the 1% of average deviation of the initial population (0-th generation).
- **Trial** : Each new computation of the tour length of tour counts as a trial.
- **INTERV** : the number of trials after which the temperature is updated.
- **2-opt Rate** : (2-opt Rate * Structure length) yields the number of 2-opt operations on a structure undergoing local improvement.

How to read a table.

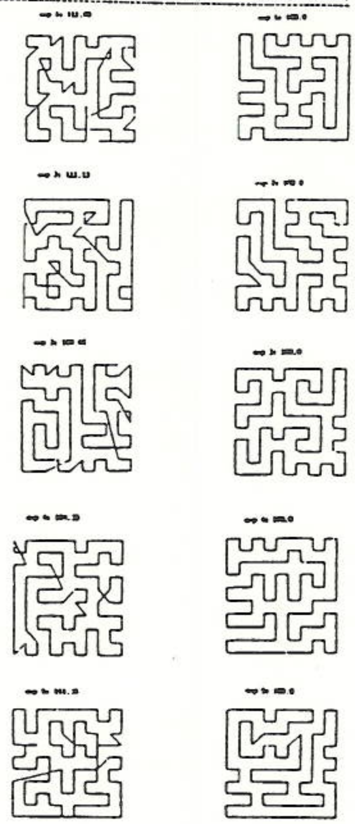
Local Rate		0 %	10 %
lattice	exp. 1	111.1 (124, 6203)	100 (190:16910)
	exp. 2	100.6 (114, 5777)	100.0 (200:17706)
	exp. 3	104.9 (209:10545)	100 (207:10065)
	exp. 4	111.3 (139, 7013)	100 (237:22530)
	exp. 5	111.6 (125, 6346)	100.0 (163:13650)

The above table gives the results for the lattice TSP. We conducted 5 experiments using two different 2-opt rates.
0 % Local Rate means that we are running cross-over only without using local improvement.
The description "exp. 1" stands for experiment 1.
The notation nnn.nn (ggg, tttt) indicates that we obtained the solution of length 111.1 after ggg generations which took tttt trials.

1. Lattice

Population Size = 100
Structure Length = 100
Crossover Rate = 0.5
DECRATIO = 0.875
INTERV = 300

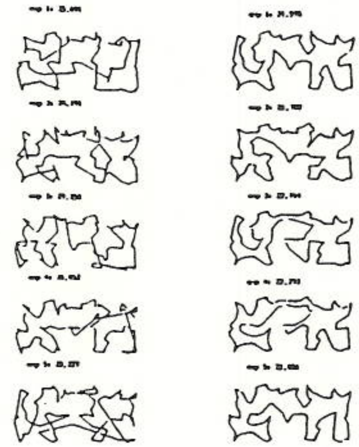
2-opt Rate		0 %	10 %
lattice	exp. 1	111.1 (124, 6203)	100 (190:16910)
	exp. 2	100.6 (114, 5777)	100.0 (200:17706)
	exp. 3	104.9 (209:10545)	100 (207:10065)
	exp. 4	111.3 (139, 7013)	100 (237:22530)
	exp. 5	111.6 (125, 6346)	100.0 (163:13650)



2. [illegible]

Population Size = 100
Structure Length = 100
Crossover Rate = 0.5
DECRATIO = 0.875
INTERV = 300

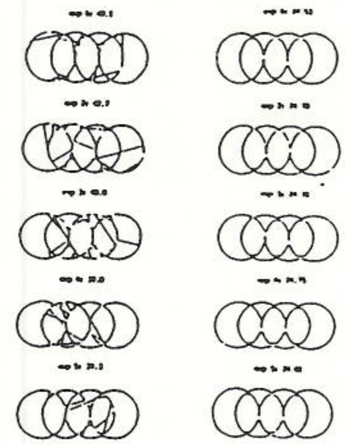
2-opt Rate		0 %	10 %
[illegible]	exp. 1	25691 (104, 5390)	22293 ([illegible])
	exp. 2	26191 (236:11055)	22719 ([illegible])
	exp. 3	27259 (199, 9750)	21702 ([illegible])
	exp. 4	26952 (111, 5602)	21976 (627:49992)
	exp. 5	29227 (117, 5941)	21651 ([illegible])



3. 4 Circles

Population Size = 100
Structure Length = 200
Crossover Rate = 0.5
DECRATIO = 0.875
INTERV = 300

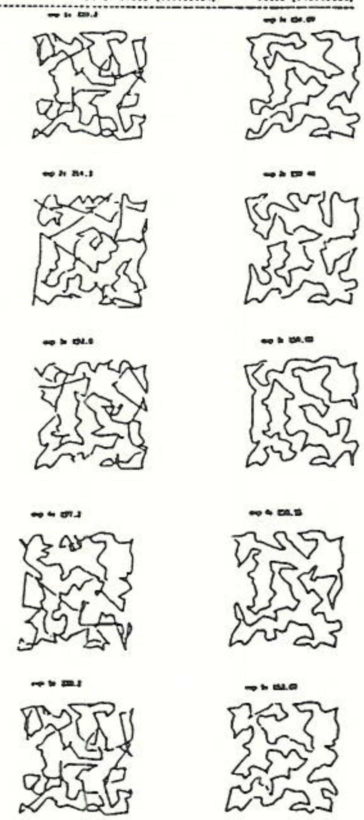
2-opt Rate		0 %	10 %
4cir	exp. 1	40.1 (255:12006)	34.5 ([illegible])
	exp. 2	42.7 (176, [illegible])	34.7 ([illegible])
	exp. 3	40.0 (173, [illegible])	34.7 ([illegible])
	exp. 4	39.0 (300:15060)	34.7 ([illegible])
	exp. 5	39.3 (290:15055)	34.6 ([illegible])



4. 200 Random Points

Population Size = 100
Structure Length = 200
Crossover Rate = 0.5
DECRATIO = 0.875
INTERV = 300

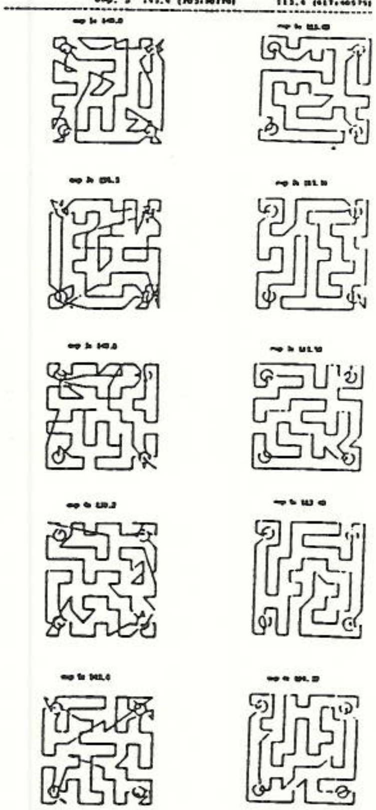
2-opt Rate		0 %	10 %
200	exp. 1	220.2 (216:10034)	154.0 ([illegible])
	exp. 2	219.3 (330:14590)	150.4 ([illegible])
	exp. 3	192.0 ([illegible])	150.0 ([illegible])
	exp. 4	199.2 ([illegible])	150.5 ([illegible])
	exp. 5	220.2 (216:10034)	152.6 (946:72553)



5. Lattice and 4 Circles

Population Size = 100
Structure Length = 200
Crossover Rate = 0.5
DECRATIO = 0.875
INTERV = 300

2-opt Rate		0 %	10 %
[illegible]	exp. 1	140.0 ([illegible])	113.6 ([illegible])
	exp. 2	155.3 ([illegible])	113.2 ([illegible])
	exp. 3	140.0 ([illegible])	113.5 ([illegible])
	exp. 4	139.3 ([illegible])	114.3 ([illegible])
	exp. 5	145.4 ([illegible])	113.6 ([illegible])



APPENDIX 2

In this appendix, we show some of the results obtained for two 3x3 puzzles and one (difficult) 4x4 puzzle.

Population Size : same as in appendix 1
Percent of Population undergoing
cross-over : same as in appendix 1
simulated annealing : same as in appendix 1

Annealing Schedule :

temperature = (initial average deviation) * (0.975 ** gen)

where gen denotes the number of generations and ** denotes: raise to the power.

Note: The Local Improvement Rate is 20 % here but is not explicitly shown.

Extension Schedule

generation	length
0 - 5	10
after 5	20

This table indicates that, between the 0-th and the 5-th generation, the length of the structures is 10. After the 5-th generation,

the structures are extended to have length 20.

1. 3 by 3 Gliding Puzzle

Population Size : 20
Percent of Population
undergoing cross-over : 70 %
undergoing simulated annealing : 30 %

Annealing Schedule: Temperature = (initial average deviation) * (0.975 ** gen)
where gen denotes the generation.

Extension Schedule:

generation	length
0 - 5	10
after 5	20

Note: 0 stands for the empty tile.

init board	goal board	distance
** task 1 **		
1 2 3 4 0 5 6 7 8	1 2 5 6 3 0 4 7 0	6
** task 2 **		
1 2 3 4 0 5 6 7 8	2 3 5 1 4 0 0 6 7	8

Best Structure obtained in each experiment

experiment #	distance from goal board	number of moves taken	number of trials taken
** task 1 **			
1	0 (solved)	14	less than 300
2	0 (solved)	14	less than 300
** task 2 **			
1	0 (solved)	8	less than 200
2	0 (solved)	8	less than 200

2. 4 by 4 Gliding Puzzle

Population Size : 30
Percent of Population
undergoing cross-over : 60 %
undergoing simulated annealing : 20 %

Annealing Schedule: Temperature = (initial average deviation) * (0.975 ** gen)
where gen denotes the generation.

Extension Schedule:

generation	length
0 - 10	20
11 - 20	30
after 20	40

Note: 0 stands for the empty tile.

init board	goal board	distance
1 2 3 4 5 6 7 8 9 0 10 11 12 13 14 15	10 1 6 2 3 8 15 4 9 7 13 11 5 0 12 14	27

Best Structure obtained in each experiment

experiment	distance from goal board	number of moves taken	number of trials taken
1	4	29	less than 2000
2	5	30	less than 2000
3	4	31	less than 2000