Adapting the Genetic Algorithm to the Travelling Salesman Problem

Author
Pullan, Wayne

Published
2003

Conference Title
The 2003 Congress on Evolutionary Computation CEC 2003

DOI
https://doi.org/10.1109/CEC.2003.1299781

Copyright Statement
Copyright 2003 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

Downloaded from
http://hdl.handle.net/10072/1782

Link to published version
http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=9096
Adapting the Genetic Algorithm to the Travelling Salesman Problem

Wayne Pullan
School of Information Technology,
Griffith University, Gold Coast,
Qld., 4215, Australia
Email: w.pullan@griffith.edu.au

Abstract- The combination of local optimisation heuristics and genetic algorithms has been shown to be an effective approach for finding near-optimum solutions to the Travelling Salesman Problem (TSP). In problem domains where the problem can be represented geometrically, such as networks and chemical structures, the combination of local optimisation operators and phenotype genetic operators has also been an effective approach. This paper evaluates the combination of local optimisation heuristics and phenotype genetic operators when applied to the TSP. The local optimisation heuristics reduce the search domain, while the phenotype genetic operators eliminate the creation of invalid tours and also assist the generation of sub-optimal schema. The implementation of the genetic algorithm is described and results presented.

1 Introduction

The Travelling Salesman Problem (TSP) [1] consists of a number of nodes, with the distances between them given. The goal is to find the minimum length closed tour that visits each node exactly once and returns to the starting node. That is, the TSP, for \( n \) nodes, is the global optimisation problem:

\[
\min f(x) = \sum_{i=1}^{n-1} d_{x_ix_{i+1}} + d_{x_{n}x_1},
\]

where \( x \) is a vector containing a permutation of the set \( \{1, 2, \ldots, n\} \), \( x_i \) is the \( i \)th element of \( x \) and \( d_{ij} \) is the distance between nodes \( i \) and \( j \).

The TSP has become a standard benchmark for combinatorial optimisation methods that attempt to find near optimal solutions for NP-hard problems. Several methods, usually based on search heuristics, have been applied including local search [2], simulated annealing [3], tabu search [4] and neural networks [5]. While some of these methods have been relatively successful in producing good results, better results seem to require the combined efforts of several methods, particularly for large TSPs. For example, near optimal solutions have been obtained by combinations of local search algorithms and simulated annealing [6] and by combinations of local search and genetic algorithms [7], [8].

The pure Genetic Algorithm (GA) has a poor performance record when applied to the TSP. This has motivated researchers to construct GA based approaches that incorporate TSP specific heuristics including, tour construction heuristics for generating the initial GA population, tour improvement heuristics for producing local optima, and special crossover operators designed for the characteristics of the GA [7]. These studies seem to indicate that the best way to obtain results comparable to good conventional local search methods is to incorporate TSP specific knowledge into a GA.

The TSP is a geometric problem in that it can be represented as a graph. One of the problems in implementing a GA for the TSP is encapsulating structural knowledge into the genotype representation so that the genetic operators have a higher probability of generating fitter individuals. Recently, phenotype genetic algorithms have been successfully applied to geometry based problems such as networks [9], clusters of atoms [10] and molecules [11], and molecular structures [12]. These genetic algorithms were innovative in that they restricted the search domain through the use of local (deterministic) optimisers and also utilised innovative genetic crossover operators operating in the phenotype domain rather than the more usual genotype domain. A measure of the success of these phenotype genetic algorithms is the well-known and well-studied Lennard-Jones atomic cluster problem where all pairs of inert atoms in a cluster interact via a Lennard-Jones pair potential and the goal is find the configuration of atoms which gives the minimum energy for the cluster. From [13], the number of local minima for this problem scales as \( m(n) = \exp(-2.5176 + 0.3572n + 0.0286n^2) \) where \( n \) is the number of atoms. This predicts that the potential energy surface for a 100 atom cluster has of the order of \( 10^{40} \) local minima! A large number of optimisation techniques have been applied to this problem but, of particular relevance to this study, Hartke [14] used a genetic algorithm to find global minimum energy structures for four atoms. An encoding method, using just four parameters to specify the four atom configuration, allowed a standard GA to find the global minimum energy. Mestres and Scuseria [15] also used a GA to find global minimum energy configurations for Lennard-Jones atomic clusters of up to 13 atoms and Gregurick et al. [16] employed a modified deterministic/stochastic genetic algorithm to find all currently accepted global minima for Lennard-Jones atomic clusters in the range 2, \ldots, 29. However their technique did require a seed growth method for generating initial structures for all clusters containing more than 20 atoms. More recently, Deaven et al. [17], using a real-coded GA with innovative phenotype genetic crossover operators, were able to find all global minimum energies for Lennard-Jones clusters in the range 2, \ldots, 100 including new minima for 38, 65, 69, 76, 88 and 98 atom clusters that had not been found by any other optimisation method. This was a major breakthrough and clearly demonstrated that phenotype GAs can be effectively applied to geometric type problems. The primary objective...
of this paper is to evaluate the performance of a combination of local optimisation and phenotype genetic operators, as used in [17], when applied to the TSP.

This paper is organised as follows: Section 2 describes the Phenotype Genetic Algorithm (PGA) used in this study while Section 3 presents the results obtained and an analysis of the phenotype genetic operators and local optimisations. Finally Section 4 contains a review, future directions and conclusion.

2 Genetic Algorithm

The PGA proceeds as follows:

The initial population is constructed by using, with equal probability, random generation or the nearest neighbour heuristic, and then applying local optimisation. In the main loop, crossover operators are applied to all possible pairings in the pool and a mutation operator applied, with some probability, to all generated individuals. To achieve local optimality, local optimisation is applied after each application of an operator, and the newly created individuals inserted into the population. In the last step of each generation, the population is reduced to its initial size by first eliminating individuals which are “close” to other individuals and then selecting individuals with the shortest tour length until the population size is reached.

In summary, this genetic algorithm differs from traditional genetic algorithms because:

- The genetic pool contains a very small number (20) of individuals.
- As this GA is specifically designed to search within the space of local optima instead of the space of all possible solutions, it is important to maintain diversity within the genetic pool. The more similar parents are, the higher probability that a genetic operator will produce children that, when locally optimised, will become identical to one of the parents. Genetic diversity is maintained by not allowing individuals in the genetic pool when they are “close”, in terms of common edges.
- All possible pairwise crossovers are performed and both the crossover and mutation genetic operators use the phenotype representation of the TSP.
- All new individuals are locally optimised using a three step process. A cache of individuals that have recently been locally optimised is maintained to prevent redundant local optimisations.

These points are described in more detail in the following sections.

2.1 Genetic Pool

The genetic pool consists of 20 individuals selected on the basis of shortest tour length. In addition, no two individuals, with the same tour lengths, are allowed in the pool if there is less than fifteen different edges in their tours. All individuals in the pool automatically became available for the next generation.

2.2 Crossover Operators

All crossover operations are performed in the phenotype space by selecting one or more cuts through the geometric representation of the parent individuals. Child individuals are constructed by recombining the resultant subtours from alternate geometric segments of the parent individuals. There are three types of crossover operators.

\(\chi_1\) A random horizontal or vertical cut is made through the geometrical representation of both parent tours and any edge that crosses the cut is deleted. This results in a number of subtours on both sides of the cut. The left (top) subtours from one parent are combined with the right (bottom) subtours from the other parent to generate the basis for a child tour. The opposite operation is performed to generate the basis for the other child.

\(\chi_2\) Between five and nine nodes are randomly selected and these become the vertices of one or more polygons. Any tour edge which intersects an edge of a polygon is deleted and subtours from within the polygons of one parent are combined with subtours from outside the polygons in the other parent to create the basis for each child.

\(\chi_3\) A randomly sized rectangle, with probability of 0.5 centred on the centre of gravity of the nodes, is used to divide each parent into two segments. Any tour edge which intersects an edge of this rectangle is deleted and subtours from within the rectangle of one parent are combined with subtours from outside the rectangle in the other parent to create the basis for each child.

The basis for child individuals are then locally optimised, in three stages, as described in sub-section 2.4.

2.3 Mutation Operator

Mutation is performed, with a probability of 0.15 to all child individuals, by applying one of the \(\chi_1\), \(\chi_2\) or \(\chi_3\) crossover operators (with equal probability) to a single individual and then performing a local optimisation.

2.4 Local Optimisation

Local optimisation of individuals occurs sequentially, in three deterministic stages:

\(\lambda_1\) A greedy optimisation algorithm is applied to reconnect subtours and create a valid tour. An important aspect of the \(\lambda_1\) local optimisation is that new edges will be introduced into the individual when reconnecting subtours. Edges not in the parent(s) and perhaps not even in the population may be introduced. This is
important as, for \( n \) nodes there are \( n(n - 1)/2 \) possible edges and with a population size of 20 there will be a maximum of 20n distinct edges present in the genetic pool. As “good” edges may occur in poor tours they can be quickly lost from the population. Hence it is desirable that there is some means of intelligently introducing “good” edges into the population. The greedy optimisation algorithm within \( \lambda_1 \) performs this function.

\( \lambda_2 \) At the subtour level, 2-OPT \([2]\) moves are performed on subtours by repeatedly using subtours or combinations of subtours and performing 2-OPT moves until no further improvement can be made.

\( \lambda_3 \) At the edge level, local optimisation of individuals is performed by repeatedly performing 2-OPT moves that result in a shorter tour length. When no shorter tour length could be found using 2-OPT moves then the local optimisation is terminated.

As local optimisation is the major computational overhead of the GA, it is important to avoid redundant optimisations. Within each generation, all individuals that have been input to the \( \lambda_2 \) are saved in a simple cache. This cache is checked prior to a \( \lambda_2 \) optimisation and the optimisation process terminated if the individual has already been \( \lambda_2 \) optimised.

Prior to the \( \lambda_3 \) optimisation, if the tour length of an individual is greater than the best tour length at generation zero, the optimisation process is terminated. In addition, a more sophisticated caching scheme was implemented where all individuals that have been input to the \( \lambda_3 \) local optimiser within the last ten generations are saved and a quick access method implemented for this cache. Prior to commencing a \( \lambda_3 \) local optimisation, a check is made on the cache to determine if this individual has already been \( \lambda_3 \) optimised. If so then the \( \lambda_3 \) optimisation is not performed.

The elimination of redundant local optimisations is important in a GA that is specifically designed to search within the space of local optima instead of the space of all possible solutions, as there is a higher probability that a genetic operator will produce children that, when locally optimised, will become identical to one of the parents.

3 Implementation and Results

The genetic algorithm was implemented in C++ on a 2.0Ghz personal computer under the Linux operating system. All test problems are from TSPLIB.

3.1 Genetic Pool

Figure 1 shows the effect the initial generation method (random only, nearest neighbour only, 50% random and 50% nearest neighbour) for the genetic pool has on the minimum tour length at each generation for the ch130 TSPLIB\(^1\) TSP. While there is little difference in the number of generations required, the execution times were 7.09, 4.64 and 3.55 seconds respectively. This difference clearly demonstrates the additional local optimisations required for the random only and nearest neighbour only pool generations. Accordingly, the method used for generating the initial pool was 50% random and 50% nearest neighbour.

![Figure 1: Minimum tour length for the ch130 TSP at each generation for three different methods of generating the initial genetic pool. The methods are random only, nearest neighbour only, and 50% random and 50% nearest neighbour.](http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/)

3.2 Crossover Operators

The performance of the crossover operators and subsequent local optimisation are shown in Figures. 2, 3 and 4 which show the input (x-axis) and output (y-axis) tour lengths at each stage of the local optimisation process. Clearly the \( \chi_1 \) crossover operator tends to produce tours which, when optimised by \( \lambda_1 \), are closer to the parent tours than is the case for \( \chi_2 \) and \( \chi_3 \). This is probably due to the fact that, on average, \( \chi_1 \) will delete fewer edges than \( \chi_2 \) or \( \chi_3 \) as it uses a single cut. However, with regard to the final tour length produced by each \( \lambda_3 \) local optimisation, there is no real distinction between the crossover operators.

The frequency at which different edges are introduced into child individuals is summarised in the histogram shown in Figure 5. The count of different edges is the average number of different edges between the child individual and each parent for the ts225 TSP. Once again, there is no clear differentiation between the crossover operators.

Counts of the number of unique (1x), duplicated (2x) and triplicated (3x) individuals generated during each generation for the ts255 TSP are shown in Figure 6. As each individual has been locally optimised and the parents are included as individuals generated, this is a measure of how successful PGA is in generating new individuals that do not lie within the same local optimisation “basin” as their parents. For PGA, this stabilises at approximately 70% of individuals being unique in each generation.

3.3 Mutation Operator

The performance of the mutation operator for the ch130 TSP is summarised in Figure 7. Clearly, during the earlier generations, when tour lengths are relatively large, mutation is effective. However, as the genetic algorithm progresses,
Figure 2: $\chi_1$ crossover operator. The graphs show, on the x-axis, the input tour lengths for the local optimisers while the y-axis show the tour lengths after optimisation. The bottom-right graph is a summary graph showing the overall change in tour length. All points beneath the diagonal line represent a reduction in tour length for this operator.

Figure 4: $\chi_3$ crossover operator. The graphs show, on the x-axis, the input tour lengths for the local optimisers while the y-axis show the tour lengths after each optimisation. The bottom-right graph is a summary graph showing the overall change in tour length. All points beneath the diagonal line represent a reduction in tour length for this operator.

Figure 3: $\chi_2$ crossover operator. The graphs show, on the x-axis, the input tour lengths for the local optimisers while the y-axis show the tour lengths after optimisation. The bottom-right graph is a summary graph showing the overall change in tour length. All points beneath the diagonal line represent a reduction in tour length for this operator.

Figure 5: The frequency at which different edges are introduced into child individuals by the crossover operators for ts225 TSP. The count of different edges is the average number of different edges between the child individual and each parent.
Figure 6: Counts of the number of unique (1x), duplicated (2x) and triplicated (3x) individuals generated during each generation for the ts255 TSP. This is a measure of how successful PGA is in generating new individuals that do not lie within the same local optimisation “basin” as their parents.

the mutation operator becomes less beneficial. For ch130 (minimum tour length is 6110), no mutation was able to produce a tour length of less than approximately 6150.

3.4 Local Optimisation

The performance of each stage of the local optimisation process is summarised in Figures 2, 3, 4 and 7. The effectiveness of the caching mechanisms is summarised in Figure 8 which gives cumulative counts, at each generation, of the number of individuals fully optimised, discarded prior to $\lambda_2$ and $\lambda_3$ optimisations and the size of the $\lambda_3$ cache. The reduction in size of the $\lambda_3$ cache can be seen at generation 10 while the major reason individuals are discarded is clearly because they exceed the best tour length found at generation zero.

3.5 Results

The TSP problems from TSPLIB for which optimal results were obtained are shown in Table 1. Basically, PGA, in its current form, is able to optimally optimise TSP problems where the number of nodes is less than 450. Table 2 show the results obtained on larger TSP problems where the “% Over” column shows the percentage by which the best tour length found was greater than the optimal tour length. For the 1002 node, TSPLIB pr1002 TSP, Figure 9 provides a comparison between the optimal tour and the best tour found in this study.

4 Conclusion

In this paper, a combination of local optimisation heuristics and a phenotype genetic algorithm was developed for the TSP. Local optimisation techniques were used to find local minima in the search space and the phenotype genetic algo-
Figure 9: Comparison of the optimal tour and the best tour produced by PGA for the pr1002 TSP.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Optimal Tour</th>
<th>Generation</th>
<th>Proc. Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ch130</td>
<td>6110</td>
<td>7</td>
<td>3.5</td>
</tr>
<tr>
<td>ch150</td>
<td>6528</td>
<td>4</td>
<td>1.7</td>
</tr>
<tr>
<td>eil51</td>
<td>426</td>
<td>9</td>
<td>0.9</td>
</tr>
<tr>
<td>eil76</td>
<td>538</td>
<td>2</td>
<td>0.2</td>
</tr>
<tr>
<td>eil101</td>
<td>629</td>
<td>6</td>
<td>1.9</td>
</tr>
<tr>
<td>kroA100</td>
<td>21282</td>
<td>4</td>
<td>0.8</td>
</tr>
<tr>
<td>kroB100</td>
<td>22141</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>kroC100</td>
<td>20749</td>
<td>5</td>
<td>1.2</td>
</tr>
<tr>
<td>kroD100</td>
<td>21294</td>
<td>10</td>
<td>2.9</td>
</tr>
<tr>
<td>kroE100</td>
<td>22068</td>
<td>24</td>
<td>9.1</td>
</tr>
<tr>
<td>kroA150</td>
<td>26524</td>
<td>9</td>
<td>8.8</td>
</tr>
<tr>
<td>kroB150</td>
<td>26130</td>
<td>8</td>
<td>5.7</td>
</tr>
<tr>
<td>kroA200</td>
<td>29368</td>
<td>10</td>
<td>10.9</td>
</tr>
<tr>
<td>kroB200</td>
<td>29437</td>
<td>10</td>
<td>32.6</td>
</tr>
<tr>
<td>rat99</td>
<td>1211</td>
<td>60</td>
<td>29.8</td>
</tr>
<tr>
<td>rd100</td>
<td>7910</td>
<td>8</td>
<td>1.7</td>
</tr>
<tr>
<td>st70</td>
<td>675</td>
<td>4</td>
<td>0.4</td>
</tr>
<tr>
<td>tsp225</td>
<td>3919</td>
<td>8</td>
<td>20.4</td>
</tr>
<tr>
<td>a280</td>
<td>2579</td>
<td>5</td>
<td>23.1</td>
</tr>
<tr>
<td>pr76</td>
<td>108159</td>
<td>9</td>
<td>1.6</td>
</tr>
<tr>
<td>pr124</td>
<td>59030</td>
<td>3</td>
<td>0.6</td>
</tr>
<tr>
<td>pr136</td>
<td>96772</td>
<td>7</td>
<td>7.8</td>
</tr>
<tr>
<td>pr144</td>
<td>58537</td>
<td>12</td>
<td>2.8</td>
</tr>
<tr>
<td>pr152</td>
<td>73682</td>
<td>3</td>
<td>0.9</td>
</tr>
<tr>
<td>pr226</td>
<td>80369</td>
<td>5</td>
<td>5.5</td>
</tr>
<tr>
<td>pr264</td>
<td>49135</td>
<td>11</td>
<td>57.9</td>
</tr>
<tr>
<td>pr299</td>
<td>48191</td>
<td>11</td>
<td>92.9</td>
</tr>
<tr>
<td>pr439</td>
<td>107217</td>
<td>67</td>
<td>2586.0</td>
</tr>
<tr>
<td>bier127</td>
<td>118282</td>
<td>4</td>
<td>3.6</td>
</tr>
<tr>
<td>d198</td>
<td>15780</td>
<td>82</td>
<td>566.0</td>
</tr>
<tr>
<td>ts225</td>
<td>126643</td>
<td>19</td>
<td>20.4</td>
</tr>
<tr>
<td>u159</td>
<td>42080</td>
<td>4</td>
<td>3.4</td>
</tr>
<tr>
<td>gil262</td>
<td>2378</td>
<td>10</td>
<td>54.4</td>
</tr>
<tr>
<td>lin318</td>
<td>42029</td>
<td>47</td>
<td>1260.0</td>
</tr>
</tbody>
</table>

Table 1: TSPLIB problems that were solved exactly within 100 generations of PGA.

Table 2: TSPLIB problems that were not solved exactly within 100 generations of PGA. The % Over column gives the percentage by which the best tour length found exceeds the optimal tour length.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Optimal Tour</th>
<th>Generation</th>
<th>% Over</th>
</tr>
</thead>
<tbody>
<tr>
<td>att532</td>
<td>27686</td>
<td>32</td>
<td>0.001</td>
</tr>
<tr>
<td>p654</td>
<td>34643</td>
<td>19</td>
<td>0.01</td>
</tr>
<tr>
<td>d657</td>
<td>48912</td>
<td>54</td>
<td>0.09</td>
</tr>
<tr>
<td>u724</td>
<td>41910</td>
<td>58</td>
<td>0.12</td>
</tr>
<tr>
<td>pr1002</td>
<td>259045</td>
<td>42</td>
<td>0.33</td>
</tr>
<tr>
<td>u1060</td>
<td>224094</td>
<td>39</td>
<td>0.50</td>
</tr>
</tbody>
</table>

There are several issues for future research. Firstly, it would be interesting to improve the local optimisation algorithm (2-OPT) by incorporating a 3-OPT or Lin-Kernighan heuristic. Secondly, other geometric based crossovers could be implemented, including adaptive variants which focused on particular segments of the tour. Finally, a parallel version of the genetic algorithm would be desirable. Once these changes have been implemented, further research on more complex TSPs and related problems (asymmetric travelling salesman, sequential ordering, capacitated vehicle routing) is required to provide a detailed assessment of the merits of this type of genetic algorithm.

**Bibliography**


