Inver-over Operator for the TSP

Guo Tao[1] and Zbigniew Michalewicz[2]

[1]State Key Labortory of Software Engineering, Wuhan University, Wuhan, Hubei, 430072 P.R. China *gt@rjgc.whu.edu.cn*

[2]Department of Computer Science, University of North Carolina, Charlotte, NC 28223, USA *zbyszek@uncc.edu*

Abstract. In this paper we investigate the usefulness of a new operator, inver-over, for an evolutionary algorithm for the TSP. Inver-over is based on simple inversion, however, knowledge taken from other individuals in the population influences its action. Thus, on one hand, the proposed operator is unary, since the inversion is applied to a segment of a single individual, however, the selection of a segment to be inverted is population driven, thus the operator displays some characterictics of recombination.
This operator outperforms all other 'genetic' operators, whether unary or binary, which have been proposed in the past for the TSP in connection with evolutionary systems and the resulting evolutionary algorithm is very fast. For test cases, where the number of cities is around 100, the algorithm reaches the optimum in every execution in a couple of seconds. For larger instances (e.g., 10,000 cities) the results stay within 3% from the estimated optimum.

1 Introduction

The traveling salesman problem (TSP) is one of the most widely studied NP-hard combinatorial optimization problems. Its statement is deceptively simple, and yet it remains one of the most challenging problems in Operational Research.

Let $G = (V, E)$ be a graph where V is a set of vertices and E is a set of edges. Let $C = (c_{ij})$ be a distance (or cost) matrix associated with E. The TSP requires determination of a minimum distance circuit (Hamiltonian circuit or cycle) passing through each vertex once and only once. C is said to satisfy the triangle inequality if and only if $c_{ij} + c_{jk} \geq c_{ik}$ for all $i, j, k \in V$ (in such a case we talk about $\triangle$TSP). Euclidean TSP problems (ETSP), i.e., problems where V is a set of points in R^2 and c_{ij} is an Euclidean (straight-line) distance between i and j, are, of course, special cases of $\triangle$TSP.

A lot of algorithms have been proposed to solve TSP. Some of them (based on dynamic programming or branch and bound methods) provide the global optimum solution (the largest nontrivial instance of the TSP solved to optimality is of 7397 cities [1], however, it required almost 4 years of CPU time on network of machines). Other algorithms are heuristic ones, which are much faster, but they do not guarantee the optimal solutions. There are well known algorithms based on 2-opt or 3-opt change operators, Lin-Kerninghan algorithm (variable change) as well algorithms based on greedy principles (nearest neighbor, spanning tree, etc). The TSP was also approached by various "modern heuristic" methods, like simulated annealing, evolutionary algorithms, tabu search, even neural networks. However, these techniques were mainly applied to test cases with relatively small number of cities (usually less than 1000), whereas such problems are now solved routinely within a few hours [10].

In this paper we investigate a new evolutionary algorithm based on a new operator *inver-over*,[1] which incorporates the knowledge taken from other individuals in the population. One can view this operator as a mixture of inversion and recombination: on one hand, the inversion is applied to a part of a single individual, however, the selection of a segment to be inverted depends on other individuals in the population.

It seems that the proposed algorithm still can't compete (at least as far as computational time is concerned) with efficient approaches based on local search [10], however, it has a few adventages. First of all, it is extremely simple and easy to implement (less than 100 lines of C code). Additionally, experimental results indicate that this operator outperforms all other evolutionary operators (whether unary or binary), which have been proposed in the past for the TSP (PMX, OX, CX, ER, Edge-2, Edge-3, MPX, RAR, GNX, 2-repair, simple inversion, swap, remove and reinsert, and many others). Moreover, the evolutionary algorithm based on the proposed operator is quite fast (in comparison with other evolutionary techniques) and the quality of results are very high. For test cases, where the number of cities is around 100, the algorithm reaches the optimum in every execution. For larger instances (10,000 cities) the results stay within 3% from the estimated optimum.

The paper is organized as follows. The next section provides a brief background information on evolutionary algorithms which have been developed for the TSP. Section 3 provides a description of the proposed algorithm with a new adaptive inversion operator. Section 4 reports on experimental results and section 5 concludes the paper.

2 TSP and evolutionary approach

Initially, main effort of researchers was directed at discovery of an appropriate recombination operator, which would produce an offspring by preserving partial tours from the parents. For example, partially matched crossover (PMX) builds an offspring by choosing a subsequence of a tour from one parent and preserving the order and position of as many cities as possible from the other parent. A subsequence of a tour is selected by choosing two random cut points, which serve as boundaries for swapping operations. Order crossover (OX) builds offspring by choosing a subsequence of a tour from one parent and preserving the relative order of cities from the other parent. Cycle crossover (CX) builds offspring in

[1] The name for this operator was invented by Bob Reynolds during the EP'98 conference.

such a way that each city (and its position) comes from one of the parents. Of course, many other binary operators (and variants of the above operators) for the path representation have been defined. On the other hand, unary operators were usually defined as a swap (of two cities) or as inversion of a segment of cities. Simple inversion selects two points along the length of the chromosome, which is cut at these points, and the substring between these points is reversed. Such simple inversion guarantees that the resulting offspring is a legal tour. It is reported [27] that in a 50-city TSP, a system with inversion outperformed a system with a "cross and correct" operator. However, an increase in the number of cut points decreases the performance of the system.

The operators listed above take into account cities (i.e., their positions and order) as opposed to edges — links between cities. Clearly, the linkage of this city with other cities might be more important than the particular position of a city in a tour. Consequently, Grefenstette [8] developed a class of heuristic operators that emphasizes edges. However, as reported in [8], such operators transfer around 60% of the edges from parents — which means that 40% of edges are selected randomly. Whitley, Starkweather, and Fuquay [27] have developed a new crossover operator: the edge recombination crossover (ER), which transfers more than 95% of the edges from the parents to the single offspring. Later, the edge recombination crossover was further enhanced [23, 16]. In [17] a new local search operator was developed, which makes a use of both crossover and mutation. In [15] a new selection method was introduced as well as new crossovers (edge exchange crossover EEX and subtour exchange crossover SXX). Similarly, a new crossover (edge assembly crossover EAX) was investigated in [20]. However, these new operators were tested on relatively small instances of TSP; in many cases the reported computational time was not encouraging (few minutes for 100 city problems).

Several researchers investigated the combination of local search heuristics and evolutionary systems. Probably this is the most popular trend and the state-of-the-art in the evolutionary field: most researchers believe that the efficient implementation of quality local optimizer is crucial for the efficiency of any evolutionary algorithm. Many researchers have been applying various crossover operators (e.g., MPX operator [19]) to locally optimal individuals [2, 7, 19, 16, 25, 5], i.e., individuals after improvement made by a local search.[2] Evolutionary techniques, extended by a local search algorithm, perform very well (better then multistart local search algorithm by itself); in many experiments such systems returned a near-optimum solution (for many test cases with e.g., 442, 532, 666 cities, the deviation from the optimal tour length was less than 1%).

There were attempts to solve the TSP by evolutionary algorithms based on paradigm of evolution strategies [9, 21] or evolutionary programming [6]; there

[2] A term *memetic algorithm* refers to an evolutionary algorithm where local optimization is applied to all solutions before evaluation. This can be thought of as evolutionary algorithm is applied in the subspace of local optima, with local optimization acting as a repair mechanism for children luing outside this subspace (i.e., not being locally optimal).

were also attempts to build evolutionary systems based on non-standard representations (e.g., matrix representations). In [26] an evolutionary system is used to improve a simple heuristic algorithms for the TSP by perturbing city coordinates; results for problem sizes up to 500 cities were reported.

Very few reports provide the computational time required for solving some instances of TSP; rather the number of function evaluations are reported. Thus it is quite hard to compare different approaches, as various proposed operators have different time complexity. Eshelman reports [5] 2.5 hours (single processor Sun SPARKstation), for a 532 city problem; Gorges-Schleuter [7]: between 1 and 2 hours on the same problem (parallel implementation, 64 T800 transputers); Braun [2]: around half an hour for a 431 city problem (SUN woirkstation). All these times indicate that evolutionary algorithm might be too slow for solving larger instances (say, 10,000 cities) of the TSP.

3 Evolutionary algorithm with the inver-over operator

Most evolutionary algorithms developed so far for the TSP (not extended by a local search routine) can not compete with other heuristic methods (e.g., Lin-Kerninghan algorithm) neither in precision of results nor in computational time. Evolutionary algorithms based on crossover operator usually are quite espensive (in terms of computational time), whereas algorithms based on mutation only (whether simple inversions or swaps) do not escape efficiently local optima. It is why, as indicated in the Introduction, most recent effort aimed at combining evolutionary engine with a local search method.

What characteristic should a 'pure' evolutionary algorithm have for the TSP to compete with local search methods? It seems that the TSP problem requires a relatively strong selection pressure, which would move the search to a promising area of the search space, and an *efficient* operator, which produces an offspring without a burden of many additional calculations and allows the algorithm to escape local optima. Clearly, unary operators require much less time (in comparison with binary operators), however, they have not produced satisfactory results. Thus it might be worthwhile to experiment with an operator which combines adventages of unary and binary operators.

A new evolutionary algorithm developed for the TSP has the following characteristics:

– each individual competes with its offspring only,
– there is only one operator used; however, this inver-over operator is adaptive: it takes a clue from the current population,
– the number of times the operator is applied to an individual during a single generation, is variable.

Such an algorithm can be perceived as a set of parallel hill-climbing procedures, which preserve the spirit of Lin-Kerninghan algorithm (each hill-climber performs a variable number of edge-swaps). However, the inver-over operator has adaptive components: (1) the number of inversions applied to a single indiuvidual and (2) the segment to be inverted is determined by another (randomly selected) individual. So it is possible to view this algorithm as an evolutionary one with a strong selective pressure and with an adaptive operator.

```
random initialization of the population $P$
**while** (not satified termination-condition) **do**
{
  **for** each individual $S_i \in P$ **do**
  {
    $S' = S_i$
    select (randomly) a city $c$ from $S'$
    **repeat**
    {
      **if** ($rand() \leq p$)
        select the city $c'$ from the remaining cities in $S'$
      **else**
      {
        select (randomly) an individual from $P$
        assign to $c'$ the 'next' city to the city $c$ in the selected individual
      }
      **if** (the next city or the previous city of city $c$ in $S'$ is $c'$)
        **exit** from repeat loop
      inverse the section from the next city of city $c$ to the city $c'$ in $S'$
      $c = c'$
    }
    **if** ($eval(S') \leq eval(S_i)$)
      $S_i = S'$
  }
}
```

Fig. 1. The outline of the algorithm

Figure 1 provides a more detailed description of the whole algorithm in general and of the proposed operator in particular. With a low probability[3] p the second city for inversion is selected randomly. This is necessary: without a possibility to generate new connections, the algorithm would search only among connections between cities present in the initial population. If $rand() > p$, a randomly selected mate provides a clue for the second marker for inversion. In that case the inversion operator resembles crossover, as part of the pattern (at least 2 cities) of the second individual appears in the offspring.

Let's illustrate a single iteration of this operator on the following example. Assume that the current individual S' is

[3] Interestingly, experimental results indicated that the value of this parameter was independent of the number of cities in a test case. Note also, that the function $rand()$ (Figure 1) generates a random float from the range [0..1].

$$S' = (2, 3, 9, 4, 1, 5, 8, 6, 7),$$

and the current city c is 3. If the generated random number $rand()$ does not exceed p, another city c' from the same individual S' is selected (say, c' is 8), and appropriate segment is inverted, producing the following offspring

$$S' \leftarrow (2, 3, 8, 5, 1, 4, 9, 6, 7)$$

(note the position of the cutting points for the selected segment, which are after cities 3 and 8). Otherwise (i.e., $rand() > p$), another individual is (randomly) selected from the population; assume, it is $(1, 6, 4, 3, 5, 7, 9, 2, 8)$. This individual is searched for the city c' "next" to city 3 (which is 5), thus the segment for inversion in S' starts after city 3 and terminates after city 5; consequently, the new offspring is

$$S' \leftarrow (2, 3, 5, 1, 4, 9, 8, 6, 7).$$

Note again, that a substring 3 – 5 arrived from the "second parent". Note also, that in either case the resulting string is intermediate in the sense that the above inversion operator is applied several times before an offspring is evaluated. This process terminates when the next city c' (to the current city c) in randomly selected individual is also "next city" in the original individual. For example, assume that after a few inversions, the current individual S' is

$$S' = (9, 3, 6, 8, 5, 1, 4, 2, 7),$$

and the current city c is 6. If $rand() > p$, a city 'next' to city 6 is recovered from a (randomly) selected individual from the population; assume, it is city 8 (if $rand() \leq p$, a random city is selected, so it may also happen that city 8 was chosen). Since city 8 already follow city 3, the sequence of inversions terminates.

4 Experiments and results

In this section we present the experimental results of the proposed algorithm. All experiments are performed on a Pentium Pro 180 machine. The unit of the time listed in the result tables is one second. The two paramaters of the algorithm had the following values: population size $m = 100$ and probability of random inversion $p = 0.02$. The termination-condition is satisfied when the best solution of the population remains unchanged for the last 10 iterations (of the while loop).

Almost all test cases (except CHN144[4]) were chosen from TSPLIB [22]. The optimal solution of each test case is known. The size of these test cases vary from 30 cities to 2,392 cities. We have also created one (random) instance (RAN10000) of 10,000 cities, and relied on the formula [11] for the expected ratio k of the Held-Karp bound to $\sqrt{n}$ for n-city random ETSP; for $n \geq 100$ it is:

$$k = 0.70805 + \frac{0.52229}{\sqrt{n}} + \frac{1.31572}{n} - \frac{3.07474}{n\sqrt{n}}.$$

[4] See http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/TSPLIB95/TSPLIB.html.

So, the length of the optimal tour is estimated as $L^* = k\sqrt{n \cdot R}$, where n is the number of cities and R is the area of the square box within which the cities were randomly placed. For our instance, the number of cities is $n = 10,000$ and the edge length (of the square box) is 400, so the approximate length of the optimum solution is 28536.3.

We list the test cases and their optimal solutions in table 1. Note that in calculating the optimal solution, each distance is rounded to integer value (except, of course, the last test case, RAN10000, where the exact distances are calculated). There were ten runs of the algorithm performed for each test case. The results (listed also in table 1) represent average scores of these ten runs. For each test case, the table provides the value of the optimum solution, the *average* value found by the algorithm, average computational time (seconds), average total number of inversions and the average total number of iterations (while loop) performed during a run.

Instance	**Optimum**	**Result**	**Time**	**Inversions**	**Iterations**
EIL30	420	420	0.31	46505	129
EIL51	426	426	1.09	147972	399
EIL76	538	538	2.11	257613	651
EIL101	629	629.2	7.52	787792	2447
ST70	675	675	1.98	239152	643
KROA100	21282	21282	2.94	319182	785
KROC100	20749	20749	3.23	344272	874
KROD100	21294	21294	4.13	432336	1221
LIN105	14379	14379	3.34	350943	876
CHN144	30347	30359.2	16.81	1432780	4839
PCB442	50778	51097.5	172.21	6961960	23265
PR2392	378032	388095	5366.23	38341600	126846
RAN10000	28536.3	29551.4	167501	207775822	676840

Table 1. Results of the algorithm with adaptive inversion

The above results demonstrate clearly the efficiency of the algorithm. Note that for the first nine test cases the optimum was found in all ten runs (except the test case EIL101, where the algorithm failed only once in ten runs). The number of cities in these test cases varies from 30 to 105. For the test case with 144 cities the average solution was only 0.04% above the optimum, for the test case with 442 cities—0.63% above the optimum, and for the test case with 2392 cities—2.66%. Moreover, for a random test case with 10,000 cities the average solution stayed within 3.56% from the Held-Karp lower bound (whereas the best solution found in these ten runs was less than 3% above this lower bound).

Note also, that the running time of the algorithm was reasonable: few seconds for problems with up to 105 cities, below 3 minutes for the test case of 442 cities, below 90 minutes for the test case with 2392 cities. These represent fraction of time needed by other evolutionary algorithms based on crossover operators.

It is also interesting to compare the proposed algorithm to two other algorithms. This first one is based on simple inversion and the second one is based on Lin-Kerninghan algorithm.[5] The comparison with the first algorithm provides information on the significance of the proposed adaptive inversion operator versus blind inversion, whereas the other one—on relative merits of the tested algorithms.

		Simple Inversion			**Lin-Kerninghan**	
Instances	**Optimum**	**Result**	**Time**	**Iterations**	**Results**	**Time**
EIL30	420	432.6	58.326	74.71	421.8	0.013
EIL51	426	451.7	77.755	93.79	427.4	0.012
EIL76	538	580.9	138.763	149.66	549.7	0.026
EIL101	629	680.6	339.364	325.06	640	0.0.39
ST70	675	720.7	104.756	116.27	684.6	0.034
KROA100	21282	23136.8	253.823	241.18	21380.9	0.04
KROC100	20749	23175.5	319.7	296.88	20961	0.034
KROD100	21294	23746.8	282.806	264.17	21417.3	0.045
LIN105	14379	15627.8	350.96	311.85	14566.5	0.039
CHN144	30347	33156.7	707.033	537.80	30602.1	0.054
PCB442	50778	—	—	—	51776.5	0.137
PR2392	378032	—	—	—	389413	0.719

Table 2. Results of the Simple Inversion and Lin-Kerninghan algorithms

Table 2 give the results of such comparisons. The results of the algorithm based on random inversion were provided by our algorithm with parameter p set to 1.0. Note that for test cases with around 100 cities, the error (percentage above the optimum) was much higher (more than 10%). Time of the run increased in a significant way, in some cases more than 100 times (the termination condition was left without a change). Because of this increase in time, we do not report the results of this algorithm for the largest test cases. On the other hand, the Lin-Kerninghan algorithm takes a fraction of time necessary for our algorithm (e.g., below 1 second for the test case with 2,392 cities). However, the precision of results is much lower. Table 2 provides averages of ten runs of the Lin-Kerninghan algorithm: note, that none of the test cases resulted with the optimum solution in all ten runs. So, the proposed evolutionary algorithm has much better consistency than the Lin-Kerninghan algorithm. On the other hand, if Lin-Kerninghan algorithm was run for the same *time* as our evolutionary system (as opposed just to the same number of runs), probably it would win the competition easily.

[5] We have experimented with the implementation provided by Bill Cook, available from ftp.caam.rice.edu/pub/people/bico/970827/.

5 Conclusions

There are a few interesting observations which can be made on the basis of the experiments:

- the proposed system is probably the quickest evolutionary algorithm for the TSP developed so far. All other algorithms based on crossover operators provide much worse results in a much longer time (the exact comparison between various methods will be given in the full version of the paper [24]);
- the proposed system has only three parameters: population size, the probability p of generating random inversion, and the number of iterations in the termination condition; most of the other evolutionary systems have many additional parameters;
- it is worthwhile to emphasise the precision and stability of the system for relatively small test cases (almost 100% accuracy for all considered test cases up to 105 cities); the computational time was also acceptable (3-4 seconds);
- the system introduces a new, interesting operator, which combines features of inversion (or mutation) and crossover. Results of experiments reported in the previous section indicate clearly that the inver-over operator is significantly better than random inversion. The probability parameter p (in all experiments kept constant at 0.02) determines a proportion of blind inversions and guided (adaptive) inversions.

Further research will concentrate on (1) the significance of the selection method in connection with the inver-over operator (e.g., it would be interesting to experiment with (μ, λ)-selection and compare it with the current one, which allows competition between parent and offspring only), (2) adaptive (or self-adaptive) change of the parameter p (if successful, the system will have only one parameter: population size, apart from termination condition), (3) the significance of the population size and the termination condition (the current version of the system has fixed population size of 100 and terminates if there is no improvement in 10 iterations of the while loop), (4) full comparison of the proposed technique with other algorithms (including other evolutionary systems, tabu search, simulated annealing, and other heuristic methods), (5) experiments with larger instances of TSP (up to 1,000,000 cities).

References

1. Applegate, D. Bixby, R.E., Chvatal, V., and Cook, W., Finding cuts in the TSP: a preliminary report. Report 95-05, DIMACS, Rutgers University, NJ.
2. Braun, H., On solving traveling salesman problems by genetic algorithms. In Proc. PPSN'90, pp.129–133.
3. Davis, L., (Editor), *Genetic Algorithms and Simulated Annealing*, Morgan Kaufmann Publishers, San Mateo, CA, 1987.
4. Davis, L., *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, NY, 1991.
5. Eshelman, L., The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In Proc. FOGA'90, pp.265–283.

6. Fogel, D.B., An evolutionay approach to the traveling salesman problem. Biol. Cybern., Vol.60, pp.139–144, 1988.
7. Gorges-Schleuter, M., ASPARAGOS: An asynchronous parallel genetic optimization strategy. In Proc. ICGA'91, pp.422–427.
8. Grefenstette, J.J., Incorporating Problem Specific Knowledge into Genetic Algorithms. In [3], pp.42–60.
9. Herdy, M., Application of the Evolution Strategy to Discrete Optimization Problems. In Proc. PPSN'90, pp.188–192.
10. Johnson, D.S., The Traveling Salesman Problem: A Case Study. In *Local Search in Combinatorial Optimization*, E. Aarts and J.K. Lenstra (Editors), John Wiley, 1996, pp.215–310.
11. Johnson, D.S., McGeoch, L.A., and Rothberg, E.E., Asymptotic experimental analysis for the Held-Karp traveling salesman bound. Proc. Seventh Annual ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, and SIAM, Philadelphia, PA, pp. 341–350.
12. Karp, R.M., Probabilistic Analysis of Partitioning Algorithm for the Traveling Salesman Problem in the Plane. Mathematics of Operations Research, Vol.2, No.3, 1977, pp.209–224.
13. Lidd, M.L., Traveling Salesman Problem Domain Application of a Fundamentally New Approach to Utilizing Genetic Algorithms. Tech. Rep., MITRE Corp., 1991.
14. Lin, S. and Kerninghan, B.W., An Effective Heuristic Algorithm for the Traveling Salesman Problem. Operations Research, pp.498–516, 1972.
15. Maekawa, K., Mori, N.,Tamaki, H., Kita, H. and Nishikawa, Y., A genetic solution for the traveling salesman problem by means of a thermodynamical selection rule. Proc. IEEE ICEC '96.
16. Matias, K. and D. Whitley, Genetic operators, the fitness landscape and the traveling salesman problem. In Proc. PPSN'92, pp.219–228.
17. Merz, P.and B. Freisleben, Genetic local search for the TSP: New results. Proc. IEEE ICEC '97.
18. Michalewicz, Z., *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, 3rd edition, 1996.
19. Mühlenbein, H., Evolution in time and space – The parallel genetic algorithm. In Proc. FOGA'90, pp.316–337.
20. Nagata, Y. and Kobayashi, S., Edge assembly crossover: A high-power genetic algorithm for the traveling salesman problem. Proc. ICGA '97.
21. Nurnber, H.-T. and Beyer, H.-G., The dynamics of Evolution Strategies in the optimization of traveling salesman problem, Proc. of EP'97.
22. Reinelt, G., TSPLIB – A Traveling Salesman Problem Library. ORSA Journal on Computing, Vol.3, No.4, pp.376–384, 1991.
23. Starkweather, T., McDaniel, S., Mathias, K., Whitley, C., and Whitley, D., A Comparison of Genetic Sequencing Operators. In Proc. ICGA'91, pp.69–76.
24. Tao, G. and Michalewicz, Z., Evolutionary Algorithms for the TSP. In preparation.
25. Ulder, N.L.J., Aarts, E.H.L., Bandelt, H.-J., van Laarhoven, P.J.M., Pesch, E., Genetic Local Search Algorithms for the Traveling Salesman Problem. In Proc. PPSN'90, pp.109–116.
26. Valenzuela, Ch.L. and Willians, L.P., Improving Simple Heuristic Algorithms for the Travelling Salesman Problem using a Genetic Algorithm. In Proc. ICGA'97, pp.458–464.
27. Whitley, D., Starkweather, T., and Fuquay, D'A., Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator. In Proc. ICGA'89, pp.133–140.