A Modified Edge Recombination Operator for the Travelling Salesman Problem

Anthony Yiu-Cheung Tang[1] and Kwong-Sak Leung[2]

Department of Computer Science
The Chinese University of Hong Kong
[1] tang028@cs.cuhk.hk
[2] ksleung@cs.cuhk.hk

Abstract. Edge recombination is a crossover operator developed to preserve edge information for the Travelling Salesman Problem. This paper describes a modified version of the operator which converges significantly faster for all the benchmark problems tested.

1 Introduction

The edge recombination crossover is one of the commonly used recombination operators for solving the Travelling Salesman Problem (TSP) using genetic algorithms [8]. This paper describes a new variant of the edge recombination operator which introduces greedy choices to the algorithm. It was found that the modified operator converges significantly faster for all the problems tested.

2 EdgeNN, A New Edge Recombination Operator

An edge recombination operator is an operator that preserves edge information between parent tours. Starkweather et al. [8] described an enhancement (named in [6] as Edge-2) to the edge recombination operator by preserving common edges of the parent tours.

EdgeNN (edge recombination, nearest neighbour) is a variant of Edge-2. It modifies Edge-2 by replacing some non-deterministic steps by a greedy heuristic. The modified edge recombination algorithm is as follows.

1. Identify the two parents as Parent1 and Parent2. Copy a segment of Parent1 to the offspring (The length of this segment is chosen to be 1/4 of the number of cities in the tour and the starting point of the segment is chosen randomly).
2. Construct an edge map using the edge information in Parent2 and the segment of Parent1 not copied to the offspring.
3. Set the last city of the inherited segment in the offspring to be the current city.
4. Remove all occurrences of the current city from all the edge lists.

5. If the current city has elements in its edge list go to step 6; otherwise go to step 8.
6. If there are negative elements (*shared edges*) in the edge list of the current city, choose one to be the current city. Ties are broken at random if more than one shared edge is available. Go to step 4. If there are no shared edges then go to step 7.
7. Determine the city in the edge list of the current city which is *nearest* to the current city. The nearest city becomes the current city. Go to step 4.
8. If there are no remaining unvisited cities, then END. Otherwise, choose an unvisited city which is *nearest* to the current city. Ties are broken at random if the two nearest cities have the same distance from the current city. Go to step 4.

In selecting the next city for the current city, an *edge failure* [6] is said to occur when there are no edges in the edge list of the current city. A new edge has to be introduced that is not found in both parents, such edges are referred to as *foreign edges*.

To illustrate the new algorithm, we use a simple symmetric TSP as an example and its cost matrix is shown in fig. 1 with the upper triangle omitted.

	a	b	c	d	e	f	g	h	i	j	k	l
a	-											
b	3	-										
c	2	9	-									
d	3	2	9	-								
e	7	3	8	1	-							
f	5	1	2	5	7	-						
g	6	6	7	9	4	6	-					
h	2	8	4	2	8	5	1	-				
i	1	7	3	5	5	2	3	9	-			
j	5	3	1	4	3	7	6	1	8	-		
k	8	7	6	3	6	3	4	2	1	7	-	
l	4	5	5	7	9	1	9	3	4	8	8	-

Figure 1. An example cost matrix of a symmetric TSP.

Consider the following tours as parents to be recombined, the tour lengths of Parent1 and Parent2 are 72 and 51 respectively :

```
Parent1 : a b c d e f g h i j k l
Parent2 : f g h c l k j b e i a d
```

The offspring first inherits a fixed segment from Parent1 :

```
        c d e
```

An edge map is constructed using Parent2 and the remaining subtour of Parent1 (fig. 2). The negative elements are *shared edges* which are present in both parents :

city	edge list
a	l,b,i
b	a,j
c	-
d	-

city	edge list
e	b,f,i
f	-g
g	-f,-h
h	-g,i

city	edge list
i	h,j,a
j	i,-k,b
k	-j,-l
l	-k,a

Figure 2. The edge table.

City *e* is now the current city. Referring to the edge list of city *e*, cities *b*, *f*, and *i* are the candidates for the next city. City *b* is chosen because the edge {*e*,*b*} is the shortest. The edge list of city *b* is [*a*,*j*]. Since edge {*b*,*a*} and edge {*b*,*j*} are of the same length, ties are broken at random. Suppose city *j* is chosen. City *k* is then chosen because it is a shared edge. As city *j* has already been selected, the edge list of city *k* now contains only city *l*, hence the new current city is city *l*. The remaining cities are selected by the same process and finally the following tour is formed with a tour length of 54 :

c d e b j k l a i h g f

3 Experimental Results

3.1 Comparing EdgeNN with Edge-2

Table 1 lists the three problems tested. The data in the column "best known solution" are collected from [2] and [7].

problem	number of cities (n)	best known solution
LIN318	318	42029
PCB442	442	50778
ATT532	532	27686

Table 1. The TSP test bed.

To illustrate the convergence rate of EdgeNN, a genetic algorithm with the following parameters was applied to the test problems :

Population size :	2000
Selection method :	Baker's SUS Selection [1]. Only one offspring is generated by crossing over two parents.
Mutation rate :	0.05
	Mutation is performed by randomly swapping elements within a randomly selected segment of the chromosome.
Generation gap :	0.1
Fitness scaling :	None

Three runs were performed on each problem and the results were averaged (the variances are not significant). A randomly generated initial population was used for each run. For the chosen parameter settings each generation corresponds to 200 recombinations (population size * generation gap = 2000 * 0.1 = 200). The percentage excesses over the best known solution taken after 200,000 recombinations are shown in table 2. The graph for ATT532 is shown in fig. 3, the tour length of the best individual in the population is plotted against the number of generations. The performance of EdgeNN is significantly better.

	LIN318	PCB442	ATT532
Percentage excess over optimal length			
Edge-2	276	459	582
EdgeNN	7	12	14
Edge failures per recombination			
Edge-2	8	12	15
EdgeNN	6	12	10

Table 2. Percentage excess over the optimal solution for Edge-2 and EdgeNN.

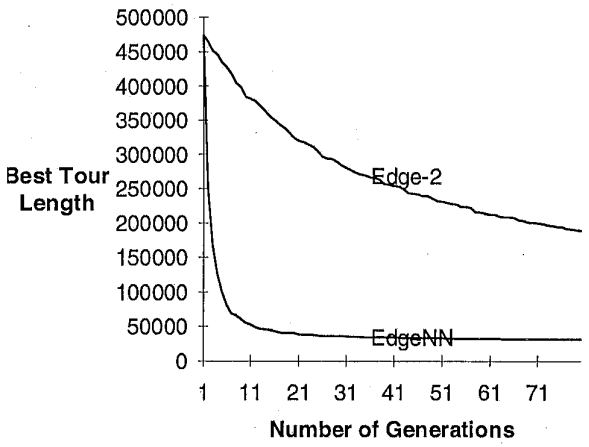

Figure 3. The convergence rate of Edge-2 and EdgeNN on ATT532.

Although EdgeNN produced better results than Edge-2, the comparison is not fair because Edge-2 is a "blind" operator : it uses no local information during recombination. Nevertheless the comparison is made to illustrate the drastic improvement in the rate of convergence by introducing greedy choices to Edge-2.

3.2 Comparing EdgeNN with Edge-3

The Edge-3 operator was designed by Mathias and Whitley [6] which enhances Edge-2 with an additional mechanism to reduce edge failures. Although Edge-3 is also a blind recombination operator, the authors hybridized Edge-3 by a procedure called *2-Repair*, a variant of 2-Opt which compares only the foreign edges produced by edge recombination operators. In view of processing cost and degree of hybridization, EdgeNN is quite similar to the 2-Repair-Edge-3 hybrid.

To compare EdgeNN with Edge-3, an experiment was performed on the problem ATT532. The setting used was chosen to be as similar as possible to that described in their paper. GENITOR [9] with a bias of 1.25 was used with a population size of 2,000. The initial population was created randomly. In the experiment performed by Mathias and Whitley the initial population was first improved by the *1-Pass of 2-Opt procedure.* In our case the initial population was used without any preprocessing because, as shown in fig. 3, the convergence rate of EdgeNN should be sufficiently fast. When 16,000 recombinations had been performed the 1-Pass of 2-Opt procedure was performed on the whole population. The process was then repeated.

Our results were averaged over 30 runs. The average tour length obtained was 28,914 with a best solution of 28,652. These results were slightly better than the average solution of 28,979 with a best of 28,752 obtained by Edge-3 reported in [6].

4 Further Improvement : A heuristic GA using EdgeNN

We agree that "from a function optimization point of view, GAs frequently don't exhibit a 'killer instinct' " [3]. Although a genetic algorithm is good at locating the region containing the global optimum, it may take a very long time to locate the optimal solution. For this reason most attempts to solve TSP using genetic algorithms incorporated some form of hill-climbing heuristics. In this section a genetic algorithm combining EdgeNN and hill-climbing heuristics is presented which produces tours that are within 2 % from the optimal for the problem ATT532.

The heuristics use the k-change operator [5]. The operator improves a tour by deleting k edges of the tour and introducing k new ones such that the tour length is reduced. We used 2-change and 3-change operators. Two objectives guided the design of our algorithm :

Diversity maintenance. If an offspring is *phenotypically* equivalent (i.e. having the same tour length) to either parents, the alleles in a small segment picked randomly are shuffled.

Minimizing the number of local optimizations performed. As the hill-climbing heuristics are CPU-intensive, they are applied under restricted circumstances. The algorithms will stop optimizing a tour after *one* exchange that can reduce tour cost is found. These heuristics are invoked under the following situations :

i) The tour length of an offspring is larger than the average tour length of its parents.
As it is a waste of time to repair very bad tours, one more constraint is added requiring that the tour length of the offspring must also be shorter than :

(mean tour length + minimum tour length) / 2

The mean and minimum are values with respect to the current population. If these conditions are satisfied, 2-change is applied to the offspring. The fixed segment inherited directly from the parent will not be optimized.

ii) The best tour length so far does not improve after a number of offsprings have been generated.
We use the population size (*popsize*) as our criterion. When there is no improvement after a *popsize* of offsprings are generated, 3-change is applied to the tour randomly picked from the best ten tours. When there is still no improvement after another 9 * *popsize* offsprings are generated, 2-change will be applied to the best half of the population.

Our algorithm is essentially GENITOR with the following set of parameters :

Population Size	500
Selection Method	Linear ranking with a bias of 1.25
Generation Gap	0.1
Mutation Rate	0
Fitness Scaling	None

The results averaged over 30 runs on ATT532 are shown in table 3. The number of recombinations performed for each run is 250,000. The best tour obtained is within 1% from the optimal and the average tour length obtained is around 2% from the optimal.

Tour length	**Best**	**Mean**	**SD**
	27949	28255	197
average number of 2-changes performed per run		8820	
average number of 3-changes performed per run		1123	

Table 3. Performance of the heuristic GA on ATT532.

5 Discussion

Since EdgeNN selects among the available cities the on nearest to the current city as the next city, the computation cost for EdgeNN will be comparatively higher than that of Edge-2 if there are many edge failures. Fig. 4 shows the averaged accumulated edge failures for the tests on ATT532 described in section 3. The average number of foreign edges produced per recombination was quite high in the beginning but gradually decreased as the tours in the population were becoming more and more similar.

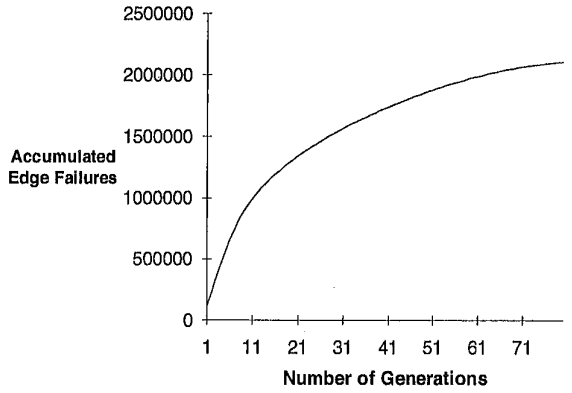

Figure 4. The accumulated number of foreign edges made by EdgeNN for ATT532.

We agree with Grefenstette [4] that for genetic searches "probabilistic choices are usually preferable to deterministic ones." The reason why a part of the offspring is inherited directly from one of the parents is that we do not want too much determinism in our algorithm. Consider the following parent tours :

```
Parent1 : a b c d e f g h i j
Parent2 : f b e i a d j g h c
```

Using EdgeNN, the offspring first inherits a fixed segment from Parent1 :

```
Offspring : c d e f
```

The edge list for city *f* using Edge-2 is [*b,c,e,g*]. Suppose city *c* is nearest to city *f* among these four cities, then EdgeNN will always choose city *c* if it is available. However, this deterministic behaviour does not always happen. As shown in this example, the edge list for city *f* using EdgeNN is [*b,g*] because the fixed segment chosen already includes city *c* and *e*.

With regard to foreign edges, suppose city *d* is the city nearest to city *j* and that an edge failure occurs in choosing the next city for city *j*. Since city *d* has already been added to the offspring, the algorithm has to select a city from the available cities that is nearest to city *j*. Since the fixed segment is picked randomly, EdgeNN retains some probabilistic choices in selecting foreign edges.

6 Conclusions

The goal of this paper is to present and evaluate the effectiveness of EdgeNN for solving TSPs, and to explore the interactions of EdgeNN with local optimization heuristics. The experimental results show that EdgeNN converges much faster than Edge-2 and produces results similar to a hybridized version of Edge-3. This suggests that any method for solving TSPs can use EdgeNN to generate initial tours of medium quality.

Although satisfactory results are produced for a medium sized problem when local optimizations are combined with EdgeNN, the number of recombinations is a bit too large. Better tour lengths with fewer recombinations may be obtained if more sophisticated local hill-climbing procedures such as the LK algorithm [5] are used.

Remarks. The experiments are constructed using TOLKIEN : *TOoLKI*t for g*EN*etics-based applications, a C++ class library developed by the first author. TOLKIEN is developed as a prototyping tool that enables GA and classifier system applications to be constructed easily. The toolkit is available in /pub/local/tolkien by anonymous ftp access to ftp.cs.cuhk.hk (137.189.4.57).

References

1. Baker, J. (1987). Reducing bias and inefficiency in the selection algorithm. *Genetic algorithms and their applications : Proceedings of the Second International Conference on Genetic Algorithms.* Lawrence Erlbaum Associates, Publishers.
2. Bixby, B., and Reinelt, G. (1990). TSPLIB 1.1.
3. De Jong, K. A. (1993). Genetic algorithms are not function optimizers. *Foundations of Genetic Algorithms 2.* Morgan Kaufmann.
4. Grefenstette, J. J. (1987). Incorporating problem specific knowledge into genetic algorithms. *Genetic Algorithms and Simulated Annealing.* Pitman.
5. Lin, S., and Kernighan, G. W. (1973). An efficient heuristic algorithm for the traveling salesman problem. *Operations Research* 21 : 498-516.
6. Mathias, K., and Whitley, D. (1992). Genetic operators, the fitness landscape and the traveling salesman problem. *Parallel Problem Solving from Nature, 2.* Elsevier Science Publishers B.V.
7. Padberg, M., and Rinaldi, G. (1991). A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review* 33 : 60-100.
8. Starkweather, T., McDaniel, S., Whitley, D., and Whitley, C. (1991). A comparison of genetic sequencing operators. *Proceedings of the Fourth International Conference on Genetic Algorithms.* Morgan Kaufmann.
9. Whitley, D. (1989). The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. *Proceedings of the Third International Conference on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann.