IT3708
Subsymbolic Methods in AI

Assignment 2

Solving Traveling Salesman Problems with Evolutionary Algorithms

Kjetil Valle
kjetilva@stud.ntnu.no

Olav Bjørkøy
olavfrih@stud.ntnu.no

NTNU, Trondheim

February 28th, 2010

In this report we describe our results in finding solutions to the Traveling Salesman Problem using evolutionary algorithms. We explore three different genetic representations: An indirect representation using a bit vector, and the two direct representations Partially-Mapped Crossover (PMX) and the Edge Recombination Operator (ERO).

1 Introduction

The Traveling Salesman Problem (TSP) is an traditional optimization problem in computer science. Since TSP belongs to the class of NP-complete problems, there exists no efficient algorithm for finding the optimal solution. The running time for solvers of TSPs are therefore assumed to be exponential in regard to the number of cities in the current problem. Evolutionary Algorithms are of course not excepted from this. EAs can not find the optimal solution any faster, neither can we formally prove that the best solution we find will always be the optimal solution.

Evolutionary Algorithms are still an interesting approach to TSP, since they find gradually better solutions. Because of this gradual increase in performance, we can use EAs to find, not necessarily the optimal, but a sufficiently good solution.

Applying EAs to solve TSPs is also interesting in the way it raises some challenges in the genetic representation. Since TSP solutions are permutations, we need to maintain this property from one generation to the next.

This can be done directly by defining mutation and crossover functions that produces only valid permutations. PMX and ERO representations are two ways to do this. It is also possible to do this in an indirect way by making sure the genotype, which needs not be permutational in nature, always develop into a valid permutation. We demonstrate the use of all three representations in our solutions.

2 Implementation

Our efforts to create a robust GA framework in the first assignment gave us a head start in this exercise, as the amount of framework code we had to change was minimal. We already had everything we needed, except the classes specific to the TSP domain; The selecting, plotting, breeding and everything else we needed for the evolutionary loop could be reused from assignment 1. The only new generic class we had to create was a wrapper for `RMagick`, a library we use for plotting the routes created by our TSP solver.

We were also able to reuse several domain specific classes from the One-Max and Colonel Blotto problems for the TSP. The bit vector from One-Max served as a base for the indirect TSP genotype representation, which we used together with a new `Developer` for converting a bit vector to an integer permutation. For the PMX and ERO representations were we able to reuse the `CopierDeveloper` when directly creating the genotype from the phenotype.

Since both ERO and PMX are integer permutations, we created an abstract class called `IntPermutationGenotype` which groups the traits shared between these genotypes. This class contains everything but the crossover method, which is overridden by the `EroGenotype` and `PmxGenotype` classes. The code for mutation and generating random genomes is inherited.

All the three representations develop into an `IntVectorPhenotype`.

The fitness of the phenotypes are in turn tested by the `TspEnvironment`. This class is responsible for reading the datasets and computing the length of the routes described by each phenotype. We use the following fitness measure:

$$fitness = \frac{1}{\sqrt[3]{length(route)}}$$

To visualize what we were doing, we also created a plotter called `RoutePlot`, extending `AbstractPlot`. This plot draw the cities from coordinates in the input files, together with the route we get from the solved TSP. The image is created by the new `RMagick` wrapper, called `Drawer`, which works just like the `Plotter` class, and gives the different `Plot` implementations another tool to use. See some of these images in Figure 4.

All the settings for customizing the framework to the task, including what dataset and which representation to use, are set and tweaked in the domain specific settings file, `settings/tsp.yml`.

3 Results

Tables 1 and 2 presents our results from running the evolutionary algorithm with the tree different genetic representations. We show the average best solution, and a 90% confidence interval for the expected result per representation.

We ran the algorithm with a population size of 500. On the Western Sahara dataset we let the algorithm run for 200 generations, and on Djibouti we ran it for 300 generations. Max fitness was used for adult selection, and sigma scaling for parent selection. The mutation rate was set to 0.05, the crossover rate to 0.7 and elitism was set to 3.

Representation	Avg. Best Solution	90% Confidence Interval
ERO	28667.7	[29810.7, 27582.4]
PMX	31815.2	[36508.8, 27892.8]
Bit	36559.0	[41308.4, 32510.6]

Table 1: Results for the Sahara dataset. Optimal solution for this dataset is 27603.

Representation	Avg. Best Solution	90% Confidence Interval
ERO	8702.9	[10079.4, 7566.0]
PMX	8698.6	[10094.0, 7548.9]
Bit	12528.5	[14661.1, 10790.2]

Table 2: Results for the Djibouti dataset. Optimal solution for this dataset is 6656.

Figures 1, 2 and 3 shows fitness plots for typical runs with ERO, PMX and the bit vector representation. We see that ERO improves its solutions at a faster rate than PMX, but that they converge to a set of good solutions pretty early in the run. The bit vector representation tends to get stuck a lot more often, and only occasionally converge to an optimal solution, dependent on how good the initial random population performed. The settings used were the same as given above.

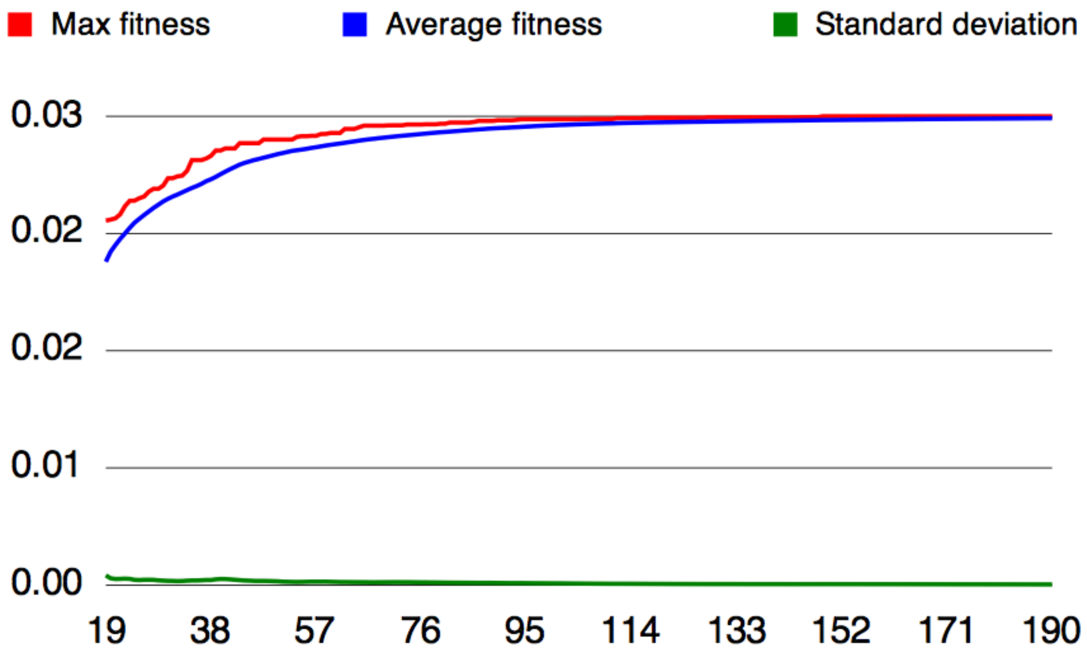

Figure 1: Fitness plot with the ERO method.

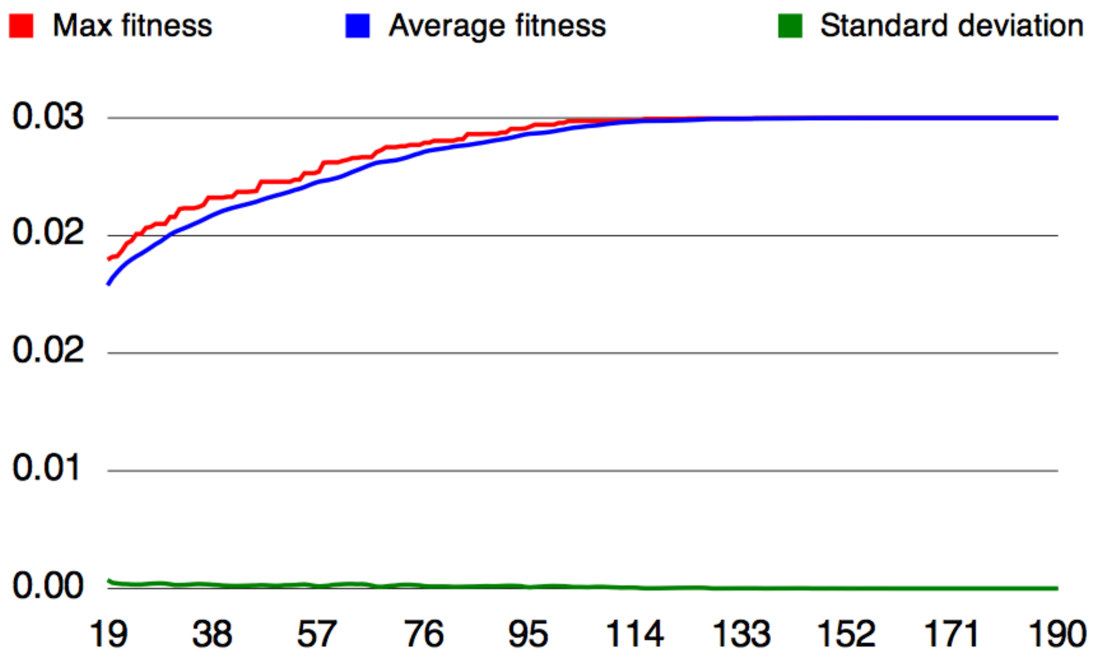

Figure 2: Fitness plot with the PMX method.

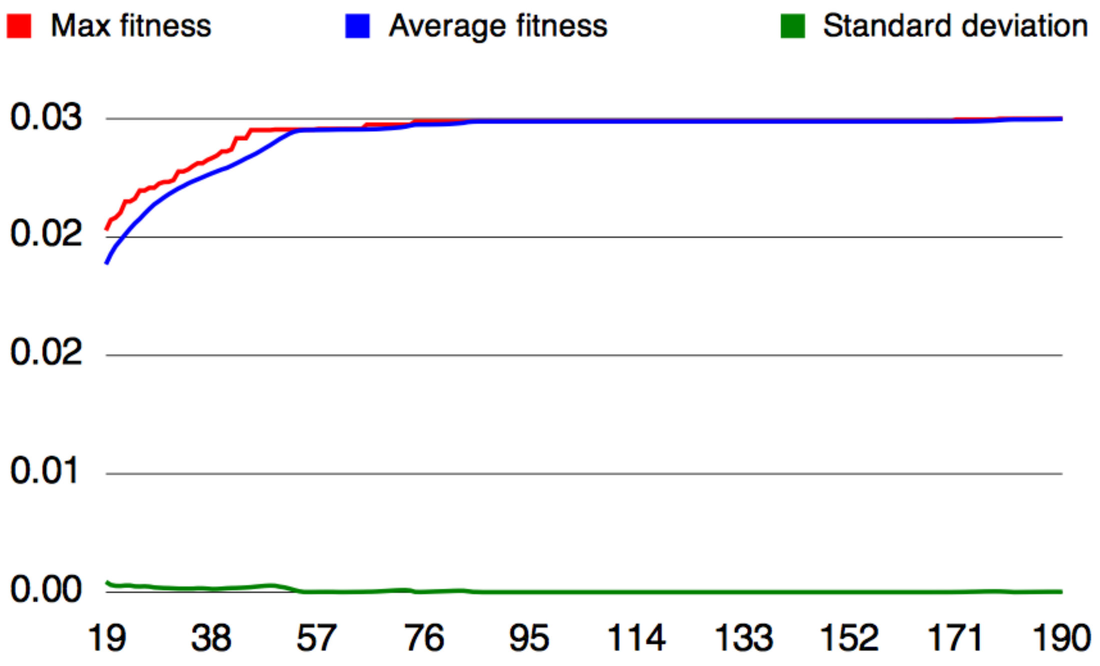

Figure 3: Fitness plot with the bit vector representation.

4 Discussion

We see from the results that all three of the representations finds solutions to the problem. ERO is clearly the best representation, with results repeatedly within 5% of the optimal solution when run with the Western Sahara dataset. Indirect representations with the bit vector or direct with PMX are not nearly as good.

We believe the reason for this is that ERO focuses on a more appropriate part of the problem than PMX and bit vector does. PMX and the bit vector representation both focuses on the order of the visited cities. The route, the edges between the cities, is then inferred from this permutation of cities.

The problem with this is of course that the representations thus differ between the very same edge occurring early vs late in the trip. This obviously have no effect on the length of the trip and the representation should be indifferent to it, as indeed ERO is.

Although all three representations find some solution, they have problems finding the optimal solution. From the fitness plots given in Figure 5 below, it seems that each representation tends to get stuck at a local optimum before finding the optimal path. We experience that the topology of the individuals in the initial random population is crucial in whether the algorithm will find the optimal path or get stuck at a local maximum.

After some initial exploration the algorithm converges to a set of relatively good solutions, and then turns to exploiting these solutions until no further improvements can be made. At this time the variance in the population fitness approaches zero, which makes further evolution impossible. To achieve better results, our representations would have to invest more time in exploring the fitness landscape before converging on a set of solutions. Our test runs demonstrate that our system converges too early.

To remedy this problem, we tried changing the evolutionary settings in favor of more exploration. The result of this, when given exploration too freely a rein, was that the fitness grew more slowly without much improvement in the final result.

In short, our biggest challenge when solving TSPs by the means of evolutionary algorithms was the prematurely convergence on local optima. We were not able to resolve this problem as well as we would have liked.

Although our algorithm do not find the optimal solution every time, it is interesting to see the process of evolution generate good solutions to a problem considered to be as hard as TSP. It is also fascinating to see how the algorithm improves on its results from generation to generation. Figure 4 shows the best solution in the population at generations 50, 100, 150 and 200 when run on the West Sahara dataset. We see here that a rather good solution have been arrived at already in generation 50, and the further generations provide improvements on its basic scheme. The solution in generation 200 is the optimal path in this dataset.

A final observation worth mentioning is that the PMX representation tends to get stuck with a lot more crossing paths than the others. If, in the early generations, the topology converges on one whith one or more long edges across the map it has a really hard time getting rid of these edges. ERO, in comparison, focuses on the best possible edges, regardless of order, and thus more easily overcomes this obstacle.

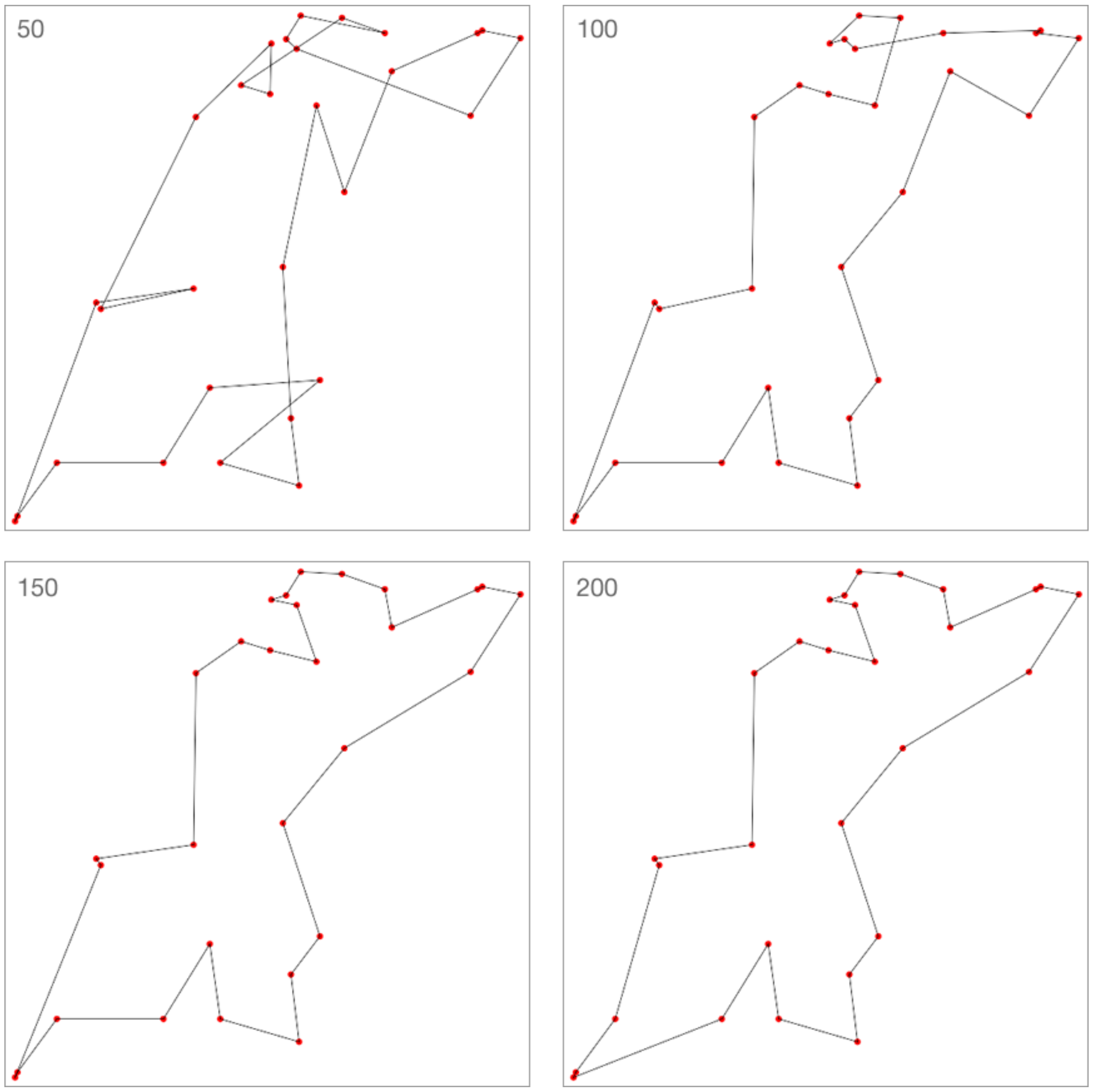

Figure 4: Progression of the best route in the population with the Edge Recombination Operator crossover method, over 200 generations with 500 individuals. Sigma scaling was used as the parent selection mechanism, and max fitness for adult. The changes are drastic in the beginning, while the final incremental changes leading to an optimal solution take more time. This particular run yields an optimal solution for the West Sahara data set.

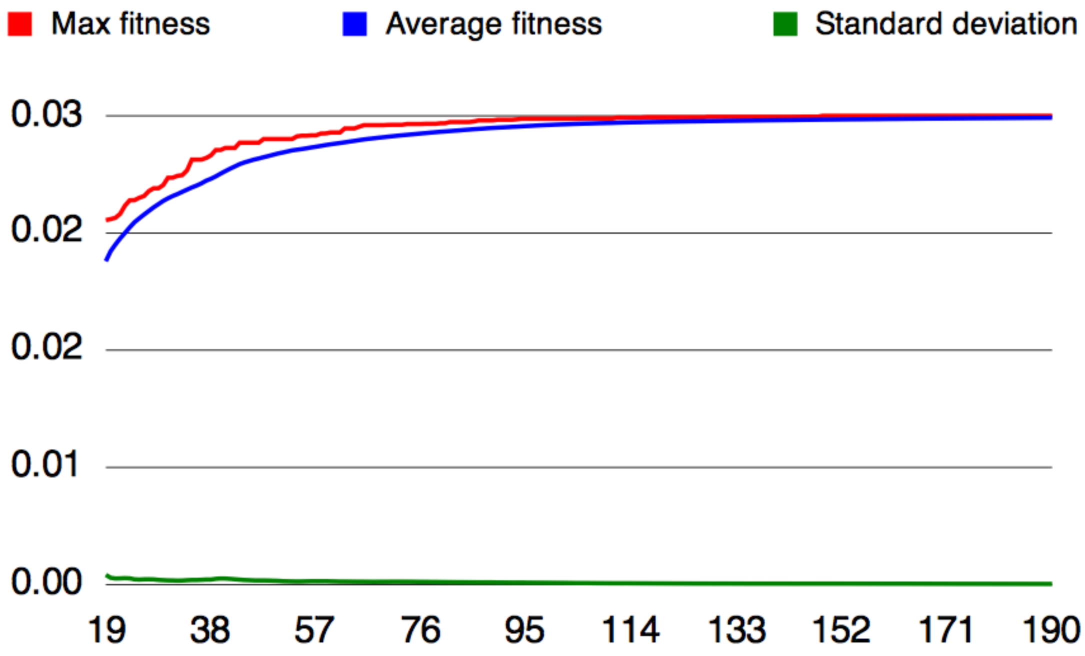

Figure 5: Fitness plot of the run in Figure 4.

Appendix

Run	Method	Best	10%	5%
1	ERO	0.03292	109	130
2	ERO	0.03287	107	120
3	ERO	0.03251	129	no
4	ERO	0.03272	99	116
5	ERO	0.03235	153	no
6	PMX	0.03204	no	no
7	PMX	0.03244	95	no
8	PMX	0.03189	no	no
9	PMX	0.03055	no	no
10	PMX	0.03088	no	no
11	BIT	0.03070	no	no
12	BIT	0.02938	no	no
13	BIT	0.03078	no	no
14	BIT	0.02942	no	no
15	BIT	0.03037	no	no

Table 3: Runs with the Sahara dataset, with three different genotype methods used. The set was run for 200 generations with a populations size of 500. The numbers in the two last columns indicate in which generation the goal was reached.

Run	Method	Best	10%	5%
1	ERO	0.04999	no	no
2	ERO	0.04929	no	no
3	ERO	0.04768	no	no
4	ERO	0.04937	no	no
5	ERO	0.04680	no	no
6	PMX	0.04723	no	no
7	PMX	0.05004	no	no
8	PMX	0.04755	no	no
9	PMX	0.04872	no	no
10	PMX	0.05003	no	no
11	BIT	0.04439	no	no
12	BIT	0.04148	no	no
13	BIT	0.04429	no	no
14	BIT	0.04249	no	no
15	BIT	0.04236	no	no

Table 4: Runs with the Djibouti dataset, with three different genotype methods used. The set was run for 300 generations with a populations size of 500. The numbers in the two last columns indicate in which generation the goal was reached.