Chapter 1

Solving the Travelling Salesman Problem with EDAs

Víctor Robles, Pedro de Miguel
Dept. of Computer Architecture and Technology
Technical University of Madrid, Spain
vrobles,pmiguel@fi.upm.es

Pedro Larrañaga
Dept. of Computer Science and Artificial Intelligence
University of Basque Country, Spain
ccplamup@si.ehu.es

Abstract In this chapter we present an approach to solve the Travelling Salesman Problem using EDAs. The approach is based on using discrete and continuous EDAs to find the best possible solution. It is also presented a way in which domain knowledge (based on local search) is combined with EDA to find better solutions. Likewise, we show the experimental results obtained with different standard examples using discrete and continuous EDAs alone or with an heuristic local search.

Keywords: EDA, Travelling Salesman Problem, Evolutionary Computation, Genetic Algorithms, local search heuristics.

1. Introduction

The Travelling Salesman Problem (TSP) objective is to find the shortest route for a travelling salesman who, starting from his home city, has to visit every city on a given list precisely once and them return to his home city. The main difficult of this problem is the immense number of possible tours: $(n-1)!/2$ for n cities.

The TSP is a relatively old problem: it was documented as early as 1759 by Euler (though not by that name), whose interest was in solving the knights' tour

problem. A correct solution would have a knight visit each of the 64 squares of a chessboard exactly once on its tour. The term "travelling salesman" was first used in 1932, in a German book written by a veteran travelling salesman. The RAND CORPORATION introduced the TSP in 1948. The Corporation's reputation helped to make the TSP a well-known and popular problem.

Through the years the Travelling Salesman Problem has occupied the thoughts of numerous researchers. There are several reasons for this. Firstly, the TSP is very easy to describe but very difficult to solve. No polynomial time algorithm is known with which it can be solved. This lack of any polynomial time algorithm is a characteristic of the class of NP-complete problems, of which the TSP is a classic example. Second, the TSP is broadly applicable to a variety of routing and scheduling problems. Thirdly, since a lot of information is already known about the TSP, it has become a kind of *test* problem; new combinatorial optimization methods are often applied to the TSP so that an idea can be formed of their usefulness.

Finally, a great number of problems actually treated with heuristics techniques in Artificial Intelligence are related with the search of the best permutations of n elements. For example, some problems in cryptoanalysis, such as the discovery of a key of a simple substitution cipher (Spillman et al., 1993), or the breaking of transportation ciphers in cryptographic systems (Matthews, 1993).

The structure of this chapter is as follows. In Section 2 we introduce the different techniques used to solve the TSP. Section 3 presents a new approach to solve the TSP with EDAs. In Section 4 we present some experimental results carried out with EDAs. Lastly, conclusions are given in Section 5.

2. Review of algorithms for the TSP

There have been many different approaches to solve the TSP. We have split these approaches in three main points: Using TSP domain knowledge with heuristics, modern heuristics and evolutionary computation.

2.1 Using TSP domain knowledge with heuristics

Several formulations and algorithms have been proposed for the TSP. Many of these approaches can be outperformed using specific domain knowledge. This domain knowledge can be divided as follows: Tour construction heuristics and tour improvement heuristics.

2.1.1 Tour construction heuristics. If we are solving a TSP problem, it is possible to use a tour construction heuristic, such as nearest neighbour, Greedy, Clarke-Wright or Christofides (Johnson and McGeoch, 1997) instead of generating random initial tours.

Nearest neighbour: The most natural heuristic is the nearest neighbour algorithm. In this algorithm the voyager always goes to the nearest location. For an n city problem, we can create n different tours, each one starting in a different city.

Greedy: In the greedy heuristic we can see a tour as an instance of a graph with the cities as vertices and with edges of distance d between each pair of cities. With this model we can see a tour as a Hamiltonian cycle in the graph. To build the tour we insert one edge at a time we start with the shortest edge, and we repeatedly add the shortest remaining available edge, if adding it would not create a degree-3 vertex or a cycle of length less than n.

Clarke-Wright: The original name is the "Saving" algorithm of Clarke and Wright (Clarke and Wright, 1964), initially thought to solve the Vehicle Routing Problem. We start with a tour in which a city is the depot, and the traveller must return to the depot after visiting each city. The savings are the amount in which the tour is shortened if we combined two cities into a single tour, bypassing the depot. We can perform the bypass if it does not create a cycle or cause a non-depot vertex to become an adjacent to more than two other non-depot vertices.

Christofides: This algorithm was developed by "Christofides (1976)". To solve the TSP we construct a minimum spanning tree T for the set of cities. Next, we compute a minimum length matching M of the vertices of odd degree in T. Combining M and T we obtain a connected graph in which every vertex has even degree. This graph must contain an Euler tour, i.e., a tour that passes through each edge exactly once. Such a cycle can be easily found.

Experimental results (Johnson, Rothberg et al., in preparation) show that best tours are obtained as follows: Firstly Christofides, then Clarke-Wright, then Greedy and finally nearest neighbour.

2.1.2 Tour improvement heuristics.

The tour improvement heuristics can be used for postprocessing, i.e., each time we have a tour we can improve it using these local improvement algorithms.

2-opt and 3-opt: The 2-opt algorithm (Croes, 1992), and the 3-opt algorithm (Lin, 1965), are the most famous local search algorithms. In the 2-opt algorithm each move consists of deleting two edges, breaking the tour into two paths, and reconnecting those paths in the other possible way. In the 3-opt algorithm we have more possibilities, breaking the tour into three paths we have two possible resulting tours.

2.5-opt and Or-opt: Based on 2-opt and 3-opt algorithms some authors have created a bit more complicated algorithms, for instance 2.5-opt and Or-opt. In the 2.5-opt algorithm (Bentley, 1992), we expand the 2-opt heuristic to include a simple form of 3-opt move that can be found with little extra effort. We have also the Or-opt heuristic (Or, 1976). Using 3-opt moves we take a

segment consisting of three or fewer consecutive cities and place it between two tour neighbours elsewhere in the tour.

Lin-Kernighan heuristic (LK): Perhaps the LK heuristic (Lin and Kernighan, 1973) is the best local search algorithm for the TSP. It is based on 2-opt and 3-opt but it also uses some ideas that will see later in tabu search. This ideas are based on the avoiding of some kinds of moves depending on two different lists. For more information about this heuristic we refer to "Johnson et al. (1996)".

The most used tour improvement heuristics are LK, 2-opt and 3-opt. The best solutions are obtained with LK.

2.2 Modern Heuristics

Besides previous heuristics, that are used to create tours and to improve existing tours, some modern heuristics techniques have been used in the TSP. Most of these heuristics use the idea of Neighbourhood Search (NS). NS is a widely used method in solving combinatorial optimization problems. A good introduction to NS can be found in "Reeves, 1993".

Table 1.1 Neighbourhood Search Method

Step 1: Select a starting solution $\mathbf{x}^{now} \in \mathbf{X}$.
Step 2: Record the current best-known solution by setting $\mathbf{x}^{best} = \mathbf{x}^{now}$ and define $best\_cost = c(\mathbf{x}^{best})$.
Step 3: Choose a solution $\mathbf{x}^{next} \in N(\mathbf{x}^{now})$. If the choice criteria can not be satisfied or other termination criteria apply, then the method stops.
Step 4: Re-set $\mathbf{x}^{now} = \mathbf{x}^{next}$, and if $c(\mathbf{x}^{now}) < best\_cost$, perform Step 2. Then return to Step 3.

A solution is specified by a vector $\mathbf{x}$, all the feasible solutions are denoted by $\mathbf{X}$, and the cost of a solution is denoted by $c(\mathbf{x})$. Each solution $\mathbf{x} \in \mathbf{X}$ has an associated set of neighbours, $N(\mathbf{x}) \subset \mathbf{X}$, called the neighbourhood of $\mathbf{x}$. Each solution $\mathbf{x}^{next}$ can be reached directly from $\mathbf{x}^{now}$ by a move. Depending on the heuristic method used we are going to have a different type of neighbourhood. Modern heuristics based on the idea of NS are Simulated Annealing (SA) and Tabu Search (TS).

Simulated Annealing (SA): This technique was born around fifteen years ago, and was originally proposed by "Kirkpatrick, Vecchi and Gelatt (1983)". It works searching the set of all possible solutions, reducing the chance of getting stuck in a poor local optimum by allowing moves to worst solutions under the control of a randomized scheme. This parameter is initially high, allowing many inferior moves to be accepted, and it is slowly reduced to a value where inferior

moves are usualy accepted. SA has a close analogy with the thermodynamic process of annealing in physics. To solve the TSP with SA "Kirkpatrick et al. (1983)" suggestes the use of a neighbourhood structure based on 2-opt moves.

Tabu Search (TS): TS tries to model human memory processes, by recording previous seen solutions using simple but effective data structures. We create a tabu list of moves which have been made in the recent past, and which are forbidden for a certain number of iterations. This helps to avoid cycling and serves to promote a diversified search of solutions. While exploring the neighbourhood of a solution TS evaluates all the moves in a candidate list. The number of moves examined is one parameter of the search. The best move on the candidate list is accepted, unless it is in the tabu list. TS introduces diversification when there are no improvements moves available. This modern heuristic was developed by "Glover (1986)", and all the basic concepts can be found on "Glover and Laguna (1993)". Some TS heuristics for the TSP are based on the use of 2-Opt exchange as their basic moves.

2.3 Evolutionary Algorithms

Evolutionary Algorithms are based on the model of natural evolution. Inside these algorithms we can identify three different branches: Genetic algorithms (GA), Evolutionary Programming (EP) and Evolution Strategies (ES). These algorithms are based on an initial population, which by means of selection, mutation and recombination evolve toward better regions in the search space. Individuals are measured thanks to an objective function.

Genetic algorithms (GA): GA (Holland, 1975) are based on the idea of biological evolution. GA operates on populations of chromosomes (strings representing possible solutions). New chromosomes are produced by combining members of the population and replacing existing chromosomes. There are two operators commonly used in GA: crossover and mutation. Using crossover we perform a type of neighbourhood search, and using mutation we can introduce some noise in the population that can help to avoid local minima. Genetic algorithms have been widely used to solve the TSP. Experimental results (Larrañaga et al., 1999) show the superiority of the following operators: Genetic Edge Recombination Crossover (ER) (Withley et al., 1989), Order Crossover (OX1) (Davis, 1985), Position Based Crossover (POS) (Syswerda, 1991) and Order Based Crossover (OX2) also developed by "Syswerda (1991)". More modern crossover operators are edge-2 and edge-3 (Mathias and Withley, 1992) as an improvement of the ER crossover operator, and the maximum preservative crossover (MPX) (Freisleben and Merz, 1996). Genetic algorithms which use some kind of interaction with local searchers (adaptative or not) are named **Memetic Algorithms (MA)** (Moscato, 1999). A key feature behind the MA implementation is the use of available knowledge about the specific

problem. In different contexts MA are also known as Hybrid GAs, Genetic Local Searches, etc.

Evolution Strategies (ES): Evolution strategies were born in 1964 at the technical University of Berlin in Germany (Rechenberg, 1973). The first example was a simple mutation-selection mechanism working on one individual, which creates one offspring per generation by means of Gaussian mutation. Another initial proposal from the University of Berlin was a multimembered ES in which one or more individuals were recombined to form one offspring. These strategies provided the ideas to $(\mu + \lambda)$-ES in which the μ best individuals are selected from a population of λ individuals. Individuals in ES are vectors of real numbers. The main loop in an ES algorithm consists of recombination, mutation, evaluation and selection. Recombination produces one new individual from the selected parent individuals. In many ways this approach is similar to GAs, except that the primary operator is mutation and parameters are adapted as the search progresses.

In "Herdy and Patone (1994)" can be found an ES to solve the TSP. In this solution four different mutation operators are created: Inversion of a segment of the tour, insertion of a town at another point of the tour, reciprocal exchange of two towns and displacement of a segment of the tour. Is also needed a new recombination operator because it must be taken in consideration that the recombination may produce only valid tours. It is important to note that this ES is based on individuals formed by vectors of integer numbers in spite of real numbers.

Evolutionary Programming (EP): A complete description of an EP algorithm is given in "Fogel (1992)". EP is similar to ES, but besides a missing recombination operator, fitness evaluation, mutation and selection are different from corresponding operators in ESs (Back et al., 1993). The operator called mutation simulates all changes that transpire between one generation and the next.

A lot of interesting information about evolutionary computation can be found in ENCORE, the Evolutionary Computation Repository Network. ENCORE is mirrored in a lot of web pages and ftp sites.

We know that there are a lot of missing algorithms around TSP in this review. For more detailed information a bibliography of TSP related papers and software can be found in the following web page

http://www.densis.fee.unicamp.br/~moscato/TSPBIB_home.html

3. A new approach: Solving the TSP with EDAs

In this section we are going to introduce a new heuristic for the TSP based on the use of EDA algorithms.

We can do two different approximations for the TSP. The first one is the use of discrete EDA, in which individuals are vectors of integer numbers. In the second one we can use continuous EDA, in which the individuals are going to be represented by vectors of real numbers. Both approximations need some kind of modifications respect to the standard EDA algorithm. These modifications are detailed in the following sections.

3.1 Using discrete EDA

With the use of discrete EDA the learning is based on Bayesian networks, and all the work is made with integer numbers. We have chosen to represent individuals using the path representation. In this representation, the n cities that should be visited are put in order according to a list of n elements, so that if the city i is the $j-th$ element of the list, city i is the $j-th$ city to be visited. The fitness function of individuals is easy to compute, just adding all distances between adjacent cities.

Table 1.2 Pseudocode for the EDA approach

Step 1: Generate M individuals (the initial population) randomly
Step 2: Repeat until a stopping criterion is met
Step 2.1: Select s individuals, $s \leq M$ from the population, according to a selection method
Step 2.2: Estimate the probability distribution of an individual being among the selected individuals
Step 2.3: Sample M individuals (the new population) from the probability distribution found before

In step 1 we must generate N individuals, being N the size of the population we are using. These individual are generated randomly and must represent a correct tour, i.e., we visit every city precisely once. Step 2 represents the main loop of the algorithm, and this loop is repeated until the stopping criteria is found. The stopping criteria can be, for example, reach to a certain numbers of generations, or stop when our Bayesian network has converged. For each loop in step 2 we must do three operations. The first one is the selection of s individuals depending in the selection method. In the realized experiments we have always selected all the individuals, that is, $s = M$. This means that we must not be worried about the selection method. The second step is the estimation of the probability function. Depending on the learning method used in EDA, we are going to estimate different Bayesian network structures. For the TSP we have used the next learning methods: UMDA (Mühlenbein, 1998), MIMIC (De Bonet et al., 1997), TREE (Chow and Liu, 1968) and EBNA

(Etxeberria and Larrañaga, 1999). The last step is the sampling of the Bayesian Network. In the standard discrete EDA algorithm we can have problems in this step, because is quite possible to generate incorrect tours, in which one or more cities are not visited, or visited more than once. To solve this problem we apply the ATM (All Time Modification) method. This method assures that all the generated individuals are correct. When doing the sampling, we must be sure that none of the numbers (cities) are repeated. To make this, ATM method dynamically modify the sampling, avoiding generating this number again. With this approximation our Bayesian network will have n variables each one with n possible values. The advantage of this method is that we are always creating correct individuals (tours), but the disadvantage is big, we are influencing decisively in the sampling, spoiling the learning realized.

The sampled individuals are introduced in the population in an elitist way, that is, replacing the worst individual in the population if is better than it.

The expected results with this approximation are not too exciting since, first, depending on the number of cities, we have a lot of variables and possible values for the learning, and second, the ATM modification is influencing decisively in the probability distribution.

3.2 Using continuous EDA

With the use of continuous EDA the learning is based on Gaussian networks, and all the work is made with real numbers. In this approximation individuals of population are represented by means of real numbers vectors. Thus, we need a method to translate these real vectors to a valid tour for the TSP. In the following table we can see one of these translations.

Table 1.3 Translation of an individual to a correct tour

Original vector:	1.34	2.14	0.17	0.05	-1.23	2.18
Resulting tour:	4	5	3	2	1	6

In table 3.2 we can see a 6-city example. In the original vector the generated reals are between 3 and -3. The obtained tour will be an integer vector in which each of the elements is the index after the values of the original vector are sorted. Thus, the calculus of the fitness function of individuals is more complex to compute. First we must obtain the resulting tour and after that we apply the same formula as in discrete EDA.

The pseudocode for the continuous EDA is the same that the one used for discrete EDA. In general two main differences can be found between discrete and continuous EDA: When estimating the probability distribution we are learning a Gaussian Network and the calculus of the fitness function must

first compute the resulting tour. The sampling of the Gaussian network will be make throught a simple method (Box and Muller, 1958).

For continuous EDA the next learning methods will be used: UMDA, EGNA, MIMIC and EMNA. For detailed information about these learning types see Chapter 3.

We still have the same problem with the great number of variables and possible values to be learned. Even though discrete EDA seems to be a better algorithm for the TSP.

3.3 Use of local optimization with EDA for the TSP

As we saw in section 2 Genetic algorithms which use some kind of interaction with local searchers (adaptative or not) are named Memetic Algorithms (MA) (Moscato, 1999). A key feature, presented in most MAs implementations, is the use of a population-based search which intends to use all available knowledge about the problem. From this point of view, if we are able to find an algorithm that introduces local search techniques inside the EDA algorithm, this will be a kind of Memetic Algorithm, which uses EDA instead of using GAs.

In "Freisleben and Merz (1996)" we can find a pseudocode that summarizes the possibilities of incorporating TSP heuristics into a GA. This pseudocode is reflected in the next table.

Table 1.4 Possibilities of incorporing TSP heuristics into a GA

Step 1: create the initial population by a tour construction heuristic
Step 2: apply a tour improvement heuristic to initial population
Step 3: selection: select parents for mating
Step 4: recombination: perform heuristic crossover
Step 5: apply a tour improvement heuristic to offspring
Step 6: mutation: mutate individuals with a given probability
Step 7: replacement: replace some parents with new offsprings
Step 8: if not converged go to Step 3
Step 9: perform postprocessing by applying a tour improvement heuristic

Several previous attempts to use TSP heuristic at particular steps of the previous template were rather discouraging. For example,"Grefenstette (1987)", used a heuristic crossover, tour construction heuristics and local hill-climbing in some experiments, and the reported results, although better than corresponding to "blind" genetic search, was worse than those produced by a simple 2-opt tour improvement heuristic. Another example were "Suh and Gucht (1987)", who applied 2-opt to some individuals of a GA population, and reached a quality of 1.73optimum for a 100-cities problem. Using only 2-opt and Or-opt as

tour improvement operator can be found solutions with a quality of 1.37 above the optimum in average.

If we want to introduce TSP heuristics into EDA, we can apply the same concept that in GAs. The resulting algorithm is shown in the table 1.5.

Table 1.5 Using Local search techniques in EDA. Heuristic EDA

Step 1:	create the initial population randomly
Step 2:	apply a tour improvement heuristic to initial population
Step 3:	select individuals according to a selection method
Step 4:	estimate the probability distribution of an individual being among the selected individuals
Step 5:	sample individuals from the probability distribution. Apply a tour improvement heuristic to each new individual
Step 6:	if not stopping criterion is met go to Step 3
Step 7:	select the best individual in the last generation

The Heuristic EDA algorithm is quite similar to the initial one, only with the difference that a tour improvement heuristic is used in the individuals of the initial population and in all the new sampled individuals. As tour improvement heuristic we have chosen the 2-opt algorithm. Despite the fact that this is a very basic algorithm, this heuristic EDA has shown better results than standard EDA as we will see in the later section.

4. Experimental results with EDAs

4.1 Introduction

Faced with the impossibility of carrying out an analytic comparison of the different EDA algorithm presented in the previous section, we have carried out an empirical comparison between the different combination of EDA algorithms, learning types and local optimization.

With these experiments we want to measure the performance of the various EDA algorithms in two main aspects: quality of the results and speed. Besides this, we also want to compare these results with other heuristics commonly used in the TSP. These experiments have been carried out in a Pentium II Xeon 500 Mhz with 1MB cache, 512 MB RAM under the Sun Solaris 2.7 operating system.

The following files have been used in the empirical study: The well known Gröstel24, Gröstel48 and Gröstel120. These are files that can be obtained via web or ftp in many sites. They represent the distances between 24, 48 and 120

imaginary cities. They are often used in TSP problems to know the fitness of the algorithm we use, and can be defined like a classical experiment in the TSP.

We are going to focus in both discrete and continuous EDA, with or without local optimization. In discrete EDA will be used the following learning methods: UMDA, MIMIC, TREE and EBNA, while in continuous EDA will be used: UMDA, EGNA, MIMIC and EMNA. Discrete EDA will be compared with GAs, the more similar heuristic that can be found in the evolutionary computation.

The results for the GAs will be taken from de literature (Larrañaga et al., 1999), which use the principle of GENITOR (Withley et al., 1989). In the mentioned algorithm, only one individual is created in each iteration of the algorithm. This new individual replaces the worst individual existing in the population, only if its evaluation function is better. This is known as steady state. The criterion used to stop the algorithm is double. In this way, if in 1000 successive iterations, the average cost of the population has not decreased, the algorithm will be stopped, not allowing, whatsoever more than 50000 evaluations. In the experiment presented here the following parameters have been established: size of population 200, probability of mutation 0.01 and selective pressure 1.90.

For each of the combinations made in the experiment, we have realized 10 searches.

4.2 Results

4.2.1 Gröstel24. Table 1.6 shows the best results and the average results obtained for each possible combination between population size, local optimization, and learning type for EDA. As a comparison the table also shows the results obtained in the GA using the crossover operators ER and OX2 and the SM (Scramble Mutation) mutation. Having into account the number of iterations needed for EDA we have also calculated the results using only local optimization.

We can analize the results' quality. In this aspect all learning types have similar results, having much better results continuous EDA. Using discrete EDA without local optimization, we have reached the optimum (1272) within the EMNA learning type, though, in general, all learning types are near the optimum.

Another interesting thing to analyse is the local optimization. Using EDA with local optimization all our tests have reached the optimum, improving the results obtained using only local optimization. Optimization takes a lot of the total execution time, but makes the algorithm to converge quicker, making the algorithm much faster. We have beat the OX2 operator and equal the ER operator, compared to GAs.

About running times, see table 1.7, we can see that, in general, MIMIC is the best learning type, in both, discrete and continuous EDA. In both kinds of

Table 1.6 Tour length for the Gröstel24

Population & Local Optimization								
	500-without		*500-with*		*1000-without*		*1000-with*	
	Best	*Aver*	*Best*	*Aver*	*Best*	*Aver*	*Best*	*Aver*
Local-optim			1272	1285			1272	1272
GA-ER*	1272	1272						
GA-OX2*	1300	1367						
UMDA	1339	1495	1272	1272	1329	1496	1272	1272
MIMIC	1391	1486	1272	1272	1328	1451	1272	1272
TREE	1413	1486	1272	1272	1429	1442	1272	1272
EBNA	1431	1528	1272	1272	1329	1439	1272	1272
UMDA_c	1289	1289	1272	1272	1289	1464	1272	1272
MIMIC_c	1289	1289	1272	1272	1300	1560	1272	1272
EGNA	1289	1306	1272	1272	1289	1307	1272	1272
EMNA	1289	1289	1272	1272	1272	1285	1272	1272

*Size of population 200, mutation used SM
Optimum 1272

Table 1.7 Number of generations and execution time for the Gröstel24 problem

Population & Local Optimization								
	500-without		*500-with*		*1000-without*		*1000-with*	
	Gen.	*Time*	*Gen.*	*Time*	*Gen.*	*Time*	*Gen.*	*Time*
UMDA	75	00:14	19	00:27	78	00:55	12	00:35
MIMIC	47	00:09	4	00:06	58	00:36	4	00:12
TREE	37	08:58	4	00:51	46	22:29	2	00:57
EBNA	72	01:00	16	00:35	79	01:50	7	00:28
UMDA_c	233	00:24	10	00:08	265	02:20	7	00:11
MIMIC_c	184	00:06	8	00:06	306	03:03	7	00:11
EGNA	263	05:31	7	00:25	298	06:05	6	00:20
EMNA	56	03:17	8	00:20	59	03:44	5	00:20

algorithm, using MIMIC, and a population of 500 individuals we reach to the optimum solution in an average time of 6 seconds. Unfortunately we cannot

compare these times with the ones obtained in GAs because we do not have this information.

There are a lot of times in which a smaller population implies obtaining better results. We can think of this as having no sense, because having more individuals the learning must be better, but, on the other hand, less individuals implies some kind of "noise" in the system, this noise can act as a certain type of mutation.

Table 1.8 Tour length for the Gröstel48

	Population & Local Optimization							
	500-without		*500-with*		*1000-without*		*1000-with*	
	Best	*Aver*	*Best*	*Aver*	*Best*	*Aver*	*Best*	*Aver*
Local-optim			5200	5290			5188	5272
GA-ER*	5074	5138						
GA-OX2*	5251	5715						
UMDA	6715	7432	5079	5149	6683	7388	5067	5139
MIMIC	6679	7083	5046	5053	6104	6717	5046	5057
TREE	-	-	5046	5071	-	-	5046	5057
EBNA	7044	7476	5165	5193	6398	7336	5114	5146
UMDA_c	5142	5248	5046	5048	5122	5245	5046	5046
MIMIC_c	5122	5176	5046	5046	5150	5228	5046	5050
EGNA	5122	5249	5046	5046	5129	5148	5046	5046
EMNA	5336	5532	5046	5048	-	-	5046	5046

*Size of population 200, mutation used SM
Optimum 5046

4.2.2 Gröstel48. While in discrete EDA results are not very good, continuous EDA has shown to be more efficient, table 1.8. In this test, we have not reached the optimum (5046) without the help of local optimization, but with MIMIC and EGNA we have reached values only 1.015 superior to the optimum. Again with the use of local optimization we get much better solutions, and using it in continuous EDA we achieve the optimum frequently.

The more significant differences are found in the running times. See table 1.9. Using UMDA and MIMIC with local optimization in continuous EDA the running time is about 5 minutes, but the running time of EMNA takes some hours. This is the reason why we have not tested EMNA with a population

Table 1.9 Number of generations and execution time for the Gröstel48 problem

Population & Local Optimization								
	500-without		*500-with*		*1000-without*		*1000-with*	
	Gen.	*Time*	*Gen.*	*Time*	*Gen.*	*Time*	*Gen.*	*Time*
UMDA	362	01:55	47	01:20	218	03:01	54	03:12
MIMIC	167	00:53	23	00:45	113	02:01	18	01:10
TREE	-	-	8	22:37	-	-	7	50:09
EBNA	306	52:16	63	12:02	261	47:50	65	14:45
UMDA_c	481	01:59	78	04:10	327	04:03	52	05:16
MIMIC_c	327	01:17	126	06:47	300	03:51	59	05:59
EGNA	381	15:46	67	16:24	1905	30:10	42	16:14
EMNA	99	4:23:05	36	1:38:44	-	-	49	2:14:03

of 1000 individuals without optimization. Again the best algorithm has been MIMIC in continuous EDA.

About the population size, the number of individuals is not decisive, with the use of smaller populations we can reach similar, or even better solutions.

Table 1.10 Tour length for the Gröstel120

Population & Local Optimization								
	500-without		*500-with*		*1000-without*		*1000-with*	
	Best	*Aver*	*Best*	*Aver*	*Best*	*Aver*	*Best*	*Aver*
UMDA	14550	15530	7171	7257	14440	15127	7287	7298
MIMIC	13644	14432	7050	7092	12739	13444	7042	7079
UMDA_c	7667	7546	7077	7113	39692	40344	7076	7103
MIMIC_c	7658	7767	7055	7078	35863	39246	7053	7101

Optimum 6942

4.2.3 Gröstel120. Image 1. Learning curve for discrete and continuous EDA without local optimization and UMDA learning.

Results obtained with the Gröstel120 problem are quite similar to the one obtained in Gröstel48. Without local optimization continuous EDA is the best algorithm, but we have found a surprising thing, with a population of 2000

Table 1.11 Number of generations and execution time for the Gröstel120 problem

	Population & Local Optimization							
	500-without		*500-with*		*1000-without*		*1000-with*	
	Gen.	*Time*	*Gen.*	*Time*	*Gen.*	*Time*	*Gen.*	*Time*
UMDA	385	22:46	55	1:42:52	368	52:10	42	2:40:58
MIMIC	306	52:08	51	1:03:09	348	1:44:59	42	1:42:55
UMDA_c	1078	32:40	95	1:11:49	425	36:30	65	1:39:30
MIMIC_c	1284	42:32	113	1:25:43	545	47:50	67	1:42:55

individuals the algorithm does not converge to a correct solution. With the use of local optimization again MIMIC is the best algorithm. The average result in continuous EDA with MIMIC, is only 1.02 superior to the optimum (6942).

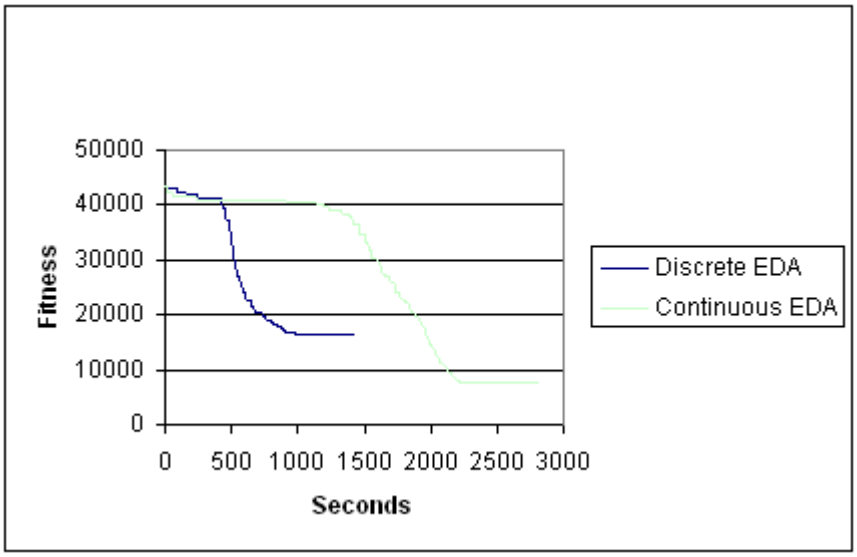

Figure 1.1 Learning curves for a 120-cities problem. Discrete EDA with UMDA learning

Another interesting thing we can see are the learning curves of discrete and continuous EDA without optimization. In figure 1.1 we can see the learning curves evolving with respect to time. Discrete EDA begin to converge quicker, but result is worst than in the continuous world. A possible solution is a

combination of discrete and continuous EDA. First generations will use discrete EDA whereus older generations would use continuous EDA.

4.3 Conclusions

Although we are aware that the experiments made over three tests files do not allow us to generalize the result obtained in other TSP problems, a certain uniformity of behaviour can be seen in the different examples. In this way UMDA and MIMIC learnings were those which had the best results.

In section 3 we spoke about the ATM modification needed to use discrete EDA with the TSP problem. We think that this modification is probably the reason of the worst solutions found by this approach.

The use of local optimization has been very successfull, leading to solutions quite near to the optimum. In my opinion EDA speed to solve the problem must be improved. The use of more specific EDA implementations for the TSP can help to improve speed, but the real problem is that a few more cities in the problem can mean much more execution time.

5. Conclusions

In this paper, an approach has been presented to use EDA, a new tool in evolutionary computing, in the traveling salesman problem (TSP). We have also incorporate domain knowledge in the resolution through the use of local search optimization based on the 2-opt algorithm. The feasibility of the proposed approach has been demonstrated by presenting performance results for a number of TSP instances between 24 and 120 cities. As this is the first approach of the use of EDA with the TSP, there are a lot of issues for future research. For example, the efficiency of the implementation can be increased to reduce computation times. Also, another kind of local search heuristics can be used in the algorithm. Additional we still need some extra tests of the proposed algorithm, insisting in the population size that must be used depending on the number of cities.

References

Or, I. (1976). *Travelling Salesman-Type Combinatorial Problems and their Relation to the Logistics of Regional Blood Banking.* Ph.D. Thesis, Deparment of Industrial Engineering and Management Sciences, Northwestern University, Evanston,IL.

Reeves, C. R. (1993). *Modern Heuristic Techniques for Combinatorial Problems.* Blackwell Scientific Publications, Oxford.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor.

Moscato, P. (1999). Memetic algorithms: A short introduction. *New ideas in optimization*. Corne,D., Glover, F. and Dorigo, M. (eds.) *New ideas in optimization*. Mc Graw Hill.

Smith, A. (1993). An Inquiry into the Nature and Causes of the Wealth of Nations. In *Classics of Economics,* Needy, C.W., ed., Moore Publishing, Amsterdam, The Netherlands.

Johnson, David S. and McGeoch, Lyle A.(1997). Local Seach in Combinatorial Optimization, Aarts, E.H.L. and Lenstra, J.K. (eds.), John Wiley and Sons, London.

Glover, F. and Laguna, M. (1993). Modern Heuristic Techniques for Combinatorial Problems, chapter 3. Blackwell Scientific Publications, Oxford.

Syswerda, G. (1991). Schedule Optimization Using Genetic Algorithms. In Davis, L. (ed.) *Handbook of Genetics Algoritms*, 332-349. New York: Van Nostrand Reinhold.

Mathias, K. and Whitley, D. (1992). Genetic Operators, the Fitness Landscape and the Traveling Salesman Problem, in *Pararell Problem Solving from Nature*, Manner, R. and Manderick, B. (eds.). 219-228, Elsevier.

Grefenstette, J. J. (1987). Incorporing Problem Specific Knowledge into Genetic Algortihm. Schedule Optimization Using Genetic Algorithms. In Davis, L. (ed.) *Genetics Algoritms and Simulated Annealing*, 42-60. Morgan Kaufmann.

Matthews, R. A. J. (1993). The Use of Genetic Algorithms in Cryptanalysis. *Cryptologia* **XVII**(2), 187-201.

Spillman, R., Janssen, M., Nelsonn, B., and Kepner, M. (1993). Use of a Genetic Algorithm in the Cryptanalysis Simple Substitution Ciphers. *Cryptologia* **XVII**(1), 31-44.

Clarke, G. and Wright, J.W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research* **12**, 568-581.

Johnson, D. S., Bentley, J. L., McGeoh, L. A., and Rothberg, E. E. Near optimal solutions to very large travelling salesman problems.

Bentley, J. L. (1992). Fast algorithm for geometric travelling salesman problem. *ORSA J. Computing* **4**, 125-128.

Croes, G. A. (1958). A method for solving travelling salesman problems. *Operation Res.* **6**, 791-812.

Johnson, D. S., Aragon, C. R., McGeoh, L. A., and Schevon, C. Optimization by Simulated Annealing: An experimental Evaluation, Part III (The Travelling Salesman Problem), in preparation.

Lin, S. (1965). Computer solutions of the travelling salesman problem. *Bell Syst. Tech. J.* **44**, 2245-2269.

Box, G. E. P., Muller, M. E. (1958). A note on the generation of random normal deviates. *Ann. Math. Static.* **29**, 610-611.

Lin, S. and Kernighan, B. W. (1973). An Effective Heuristic Algorithm for the Travelling Salesman Problem. *Operation Res.* **21**, 498-516.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Ops. Res.* **5**, 533-549.

Kirkpatrick, S., Gellat, C. D. and Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science* **220**, 671-680.

Larrañaga, P., Kuijpers, C. M. H., Murga, R. H., Inza, I. and Dizdarevic, S. (1999). Genetic Algorithms for the Travelling Salesman Problem: A review of Representations and Operators. *Artificial Intelligence Review* **13**, 129-170.

Mühlenbein, H. (1998). The Equation for Response to Selection and its Use for Prediction. *Evolutionary Computation* **5**, 303-346.

Rechenberg, I. (1973). *Optimierung Technischer Systeme Nach Prinzipien der Biologischen Information*. Fromman Verlag, Sttutgart.

Chow, C. and Liu, C. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory* **14**, 462-467.

De Bonet, J. S. and Isbell, C. L. and Viola, P. (1997). MIMIC: Finding Optima by Estimating Probability Densities. Mozer, M., Jordan, M. and Petsche, Th. (eds). Approximating discrete probability distributions with dependence trees. *Advances in Neural Information Processing Systems*, Vol **9**.

Christofides, N. (1976). Worst-case analysis of a new heuristic for the travelling salesman problem. Report No.388, GSIA, Carnegie-Mellon University, Pittsburgh, PA.

Davis, L. (1985). Applying Adaptive Algorithms to Epistatic Domains. *Proceedings of the International Joint Conference on Artificial Intelligence*, 162-164.

Withley, D., Starkweather, D. and Fuquay, D. (1989). Scheduling Problems and Travelling Salesman: The Genetic Edge Recombination Operator. In Schaffer, J. (ed) *Proceedings of the Third International Conference on Genetic Algorithms*, 133-140. Los Altos, CA: Morgan Kaufmann Publishers.

Freisleben, B. and Merz, P. (1996). A Genetic Local Search Algorithm for Solving Symmetric and Asymmetric Traveling Salesman Problems, in *Proc. IEEE Int. Conf. on Evolutionary Computation, Nagoya, Japan*. 616-621.

Fogel, D. B. (1992). An analysis of evolutionary programming. *Proc. of the First Annual Conf. on Evolutionary Computation, La Jolla, CA*. 43-51.

Back, T., Rudolph, G. and Schwefel, H. (1993). Evolutionary Programming and Evolution Strategies: Similarities and Differences. *University of Dortmund, Deparment of Computer Science, Germany*.

Herdy, M. and Patone, G. (1994). Evolution Strategy in Action: 10 ES-Demonstrations. *International Conference On Evolutionary Computation*. The Third Pararell Problem Solving From Nature. Jerusalem, Israel.

Etxeberria, R. and Larrañaga (1999). Global Optimization with Bayesian networks, *II Symposium on Artifial Intelligence. CIMAF99. Special Session on Distributions and Evolutionary Optimization* 322-339.

Suh, J. Y., and Gucht, D. van (1987). Incorporing Heuristic Information into Genetic Search. *Proc. of the second Int. Conf. on Genetic Algorithm*, 100-107, Lawrence Erlbaum.