4

Chaos Driven Evolutionary Algorithm for the Traveling Salesman Problem

Donald Davendra[1*], Ivan Zelinka[1],
Roman Senkerik[2] and Magdalena Bialic-Davendra[3]
[1]*Department of Informatics, Faculty of Electrical Engineering and Computing Science, Technical University of Ostrava, Tr. 17. Listopadu 15, Ostrava*
[2]*Department of Informatics and Artificial Intelligence, Faculty of Informatics, Tomas Bata University in Zlin, Nad Stranemi 4511, Zlin 76001*
[3]*Department of Finance and Accounting, Faculty of Management and Economics, Mostni 5139, Zlin 76001*
Czech Republic

1. Introduction

One of the most studied problem in operations research and management science during the past few decades has been the Traveling Salesman Problem (TSP). The TSP is rather a simple problem formulation, but what has garnered it much attention is the fact that it belongs to a specific class of problems which has been labelled as "non-deterministic polynomial" or NP. What this implies is that no algorithm currently exists which can find the exact solution in "polynomial time". A number of current engineering applications are formed around this premise; such as cryptography.
TSP manages to capture the imagination of theoretical computer scientists and mathematicians as it simply describes the complexity of NP Completeness. A number of resources exists on the internet such as the TSPLIB , which allow any novice user to understand and incorporate the TSP problem.
Since no concrete mathematical algorithm exists to solve the TSP problem, a specific branch of research, namely evolutionary science, has been applied rather effectively to find solutions. Evolutionary science itself is divided into many scopes, but the most effective ones have been the deterministic approaches and random approaches. Deterministic approaches like Branch and Bound (Land & Doig, 1960) and Lin-Kernighan local searches (Lin & Kernighan, 1973) have proven very effective over the years. Random based approaches, incorporated in heuristics have generally provided a guided search pattern. Therefore the most effective algorithms have been a hybrid of the two approaches.
This research introduces another approach, which is based on a chaotic map (Davendra & Zelinka, 2010). A chaotic system is one which displays a chaotic behavior and it based on a function which in itself is a dynamical system. What is of interest is that the map iterates across the functional space in discrete steps, each one in a unique footprint. What this implies is that the same position in not iterated again. This provides a great advantage as the number

*donald.davendra@vsb.cz

generated in unique and when input into an evolutionary algorithm, it provides a unique mapping schema. The question that remains to be answered is that whether this system improves on a generic random number generator or not.
This chapter is divided into the following sections. Section 2 introduces the TSP problem formulation. Section 3 describes the algorithm used in this research; Differential Evolution (DE) and Section 4 outlines its permutative varient EDE. The chaotic maps used in this research are described in Section 5 whereas the experimentation is given in Section 6. The chapter is concluded in Section 7.

2. Travelling salesman problem

A TSP is a classical combinatorial optimization problem. Simply stated, the objective of a traveling salesman is to move from city to city, visiting each city only once and returning back to the starting city. This is called a *tour* of the salesman. In mathematical formulation, there is a group of distinct cities $\{C_1, C_2, C_3, ..., C_N\}$, and there is given for each pair of city $\left\{C_i, C_j\right\}$ a distance $d\left\{C_i, C_j\right\}$. The objective then is to find an ordering π of cities such that the total time for the salesman is minimized. The lowest possible time is termed the optimal time. The objective function is given as:

$$\sum_{i=1}^{N-1} d\left(C_{\pi(i)}, C_{\pi(i+1)}\right) + d\left(C_{\pi(N)}, C_{\pi(1)}\right) \tag{1}$$

This quality is known as the *tour length*. Two branches of this problem exist, symmetric and asymmetric. A symmetric problem is one where the distance between two cities is identical, given as: $d\left\{C_i, C_j\right\} = d\left\{C_j, C_i\right\}$ for $1 \leq i, j \leq N$ and the asymmetric is where the distances are not equal. An asymmetric problem is generally more difficult to solve.
The TSP has many real world applications; VSLA fabrication (Korte, 1988) to X-ray crystallography (Bland & Shallcross, 1989). Another consideration is that TSP is *NP-Hard* as shown by Garey (1979), and so any algorithm for finding optimal tours must have a worst-case running time that grows faster than any polynomial (assuming the widely believed conjecture that $P \neq NP$).
TSP has been solved to such an extent that traditional heuristics are able to find good solutions to merely a small percentage error. It is normal for the simple 3-Opt heuristic typically getting with 3-4% to the optimal and the *variable-opt* algorithm of Lin & Kernighan (1973) typically getting around 1-2%.
The objective for new emerging evolutionary systems is to find a guided approach to TSP and leave simple local search heuristics to find better local regions, as is the case for this chapter.

3. Differential evolution algorithm

Differential evolution (DE) is one of the evolutionary optimization methods proposed by Price (1999) to solve the Chebychev polynomial fitting problem. DE is a population-based and stochastic global optimizer, and has proven to be a robust technique for global optimization.
In order to describe DE, a schematic is given in Figure 1.
There are essentially five sections to the code. Section 1 describes the input to the heuristic. D is the size of the problem, $G_{\max}$ is the maximum number of generations, NP is the total

Canonical Differential Evolution Algorithm

$$
\begin{array}{l}
\text{1.Input :} D, G_{\max}, NP \geq 4, F \in (0,1+), CR \in [0,1], \text{and initial bounds :} x^{(lo)}, x^{(hi)}. \\
\text{2.Initialize :} \left\{ \begin{array}{l} \forall i \leq NP \wedge \forall j \leq D : x_{i,j,G=0} = x_j^{(lo)} + rand_j[0,1] \bullet \left(x_j^{(hi)} - x_j^{(lo)} \right) \\ i = \{1,2,...,NP\}, j = \{1,2,...,D\}, G = 0, rand_j[0,1] \in [0,1] \end{array} \right. \\
\left\{ \begin{array}{l} \text{3.While} \quad G < G_{\max} \\ \quad \forall i \leq NP \left\{ \begin{array}{l} \text{4. Mutate and recombine :} \\ \text{4.1} \quad r_1, r_2, r_3 \in \{1,2,....,NP\}, \\ \quad \text{randomly selected, except :} r_1 \neq r_2 \neq r_3 \neq i \\ \text{4.2} \quad j_{rand} \in \{1,2,...,D\}, \text{randomly selected once each } i \\ \text{4.3} \quad \forall j \leq D, u_{j,i,G+1} = \left\{ \begin{array}{l} x_{j,r_3,G} + F \cdot (x_{j,r_1,G} - x_{j,r_2,G}) \\ \text{if} \quad (rand_j[0,1] < CR \vee j = j_{rand}) \\ x_{j,i,G} \quad \text{otherwise} \end{array} \right. \\ \text{5. Select} \\ x_{i,G+1} = \left\{ \begin{array}{l} u_{i,G+1} \text{ if} \quad f(u_{i,G+1}) \leq f(x_{i,G}) \\ x_{i,G} \quad \text{otherwise} \end{array} \right. \end{array} \right. \\ G = G + 1 \end{array} \right.
\end{array}
$$

Fig. 1. Canonical Differential Evolution Algorithm

number of solutions, F is the scaling factor of the solution and CR is the factor for crossover. F and CR together make the internal tuning parameters for the heuristic.

Section 2 outlines the initialization of the heuristic. Each solution $x_{i,j,G=0}$ is created randomly between the two bounds $x^{(lo)}$ and $x^{(hi)}$. The parameter j represents the index to the values within the solution and i indexes the solutions within the population. So, to illustrate, $x_{4,2,0}$ represents the second value of the fourth solution at the initial generation.

After initialization, the population is subjected to repeated iterations in section 3.

Section 4 describes the conversion routines of DE. Initially, three random numbers r_1, r_2, r_3 are selected, unique to each other and to the current indexed solution i in the population in 4.1. Henceforth, a new index j_{rand} is selected in the solution. j_{rand} points to the value being modified in the solution as given in 4.2. In 4.3, two solutions, $x_{j,r_1,G}$ and $x_{j,r_2,G}$ are selected through the index r_1 and r_2 and their values subtracted. This value is then multiplied by F, the predefined scaling factor. This is added to the value indexed by r_3 .

However, this solution is not arbitrarily accepted in the solution. A new random number is generated, and if this random number is less than the value of CR, then the new value replaces the old value in the current solution. Once all the values in the solution are obtained, the new solution is vetted for its fitness or value and if this improves on the value of the previous solution, the new solution replaces the previous solution in the population. Hence the competition is only between the new *child* solution and its *parent* solution.

Price (1999) has suggested ten different working strategies. It mainly depends on the problem on hand for which strategy to choose. The strategies vary on the solutions to be perturbed, number of differing solutions considered for perturbation, and finally the type of crossover used. The following are the different strategies being applied.

Strategy 1: DE/best/1/exp: $u_{i,G+1} = x_{best,G} + F \bullet (x_{r_1,G} - x_{r_2,G})$

Strategy 2: DE/rand/1/exp: $u_{i,G+1} = x_{r_1,G} + F \bullet \left(x_{r_2,G} - x_{r_3,G} \right)$

Strategy 3: DE/rand−best/1/exp: $u_{i,G+1} = x_{i,G} + \lambda \bullet (x_{best,G} - x_{r_1,G}) + F \bullet (x_{r_1,G} - x_{r_2,G})$

Strategy 4: DE/best/2/exp: $u_{i,G+1} = x_{best,G} + F \bullet (x_{r_1,G} - x_{r_2,G} - x_{r_3,G} - x_{r_4,G})$

Strategy 5: DE/rand/2/exp: $u_{i,G+1} = x_{5,G} + F \bullet (x_{r_1,G} - x_{r_2,G} - x_{r_3,G} - x_{r_4,G})$

Strategy 6: DE/best/1/bin: $u_{i,G+1} = x_{best,G} + F \bullet (x_{r_1,G} - x_{r_2,G})$

Strategy 7: DE/rand/1/bin: $u_{i,G+1} = x_{r_1,G} + F \bullet (x_{r_2,G} - x_{r_3,G})$

Strategy 8: DE/rand−best/1/bin: $u_{i,G+1} = x_{i,G} + \lambda \bullet (x_{best,G} - x_{r_1,G}) + F \bullet (x_{r_1,G} - x_{r_2,G})$

Strategy 9: DE/best/2/bin: $u_{i,G+1} = x_{best,G} + F \bullet (x_{r_1,G} - x_{r_2,G} - x_{r_3,G} - x_{r_4,G})$

Strategy 10: DE/rand/2/bin: $u_{i,G+1} = x_{5,G} + F \bullet (x_{r_1,G} - x_{r_2,G} - x_{r_3,G} - x_{r_4,G})$

The convention shown is DE/x/y/z. DE stands for Differential Evolution, x represents a string denoting the solution to be perturbed, y is the number of difference solutions considered for perturbation of x, and z is the type of crossover being used (exp: exponential; bin: binomial).

DE has two main phases of crossover: binomial and exponential. Generally, a child solution $u_{i,G+1}$ is either taken from the parent solution $x_{i,G}$ or from a mutated donor solution $v_{i,G+1}$ as shown : $u_{j,i,G+1} = v_{j,i,G+1} = x_{j,r_3,G} + F \bullet \left(x_{j,r_1,G} - x_{j,r_2,G}\right)$.

The frequency with which the donor solution $v_{i,G+1}$ is chosen over the parent solution $x_{i,G}$ as the source of the child solution is controlled by both phases of crossover. This is achieved through a user defined constant, crossover CR which is held constant throughout the execution of the heuristic.

The *binomial* scheme takes parameters from the donor solution every time that the generated random number is less than the CR as given by $rand_j[0,1] < CR$, else all parameters come from the parent solution $x_{i,G}$.

The *exponential* scheme takes the child solutions from $x_{i,G}$ until the first time that the random number is greater than CR, as given by $rand_j[0,1] < CR$, otherwise the parameters comes from the parent solution $x_{i,G}$.

To ensure that each child solution differs from the parent solution, both the exponential and binomial schemes take at least one value from the mutated donor solution $v_{i,G+1}$.

3.1 Tuning parameters

Outlining an absolute value for CR is difficult. It is largely problem dependent. However a few guidelines have been laid down by Price (1999). When using binomial scheme, intermediate values of CR produce good results. If the objective function is known to be separable, then $CR = 0$ in conjunction with binomial scheme is recommended. The recommended value of CR should be close to or equal to 1, since the possibility or crossover occurring is high. The higher the value of CR, the greater the possibility of the random number generated being less than the value of CR, and thus initiating the crossover.

The general description of F is that it should be at least above 0.5, in order to provide sufficient scaling of the produced value.

The tuning parameters and their guidelines are given in Table 1.

4. Enhanced differential evolution algorithm

Enhanced Differential Evolution (EDE) (Davendra, 2001; Davendra & Onwubolu, 2007a; Onwubolu & Davendra, 2006; 2009), heuristic is an extension of the Discrete Differential Evolution (DDE) variant of DE (Davendra & Onwubolu, 2007b). One of the major drawbacks

Control Variables	Lo	Hi	Best?	Comments
F: Scaling Factor	0	1.0+	0.3 – 0.9	$F \geq 0.5$
CR: Crossover probability	0	1	$0.8 - 1.0$	CR = 0, seperable CR = 1, epistatic

Table 1. Guide to choosing best initial control variables

of the DDE algorithm was the high frequency of in-feasible solutions, which were created after evaluation. However, since DDE showed much promise, the next logical step was to devise a method, which would repair the in-feasible solutions and hence add viability to the heuristic. To this effect, three different repairment strategies were developed, each of which used a different index to repair the solution. After repairment, three different enhancement features were added. This was done to add more depth to the DDE problem in order to solve permutative problems. The enhancement routines were standard mutation, insertion and local search. The basic outline is given in Figure 2.

4.1 Permutative population

The first part of the heuristic generates the permutative population. A permutative solution is one, where each value within the solution is unique and systematic. A basic description is

1. **Initial Phase**
   (a) *Population Generation*: An initial number of discrete trial solutions are generated for the initial population.
2. **Conversion**
   (a) *Discrete to Floating Conversion*: This conversion schema transforms the parent solution into the required continuous solution.
   (b) *DE Strategy*: The DE strategy transforms the parent solution into the child solution using its inbuilt crossover and mutation schemas.
   (c) *Floating to Discrete Conversion*: This conversion schema transforms the continuous child solution into a discrete solution.
3. **Mutation**
   (a) *Relative Mutation Schema*: Formulates the child solution into the discrete solution of unique values.
4. **Improvement Strategy**
   (a) *Mutation*: Standard mutation is applied to obtain a better solution.
   (b) *Insertion*: Uses a two-point cascade to obtain a better solution.
5. **Local Search**
   (a) *Local Search*: 2 Opt local search is used to explore the neighborhood of the solution.

Fig. 2. EDE outline

given in Equation 2.

$$\begin{aligned} P_G &= \{x_{1,G}, x_{2,G}, ..., x_{NP,G}\},\ x_{i,G} = x_{j,i,G} \\ x_{j,i,G=0} &= (\text{int})\left(rand_j[0,1] \bullet \left(x_j^{(hi)} + 1 - x_j^{(lo)}\right) + \left(x_j^{(lo)}\right)\right) \\ &\quad if\ x_{j,i} \notin \left\{x_{0,i}, x_{1,i}, ..., x_{j-1,i}\right\} \\ &\quad i = \{1,2,3,...,NP\}, j = \{1,2,3,..,D\} \end{aligned} \tag{2}$$

where P_G represents the population, $x_{j,i,G=0}$ represents each solution within the population and $x_j^{(lo)}$ and $x_j^{(hi)}$ represents the bounds. The index i references the solution from 1 to NP, and j which references the values in the solution.

4.2 Forward transformation

The transformation schema represents the most integral part of the DDE problem. Onwubolu (Onwubolu, 2005) developed an effective routine for the conversion.
Let a set of integer numbers be represented as in Equation 3:

$$x_i \in x_{i,G} \tag{3}$$

which belong to solution $x_{j,i,G=0}$. The equivalent continuous value for x_i is given as $1 \bullet 10^2 < 5 \bullet 10^2 \leq 10^2$.
The domain of the variable x_i has length of 5 as shown in $5 \bullet 10^2$. The precision of the value to be generated is set to two decimal places (2 d.p.) as given by the superscript two (2) in 10^2 . The range of the variable x_i is between 1 and 10^3. The lower bound is 1 whereas the upper bound of 10^3 was obtained after extensive experimentation. The upper bound 10^3 provides optimal filtering of values which are generated close together (Davendra & Onwubolu, 2007b). The formulation of the forward transformation is given as:

$$x_i' = -1 + \frac{x_i \bullet f \bullet 5}{10^3 - 1} \tag{4}$$

Equation 4 when broken down, shows the value x_i multiplied by the length 5 and a scaling factor f. This is then divided by the upper bound minus one (1). The value computed is then decrement by one (1). The value for the scaling factor f was established after extensive experimentation. It was found that when f was set to 100, there was a tight grouping of the value, with the retention of optimal filtration's of values. The subsequent formulation is given as:

$$x_i' = -1 + \frac{x_i \bullet f \bullet 5}{10^3 - 1} = -1 + \frac{x_i \bullet f \bullet 5}{10^3 - 1} \tag{5}$$

4.3 Backward transformation

The reverse operation to forward transformation, backward transformation converts the real value back into integer as given in Equation 6 assuming x_i to be the real value obtained from Equation 5.

$$\text{int}[x_i] = \frac{(1 + x_i) \bullet (10^3 - 1)}{5 \bullet f} = \frac{(1 + x_i) \bullet (10^3 - 1)}{500} \tag{6}$$

The value x_i is rounded to the nearest integer.

4.4 Recursive mutation

Once the solution is obtained after transformation, it is checked for feasibility. Feasibility refers to whether the solutions are within the bounds and unique in the solution.

$$x_{i,G+1} = \begin{cases} u_{i,G+1} \text{ if } \begin{cases} u_{j,i,G+1} \neq \left\{u_{1,i,G+1},...,u_{j-1,i,G+1}\right\} \\ x^{(lo)} \leq u_{j,i,G+1} \leq x^{(lo)} \end{cases} \\ x_{i,G} \end{cases} \tag{7}$$

Recursive mutation refers to the fact that if a solution is deemed in-feasible, it is discarded and the parent solution is retained in the population as given in Equation 7.

4.5 Repairment

In order to repair the solutions, each solution is initially vetted. Vetting requires the resolution of two parameters: firstly to check for any bound offending values, and secondly for repeating values in the solution. If a solution is detected to have violated a bound, it is dragged to the offending boundary.

$$u_{j,i,G+1} = \begin{cases} x^{(lo)} \text{ if } u_{j,i,G+1} < x^{(lo)} \\ x^{(hi)} \text{ if } u_{j,i,G+1} > x^{(hi)} \end{cases} \tag{8}$$

Each value, which is replicated, is tagged for its value and index. Only those values, which are deemed replicated, are repaired, and the rest of the values are not manipulated. A second sequence is now calculated for values, which are not present in the solution. It stands to reason that if there are replicated values, then some feasible values are missing. The pseudocode is given in Figure 3

Three unique repairment strategies were developed to repair the replicated values: *front mutation*, *back mutation* and *random mutation*, named after the indexing used for each particular one.

Algorithm for Replication Detection

Assume a problem of size n, and a schedule given as $X = \{x_1,..,x_n\}$. Create a *random* solution schedule $\exists! x_i : R(X) := \{x_1,..,x_i..,x_n\}; i \in Z^+$, where each value is unique and between the bounds $x^{(lo)}$ and $x^{(hi)}$.

1. Create a partial empty schedule $P(X) := \{\}$
2. For $k = 1,2,....,n$ do the following:
   (a) Check if $x_k \in P(X)$.
   (b) **IF** $x_k \notin P(X)$
   Insert $x_k \rightarrow P(X_k)$
   ELSE
   $P(X_k) = \emptyset$
3. Generate a missing subset $M(X) := R(X) \setminus P(X)$.

Fig. 3. Pseudocode for replication detection

Algorithm for Random Mutation

Assume a problem of size n, and a schedule given as $X = \{x_1, .., x_n\}$. Assume the missing subset $M(X)$ and partial subset $P(X)$ from Figure 3.

1. For $k = 1, 2, ..., n$ do the following:
   (a) **IF** $P(X_k) = \emptyset$
   Randomly select a value from the $M(X)$ and insert it in $P(X_k)$ given as $M(X_{Rnd}) \rightarrow P(X_k)$
   (b) Remove the used value from the $M(X)$.
2. Output $P(X)$ as the obtained complete schedule.

Fig. 4. Pseudocode for random mutation

4.5.1 Random mutation

The most complex repairment schema is the random mutation routine. Each value is selected randomly from the replicated array and replaced randomly from the missing value array as given in Figure 4.
Since each value is randomly selected, the value has to be removed from the array after selection in order to avoid duplication. Through experimentation it was shown that random mutation was the most effective in solution repairment.

4.6 Improvement strategies

Improvement strategies were included in order to improve the quality of the solutions. Three improvement strategies were embedded into the heuristic. All of these are one time application based. What this entails is that, once a solution is created each strategy is applied only once to that solution. If improvement is shown, then it is accepted as the new solution, else the original solution is accepted in the next population.

4.6.1 Standard mutation

Standard mutation is used as an improvement technique, to explore random regions of space in the hopes of finding a better solution. Standard mutation is simply the exchange of two values in the single solution.
Two unique random values are selected $r_1, r_2 \in rand[1, D]$, where as $r_1 \neq r_2$. The values indexed by these values are exchanged: $Solution_{r_1} \overset{exchange}{\leftrightarrow} Solution_{r_1}$ and the solution is evaluated. If the fitness improves, then the new solution is accepted in the population. The routine is shown in Figure 5.

4.6.2 Insertion

Insertion is a more complicated form of mutation. However, insertion is seen as providing greater diversity to the solution than standard mutation.
As with standard mutation, two unique random numbers are selected $r_1, r_2 \in rand[1, D]$. The value indexed by the lower random number $Solution_{r_1}$ is removed and the solution from that value to the value indexed by the other random number is shifted one index down. The removed value is then inserted in the vacant slot of the higher indexed value $Solution_{r_2}$ as given in Figure 6.

Algorithm for Standard Mutation

Assume a schedule given as $X = \{x_1, .., x_n\}$.

1. Obtain two random numbers r_1 and r_2, where $r_1 = rnd\left(x^{(lo)}, x^{(hi)}\right)$ and $r_2 = rnd\left(x^{(lo)}, x^{(hi)}\right)$, the constraint being $r_1 \neq r_2$
   (a) Swap the two indexed values in the solution
      i. $x_{r_1} = x_{r_2}$ and $x_{r_2} = x_{r_1}$.
   (b) Evaluate the new schedule X' for its objective given as $f(X')$.
   (c) **IF** $f(X') < f(X)$
      i. Set the old schedule X to the new improved schedule X' as $X = X'$.
2. Output X as the new schedule.

Fig. 5. Pseudocode for standard mutation

4.7 Local search

There is always a possibility of stagnation in evolutionary algorithms. DE is no exemption to this phenomenon.

Stagnation is the state where there is no improvement in the populations over a period of generations. The solution is unable to find new search space in order to find global optimal solutions. The length of stagnation is not usually defined. Sometimes a period of twenty generation does not constitute stagnation. Also care has to be taken as not be confuse the local optimal solution with stagnation. Sometimes, better search space simply does not exist. In EDE, a period of five generations of non-improving optimal solution is classified as stagnation. Five generations is taken in light of the fact that EDE usually operates on an average of a hundred generations. This yields to the maximum of twenty stagnations within one run of the heuristic.

Algorithm for Insertion

Assume a schedule given as $X = \{x_1, .., x_n\}$.

1. Obtain two random numbers r_1 and r_2, where $r_1 = rnd\left(x^{(lo)}, x^{(hi)}\right)$ and $r_2 = rnd\left(x^{(lo)}, x^{(hi)}\right)$, the constraints being $r_1 \neq r_2$ and $r_1 < r_2$.
   (a) Remove the value indexed by r_1 in the schedule X.
   (b) For k=r_1,.....,$r_2 - 1$, do the following:
      i. $x_k = x_{k+1}$.
   (c) Insert the higher indexed value r_2 by the lower indexed value r_1 as: $X_{r_2} = X_{r_1}$.
2. Output X as the new schedule.

Fig. 6. Pseudocode for Insertion

To move away from the point of stagnation, a feasible operation is a neighborhood or local search, which can be applied to a solution to find better feasible solution in the local neighborhood. Local search in an improvement strategy. It is usually independent of the search heuristic, and considered as a plug-in to the main heuristic. The point of note is that local search is very expensive in terms of time and memory. Local search can sometimes be considered as a brute force method of exploring the search space. These constraints make the insertion and the operation of local search very delicate to implement. The route that EDE has adapted is to check the optimal solution in the population for stagnation, instead of the whole population. As mentioned earlier five (5) non-improving generations constitute stagnation. The point of insertion of local search is very critical. The local search is inserted at the termination of the improvement module in the EDE heuristic.
Local search is an approximation algorithm or heuristic. Local search works on a *neighborhood*. A complete *neighborhood* of a solution is defined as the set of all solutions that can be arrived at by a move. The word solution should be explicitly defined to reflect the problem being solved. This variant of the local search routine is described in Onwubolu (2002) and is generally known as a 2-opt local search.
The basic outline of a local search technique is given in Figure 7. A number α is chosen equal to zero (0) ($\alpha = \emptyset$). This number iterates through the entire population, by choosing each progressive value from the solution. On each iteration of α, a random number i is chosen which is between the lower (1) and upper (n) bound. A second number β starts at the position i, and iterates till the end of the solution. In this second iteration another random number j is chosen, which is between the lower and upper bound and not equal to value of β. The values in the solution indexed by i and j are swapped. The objective function of the new solution is calculated and only if there is an improvement given as $\Delta(x,i,j) < 0$, then the new solution is accepted.
The complete template of EDE is given in Figure 8.

Algorithm for Local Search

Assume a schedule given as $X = \{x_1, .., x_n\}$, and two indexes α and β. The size of the schedule is given as n. Set $\alpha = 0$.

1. **While** $\alpha < n$
   (a) Obtain a random number $i = rand[1,n]$ between the bounds and under constraint $i \notin \alpha$.
   (b) Set $\beta = \{i\}$
      i. **While** $\beta < n$
         A. Obtain another random number $j = rand[1,n]$ under constraint $j \notin \beta$.
         B. **IF** $\Delta(x,i,j) < 0$; $\begin{cases} x_i = x_j \\ x_j = x_i \end{cases}$
         C. $\beta = \beta \cup \{j\}$
      ii. $\alpha = \alpha \cup \{j\}$

Fig. 7. Pseudocode for 2 Opt Local Search

Enhanced Differential Evolution Template

$$
\begin{array}{l}
\text{Input}: D, G_{\max}, NP \geq 4, F \in (0,1+), CR \in [0,1], \text{and bounds}: x^{(lo)}, x^{(hi)}. \\
\text{Initialize}: \left\{ \begin{array}{l} \forall i \leq NP \wedge \forall j \leq D \left\{ \begin{array}{l} x_{i,j,G=0} = x_j^{(lo)} + rand_j[0,1] \bullet \left(x_j^{(hi)} - x_j^{(lo)} \right) \\ if \;\; x_{j,i} \notin \left\{ x_{0,i}, x_{1,i}, ..., x_{j-1,i} \right\} \end{array} \right. \\ i = \{1,2,...,NP\}, j = \{1,2,...,D\}, G = 0, rand_j[0,1] \in [0,1] \end{array} \right. \\
\text{Cost}: \forall i \leq NP : f\left(x_{i,G=0} \right) \\
\left\{ \begin{array}{l} \text{While } G < G_{\max} \\ \quad \left\{ \begin{array}{l} \text{Mutate and recombine}: \\ r_1, r_2, r_3 \in \{1,2,....,NP\}, \text{randomly selected, except}: r_1 \neq r_2 \neq r_3 \neq i \\ j_{rand} \in \{1,2,...,D\}, \text{randomly selected once each } i \\ \forall j \leq D, u_{j,i,G+1} = \left\{ \begin{array}{l} \left(\gamma_{j,r_3,G} \right) \leftarrow \left(x_{j,r_3,G} \right) : \left(\gamma_{j,r_1,G} \right) \leftarrow \left(x_{j,r_1,G} \right) : \\ \left(\gamma_{j,r_2,G} \right) \leftarrow \left(x_{j,r_2,G} \right) \quad \text{Forward Transformation} \\ \gamma_{j,r_3,G} + F \cdot \left(\gamma_{j,r_1,G} - \gamma_{j,r_2,G} \right) \\ \qquad \text{if } \left(rand_j[0,1] < CR \vee j = j_{rand} \right) \\ \left(\gamma_{j,i,G} \right) \leftarrow \left(x_{j,i,G} \right) \quad \text{otherwise} \end{array} \right. \\ \forall i \leq NP \left\{ \begin{array}{l} \left(u'_{i,G+1} \right) = \left\{ \begin{array}{l} \left(\rho_{j,i,G+1} \right) \leftarrow \left(\varphi_{j,i,G+1} \right) \text{Backward Transformation} \\ \left(u_{j,i,G+1} \right) \overset{mutate}{\leftarrow} \left(\rho_{j,i,G+1} \right) \text{Mutate Schema} \\ \text{if } \left(u'_{j,i,G+1} \right) \notin \left\{ \left(u_{0,i,G+1} \right), \left(u_{1,i,G+1} \right), .. \left(u_{j-1,i,G+1} \right) \right\} \end{array} \right. \\ \left(u_{j,i,G+1} \right) \leftarrow \left(u'_{i,G+1} \right) \text{Standard Mutation} \\ \left(u_{j,i,G+1} \right) \leftarrow \left(u'_{i,G+1} \right) \text{Insertion} \\ \text{Select}: \\ x_{i,G+1} = \left\{ \begin{array}{l} u_{i,G+1} \text{ if } f(u_{i,G+1}) \leq f(x_{i,G}) \\ x_{i,G} \quad \text{otherwise} \end{array} \right. \end{array} \right. \\ G = G + 1 \\ \text{Local Search} \quad x_{best} = \Delta(x_{best}, i, j) \quad \text{if stagnation} \end{array} \right.
\end{array}
$$

Fig. 8. EDE Template

5. Chaotic systems

Chaos theory has its manifestation in the study of dynamical systems that exhibit certain behavior due to the perturbation of the initial conditions of the systems. A number of such systems have been discovered and this branch of mathematics has been vigorously researched for the last few decades.

The area of interest for this chapter is the embedding of chaotic systems in the form of *chaos number generator* for an evolutionary algorithm.

The systems of interest are *discrete dissipative* systems. The two common systems of Lozi map and Delayed Logistic (DL) were selected as mutation generators for the DE heuristic.

5.1 Lozi map

The Lozi map is a two-dimensional piecewise linear map whose dynamics are similar to those of the better known Henon map (Hennon, 1979) and it admits strange attractors.
The advantage of the Lozi map is that one can compute every relevant parameter exactly, due to the linearity of the map, and the successful control can be demonstrated rigorously.
The Lozi map equations are given in Equations 9 and 10.

$$y_1(t+1) = 1 - a|y_1(t)| + y_2(t) \tag{9}$$

$$y_2(t+1) = by_1(t) \tag{10}$$

The parameters used in this work are $a = 1.7$ and $b = 0.5$ as suggested in Caponetto et al. (2003). The Lozi map is given in Figure 9.

5.2 Delayed logistic map

The Delayed Logistic (DL) map equations are given in Equations 11 and 12.

$$y_1(t+1) = y_2 \tag{11}$$

$$y_2(t+1) = ay_2(1 - y_1) \tag{12}$$

The parameters used in this work is $a = 2.27$. The DL map is given in Figure 10.

6. Experimentation

The experimentation has been done on a few representative instances of both symmetric and asymmetric TSP problems. The chaotic maps are embedded in the place of the random

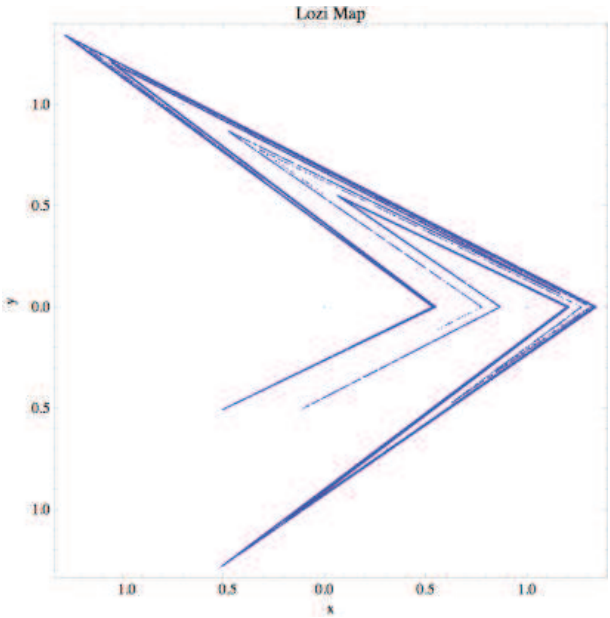


Fig. 9. Lozi map

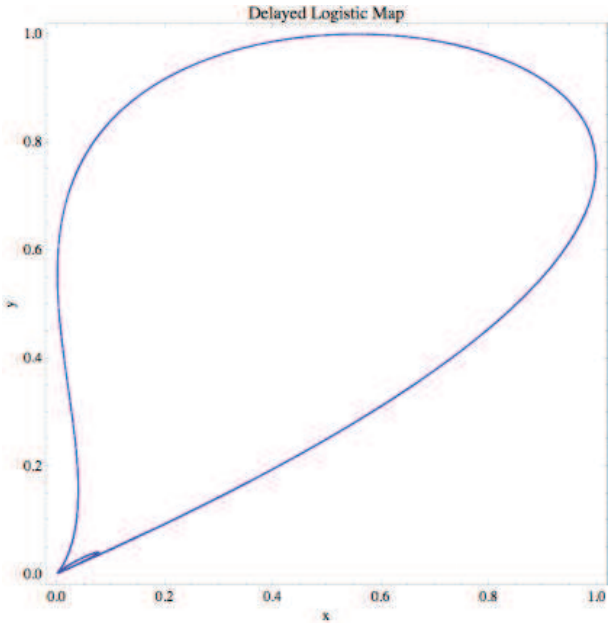


Fig. 10. Delayed Logistic

number generator in the EDE algorithm and the new algorithm is termed EDE_C. Five repeated experimentation of each instance is done by the two different chaotic embedded algorithms. The average results of all the ten experimentation is compared with EDE and published results in literature.

6.1 Symmetric TSP

Symmetric TSP problem is one, where the distance between two cities is the same *to* and *fro*. This is considered the easiest branch of TSP problem.

The operational parameters for TSP is given in Table 2.

Experimentation was conducted on the *City* problem instances. These instances are of 50 cities and the results are presented in Table 3. Comparison was done with Ant Colony (ACS) (Dorigo & Gambardella, 1997), Simulated Annealing (SA) (Lin et al., 1993), Elastic Net (EN) (Durbin & Willshaw, 1987), Self Organising Map (SOM) (Kara et al., 2003) and EDE of Davendra & Onwubolu (2007a). The time values are presented alongside.

In comparison, ACS is the best performing heuristic for TSP, and EDE_C is second best performing heuristic. On average EDE_C is only 0.02 above the values obtained by ACS. It

Parameter	Value
Strategy	5
CR	0.8
F	0.5
NP	100
Generation	50

Table 2. EDE_C TSP operational values

Instant	ACS	SA	EN	SOM	EDE	EDE_C
City set 1	5.88	5.88	5.98	6.06	5.98	5.89
City set 2	6.05	6.01	6.03	6.25	6.04	6.02
City set 3	5.58	5.65	5.7	5.83	5.69	5.61
City set 4	5.74	5.81	5.86	5.87	5.81	5.78
City set 5	6.18	6.33	6.49	6.7	6.48	6.21
Average	5.88	5.93	6.01	6.14	6	5.9

Table 3. Symmetric TSP comparison

must be noted that all execution time for EDE_C was under 10 seconds. Extended simulation would possibly lead to better results.

6.2 Asymmetric TSP

Asymmetric TSP is the problem where the distance between the different cities is different, depending on the direction of travel. Five different instances were evaluated and compared with Ant Colony (ACS) with local search (Dorigo & Gambardella, 1997) and EDE of Davendra & Onwubolu (2007a). The results are given in Table 4.

Instant	Optimal	ACS 3-OPT best	ACS 3-OPT average	EDE average	EDE_C average
p43	5620	5620	5620	5639	5620
ry48p	14422	14422	14422	15074	14525
ft70	38673	38673	38679.8	40285	39841
kro124p	36230	36230	36230	41180	39574
ftv170	2755	2755	2755	6902	4578

Table 4. Asymmetric TSP comparison

ACS heuristic performs very well, obtaining the optimal value, whereas EDE has an average performance. EDE_C significantly improves the performance of EDE. One of the core difference is that ACS employs $3-$Opt local search on each generation of its best solution, where as EDE_c has a $2-$Opt routine valid only in local optima stagnation.

7. Conclusion

The chaotic maps used in this research are of dissipative systems, and through experimentation have proven very effective. The results clearly validate that the chaotic maps provide a better alternative to random number generators in the task of sampling of the fitness landscape.

This chapter has just introduced the concept of chaotic driven evolutionary algorithms. Much work remains, as the correct mapping structure has to be investigated as well as the effectiveness of this approach to other combinatorial optimization problems.

8. Acknowledgement

The following two grants are acknowledged for the financial support provided for this research.

1. Grant Agency of the Czech Republic - GACR 102/09/1680
2. Grant of the Czech Ministry of Education - MSM 7088352102

9. References

Bland, G. & Shallcross, D. (1989). Large traveling salesman problems arising fromexperiments in x-ray crystallography: A preliminary report on computation, *Operation Research Letters* Vol. 8: 125–128.

Caponetto, R., Fortuna, L., Fazzino, S. & Xibilia, M. (2003). Chaotic sequences to improve the performance of evolutionary algorithms, *IEEE Transactions on Evolutionary Computation* Vol. 7: 289–304.

Davendra, D. (2001). *Differential evolution algorithm for flow shop scheduling*, Master's thesis, University of the South Pacific.

Davendra, D. & Onwubolu, G. (2007a). Enhanced differential evolution hybrid scatter search for discrete optimisation, *Proc. of the IEEE Congress on Evolutionary Computation*, Singapore, pp. 1156–1162.

Davendra, D. & Onwubolu, G. (2007b). Flow shop scheduling using enhanced differential evolution, *Proc.21 European Conference on Modeling and Simulation*, Prague, Czech Rep, pp. 259–264.

Davendra, D. & Zelinka, I. (2010). Controller parameters optimization on a representative set of systems using deterministic-chaotic-mutation evolutionary algorithms, *in* I. Zelinka, S. Celikovsky, H. Richter & C. G (eds), *Evolutionary Algorithms and Chaotic Systems*, Springer-Verlag, Germany.

Dorigo, M. & Gambardella, L. (1997). Ant colony system: A co-operative learning approach to the traveling salesman problem, *IEEE Transactions on Evolutionary Computation* Vol. 1: 53–65.

Durbin, R. & Willshaw, D. (1987). An analogue approach to the travelling salesman problem using the elastic net method, *Nature* Vol. 326: 689–691.

Garey, M. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco.

Hennon, M. (1979). A two-dimensional mapping with a strange attractor, *Communications in Mathematical Physics* Vol. 50: 69–77.

Kara, L., Atkar, P. & Conner, D. (2003). Traveling salesperson problem (tsp) using shochastic search. advanced ai assignment, *Carnegie Mellon Assignment* Vol. 15213.

Korte, B. (1988). Applications of combinatorial optimization, *13th International Mathematical Programming Symposium*, Tokyo.

Land, A. & Doig, A. (1960). An automatic method of solving discrete programming problems, *Econometrica* Vol. 28(3): 497–520.

Lin, F., Kao, C. & Hsu (1993). Applying the genetic approach to simulated annealing in solving np- hard problems, *IEEE Transactions on Systems, Man, and Cybernetics - Part B* Vol. 23: 1752–1767.

Lin, S. & Kernighan, B. (1973). An effective heuristic algorithm for the traveling-salesman problem, *Operations Research* Vol. 21(2): 498–516.

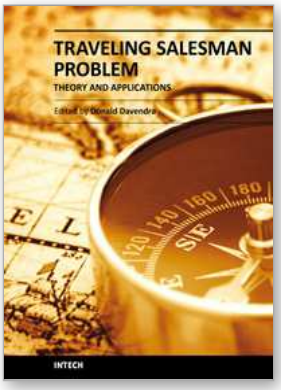
Onwubolu, G. (2002). *Emerging Optimization Techniques in Production Planning and Control*, Imperial Collage Press, London, England.

Onwubolu, G. (2005). Optimization using differential evolution, *Technical Report TR-2001-05*, IAS, USP, Fiji.

Onwubolu, G. & Davendra, D. (2006). Scheduling flow shops using differential evolution algorithm, *European Journal of Operations Research* 171: 674–679.

Onwubolu, G. & Davendra, D. (2009). *Differential Evolution: A Handbook for Global Permutation-Based Combinatorial Optimization*, Springer, Germany.

Price, K. (1999). An introduction to differential evolution, *New ideas in Optimisation*, US.



Traveling Salesman Problem, Theory and Applications
Edited by Prof. Donald Davendra

ISBN 978-953-307-426-9
Hard cover, 298 pages
Publisher InTech
Published online 30, November, 2010
Published in print edition November, 2010

This book is a collection of current research in the application of evolutionary algorithms and other optimal algorithms to solving the TSP problem. It brings together researchers with applications in Artificial Immune Systems, Genetic Algorithms, Neural Networks and Differential Evolution Algorithm. Hybrid systems, like Fuzzy Maps, Chaotic Maps and Parallelized TSP are also presented. Most importantly, this book presents both theoretical as well as practical applications of TSP, which will be a vital tool for researchers and graduate entry students in the field of applied Mathematics, Computing Science and Engineering.

How to reference
In order to correctly reference this scholarly work, feel free to copy and paste the following:

Ivan Zelinka, Roman Senkerik, Magdalena Bialic-Davendra and Donald Davendra (2010). Chaos Driven Evolutionary Algorithm for the Traveling Salesman Problem, Traveling Salesman Problem, Theory and Applications, Prof. Donald Davendra (Ed.), ISBN: 978-953-307-426-9, InTech, Available from: http://www.intechopen.com/books/traveling-salesman-problem-theory-and-applications/chaos-driven-evolutionary-algorithm-for-the-traveling-salesman-problem

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821