A Comparative Study on Differential Evolution and Genetic Algorithms for Some Combinatorial Problems

Brian Hegerty, Chih-Cheng Hung, and Kristen Kasprak

Southern Polytechnic State University, Marietta GA 30060, USA,
chung@spsu.edu

Abstract. This paper compares the performance of optimization techniques, differential evolution and genetic algorithm, for solving some combinatorial problems. Due to the NP-Complete nature of the N-Queen and traveling salesman problems, the differential evolution and genetic algorithm are used for optimizing the performance of the problems. The criteria used for comparison include 1) convergence speed, 2) computational complexity, 3) accuracy (near the optimal solution) and 4) stability. The empirical study shows that the differential evolution algorithm has a greater degree of computational complexity for combinatorial problems than the genetic algorithm. This largely arises from the encoding scheme used to represent the permutations as vectors and from redefining those vector operations inherent to differential evolution. The time required to reach convergence increases greatly as the problem size increases in both problems. The biggest advantage of the differential evolution approach over the genetic algorithm approach is its stability. The greatest setback for genetic algorithm's approach is problems with premature convergence. Our study shows that both approaches are able to find an optimal solution over time provided that premature convergence does not occur.

Key words: N-Queen, Traveling Salesman Problem, Differential Evolution, Optimization Techniques, Genetic Algorithms

1 Introduction

The purpose of the evaluation we performed was to provide an analysis of the performance of the genetic algorithm in comparison to the performance of the differential evolution algorithm in the realm of combinatorial problem spaces. The traveling salesman problem and the N-Queens problem are both classic examples of a combinatorial problem which is NP-Complete. The problems which are NP-Complete are presumed to have no complete solutions which can be implemented in polynomial time and are therefore good candidates for evolutionary algorithms. Furthermore the interest in NP-Complete problems is due to the theoretical nature of the NP-Complete problem which states that any problem in NP can be reduced to an instance of an NP-Complete problem in polynomial

time. This means that if we can find an efficient way to solve these combinatorial problems using evolutionary algorithms we should be able to apply these same algorithms to many different problems which have no fast algorithmic approach. We sampled various encodings of the traveling salesman problem and the N-Queens problem as well as various implementations of the genetic algorithm's design and values and the differential evolution's design and approach.

The paper is organized as follows. Section 2 introduces the Traveling Salesman Problem and the N-Queens Problem. Section 3 gives a brief introduction to Genetic Algorithms and Differential Evolution. Section 4 compares and contrasts Differential Evolution's approach to Genetic Algorithm's approach. Section 5 discusses the methods used to perform the experiment and the results which were obtained. Section 6 analyzes the experimental results and draws conclusions from those findings.

2 Traveling Salesman and N-Queens Problems

The traveling salesman problem is a classic NP-Complete combinatorial optimization problem in which the 'salesman' attempts to find the optimal path through a given set of cities visiting each city only once and ending in the starting city in an attempt to minimize the total cost, in time or distance, of travel between the cities. This problem can be modeled by the attempt to find a Hamiltonian tour through a set of vertices in a graph. Much research has been devoted to the development of efficient heuristics and approximations of lower bounds for the TSP [7, 9] and a significant amount of research has also been invested into the use of genetic algorithms and differential evolution in the solving of the TSP [5, 6]. The applications of solutions to the TSP can be found in many fields such as schedule planning, processor design, genome sequencing, and in a very direct manner logistics.

The N-Queens problem has been around since the mid 1800's and was initially proposed as a chess puzzle. Later studied by Gauss and Dijkstra and many other mathematicians and computer scientists the problem has become a classical one and is frequently used to teach about backtracking algorithms. The N-Queens problem can be stated formally, given an empty $N \times N$ chessboard we must position N queens on the board in a manner such that no queen can be attacked by another queen. Since the queen on a chessboard can move in any straight line in one of 8 directions there are multiple constraints. The N-Queens problem is also in the group of NP-Complete problems which means that algorithms which are efficient in the solution of this problem space should be able to be adapted to many other problems with relative ease.

3 Differential Evolution and Genetic Algorithm

The term genetic algorithm was popularized by Holland [4] in the early 1970's and further in the late 1980's by Goldberg [1]. The technique behind the genetic

algorithm is summarized in the Pseudo-code 3.1. The technique comes from Darwin's evolution, creating an initial set of 'chromosomes' and then letting 'natural selection' and 'genetic drift' take place to evolve the 'chromosome' into a better candidate for 'survival'. The specific means by which a genetic algorithm works lies heavily in the specific coding which is implemented and the implementations which were used will be discussed in section 5. Each 'chromosome' in a genetic algorithm is a single possible solution to the problem which is being solved, and the 'natural selection' is a process of crossover where some traits of better performing 'chromosomes' are incorporated with the traits of other 'chromosomes' to produce the next generation. The other force present in this evolution is the 'genetic drift' which is a type of mutation of a 'chromosome' and is usually represented by a probability which dictates the chance of random mutation in the form of inversion of a bit or a similar random change to the 'chromosome'.

Pseudo-code 3.1 Genetic Algorithm

```
Begin
   Generate randomly an initial population of solutions.
   Calculate the fitness of the initial population.
   Repeat
      Select a pair of parents based on fitness.
      Create two offspring using crossover.
      Apply mutation to each child.
      Evaluate the mutated offspring.
      All the offspring will be the new population, the parents will die.
   Until a stop condition is satisfied.
End.
```

The term differential evolution was coined by R. Storn and K. Price in their paper [2] as a product of their search to apply simulated annealing to the Chebyshev polynomial fitting problem. The method of differential evolution's functioning is similar to genetic algorithm's approach and is summarized in the Pseudo-code 3.2. Differential Evolution like the method of Genetic Algorithms allows each successive generation of solutions to 'evolve' from the previous generations strengths. The method of differential evolution can be applied to real-valued problems over a continuous space with much more ease than a genetic algorithm. The idea behind the method of differential evolution is that the difference between two vectors yields a difference vector which can be used with a scaling factor to traverse the search space. As in the genetic algorithm's beginning a random population is chosen, equally over the problem space, and to create the next generation an equal number of donor vectors are created through means of

$$\forall i \in n : D_i = X_a + F(X_b - X_c) \text{ where } i, a, b, c \text{ are distinct } . \qquad (1)$$

where X_b and X_c are randomly chosen and X_a is chosen either randomly or as one of the best members of the population (depending on individual encodings).

A trial vector $T_{i,j}$ is created by choosing between the donor vector and the previous generation for each element (j) according to the crossover rate CR[0–1], for each element in the vector we choose either the corresponding element from the previous generation vector or from the donor vector such that

$$\forall i, j \ : \text{ if } (random < CR \| j = J_{rand}) \text{ then } T_{i,j} = D_{i,j} \text{ otherwise } T_{i,j} = X_{i,j} \tag{2}$$

where J_{rand} is randomly chosen for each iteration through i and ensures that no T_i is exactly the same as the corresponding X_i. Then the trial vector's fitness is evaluated, and for each member of the new generation, X'_i, we choose the better performing of the previous generation, X_i, or the trial vector, T_i.

Pseudo-code 3.2 Differential Evolution

```
Begin
   Generate randomly an initial population of solutions.
   Calculate the fitness of the initial population.
   Repeat
      For each parent, select three solutions at random.
      Create one offspring using the DE operators.
      Do this a number of times equal to the population size.
      For each member of the next generation
         If offspring(x) is more fit than parent(x)
            Parent(x) is replaced.
   Until a stop condition is satisfied.
End.
```

4 A Comparison of Genetic Algorithms and Differential Evolution

The strength of a genetic algorithm lies in its ability to find a good solution to a problem where the iterative solution is too prohibitive in time and the mathematical solution is not attainable. [1] The way that the Genetic Algorithm works allows it to find this solution in a fast manner. Another value of the genetic algorithm's approach is that there can be many different constraints to the problem based on the specifics of the solution for which one is searching. One of the most noteworthy facets of the genetic algorithm is the Schema Theorem[4] which explains the relationship between groups of similar 'chromosomes' and their fitness. The theorem shows that a group with above average fitness should continue to increase it's fitness over successive generations. Another strength of genetic algorithm's approach is that there exists a proof of convergence for an elitist version of the genetic algorithm. While the concept of the genetic algorithm is not overly complicated, the individual parameters and implementation of the genetic algorithm usually require a large amount of tuning.

The strength of differential evolution's approach is that it often displays better results than a genetic algorithm and other evolutionary algorithms [10, 13, 14] and can be easily applied to a wide variety of real valued problems despite noisy, multi-modal, multi-dimensional spaces, which usually make the problems very difficult for optimization. Another impressive trait of differential evolution is that the parameters CR and F do not require the same fine tuning which is necessary in many other evolutionary algorithms[11]. Differential evolution has been used effectively in many applications on various domains such as neural network learning, digital signal processing, and image processing.

5 Experimental Results

Both the traveling salesman problem and the N-Queens problem offer a variety of different choices in implementations, each with its own difficulties. One must choose between different representations such as a permutation array, a permutation matrix, or a stack. There are many trade-offs to consider when choosing the encoding scheme for a potential solution, in this experiment we chose to use permutation array encoding to represent both problems. One reason this encoding was chosen was in order to provide relatively similar levels of complexity for use with the genetic algorithm's and the differential evolution's approaches. So each member of the population was stored as an integer array with N elements, equal to the number of cities in the traveling salesman problem or the size of the chessboard in the N-Queens problem. The permutation encoding represents the path between cities in the traveling salesman problem, and in the N-Queens problem the i^{th} element in the array represents the i^{th} column on the chessboard and the i^{th} element's value, j, represents the j^{th} row on the chessboard. The genetic algorithm and differential evolutionary algorithm were both applied exhaustively to the traveling salesman problem with the number of cities being 10, 15, 20, 25, and 50, and to the N-Queens problem with similar sizes for N. When possible the same functions were used in both algorithms, for example in the generation of a random sample population for the traveling salesman problem a single method call was developed to use the Fischer-Yates shuffle [12]. This was done in an attempt to create similar computational complexity and only measure the differences in the algorithms not the encodings.

In the specific encoding scheme for the genetic algorithm crossover technique candidates were chosen using roulette wheel selection and the crossover was accomplished by a modified partial order crossover. The mechanism of that modified partial order crossover is as follows: a crossover point was chosen in the parents X and Y at random and the child A was generated by copying the elements from X before the crossover point into A and removing those cities from Y. Then the remaining cities in Y were copied into A relative to their respective order in Y, this guarantees that no cycles are formed. Mutation was accomplished by switching the order of the city to be mutated with another city which was randomly chosen thus ensuring a valid path. Because it is very likely as the genetic algorithm converges that the best solutions will be lost in the

successive crossovers and mutations the scheme of Elitism was used in order to increase the efficiency of the algorithm. In Elitism the best performing member of the previous generation is copied into the next generation. In this encoding no effort was made to make use of any traveling salesman problem heuristics such as the n-opt switch, nearest neighbor heuristics, Lin-Kerringhan etc, because this was an attempt to compare the basic functioning of the genetic algorithm on a combinatorial problem space, not for a specific traveling salesman problem genetic algorithm.

The encoding used for the differential evolution's approach to the traveling salesman problem also used arrays of permutations, however an additional level of abstraction was used to allow us to perform addition, subtraction and multiplication on permutations. Lehmer code [3] is used to represent the permutations during the generation of the donor vector. The Lehmer code of permutations forms a bijection between the normal representation of a permutation and a representation formed by counting the numbers to the right which are larger than the current number, this relationship is formed by the use of modulo operations. This results in a greater level of computational complexity, as will be seen in the results below. The generation of the donor vector uses *DE/best/1*[8] in which the most fit member of the population is chosen for X_a, and two randomly chosen other members are chosen for X_b and X_c, according to the function

$$D_i = X_{best} + F(X_b - X_c). \tag{3}$$

The addition and subtraction is a simplistic task due to the nature of the Lehmer code, the numbers are all added, subtracted and multiplied by the scaling factor with no restrictions, and at the end of the generation of a donor vector member we use a modulo operation on each element in the Lehmer coded version of the permutation such that it ensures a valid transition back to a non-Lehmer code permutation. The generation of the trial vector is a little more complex. Starting with the element at index 0 in the array we choose the city, A, from the donor vector or the previous generation based on the CR as is typical for differential evolution. For each selection after that the selection process is the same, but we select the city which follows A in the donor vector or previous generation, depending on CR. In the case that this city is already present in the 'child' we choose the city following that city. This ensures that we do not create an invalid permutation but also adds to the computational complexity of the differential evolution's method. However, this method adheres to the basic logic of differential evolution and so it was chosen to maintain the integrity of the experiment. As in the genetic algorithm's design we made no effort to implement any heuristics.

The functioning of the genetic algorithm used in the N-Queens problem is similar to the implementation used for the traveling salesman problem. A tournament-based selection for obtaining crossover candidates was used. First, a random selection of 10 solutions was created as the tournament pool. Next, the top four in terms of fitness were chosen for potential mating. Finally , a random number was drawn, if this number was less than the crossover rate then the

pairs would mate. The crossover algorithm used is a partially mapped crossover as demonstrated in Figure 1. Two points A and B were chosen at random such that $A < B$. The first parent passed in, X, is considered to be the dominant parent and the child will most likely contain similar genes to that parent. The first step begins by copying all of the elements in X between A and B and giving them to the child in the range of A to B. Next, we start scanning the second parent, Y, again starting in the range A to B. Taking the i^{th} element's value, j, from the X, we find Y's j^{th} element value k. We then give Y's i^{th} element value to the child's kth element value. This step continues until the range A to B is exhausted. If a value is already in the kth element of the child, then the mapping begins again with the kth value in X until an open slot is found in the child solution. Finally, the rest of the element values in Y are copied into the child. Because this procedure produces only one child the algorithm will be repeated with X and Y switching roles to produce the second offspring. The mutation operator is represented by simply choosing two elements at random in the solution and swapping their values. The genetic algorithm used a selection scheme that represented a queue. Basically, the oldest solutions were lost. This technique was chosen after unsuccessful approaches of elitism and roulette-wheel. The need for a different selection criteria was due to problems with premature convergence. It is believed that the newer solutions did not have enough time to get an opportunity to cross and mutate.

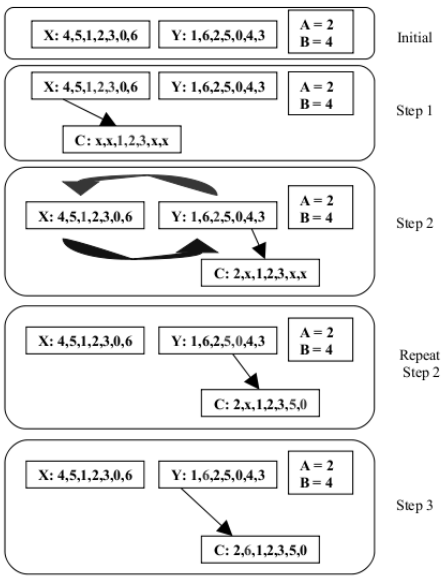

Fig. 1. Implementation of Partially Mapped Crossover

The differential evolution's 'crossover' technique is similar to that of the genetic algorithm. In fact the differential evolution uses the partially-mapped crossover, but the selection of the dominant parent is determined by the variable CR. Ordinarily, the crossover approach is to scan the solution arrays and to select the donor vector element value if a random number is less than the CR. But this would not preserve the permutation, so this approach was adjusted. A random value is still generated, but the solutions are not scanned element by element. Instead, if the number is less than the CR the original solution is set as the dominant parent in the partially-mapped crossover. Otherwise, it is the donor vector's solution whose genetic makeup we try to preserve. Since the differential evolution's mutation equation uses vector addition and subtraction, these operators needed to be defined appropriately for the permutation arrays. The vector addition/subtraction was constructed to utilize the definition of a permutation matrix. The second operand would be the column permutation mapping matrix for the first. For example, to perform $X - Y$, X will be used as a $1 \times N$ row matrix and Y will become an $N \times N$ identity matrix. First, the values of Ys elements are reversed and placed in an $N \times 1$ column matrix. The column matrix is expanded into a $N \times N$ identity matrix with a 1 being placed in the column that represents the original value of the column matrix. All other columns in the row contain a 0. The matrices are multiplied to produce a new $1 \times N$ row matrix as seen in Figure 2. Since the values in the permutations are integers the scalar multiplication of the mutation factor F also needed reconsideration. This operation was defined as a sweep over the elements in the intermediate solution and swapping the element value with another random element if a randomly generated number was less than F.

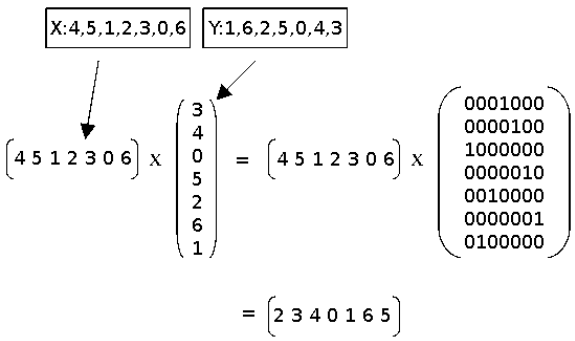

Fig. 2. Implementation of Permutation Matrix Operations

For the Traveling Salesman Problem the testing was performed on three different types of city sets: a circle, a rectangular grid, and a random group. The circle is a group of evenly distributed points on the circumference of the

circle, this test approximates a single optimal solution with no local minima to stall the algorithm. The rectangular grid offers a variety of local minima and several separate but equal global minima. The random group of cities offers the greatest challenge to the algorithms due to fact that there are often various local minima of varying proximity to the global minimum. The fitness function for both algorithms was a geometric calculation of the path between the cities, with a known optimum for the circular and rectangular grid sets. For the random group of cities the lowest value found by the genetic algorithm or differential evolution, each run 50 times on the problem, was used as the relative optimum for calculations of stability. Various sizes were used for all three types of city sets and for each city set size a stopping point was set that would terminate the algorithm after a certain number of generations. The number of generations which was chosen as a stopping point was significantly higher than either algorithm needed based on repeated tests with extremely long runs. Multiple tests were performed in order to determine the best values of CR and F for the Differential Evolution variables and similarly for the Genetic Algorithm's variables. In all tests the population was kept equivalent between the Genetic Algorithm and Differential Evolution. For the Differential Evolution CR was set to 0.3 and F was set at 1.1. For the Genetic Algorithm we chose a crossover rate of 45% and a mutation rate of 2.5% based on similar testing. Evaluation of the Differential Evolution and the Genetic Algorithm in the TSP are as follows. The Differential Evolution has an extremely low percentage of finding non-optimal solutions, and when they occur the average Differential Evolution non-optimal solution is better than the Genetic Algorithm's average non-optimal solution. The Differential Evolution converges between two to five times slower (in number of generations) and seems like the gap will continue to increase as the problem size increases. The Genetic Algorithm gets quick convergence with the trade-off of an increased probability of finding a local minimum and not a global minimum. Furthermore the Genetic Algorithm has a problem with staying in that non-optimal converged point and never leaving that local minimum even when given a surplus of running time (in generations). The Differential Evolution, on the other hand, proves much more robust and will continue to improve until an optimal solution is found, very rarely staying in a non-optimal solution for a significant period of time.

For the N-Queens problem the testing was performed in a loop with the stopping condition set to a fitness of 0. It was found that if the crossover rate for the genetic algorithm was any lower than 100% and the probability for crossover was not met then the previous generation and current generation would be exactly the same. This would cause a wasted iteration of the algorithm. To prevent this the crossover rate for the Genetic Algorithm was kept at 100%. Similar testing resulted in a value of 20% for the mutation rate. Even so, the genetic algorithm suffered from premature convergence for several population sizes. The values for the Differential Evolution's variables were set to $CR = 0.9$ and $F = 0.5$ based on tests to determine the optimal settings for those parameters. The differential evolution approach did not suffer from premature convergence at all. Both algorithms were tested for N sizes of 10, 15, 20, and 25. First, each value of N was

tested with population sizes of 5, 10, 25, 100, 500, and 1000. Next, each combination of N size and population size was given 10 trial runs recording the number of generations, running-time, and whether the solution discovered an optimal solution. The number of generations and runtime numbers were averaged. In most accounts differential evolution proved to find an optimal solution in fewer generations. The exception occurred with a population size of 5. Both algorithms iterated through a large number of generations for that population size, ranging from the 30,000 to 200,000. However, just by increasing the population size to 10, differential evolution saw a great improvement visible in a reduction in the number of generations. Dropping by about 75% in all cases. Another important number calculated was the standard deviation of the number of generations in each case. For all population sizes, except for 5, the standard deviation for differential evolution was very low compared to that found for genetic algorithm's.

6 Conclusions

It would seem from this set of experiments that the approach of Differential Evolution proved to be much more robust. In both problem spaces the Genetic Algorithm's results were less valuable than those of Differential Evolution's results. However, there is value in the relative speed of the Genetic Algorithm's results in the Traveling Salesman Problem if a local minimum will be sufficient. In the Traveling Salesman Problem both algorithms produced good results quickly, but the Differential Evolutionary Algorithm continued to improve on the tour of the cities, where the Genetic Algorithm could be observed to stall in a non-optimum solution with much greater frequency (Table 1). As both algorithms produced good results for small sized N, differential evolution had the advantage of stability as can be seen by the standard deviation (Table 2) of generations. In either algorithm, increasing the population size along with the value of N has an impact on the iteration time for a generation. This is as expected. It is important to note that with larger N values the number of generations was larger for genetic algorithm but the running time was comparable and in some cases better than differential evolution. This is due to the overhead caused by the computation complexity of differential evolution, in fact a measurement of the time to complete a single generation for Differential Evolution is on average four times larger than it takes to complete a similar generation (based on population size and problem size) for the Genetic Algorithm. But for larger population sizes it seems that differential evolution can outperform the genetic algorithm in both generations and runtime, as can be seen with the results from $N = 20$, population sizes = 100, 500, 1000. All of these facts make Differential Evolution's approach a better solution to these combinatorial problems.

The next question is why Differential Evolution's results are superior to Genetic Algorithm's. In the Traveling Salesman Problem a likely reason lies in the depth of the operations which are performed on the permutations. In the Differential Evolution approach each permutation goes through a much more involved type of mutation in the generation of the donor vector than in the mutation

Table 1. Testing Random City Sets 50 Evaluations on each set.

Traveling Salesman Problem			
Optimum Found (50 tests)		Average Generations	
DE	GA	DE	GA
94%	86%	4770.84	1970.76
100%	100%	3799.60	1220.62
98%	50%	3707.20	2182.70
100%	98%	3371.96	1034.36
100%	98%	3696.26	1333.08
94%	92%	3443.04	1448.44
100%	72%	4254.22	2568.72
88%	76%	4378.88	2202.02
100%	98%	3240.20	1108.44
94%	82%	3541.20	1489.94
98%	98%	3734.76	1447.90
90%	42%	3992.72	1875.68
80%	22%	4206.06	1787.18
82%	34%	3972.82	1829.40
94%	72%	4095.02	1790.36
96%	68%	2648.48	1822.92
100%	86%	3344.60	1267.10
96%	66%	4567.56	1600.38
96%	98%	3010.32	974.78
94%	84%	2941.14	2262.58
100%	78%	3228.90	1418.54

step of the Genetic Algorithm. The generation of the trial vector also allows for a greater reach in to the search space by taking individual elements from each permutation than the crossover stage permits with a single break and switch in the Genetic Algorithm. In the N-Queens problem the issue seems to be limited to genetic algorithm's tendency to find a good but non-optimal solution quickly but require a long time to transition towards the optimal solution from this point. In both problems the application of heuristics specific to that problem would greatly increase the efficiency and thus value of both the Genetic Algorithm and Differential Evolution. It would seem though that while both algorithm's have difficulties in adapting to a combinatorial and non-continuous problem space the basic mechanisms of differential evolution's optimization techniques provides better results. It is likely the case that further investigation into the adaptation of those optimization techniques from vectors into combinatorial types such as permutations and sets would result in large gains in the value of Differential Evolution's contribution to the optimization of combinatorial problem spaces.

Table 2. Testing Different Problem Sizes, 10 Evaluations on each set with standard deviation of generations.

N-Queens Problem				
Problem Size and Population Size	Optimum Found (10 tests)		Average Generations	
	DE	GA	DE	GA
15×5	100%	100%	116972.60	92625.50
15×10	100%	100%	21482.90	99246.60
15×25	100%	100%	7993.10	58475.60
15×100	100%	100%	800.70	6129.00
15×500	100%	90%	55.00	11960.11
15×1000	100%	100%	40.80	822.80
20×5	100%	90%	204162.80	49933.33
20×10	100%	100%	52784.20	81444.50
20×25	100%	100%	10870.60	27165.70
20×100	100%	100%	1421.10	8028.80
20×500	100%	80%	306.50	3085.75
20×1000	100%	80%	235.40	1575.50
Standard Deviation			7428.66	64112.45

References

1. Goldberg, David E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Kluwer Academic Publishers, Boston, MA. (1989)

2. Storn, R. and Price, K., "Differential Evolution — a Simple and Effecient Heuristic for Global Optimization over Continuous Spaces", Journal of Global Optimization, Kluwer Academic Publishers, **11** (1997) pp. 341–359.
3. Lehmer, H., "Teaching combinatorial tricks to a computer", In Proc. Sympos. Appl. Math. Combinatorial Analysis, **10**, Amer. Math. Soc., Providence, RI, (1960), pp. 179–193.
4. Holland, J. H., *Adaptation in natural and artificial systems.* University of Michigan Press, Ann Arbor, MI, (1975).
5. Peško, Š., "Differential Evolution for Small TSPs with Constraints", *Proceedings of the fourth International Scientific Conference: Challenges in Transport and Communications, Part III*, September 14–15, Pardubice, Czech Republic, (2006), pp. 989–994.
6. Sauer, J. G. and Coelho, L., "Discrete Differential Evolution with local search to solve the Traveling Salesman Problem: Fundamentals and case studies," *Cybernetic Intelligent Systems, 2008. CIS 2008. 7th IEEE International Conference on,* (2008) pp. 1–6.
7. Applegate, D., Cook, W., Rohe, A., "Chained Lin-Kernighan for large traveling salesman problems." INFORMS J. Comp. **15** (2003) pp. 82–92.
8. Price, K., Storn, R., Lampinen, J., *Differential Evolution — A Practical Approach to Global Optimization*, Springer, Berlin, (2005).
9. Applegate, D., Bixby, R., Chvatal, V., Cook, W., *The Traveling Salesman Problem: a Computational Study (Princeton Series in Applied Mathematics)*, Princeton University Press, (2007).
10. Xu, Xing and Li, Yuanxiang, "Comparison between Particle Swarm Optimization, Differential Evolution and Multi-Parents Crossover," *Computational Intelligence and Security, 2007 International Conference on*, (2007), pp. 124–127.
11. Mezura-Montes, E., "Nature-Inspired Algorithms Evolutionary and Swarm Intelligence Approaches", A Tutorial in MICAI 2008, (2008).
12. Fisher, R. A. and Yates, F., *Statistical tables for biological, agricultural and medical research* (3rd ed.). Oliver & Boyd, London. (1948), pp. 26–27.
13. Codreanu, I., "A parallel between differential evolution and genetic algorithms with exemplification in a microfluidics optimization problem." *Semiconductor Conference, 2005. CAS 2005 Proceedings. 2005 International* **2**, (2005), pp. 421–424.
14. Sentinella, M. R., "Comparison and integrated use of differential evolution and genetic algorithms for space trajectory optimization," *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on,*, (2007), pp. 973–978.
15. Bhandari, D., Murthy, C. A., Pal, S. K., "Genetic algorithm with elitist model and its convergence.", International journal of pattern recognition and artificial intelligence, **10**, (1996), pp. 731–745.