Tuning Genetic Algorithm Parameters using Design of Experiments

Mohsen Mosayebi
University of Rhode Island
Kingston, Rhode Island
mosayebi@uri.edu

Manbir Sodhi
University of Rhode Island
Kingston, Rhode Island
sodhi@uri.edu

ABSTRACT

Tuning evolutionary algorithms is a persistent challenge in the field of evolutionary computing. The efficiency of an evolutionary algorithm relates to the coding of the algorithm, the design of the evolutionary operators and the parameter settings for evolution. In this paper, we explore the effect of tuning the operators and parameters of a genetic algorithm for solving the Traveling Salesman Problem using Design of Experiments theory. Small scale problems are solved with specific settings of parameters including population size, crossover rate, mutation rate and the extent of elitism. Good values of the parameters suggested by the experiments are used to solve large scale problems. Computational tests show that the parameters selected by this process result in improved performance both in the quality of results obtained and the convergence rate when compared with untuned parameter settings.

CCS CONCEPTS

• **General and reference → Performance**; **Experimentation**; • **Computing methodologies → Concurrent algorithms**; *Combinatorial algorithms*;

KEYWORDS

Genetic Algorithm, Tuning Parameters, Design of Experiments

ACM Reference Format:
Mohsen Mosayebi and Manbir Sodhi. 2020. Tuning Genetic Algorithm Parameters using Design of Experiments. In *Genetic and Evolutionary Computation Conference Companion (GECCO '20 Companion), July 8–12, 2020, Cancún, Mexico.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3377929.3398136

1 INTRODUCTION

Genetic Algorithms (*GA*) are random search algorithms based on evolutionary principles, introduced by Holland [21] and developed by Goldberg [15]. *GA*s have been used for solving many NP-hard combinatorial optimization problems [25, 43]. Evolutionary computing (*EC*) specialists all acknowledge that good parameter values are essential for efficient evolutionary algorithms (*EA*), and *GA* is not an exception, but in practice parameters are mostly selected by the implementer's expertise and conventions in which crossover is high and mutation is low [9, 22, 44]. The parameters are detailed in figure 1 and categorised as follows:

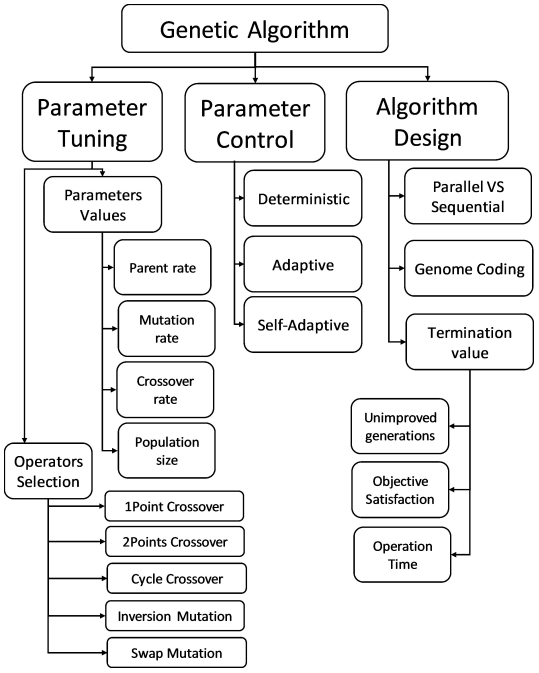

Figure 1: A framework of *GA* parameters.

1) The performance of a *GA* is greatly affected by the settings of parameters such as the mutation rate, the cross over rate, population size, elitism rate (parent rate) etc. Determining good parameter values in advanced is referred to as the parameter tuning problem.

2) To achieve (near-) optimal solutions, the algorithm may use different values of parameters in different stages of the algorithm. The strategy by which this change is controlled is known as the parameter control problem.

3) The same *GA* algorithm can be coded in different ways, and these can affect the performance considerably. For example, some *GA*s can be coded for parallel programming, whilst others are difficult to parallelize. Another fixed parameter that can make a significant difference is genome coding, and this is also a design parameter. Other factors that may have an impact include objective selection, termination conditions etc.

The tuning problem seeks settings for static parameters, while the control problem searches for a strategy to dynamically generate parameter settings. The control parameter procedures improve

GECCO '20, July 8–12, 2020, Cancun, Mexico

ACM ISBN 978-1-4503-7127-8/20/07…$15.00
https://doi.org/10.1145/3377929.3398136

Table 1: 10 most recent *GA* related papers and reviewing the tuning parameters explanation

Author	Population	Crossover-rate	Mutation-rate	Termination	Focuses
Abbasi [1]	1024-16384	0.8	0.02	N/A	A parallelizable *GA*
Ahmed [2]	50	1	0.09	Iteration	A new crossover operator
Alzyadat [4]	80-100	N/A	N/A	Iteration	Static and dynamic crossover
Ha [17]	60	N/A	N/A	Iteration	dynamic population
Hariyadi et al. [18]	N/A	0.25	0.2	No-improvement	Hybrid *GA* and *AI*
Jiang [20]	100	N/A	N/A	Iteration	Hybrid models of *GA* and *AC*
Koohestani [24]	500	1	0	N/A	A new crossover operator
Mousakazemi [32]	30	0.7	0.2	N/A	*PID* parameter control
Paul [35]	N/A	N/A	N/A	N/A	Comparing the crossovers
Thanh [46]	100	0.9	1/Nodes	Iteration	Two new crossovers

the *GA* performance by using real time data to change parameter settings for the run in progress. Static parameter tuning for initializing the algorithm is also important for this gives a good starting point for the control procedures to continue improvements from. The design parameter problem relates to decisions that an algorithm designer makes. As an example, developers may have different interpretations of the cycle crossover operator - some designers implementations of cycle crossovers with a parent and its position [34], while the others may use two parents for cycle crossover [14]. These design decisions can impact the performance of the *GA* significantly.

A review of ten papers reporting the use of genetic algorithms for solving different problems, and published in 2020, is summarized in Table 1. Except for Mousakazemi [32], who used the proportional–integral–derivative (*PID*) controller, the others have not explained how the parameters were tuned and the rationale for selecting the ranges used for their parameters. [1, 2, 4, 7, 17, 18, 20, 24, 35, 46]. In this paper, we propose a systematic approach based on well established Design of Experiments methods for determining the settings of parameters for a *GA* for solving *TSP*s.

Many parameter tuning methods [9, 38, 44], operator selection approaches [19, 29], and parameter control methods [8, 22, 32] have been reported over the past several decades. However, most have analyzed the effect of the parameter values on the objective function independently. There are some efforts to use design of experiments for evolutionary algorithm such as ant colony algorithm [40], genetic algorithm for scheduling problem [5], and vehicle routing problem with backhauls[42]. They just focused on the parameter values. In this paper we use the design of experiments, specifically general full factorial design, with small scale *GA* runs to analyze not only the effect of each parameter and operator but also the effects of the interactions between the parameters and operators. We then test the effect of the chosen parameter values and operators with large scale runs to evaluate the efficiency of this tuning.

Genetic Improvement (*GI*) utilizes automated search to evaluate software variations in order to find one that improves the criteria while preserves the desired properties [36]. The improved version of the software should behave identically to the previous version but is better because, for instance: it executes faster [27], with less memory [49], and less energy [6]. Although we have selected a well-known example, the *TSP*, to show the advantageous effect of parameter tuning in advance, the goal of this paper is to show how parameter settings based on *DOE* and pilot runs improve the performance of genetic algorithms, and the *GI* is not an exception. So, using a small-scale sample set to analyze the properties of a *GI* algorithm and searching for good initial parameter settings will improve the *GI* performance in both the speed and power.

The remaining parts of this paper are organized as follows: Section 2 details the methodology including the Design of Experiments approach. The results of the pilot runs for small scale problem and parameter values presented in Section 3. Computational results obtained from large scale runs using parameters prescribed in Section 4, followed by a conclusion in Section 5.

2 METHODOLOGY

Generally, *GA* generates a set of solutions called population of individuals (chromosomes) that are evaluated by a fitness function. After evaluating all individuals, parents are selected and a crossover mechanism is applied to obtain a new generation of offspring. A mutation function is also applied in order to preserve diversification (variation of genotypes) in the population. In the context of using *GA*'s for solving *TSP*s, a great deal of effort has been invested in developing new operators to improve the solution performance. Some of the operators developed include Edge Recombination Crossover (ERX) [48] , Order Based Crossover (OX2) [48], Distance Preserving Crossover (DPX) [13], Partially Mapped Crossover (PMX) [16], Improved PMX (IPMX) [24], Partition Crossover (PX) [48], Maximal Preservative Crossover (MPX) [33], Adaptive Sequential Constructive Crossover [2], ODV-based crossover operator [35], as well as different mutation operators such as Scramble mutation (SM) [45], Inversion mutation (IVM) [12], Insertion Mutation (ISM) [11], Greedy Sub-Tour Mutation (GSTM) [3].

Eiben and Smit [9] distinguished between operator design/ selection and value setting. The choice of operators is an integer choice, whereas the parameter values can be real values in $\mathbb{R}$ such as crossover rate (p_c), which can be $p_c \in [0, 1]$. Optimization methods can be used to find best values for continuously varying parameters, but for operator selection, the only option for exhaustive evaluation is enumeration . Different approaches for tuning parameters have been developed such as Taguchi Orthogonal Arrays [41], ANOVA [26], Racing [37], Meta-*GA* + Racing [50], REVAC [47], and F-RACE [47] (see [44] P 75).

Table 2: DOE factors and their levels

Parameter	level 1	level 2	Level3
Crossover operator	1-Point	2-Point	Cycle
Mutation operator	Inversion	Swap	
Population rate	10	20	-
Parent rate	0.7	0.9	-
Crossover rate	0.4	0.6	-
Mutation rate	0.05	0.1	-

Many researchers have tried to compare the effectiveness of operators used in Genetic Algorithms [4, 28, 34]. This is difficult, even for the same problem type, because of differences between the coding approaches and the hardware involved. While the quality of the solutions for test problems can be evaluated across different approaches, the solution times and iterations are not comparable. When faced with a new problem type, guidance from past problems is not easily applicable. Operators have to be reviewed and adapted for new problem types. Finding a good range of initial values for common parameters, such as population size, mutation rate etc. is crucial. This paper consider the *TSP* problem as an example to show the effect of tuning on operators, operator values, and design parameters. In this regard, we select three famous crossover operators: 1-point, 2-point [28], and Cycle crossover [14]; and two mutation operators: Inversion mutation and Swap mutation [10]. Also, we use the relative population rate to generate the population size as a fixed value in design parameters.

This paper considers two population sizes: 10 and 20 time the number of nodes in the *TSP*. Most *TSP* implementations consider the elitism rate for *GA* operators to be between 0.5 and 1.0. When the mutation rate is low, it is recommended that the crossover rate be high [9, 44]. In addition, the operators may create the offspring that are completely the same as one or some of the current members in the population. Duplicates must be removed, either after each iteration, or, for efficiencies sake, after a certain number of unimproved generations. The number of generations between removal of duplicates can be considered a design parameter that affects the solution but it is not a conventional parameter of the *GA*.

Design of experiments (*DOE*) methodology is a systematic method for experimental data analysis that ranges from a full factorial approach to a fractional factorial design. For all parameters of interest, different operating levels are first identified. It is recommended that these be two to three levels only, although uniform Latin Hypercube designs work on multiple levels of parameters in many dimensions. A full factorial design uses all combinations of the the parameters at different levels, whereas a fractional factorial selects a subset of level combinations. The goal of the design is to evaluate the effects and interactions of the parameters to determine the statistical significance of the main and interaction effects of parameters. In general, experiments are designed by adopting an orthogonal array from which experimental runs will be determined aiming at the specific results [23].

In order to find the efficient combination of the parameters and their values for using a *GA* to solve a *TSP*, an experiment is designed using 6 parameters, listed in Table 2 along with the level settings. Because the execution time for *GA* heuristics for small problems is short, a full factorial design is used as opposed to a fractional factorial design. Also, ten replications are used for each setting, result in a total of 960 ($3 * 2 * 2 * 2 * 2 * 2 * 5 = 960$) runs. The results are analyzed using Minitab 17.0 [30]. The termination condition used for all runs is when there is no improvement of the fitness value for a set number of generations, and this parameter was set to 30. The value of this parameter is a design parameter that can affect the quality of the objective function.

3 DESIGN OF EXPERIMENT RESULTS

As a test problem, gr17 from the well known *TSPLIB* problems [39] has been used for the pilot runs. The optimum solution for gr17 is 2085 units. Based on the results from the runs, Minitab determines the regression equation (1) relating the fitness value to the parameters. We show only the coefficients that exceed a value of 3.00 for brevity. Note, the sum of the coefficients of each parameter or combination of parameters totals zero, and the reader is referred to [31] for further discussion. Equation (1) shows the effect of the population rate (36.37) is much larger than the other coefficients. The implication of this is that keeping the population rate to 20 as opposed to 10 results in a much greater improvement in the objective function. The mutation operator coefficient (27.25) is second highest in value, and the interpretation in this case is that lower is better since the objective is to be minimized, and the first mutation option, i.e. inversion selection is preferred to the swap mutation. For the crossover operators, three settings are used, and the coefficients for the three levels are -8.31, -6.69 and +15. This indicates that when the first crossover option (1 point crossover) was used, the objective improved (reduced), on the average, by 8.31 units, as compared with an improvement of 6.69 for the second option (2 point crossover) and an increase (detriment) of the objective by 15 for the third level (cycle crossover). Thus, the first for crossovers is preferred. Based on the remaining coefficients, 90% of the parents are retained per generation, the mutation rate of 0.1 is preferable and the crossover rate of 0.6 is recommended.

Figure 2 and 3 present the interactions between the parameters and the main effect of the responses. In figure 2, the first column from right shows the interaction between the crossover operators and the other parameters. In all the graphs of this column, the 1-point crossover (operator 1) is shows a better (lower) fitness, and is therefore the preferred setting. Equation (1) shows the interactions coefficient between Mutation selection (choice) and Population Rate is 6.99. Recall, the recommended choice of Mutation selection was operator 1, and the Population Rate was 2. For this combination, the effect on the objective is positive, i.e. a deterioration. However, since the main effects are considerably larger than the interaction effects, the original choices are retained. A similar discussion can be proposed for the other parameters and interactions.

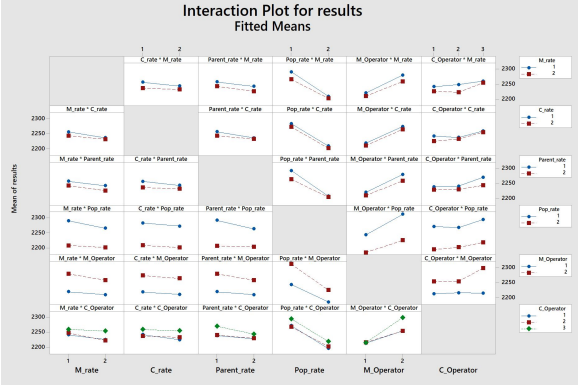

Figure 2: Interaction Plot for results.

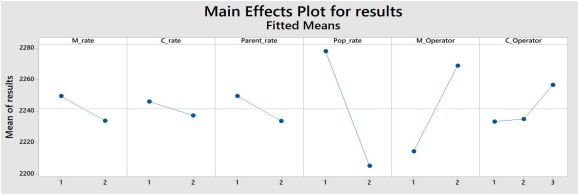

Figure 3: Main Effects Plot for results.

$$
\begin{aligned}
results = {} & 2241.71 + 7.82\, M_{rate_1} - 7.82\, M_{rate_2} \\
& + 4.39\, C_{rate_1} - 4.39\, C_{rate_2} \\
& + 7.96\, Parent_{rate_1} - 7.96\, Parent_{rate_2} \\
& + 36.37\, Pop_{rate_1} - 36.37\, Pop_{rate_2} \\
& - 27.25\, M_{Operator_1} + 27.25\, M_{Operator_2} \\
& - 8.31\, C_{Operator_1} - 6.69\, C_{Operator_2} + 15.00\, C_{Operator_3} \\
& + 6.47\, C_Operator * M_Operator_1 1 - 6.47\, C_Operator * M_Operator_1 2 \\
& + 8.08\, C_Operator * M_Operator_2 1 - 8.08\, C_Operator * M_Operator_2 2 \\
& - 14.55\, C_Operator * M_Operator_3 1 + 14.55\, C_Operator * M_Operator_3 2 \\
& - 3.65\, C_Operator * Pop_rate_2 1 + 3.65\, C_Operator * Pop_rate_2 2 \\
& - 3.00\, C_Operator * Parent_rate_1 1 + 3.00\, C_Operator * Parent_rate_1 2 \\
& + 5.40\, C_Operator * Parent_rate_3 1 - 5.40\, C_Operator * Parent_rate_3 2 \\
& + 4.40\, C_Operator * C_rate_1 1 - 4.40\, C_Operator * C_rate_1 2 \\
& + 4.78\, C_Operator * M_rate_2 1 - 4.78\, C_Operator * M_rate_2 2 \\
& - 4.78\, C_Operator * M_rate_3 1 + 4.78\, C_Operator * M_rate_3 2 \\
& - 6.99\, M_Operator * Pop_rate_1 1 + 6.99\, M_Operator * Pop_rate_1 2 \\
& + 6.99\, M_Operator * Pop_rate_2 1 - 6.99\, M_Operator * Pop_rate_2 2 \\
& + 6.33\, Pop_rate * Parent_rate_1 1 - 6.33\, Pop_rate * Parent_rate_1 2 \\
& - 6.33\, Pop_rate * Parent_rate_2 1 + 6.33\, Pop_rate * Parent_rate_2 2 \\
& + 4.61\, Pop_rate * M_rate_1 1 - 4.61\, Pop_rate * M_rate_1 2 \\
& - 4.61\, Pop_rate * M_rate_2 1 + 4.61\, Pop_rate * M_rate_2 2
\end{aligned}
\tag{1}
$$

Based on the interactions between the parameters and the main effect of the responses, the higher level of population, parent rate, crossover rate, and mutation rate with the inversion mutation and 1-point crossover result in a better response. Table 3 lists the preferred values of the tuned parameters.

One may argue that two crossover operators (1-point and 2-point) are close, and the third, cycle crossover can be disregarded from further investigation. We now design a second experiment full factorial design with five factors (parameter rate) and 2 levels for each factor, which is presented in Table 4.

Figure 4 shows the residual plot for results in which the residual data versus the fitted value and observation well dispersed as is desired.

Figure 5 shows that the population rate (symbol D); mutation operator (symbol E); mutation rate (symbol A); and the interaction between parent rate and population rate (symbol CD); as well as interaction between the mutation rate, population rate, and mutation operator (symbol ADE) are significant.

Table 3: General full factorial tuning results

Parameter	tuned level
Crossover operator	1-Point
Mutation operator	Inversion
Population rate	20
Parent rate	0.9
Crossover rate	0.6
Mutation rate	0.1

Table 4: DOE factors with 2 levels

Parameter	level 1	level 2
Crossover operator	1-Point	2-Point
Population rate	10	20
Parent rate	0.9	1.0
Crossover rate	0.6	0.8
Mutation rate	0.1	0.3

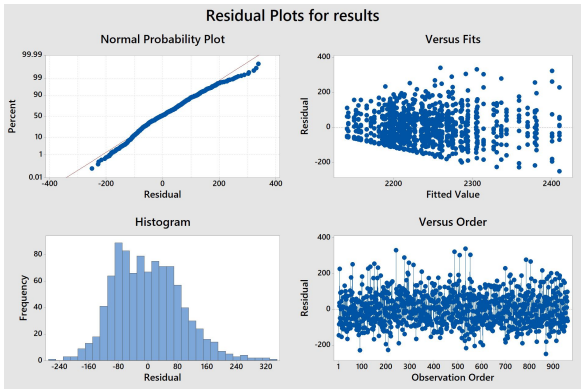

Figure 4: Residual Plots for results.

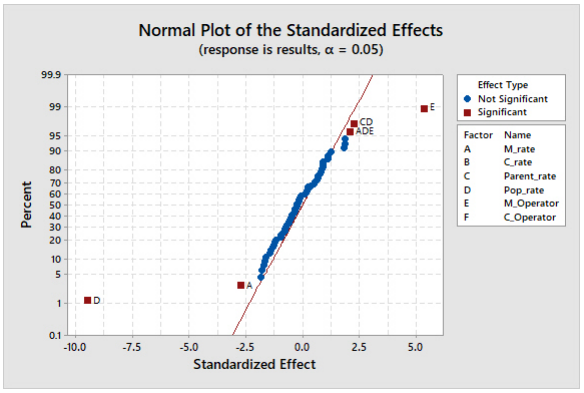

Figure 5: Effects Plot for results.

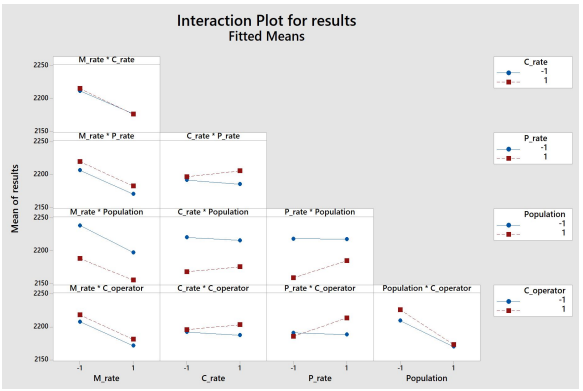

Figure 6: Interaction Plot for results.

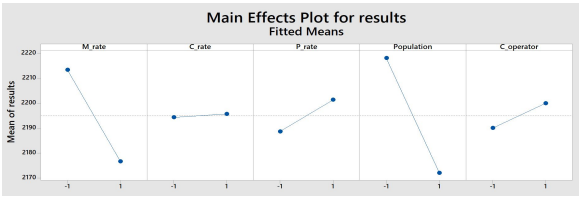

Figure 7: Main Effects Plot for results.

$$
\begin{aligned}
results = {} & 2194.94 - 4.96\, C_operator\_1 + 4.96\, C_operator_1 \\
& + 23.05\, Population\_1 - 23.05\, Population_1 \\
& - 6.33\, P_rate\_1 + 6.33\, P_rate_1 \\
& - 0.67\, C_rate\_1 + 0.67\, C_rate_1 \\
& + 18.42\, M_rate\_1 - 18.42\, M_rate_1 \\
& - 3.36\, C_operator * Population\_1 - 1 + 3.36\, C_operator * Population\_11 \\
& + 3.36\, C_operator * Population_1 - 1 - 3.36\, C_operator * Population_1 1 \\
& + 7.60\, C_operator * P_rate\_1 - 1 - 7.60\, C_operator * P_rate\_11 \\
& - 7.60\, C_operator * P_rate_1 - 1 + 7.60\, C_operator * P_rate_1 1 \\
& + 3.05\, C_operator * C_rate\_1 - 1 - 3.05\, C_operator * C_rate\_11 \\
& - 3.05\, C_operator * C_rate_1 - 1 + 3.05\, C_operator * C_rate_1 1 \\
& + 6.60\, Population * P_rate\_1 - 1 - 6.60\, Population * P_rate\_11 \\
& - 6.60\, Population * P_rate_1 - 1 + 6.60\, Population * P_rate_1 1 \\
& + 3.82\, P_rate * C_rate\_1 - 1 - 3.82\, P_rate * C_rate\_11 \\
& - 3.82\, P_rate * C_rate_1 - 1 + 3.82\, P_rate * C_rate_1 1
\end{aligned}
\tag{2}
$$

Figure 6 and 7 present the interactions between the parameters and the main effect of the responses. In figure 6, the first column from left shows the interaction between the mutation rate and the other parameters. In all graphs the higher mutation rate results in lower (lower objective value) fitness. So, the first selection is upper level of mutation rate. The graphs of crossover rate, in the second column from left, show no preference between the crossover rates. To decrease the processing time and based on the main effect of the crossover rate, we select the lower value. The graphs in the third row show the upper level of the population has a better fitness. The graphs in the last row show the first crossover operator has the better responses. Finally, based on the graphs in the third column from left as well as the result of the main effect, after selecting the upper level of population and the first crossover operation, the lower level of parent rate is chosen.

Based on the interactions between the parameters and the main effect of the responses, the larger population size and mutation rate along with inversion mutation, 1-point crossover, the lower level elitism and crossover rates result in better responses. Table 5 lists the tuned values.

Table 5: Full factorial tuning results

Parameter	tuned level
Crossover operator	1-Point
Mutation operator	Inversion
Population rate	20
Parent rate	0.9
Crossover rate	0.6
Mutation rate	0.3

Table 6: Computational results gr17

	Tuned		UnTuned	
Run	Obj.	# Gen	Obj.	# Gen
1	2309	62	2437	52
2	2249	68	2324	82
3	2157	66	2374	64
4	2085	116	2328	123
5	2269	93	2236	87
6	2138	81	2331	89
7	2085	75	2343	67
8	2331	74	2437	74
9	2213	86	2415	60
10	2090	92	2422	78
best	2085	116	2236	87

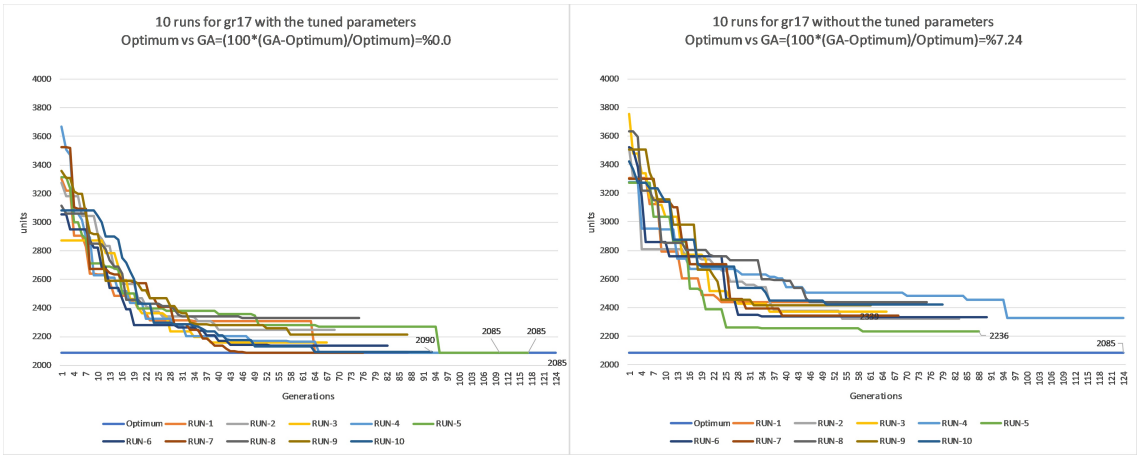

Figure 8: *GA* results, comparing tuned and untuned solutions for gr17 problem.

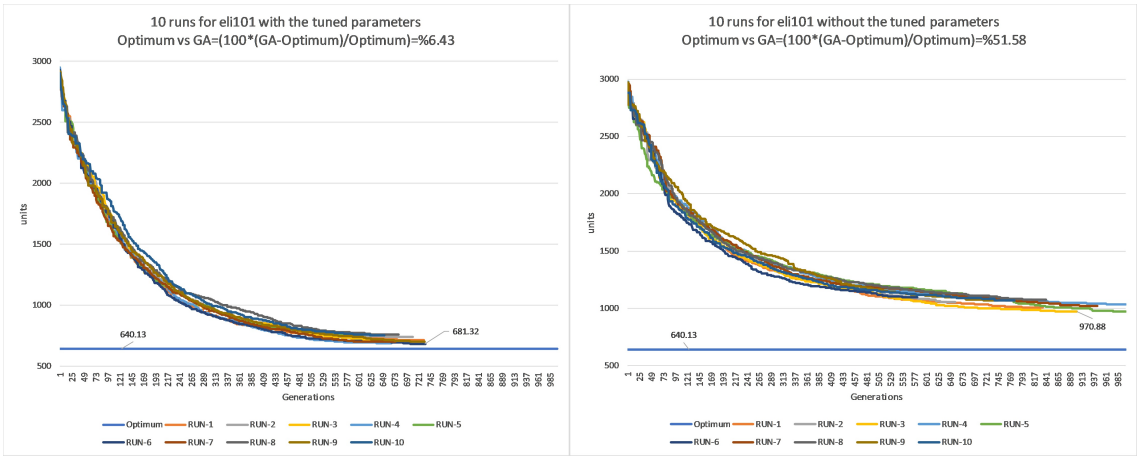

Figure 9: *GA* results, comparing tuned and untuned solutions for eli101 problem.

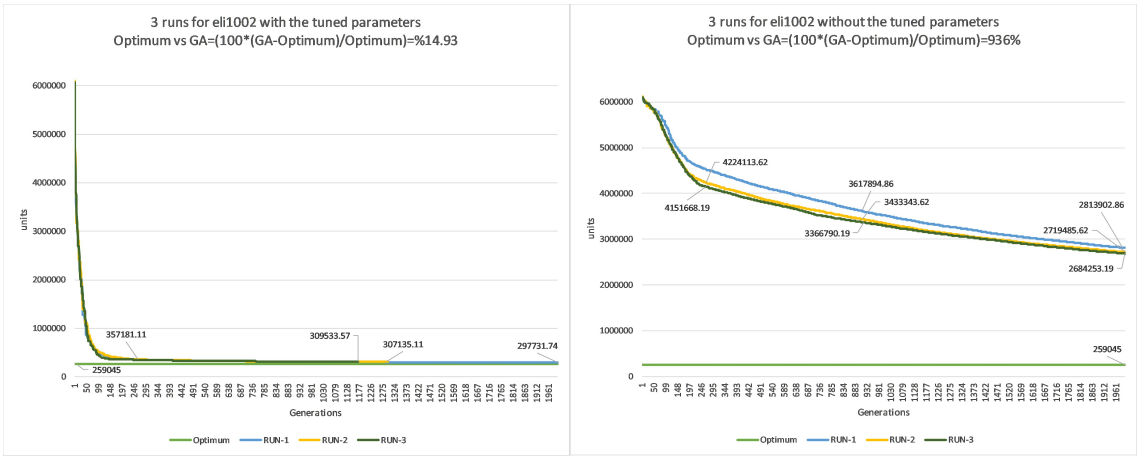

Figure 10: *GA* results, comparing tuned and untuned solutions for pr1002 problem.

Table 7: Computational results eli101

	Tuned		UnTuned	
Run	Obj.	# Gen	Obj.	# Gen
1	712.63	730	1003.06	833
2	735.56	708	1056.82	658
3	712.63	640	970	901
4	686.04	665	1030	998
5	755.37	604	971.63	999
6	681.32	733	1096	581
7	698.88	674	1019.54	942
8	758.17	680	1073	839
9	702.3	732	1064.84	743
10	752.05	652	1069.63	776
best	681.32	733	970	901

Table 8: Computational results pr1002

	Tuned		UnTuned	
Run	Obj.	# Gen	Obj.	# Gen
1	297731	1999	2364934	2999
2	307135	1298	2340669	2999
3	309533	1177	2508922	2760
best	297731	1999	2340669	2999

4 APPLICATION TO LARGE SCALE PROBLEM

Using small scale problems for parameter tuning is based on the idea that the size of the problem does not affect the original properties of the solutions. In this case, parameter values can be obtained from a small scale and applied to the medium-scale and large-scale problems. To analyze this functionality, this paper tests eli101 and pt1002 from well-known *TSPLIB* using the tuned parameters from gr17 from the same library.

The utility of tuning parameters using a design of experiments approach has been investigated by solving different *TSP*s with several settings. The first test involves the original problem - gr17. Ten runs were made with the tuned parameters, and ten runs were made with antithetic parameters - i.e. for two level parameters, the opposite setting was used. The results of these tests are shown in Figure 8. In these figures, it can be seen that the runs with tuned parameters have a greater convergence, terminate sooner, and obtain better terminal solutions. Table 6 shows the detailed results for both cases. A t-test for differences of means determines that the probability of the means being equal is rejected at the 95% level of confidence.

Figure 9 shows the results of ten runs for solving a 101 node problem (eli101 from *TSPLIB*) with and without tuned parameters. From this figure and from Table 7, again it can be seen that the results obtained using the tuned parameters are better than those obtained using the antithetical settings. Here too, a t-test for differences of means determines that the probability of the means being equal is rejected at the 95% level of confidence.

Finally, this approach has been used to solve a 1002 node *TSP* problem (pr1002 in *TSPLIB*). Because of the problem size, only three runs have been carried out. Figure 10 graphs the result of three runs of the *GA* for pr1002. For the tuned parameter runs, the gap between the final result of *GA* and optimum value (published with the test library) is %14.93 whereas the gap for the untuned *GA* is %950. The convergence rates for the tuned runs also appear to be much steeper.

5 CONCLUSIONS

This paper explored the effect of parameter tuning with genetic algorithms for solving the traveling salesman problem. It utilized a design of experiments approach. It showed that not only should the main effects related to operator selection and value setting be considered from pilot runs, but the interactions of parameters are also important. The process of working with small scale (pilot) runs for obtaining parameter settings was carried out, and the effectiveness of the settings so determined was shown with medium and large scale problems. Future work on the effectiveness of this algorithm to other evolutionary approaches for optimization are under investigation.

REFERENCES

[1] Mahdi Abbasi, Milad Rafiee, Mohammad R Khosravi, Alireza Jolfaei, Varun G Menon, and Javad Mokhtari Koushyar. 2020. An efficient parallel genetic algorithm solution for vehicle routing problem in cloud implementation of the intelligent transportation systems. *Journal of Cloud Computing* 9, 1 (2020), 6.

[2] Zakir Hussain Ahmed. 2020. Adaptive Sequential Constructive Crossover Operator in a Genetic Algorithm for Solving the Traveling Salesman Problem. *GAs* 11, 2 (2020).

[3] Murat Albayrak and Novruz Allahverdi. 2011. Development a new mutation operator to solve the traveling salesman problem by aid of genetic algorithms. *Expert Systems with Applications* 38, 3 (2011), 1313–1320.

[4] Tariq Alzyadat, Mohammad Yamin, and Girija Chetty. 2020. Genetic algorithms for the travelling salesman problem: a crossover comparison. *International Journal of Information Technology* 12, 1 (2020), 209–213.

[5] Arif Arin, Ghaith Rabadi, and Resit Unal. 2011. Comparative studies on design of experiments for tuning parameters in a genetic algorithm for a scheduling problem. *International Journal of Experimental Design and Process Optimisation* 2, 2 (2011), 102–124.

[6] Bobby R. Bruce, Justyna Petke, and Mark Harman. 2015. Reducing Energy Consumption Using Genetic Improvement. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO '15)*. Association for Computing Machinery, New York, NY, USA, 1327–1334. https://doi.org/10.1145/2739480.2754752

[7] Valentina Cacchiani, Carlos Contreras-Bolton, and Paolo Toth. 2020. Models and algorithms for the Traveling Salesman Problem with Time-dependent Service times. *European Journal of Operational Research* 283, 3 (2020), 825–843.

[8] Elena-Niculina Dragoi and Vlad Dafinescu. 2016. Parameter control and hybridization techniques in differential evolution: a survey. *Artificial Intelligence Review* 45, 4 (2016), 447–470.

[9] Agoston Endre Eiben and Selmar K Smit. 2011. Evolutionary algorithm parameters and methods to tune them. In *Autonomous search*. Springer, 15–36.

[10] Agoston E Eiben, James E Smith, et al. 2003. *Introduction to evolutionary computing*. Vol. 53. Springer.

[11] David B Fogel. 1988. An evolutionary approach to the traveling salesman problem. *Biological Cybernetics* 60, 2 (1988), 139–144.

[12] David B Fogel. 1993. Applying evolutionary programming to selected traveling salesman problems. *Cybernetics and systems* 24, 1 (1993), 27–36.

[13] Bernd Freisleben and Peter Merz. 1996. A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems. In *Proceedings of IEEE international conference on evolutionary computation*. IEEE, 616–621.

[14] DE Goldberg. 1989. Genetic algorithms in search, optimization, and machine learning, addison-wesley, reading, ma, 1989. *NN Schraudolph and J* 3, 1 (1989).

[15] David E Goldberg and John Henry Holland. 1988. Genetic algorithms and machine learning. (1988).

[16] David E Goldberg, Robert Lingle, et al. 1985. Alleles, loci, and the traveling salesman problem. In *Proceedings of an international conference on genetic algorithms and their applications*, Vol. 154. Lawrence Erlbaum, Hillsdale, NJ, 154–159.

[17] Quang Minh Ha, Yves Deville, Quang Dung Pham, and Minh Hoàng Hà. 2020. A hybrid genetic algorithm for the traveling salesman problem with drone. *Journal of Heuristics* 26, 2 (2020), 219–247.

[18] Putri Mutira Hariyadi, Phong Thanh Nguyen, Iswanto Iswanto, and Dadang Sudrajat. 2020. Traveling Salesman Problem Solution using Genetic Algorithm. *Journal of Critical Reviews* 7, 1 (2020), 56–61.

[19] Khalid Jebari and Mohammed Madiafi. 2013. Selection methods for genetic algorithms. *International Journal of Emerging Sciences* 3, 4 (2013), 333–344.

[20] Chao Jiang, Zhongping Wan, and Zhenhua Peng. 2020. A new efficient hybrid algorithm for large scale multiple traveling salesman problems. *Expert Systems with Applications* 139 (2020), 112867.

[21] Holland John. 1992. Holland, Adaptation in natural and artificial systems. (1992).

[22] Giorgos Karafotias, Mark Hoogendoorn, and Ágoston E Eiben. 2014. Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation* 19, 2 (2014), 167–187.

[23] John Kechagias, Konstantinos Kitsakis, and Nikolaos Vaxevanidis. 2017. Comparison of Full versus Fractional Factorial Experimental Design for the prediction of Cutting Forces in Turning of a Titanium alloy: A case study. *Int. J. Mater.* 4 (2017), 1–4.

[24] Behrooz Koohestani. 2020. A Crossover Operator for Improving the Efficiency of Permutation-Based Genetic Algorithms. *Expert Systems with Applications* (2020), 113381.

[25] Manoj Kumar, Mohammad Husian, Naveen Upreti, and Deepti Gupta. 2010. Genetic algorithm: Review and application. *International Journal of Information Technology and Knowledge Management* 2, 2 (2010), 451–454.

[26] Tangudu Bharat Kumar, Apurbaranjan Panda, Gaurav Kumar Sharma, Arun Kishor Johar, Sougata Kumar Kar, and Dharmendar Boolchandani. 2020. Taguchi DoE and ANOVA: A systematic perspective for performance optimization of cross-coupled channel length modulation OTA. *AEU-International Journal of Electronics and Communications* 116 (2020), 153070.

[27] W. B. Langdon and M. Harman. 2015. Optimizing Existing Software With Genetic Programming. *IEEE Transactions on Evolutionary Computation* 19, 1 (2015), 118–135.

[28] Siew Mooi Lim, Abu Bakar Md Sultan, Md Nasir Sulaiman, Aida Mustapha, and KY Leong. 2017. Crossover and mutation operators of genetic algorithms. *International journal of machine learning and computing* 7, 1 (2017), 9–12.

[29] Qiuzhen Lin, Zhiwang Liu, Qiao Yan, Zhihua Du, Carlos A Coello Coello, Zhengping Liang, Wenjun Wang, and Jianyong Chen. 2016. Adaptive composite operator selection and parameter control for multiobjective evolutionary algorithm. *Information Sciences* 339 (2016), 332–352.

[30] Minitab-Inc. 2016. Minitab -17.0. (Accessed on 02/03/2020).

[31] Douglas C Montgomery. 2017. *Design and analysis of experiments*. John wiley & sons.

[32] Seyed Mohammad Hossein Mousakazemi. 2020. Computational effort comparison of genetic algorithm and particle swarm optimization algorithms for the proportional–integral–derivative controller tuning of a pressurized water nuclear reactor. *Annals of Nuclear Energy* 136 (2020), 107019.

[33] Heinz Mühlenbein. 1989. Parallel genetic algorithms, population genetics and combinatorial optimization. In *Workshop on Parallel Processing: Logic, Organization, and Technology*. Springer, 398–406.

[34] IM Oliver, DJd Smith, and John RC Holland. 1987. Study of permutation crossover operators on the traveling salesman problem. In *Genetic algorithms and their applications: proceedings of the second International Conference on Genetic Algorithms: July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA*. Hillsdale, NJ: L. Erlhaum Associates, 1987.

[35] P Victer Paul, C Ganeshkumar, P Dhavachelvan, and R Baskaran. 2020. A novel ODV crossover operator-based genetic algorithms for traveling salesman problem. *Soft Computing* (2020), 1–31.

[36] Justyna Petke, Saemundur O Haraldsson, Mark Harman, William B Langdon, David R White, and John R Woodward. 2017. Genetic improvement of software: a comprehensive survey. *IEEE Transactions on Evolutionary Computation* 22, 3 (2017), 415–432.

[37] Saleem Ramadan and Mahmoud Barghash. 2016. Three-Step Parameters Tuning Model for Time-Constrained Genetic Algorithms. *Modern Applied Science* 10, 10 (2016).

[38] Iloneide CO Ramos, Marco César Goldbarg, Elizabeth G Goldbarg, and Adriao Duarte Dória Neto. 2005. Logistic regression for parameter tuning on an evolutionary algorithm. In *2005 IEEE Congress on Evolutionary Computation*, Vol. 2. IEEE, 1061–1068.

[39] Gerhard Reinelt. 1991. TSPLIB—A traveling salesman problem library. *ORSA journal on computing* 3, 4 (1991), 376–384.

[40] Enda Ridge. 2007. *Design of experiments for the tuning of optimisation algorithms*. Citeseer.

[41] M Sadeghifar, R Sedaghati, and V Songmene. 2016. FE modeling and optimization of cutting temperature in orthogonal turning. *Simulation* 922, 482 (2016), 178.

[42] A Saremi, TY ElMekkawy, and GG Wang. 2007. Tuning the parameters of a memetic algorithm to solve vehicle routing problem with backhauls using design of experiments. *International Journal of Operations Research* 4, 4 (2007), 206–219.

[43] N Sathya and A Muthukumaravel. 2015. A review of the optimization algorithms on traveling salesman problem. *Indian Journal of Science and Technology* 8, 29 (2015), 1–4.

[44] Selmar Kagiso Smit. 2012. *Parameter tuning and scientific testing in evolutionary algorithms*. Vrije Universiteit.

[45] Gilbert Syswerda. 1991. Scheduling optimization using genetic algorithms. *Handbook of genetic algorithms* (1991).

[46] Pham Dinh Thanh, Huynh Thi Thanh Binh, and Bui Thu Lam. 2015. New mechanism of combination crossover operators in genetic algorithm for solving the traveling salesman problem. In *Knowledge and Systems Engineering*. Springer, 367–379.

[47] Niki Veček, Marjan Mernik, Bogdan Filipič, and Matej Črepinšek. 2016. Parameter tuning with Chess Rating System (CRS-Tuning) for meta-heuristic algorithms. *Information Sciences* 372 (2016), 446–469.

[48] Darrell Whitley, Doug Hains, and Adele Howe. 2009. Tunneling between optima: partition crossover for the traveling salesman problem. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. 915–922.

[49] Fan Wu, Westley Weimer, Mark Harman, Yue Jia, and Jens Krinke. 2015. Deep Parameter Optimisation. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO '15)*. Association for Computing Machinery, New York, NY, USA, 1375–1382. https://doi.org/10.1145/2739480.2754648

[50] Bo Yuan and Marcus Gallagher. 2007. Combining meta-EAs and racing for difficult EA parameter tuning tasks. In *Parameter Setting in Evolutionary Algorithms*. Springer, 121–142.