DETERMINING THE BEST SETTINGS FOR THE OPERATORS AND PARAMETERS OF GENETIC ALGORITHMS: A METHODOLOGY AND ITS APPLICATION TO TRAVELING SALESPERSON PROBLEM

GENETİK ALGORİTMALARIN OPERATÖRLERİ VE PARAMETRELERİ İÇİN EN İYİ AYARLARI BELİRLEME: BİR METODOLOJİ VE GEZGİN SATICI PROBLEMİNDE UYGULAMASI

YAVUZHAN AKDURAN

PROF. DR MURAT CANER TESTİK

Supervisor

Submitted To
The Graduate School of Science and Engineering of Hacettepe University
as a Partial Fulfilment of the Requirements
for the Award of the Degree of Master of Science
in Industrial Engineering

2020

ABSTRACT

DETERMINING THE BEST SETTINGS FOR THE OPERATORS AND PARAMETERS OF GENETIC ALGORITHMS: A METHODOLOGY AND ITS APPLICATION TO TRAVELING SALESPERSON PROBLEM

Yavuzhan AKDURAN

Master of Science, Department of Industrial Engineering

Supervisor: Prof. Dr. Murat Caner TESTİK

July 2020, 101 pages

Genetic Algorithms (GAs) are heuristic algorithms that are used to approximate the optimal solutions of optimization problems. They are inspired by the theory of natural evolution, where a population of solutions evolves through generations and only the fittest individuals survive at the end. GAs perform very well in many optimization problems in terms of approximation quality and run time. However, a typical GA has several operators such as mutation and crossover, and parameters such as population size and generation number that affect the performance of the GA significantly. In the literature, the operators and parameters of GAs are set based on either the previous experiences of users or trial-error experiments since finding optimum settings of GAs is quite difficult.

In this thesis, a methodology is developed for effectively setting the operators and parameters of GAs. Hence, the best settings that will exploit the potential of the used genetic algorithm can be determined. Typically, performance of a GA is evaluated based on two criteria: (1) approximation quality and (2) run time. Approximation quality of an algorithm is determined based on the closeness of the solution found by the algorithm to the optimal solution. Run time is measured by the computational time the algorithm consumes until termination. In general, there are trade-offs between these two criteria, i.e. higher approximation quality requires more run time, and a GA is expected to find a solution with high approximation quality in a short time. Settings of the operators and parameters affect both criteria, and different settings can be advantageous in terms of different criteria. Therefore, we model the problem of effectively setting the parameters of a GA as a multi-objective optimization problem, using approximation quality and run time as the objectives.

In the thesis, we employ a multi-objective evolutionary algorithm (MOEA) to solve the problem and discover the trade-offs between approximation quality and run time of GAs. MOEAs are population-based heuristics that mimic natural evolution process and find a well-converged and well-diversified set of nondominated solutions. In our approach, each solution of MOEA represents a setting for operators and parameters of the GA considered. To evaluate a solution in the population, the GA is run using the settings defined in the solution. The fittest settings in terms of approximation quality and run time survive through generations. At the end, a set of settings, each of which is better on another criterion, is found.

The developed methodology is demonstrated on travelling salesperson problems (TSP). A GA that is used to solve TSP is selected. Several alternatives for operators and parameters are considered for the GA, and the best settings are investigated by experimenting on 31 TSP instances selected from the literature. The set of best settings is searched using NSGA-II, a well-known MOEA. Then a greedy heuristic is developed to help decision makers to reduce the size of the set of final solutions based on their preferences.

Keywords: Genetic algorithms, operator and parameter settings, multi-objective optimization, multi-objective evolutionary algorithms, travelling salesperson problem.

ÖZET

GENETİK ALGORİTMALARIN OPERATÖRLERİ VE PARAMETRELERİ İÇİN EN İYİ AYARLARI BELİRLEME: BİR METODOLOJİ VE GEZGİN SATICI PROBLEMİNDE UYGULAMASI

Yavuzhan AKDURAN

Yüksek Lisans, Endüstri Mühendisliği Bölümü

Tez Danışmanı: Prof. Dr. Murat Caner TESTİK

Temmuz 2020, 101 Sayfa

Genetik Algoritmalar (GA'lar), optimizasyon problemlerinin en uygun çözümlerine yaklaşmak için kullanılan sezgisel algoritmalardır. Bir çözüm popülasyonunun nesiller boyunca evrimleştiği ve sonunda sadece en uygun bireylerin hayatta kaldığı doğal evrim teorisinden ilham almıştır. GA'lar çözüm kalitesi ve çalışma süresi açısından birçok optimizasyon probleminde başarılı olarak uygulanmıştır. Bununla birlikte, tipik bir GA'nın mutasyon ve çaprazlama gibi çeşitli operatörleri ve performansını önemli ölçüde etkileyen nüfus büyüklüğü ve üretim sayısı gibi parametreleri vardır. Literatürde, GA'ların operatörleri ve parametreleri belirlenirken genellikle kullanıcıların önceki deneyimleri veya deneme yanılma deneyleri kullanılmaktadır.

Bu tezde, GA'ların operatörlerini ve parametrelerini etkili bir şekilde ayarlayan bir metodoloji geliştirilmiştir. Tipik olarak bir GA'nın performansı iki kritere göre değerlendirilir: (1) yaklaşık çözüm kalitesi ve (2) çalışma süresi. Bir algoritmanın yaklaşım kalitesi, algoritma tarafından bulunan çözümün optimal çözüme yakınlığına bağlı olarak belirlenir. Çalışma süresi, algoritmanın çözümü sonlandırmaya kadar kullandığı hesaplama süresiyle ölçülür. Genel olarak, bu iki kriter arasında ödünleşme vardır, daha iyi çözüm kalitesi daha fazla çalışma süresi gerektirir. Bir GA'nın kısa sürede yüksek kaliteye sahip bir çözüm bulması beklenmektedir. Operatörlerin ve parametrelerin ayarları her iki kriteri de etkiler ve farklı ayarlar farklı kriterler açısından avantajlı olabilir. Bu nedenle, etkili ayarların belirlenmesi çok amaçlı bir optimizasyon problemidir.

Bu tez kapsamında, GA'ların yaklaşık kalitesi ve çalışma süresi arasındaki dengeleri keşfetmek için çok amaçlı bir evrimsel algoritma (MOEA) kullanılmaktadır. MOEA'lar doğal evrim sürecini taklit eden nüfus tabanlı sezgisel yöntemlerdir ve iyi yakınsamış ve iyi çeşitlendirilmiş baskın olmayan çözümler bulurlar. Yaklaşımımızda, MOEA'nın her bir çözümü, GA'nın operatörleri ve parametreleri için bir ayarı temsil etmektedir. Popülasyondaki bir çözümü değerlendirmek için GA çözümde tanımlanan ayarlar kullanılarak çalıştırılır. Yaklaşık kalite ve çalışma süresi açısından en uygun çözümler nesiller boyunca varlığını sürdürmektedir. Sonunda, her biri başka bir kriterde daha iyi olan bir dizi ayar bulunur.

Geliştirilen metodoloji, gezici satıcı probleminde (TSP) denenmiştir. Farklı problem boyutlarına sahip 31 TSP örneği literatürden seçilmiştir. TSP'yi çözmek için kullanılan GA’nın operatörleri ve parametreleri için çeşitli alternatifler dikkate alınmakta ve en iyi ayarlar aranmaktadır. En iyi ayarlar kümesinin belirlenmesi için tanınmış bir MOEA olan NSGA-II kullanılmaktadır. Karar vericilerin tercihlerine göre nihai çözüm kümesinin boyutunu azaltmalarına yardımcı olmak için bir sezgisel tarama da geliştirilmiştir.

Anahtar Kelimeler: Genetik algoritma, operatör ve parametre ayarları, çok amaçlı optimizasyon, çok amaçlı evrimsel algoritma, gezgin satıcı problemi.

ACKNOWLEDGMENTS

I would like to extend my thanks to the following people who have an important contribution to my master thesis, education, and career. First of all, I would like to thank my dear supervisor Prof. Dr. Murat Caner TESTİK who has guided my undergraduate and graduate life with his unique knowledge and experience. He has guided me not only in this thesis, but also in my academic life and career, and I have always taken himself as a role model. Furthermore, I would like to thank Erdi DAŞDEMİR for his support and guidance. He showed a great effort and always supported to me during this thesis. He provided many research directions throughout the study and guided me for their implementations. I also would like to thank to both of my teachers for their sincerity.

I am also grateful to the University at Buffalo and Prof. Dr. Rajan Batta for allowing us to access their Center for Computational Research for high performance computing resources.

I would like to express my gratitude to my family members: my father Nihat AKDURAN, my mother Gülsüm AKDURAN and my brother Alperen AKDURAN. I owe all my successes and careers in my life to them and their loves.

Finally, I would like to thank my dear friends Büşra YÜCEL, Erdem BALTACI, İsmail BAL, Ömer Faruk ŞENOCAK, Tuğçe Eda SİNANGİL, Büşra BİŞKİN and Kübra ÇİFTÇİER for their invaluable friendships, support, and their encouragements to me for an academic career. I will always remember and miss our time at the university, the lessons we work together until late at night, and our wonderful travel memories.

Yavuzhan AKDURAN

July 2020, Ankara

CONTENTS

ABSTRACT i

ÖZET iii

ACKNOWLEDGMENTS v

FIGURES ix

TABLES x

SYMBOLS AND ABBREVIATIONS xi

1. INTRODUCTION 1

1.1. Problem Definition 2
1.2. Motivation 3
1.3. Contribution of the Thesis 4
1.4. Thesis's Organization 4

2. BACKGROUND AND LITERATURE REVIEW 6

2.1. Genetic Algorithm 6
2.2. Setting Operators and Parameters of Genetic Algorithms 7
2.3. Multi-Objective Optimization 10
2.3.1. Definitions 10
2.3.2. Multi-objective Evolutionary Algorithm 12
2.4. Travelling Salesperson Problem 13

3. METHODOLOGY AND ITS IMPLEMENTATION TO TSP 16

3.1. Searching Effective Settings for GA Using MOEA 16
3.2. NSGA-II 17
3.3. Encoding in NSGA-II 19
3.4. Operators and Parameters Considered 20
3.4.1. Selection Operator 20
3.4.2. Crossover Operator 20
3.4.3. Mutation Operator 21
3.4.4. Crossover and Mutation Probabilities 21

3.4.5. Population Size ... 21
3.4.6. Number of Generations ... 22
3.5. Computing Fitness Value of a Solution ... 22
3.6. Computational Settings of NSGA-II ... 23
3.7. Demonstration on TSP ... 24
3.7.1. Encoding of Solutions When Solving TSP ... 26
3.7.2. Operators and Parameters of Application ... 26
3.7.3. Alternatives Considered for the Selection Operator ... 27
3.7.4. Alternatives Considered for the Crossover Operator ... 28
3.7.5. Alternatives Considered for the Mutation Operator ... 31
3.7.6. Alternatives Considered for the Parameters: Crossover and Mutation Probability, Population Size and Number of Generations ... 33
3.7.7. Computing Fitness Value of Solutions When Solving TSP ... 34
4. RESULTS AND DISCUSSION ... 35
4.1. Finding the Pareto-Optimal Sets of the TSP Instances ... 35
4.1.1. The Results of “dantzig42” ... 38
4.1.2. The Results of “st70” ... 41
4.1.3. The Results of “u159” ... 43
4.2. Reducing the Size of the Final Population of NSGA-II for the DM ... 47
4.2.1. Results of Small-Sized Problems ... 49
4.2.2. Results of Medium-sized Problems ... 51
4.2.3. Results of Large-Sized Problems ... 52
4.2.4. Testing the Effective Settings of the Small-sized Instances on the Medium and Large-sized Instances ... 54
5. CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS ... 57
6. REFERENCES ... 62
APPENDICES ... 69
Appendix 1 – List of Problems ... 69
Appendix 2 – The final solution set of MOEA for TSP problems in the optimality gap tolerance (Seed=70646) ... 71
Appendix 3 – Settings for TPS problems with 0-50 nodes (small-sized problems) ... 77

Appendix 4 – Settings for TPS problems with 50-100 nodes (medium-sized problems) 78
Appendix 5 – Settings for TPS problems with 100-200 nodes (large-sized problems) 80
Appendix 6 – Thesis Originality Report 82
CURRICULUM VITAE 83

FIGURES

Figure 1 An illustration of the procedures applied in one generation of NSGA-II 18
Figure 2 NSGA-II procedure (adapted from Deb et al. [19].) 19
Figure 3 Chromosome structure of the solutions of NSGA-II 19
Figure 4 Crossover operator used in NSGA-II 24
Figure 5 An example chromosome structure for a TSP instance having 13 nodes 26
Figure 6 An example cycle crossover operation 29
Figure 7 An example partially matched (mapped) crossover operation 30
Figure 8 An example order crossover operation 31
Figure 9 An example position-based crossover operation 31
Figure 10 An example simple inversion mutation operation 32
Figure 11 An example insertion mutation operation 32
Figure 12 An example exchange mutation operation 33
Figure 13 An example displacement mutation operation 33
Figure 14 An example scramble mutation operation 33
Figure 15 Final solution set of NSGA-II when GA is used to solve “dantzig42” and some selected efficient solutions for illustration 39
Figure 16 Final solution set of NSGA-II when GA is used to solve “st70” and some selected sample solutions for illustration 41
Figure 17 Final solution set of NSGA-II when GA is used to solve “u159” and some selected sample solutions for illustration 44
Figure 18 Preferred Pareto-optimal solutions of “dantzig42” 50
Figure 19 Preferred Pareto-optimal solutions of “st70” 51
Figure 20 Preferred Pareto-optimal solutions of “u159” 53

TABLES

Table 1 Selected alternatives for operator and parameter settings 27
Table 2 Percentage of times operators and parameters appear in the 125 settings of small-sized problem class 36
Table 3 Percentage of times operators and parameters appear in the 602 efficient settings of medium-sized problem class 37
Table 4 Percentage of times operators and parameters appear in the 1294 efficient settings of large-sized problem class 37
Table 5 Selected efficient solutions for illustration when GA is used to solve “dantzig42” 39
Table 6 Percentage of times operators and parameters appear in the 30 efficient settings of “dantzig42” 41
Table 7 Selected efficient solutions for illustration when GA is used to solve “st70” 42
Table 8 Percentage of times operators and parameters appear in the 49 efficient settings of “st70” 43
Table 9 Selected efficient solutions for illustration when GA is used to solve “u159” 44
Table 10 Percentage of times operators and parameters appear in the 99 efficient settings of “u159” 45
Table 11 Best five setting for small-sized TPS problems 51
Table 12 Best five setting for medium-sized TPS problems 52
Table 13 Best five setting for large-sized TPS problems 54
Table 14 The average optimality gaps of the five best settings of the small-sized problem class for the medium-sized problem instances 55
Table 15 The average optimality gaps of the five best settings of the small-sized problem class for the large-sized problem instances 55
Table 16 Percentage of times operators and parameters appear in the efficient solutions having less than 10% optimality gap of the small (sm), medium (md) and large-sized (lg) problem classes (values in the table are in terms of %) 56
Table 17 List of problems 69
Table 18 The final solution set of MOEA for TSP problems 71
Table 19 All final settings for TPS problems with 0-50 nodes 77
Table 20 All final settings for TSP problems with 50-100 nodes 78
Table 21 All final settings for TSP problems with 100-200 nodes 80

SYMBOLS AND ABBREVIATIONS

Symbols

$f_i(x)$	i^{th} objective function with value *x*
d_{ij}	Distance between *i* and *j*
x_{ij}	Decision variable for an arc between *i* and *j*
G	A complete TSP graph
A	Set of arcs
N	Set of nodes
ps	Population size
P_t	Patent population at generation *t*
Q_t	Offspring population at generation *t*
R_t	Global population at generation *t*
M_t	Mating pool population at generation *t*
P_{mut}	Probability of mutation
P^i	i^{th} parent
Q^i	i^{th} offspring
$G(x)$	The optimality gap function
x_i	Coordinate of node i on x-axis
y_i	Coordinate of node i on y-axis
$P_s(x)$	Selection probability of solution *x*

Abbreviations

EA	Evolutionary Algorithm
GA	Genetic Algorithm
TSP	Travelling Salesman Problem
ASTP	Asymmetrical Travelling Salesman Problem
VRP	Vehicle Routing Problem
NSGA-II	Nondominated Sorting Genetic Algorithm
MOEA	Multi-objective Evolutionary Algorithm
s	Selection Operator
c	Crossover Operator
pc	Probability of Crossover
m	Mutation Operator
pm	Probability of Mutation
ps	Population Size
ng	Number of Generations

1. INTRODUCTION

Genetic Algorithms (GA) are heuristic search methods that mimic the randomness in the genetic functioning of nature. They simulate the gene transfer and mutation present in the theory of evolution. Typically, a GA starts with an initial population of solutions. This initial population evolves throughout generations. In each generation, good solutions are selected to produce new solutions to form the population of the next generation. When producing new solutions, chromosomes of the selected good solutions are crossed or mutated. Better solutions remain alive through generations and only the best ones survive at the termination. Similar to the evolution in nature, solutions with better gene structures survive and lead to the production of stronger generations.

GAs are generally successful in quickly approximating the optimal solutions of difficult optimization problems. However, there are some factors affecting the performance of a GA. A typical GA uses many operators and parameters during the solution process. These operators and parameters need to be set before using the algorithm to solve an optimization problem. Yet, there are various operators and parameters, and many alternatives need to be evaluated to determine their best settings. Moreover, the effect of an operator or parameter is often problem dependent, and an effective setting for an algorithm may not be as good for another algorithm. In the following, we will use the term “settings” to refer to the settings of the operator and parameters of an algorithm.

Settings often have significant effects on the performance of a GA in terms of approximation quality and run time. For instance, while a large selection of population size reduces the convergence ability of a GA and increases computational requirements, a small population size may lead to having a small search space and getting stuck on local optimum. Therefore, determining the best settings is substantial to optimize the performance of the algorithm.

In general, operator and parameter settings are determined according to the previous experiences of the users and trial and error experiments. There are few studies that address the optimization of operator and parameter settings of GAs. Although these studies show the effect of one or more factors on the algorithm, they are weak in providing a specific setting to the user. In this study, we develop an approach to investigate the optimum settings of a GA. The developed approach is used to solve traveling salesperson problems (TSP) from the literature. In the following subsections, we define the problem, present our motivation and contributions, and report the organization of the thesis.

1.1. Problem Definition

Determining the best settings of the operators and parameters of a GA is often at least as important as developing the GA. In general, GAs have several operators and parameters that need to be determined. In this study, we experiment with the most common three operators and four parameters:

- **Operators:** Selection operator, crossover operator, mutation operator.
- **Parameters**: Probability of crossover, probability of mutation, population size and number of generations.

A setting affects the performance of an algorithm in terms of two criteria:

1. **Approximation Quality:** Approximation quality measures the closeness of the solution found by the algorithm to the optimal solution.
2. **Run Time:** Run time measures the computational time the algorithm consumes until termination.

Typically, a GA is expected to approximate the optimal solution well and quickly. Yet, there are trade-offs among the two criteria, i.e. better approximations require more run time. Depending on the requirements of the problem solved and preferences of users, different settings may be advantageous. For example, a GA can be used to find a good initial solution for a commercial solver. For such a case, a user may demand quick solutions and sacrifice from the approximation quality since the solver will improve the initial solution.

In this study, we define the problem of finding effective settings of genetic algorithms as a bi-objective optimization problem. We develop an approach that discovers the trade-offs between approximation quality and run time to help users when setting the operators and parameters of the GA they use. In our approach, we use multi-objective evolutionary algorithms (MOEA) to find the best set of settings, each of which presents an advantage in terms of either approximation quality or run time. MOEAs are population-based heuristics that mimic natural evolution process and find a well-converged and well-diversified set of nondominated solutions. In our approach, each solution in a population is a setting of operators and parameters of the GA considered. Each is evaluated in population based on its approximation quality and run time. Hence, to evaluate a solution, the GA considered is run using the settings the solution contains.

The developed approach is implemented on well-known TSPs. 31 different TSP instances having different problem sizes are selected from the literature, and optimal settings for operators and parameters are searched. In terms of operators, we evaluate four alternatives for each of selection and crossover operators, and five alternatives for mutation operator. In terms of parameters, we evaluate six levels for crossover and mutation, and twelve levels for population size and generation number. Theoretically, there are 414,720 different settings considering all combinations of the alternatives. We search nondominated set of these 414,720 alternatives using NSGA-II, a well-known MOEA. Then a greedy heuristic is developed to help decision makers (DMs) to reduce the size of the set of final solutions based on their preferences.

1.2. Motivation

The literature focuses on development and implementation of GAs, and often ignores the selection of the settings of operators and parameters. Typically, parameters and operators are set based on either the past experiences of the users or trial-error experiments. Considering that a well-chosen setting for parameters and operators can significantly increase the performance of a GA, finding the optimum settings turns out to be a substantial issue.

Our motivation is to develop an approach that help users for effectively setting the operators and parameters of the GA they use. We aim to fill the gap in deciding the settings of parameters and operators of GAs in the literature. Developed approach is demonstrated on TSP, which is a well-studied optimization problem in the literature. There are many GAs developed for TSP and we aim to present an approach that can be used to effectively determine the settings of the algorithms.

1.3. Contribution of the Thesis

This study contributes to the literature as follows:

- A methodology to effectively determine the settings of the operators and parameters of GAs is developed. Two performance criteria, i.e. approximation quality and run time, are used when determining the settings. Several operators and parameters are considered simultaneously and the effects of different settings on the approximation quality and run time of GAs are investigated.
- Instead of a single setting, a final set of settings is presented to decision makers in order to present the trade-offs between approximation quality and run time. Decision makers may choose the settings to use from this set based on their preferences.
- Developed methodology is demonstrated on finding the best settings for a GA when solving TSP. 31 TSP problem instances having different sizes are studied and a greedy heuristic is developed for reducing the size of the set of efficient parameter settings.

1.4. Thesis's Organization

This thesis is organized as follows. In Chapter 2, the literature, some fundamental definitions of multi-objective optimization, and some background about GA, MOEA and TSP are reviewed. In Chapter 3, the developed methodology is presented. This section presents step by step how the best settings of operators and parameters of GAs are searched. In Chapter 4, TSP implementation and its results are presented. Recommended settings are shared according to the results of the problem instances we use. Finally, Chapter 5 concludes the thesis and discuss future research directions.

2. BACKGROUND AND LITERATURE REVIEW

This study develops an approach to effectively set the operators and parameters of the GAs. The problem is defined as a bi-objective optimization problem. The developed approach is implemented on a set of TSP instances to determine the best settings when solving TSP. In the following, some background about genetic algorithms and multi-objective optimization are presented as well as the relevant literature is briefly reviewed.

2.1. Genetic Algorithm

GA is one of the most popular heuristics used for solving optimization problems. A GA provides ease and speed for solving complex and difficult problems and has an important place in the optimization literature. It has a wide field of applicability.

GA basically mimic the evolution and selection process in nature. It adopts the struggle for survival in the wild. While stronger individuals transfer their genes to future generations, weaker ones are eliminated. The fundamentals of GAs are presented by John Holland in 1975 [1].

Before explaining how GA works, some basic concepts need to be known. Firstly, GA creates a set of random solutions. These solutions constitute a population and the number of solutions in the population is called as population size. Each solution in the population has a chromosome consisting of several genes. In GAs, populations behave similar to the nature. Some solutions are selected as parents that are matched and mated between each other through selection and crossover operators, respectively, to produce new offspring solutions. Apart from these, genes may be damaged in nature. This process is called as mutation in the GA terminology. Solutions are subject to mutation through generations. At every generation, GA tries to produce better offspring that will form the next generations. GA converges to the optimal solution eventually [2].

The basic procedures implemented in one generation of a GA are as follows [3] [4]:

1. Objective function value of each solution in the population is computed.
2. Two solutions (parents) from the population are selected.
3. Selected parents are crossed to create new solutions (offspring).
4. Mutation is implemented to the offspring solutions.

In general, Steps 2-4 are repeated until the number of generated offspring solutions is equal to the population size. Then the algorithm proceeds to the next generation where Steps 1-4 are repeated. Whole procedure is repeated until a termination criterion is met. Typically, the number of allowed generations is used as the termination criteria. Other termination criteria such as the minimum required improvement between the best solutions of subsequent generations are also used in the literature. Interested readers are referred to the Mitchell [2], Deb [5], Coley [6], Davis [7] for further details of GAs.

GAs are strong alternatives to exact optimization approaches with their approximation power and computational advantage. Yet, setting their operators and parameters is a challenging problem on its own. The performance of a GA often depends on the settings of its operators and parameters. These settings have significant effects on their approximation power and computational requirements [8] [9]. In general, the literature determines the settings to use based on trial and error experiments and previous experiences of the users in the field. There are few studies in the literature that addresses operator and parameter selection of GAs, which we discuss in the following section.

2.2. Setting Operators and Parameters of Genetic Algorithms

There are several parameters and operators that genetic algorithms have. Population size, selection, crossover, crossover probability, mutation, mutation probability and number of generations are the basic ones [10]. There may be more or less operators and parameters based on the developed GA, and requirements of the problem solved. Selection of operators and parameters may cause the algorithm to terminate earlier than it should be, or to take too long and not to converge to the optimal solution, or to be stuck on local optimums.

Developing a standard approach to find the best settings is a challenging problem. There are few studies in the literature that address this issue. Coy et al. [9] develop an approach using design of experiments and gradient decent to find the effective parameter settings for two vehicle routing heuristics, which are Lagrangean relaxed version of two-opt (LT) and Lagrangean-relaxed sequential smoothing (LS) and compare them with tabu search heuristic (TS) of Xu and Kelly [11] and record-to-record travel heuristic (RTR) of Golden et al. [12]. They implement their approach on 34 route-length-constrained vehicle routing problems using a fractional experimental design with 2^{n-1} runs, where $n = 6$ due to the consideration of six factors. They examine the response surfaces of the designs, where average distance from the optimal solution and the run time are the response variables similar to our consideration. The main differences of our study to that of Coy et al. [9] are the heuristic algorithms, operators and parameters considered. Their heuristic is not a GA and hence the operators and parameters are not similar to the ones that we consider. Moreover, they use design of experiments to investigate effective settings. Grefenstette [13] also employs experimental design to tune the control parameters of GA. He considers six parameters (population size, probability of crossover, probability of mutation, generation gap, scaling window, selection strategy) and measures the performance of experiments with two performance metrics (online performance and offline performance) [14]. The main differences of our study to that of Grefenstette [13] is that we consider different alternatives for the parameters and use different performance metrics. For example, unlike us, they consider generation gap, which is control parameter for the percentage of the population to be replaced in each generation, so it shows how many parents survived into the next generation. Moreover, they do not consider run time as a performance metric and focus on the average difference between the objective functions of the final solutions and the solution with minimum objective function value.

Starkweather et al. [15] investigate the impact of six genetic sequencing operators which are edge recombination, two different order crossovers, partially mapped crossover, cycle crossover, and position-based crossover methods of GA. By changing the settings of only one operator and keeping the remaining operators fixed, they investigate the effects of different settings of the operators on the result. They consider a 30-city TSP problem and a real-world warehouse/shipping scheduling problem and compare operators based on their impact on the tour length and average inventory level. The tour length is for TSP and average inventory is for the scheduling problem. The main difference of our study to that of Starkweather et al. [15] is that they investigate only one operator at a time, and do not consider all operators and parameters simultaneously as we do. Furthermore, performance of a setting is measured based on the approximation quality and run time is not considered.

Parsons and Johnson [16] use experimental design and response surface methodology to tune the parameters of an optimization approach based on genetic algorithms. They consider several operator and parameters, some of which are same as the ones we consider. For example, they also consider crossover and population size. They consider two alternatives (high and low) for each operator and parameters and consider approximation quality as the performance metric. In general, their motivation is similar to that of us. However, we consider different operators and alternatives and more alternatives for each of them. Using a MOEA instead of design of experiments allow us to consider more alternatives for operators and parameters and consider multiple objectives, simultaneously.

Magalhaes-Mendes [17] studies impact of several crossover operators when minimizing some jobshop scheduling problems with GA. They consider four alternatives for the crossover operator, which are single-point, two-point, uniform, and flat crossover. The experiment works according to a decision support system. By providing the same conditions to all operator methods, the decision support system can make comparisons based on the impact of crossover methods on the approximation quality.

To conclude, in the literature, there is no study treating the problem of selecting the operators and parameters of GAs as a multi-objective optimization problem as we do. Some studies investigate the effect of an operator or parameter at a time and ignore their interactions with each other. Some studies employ design of experiments to include these interactions as well. However, less alternatives for operators and parameters are considered to decrease the number of required experiments. Moreover, run time is ignored and the approximation quality is used as the main performance measure. We include several operators and parameters simultaneously and investigate the effect of different settings on the approximation quality and run time of GAs. Using a MOEA allows us to consider more alternatives for each operator and parameter. Moreover, instead of a single setting, we present a final set of settings to decision makers in order to present the trade-offs between approximation quality and run time.

2.3. Multi-Objective Optimization

We search the best settings for operators and parameters of GAs under two criteria (approximation quality and run time). Hence, we handle the problem as a multi-objective optimization problem and employ a MOEA for the solution. Due to the lack of suitable solution approaches, multi-objective optimization problems are often solved as single objective optimization problems. However, multi-objective optimization problems have important distinctions, some of which are as follows [18] [19] :

- Several optimization goals regardless of their type (minimization or maximization) are considered simultaneously.
- A single-objective optimization problem has a decision variable space and an optimum solution. On the other hand, a multi-objective optimization problem has an objective space in addition to the decision variable space. For each solution in the decision space there is a corresponding point in the objective space.
- Instead of a single optimum solution, multi-objective problems have multiple solutions, each of which presents a different trade-off among objectives.

Multi-objective optimization is a broad field with many fundamental definitions and principles. We next introduce some multi-objective optimization terminology that are used throughout the text, and refer readers to Deb [20] for more details.

2.3.1. Definitions

In this section, we present some fundamental concepts of multi-objective optimization [18] [21]. The main issues of multi-objective optimization are efficiency and non-dominance. The concept of dominance is used when solving multi-objective optimization algorithms as it allows solutions to be compared. Dominated solutions are eliminated during optimization process until non-dominated solutions are found.

Let $\boldsymbol{x}$ and $f(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x}))$ denote the decision variable and the corresponding objective function vectors, respectively, $f_i(\boldsymbol{x})$ represents the performance of solution $\boldsymbol{x}$ in objective *i,* and there are *m* objectives. *X* and $\mathbb{R}^m$ are the corresponding feasible sets in the solution and objective spaces. Then a minimization type problem can be formulated as follows:

$$\min_{i=1,\ldots,m} f_i(x) \qquad 1$$

$$subject\ to\ x \in X \qquad 2$$

The dominance relation between two solutions depends on their performances on the objective function. Suppose that $\boldsymbol{x^{(1)}}$ and $\boldsymbol{x^{(2)}}$ are two solutions in *X*. Solution $\boldsymbol{x^{(1)}}$ is said to dominate solution $\boldsymbol{x^{(2)}}$ when the following conditions are met:

1. The solution $\boldsymbol{x^{(1)}}$ does not perform worse than $\boldsymbol{x^{(2)}}$ in all objectives:

$$f_i(\boldsymbol{x}^{(1)}) \leq f_i(\boldsymbol{x}^{(2)}) \text{ for all } i \in \{1,2,\ldots,m\} \qquad 3$$

2. The solution $\boldsymbol{x^{(1)}}$ performs better than $\boldsymbol{x^{(2)}}$ in at least one objective:

$$f_i(\boldsymbol{x}^{(1)}) < f_i(\boldsymbol{x}^{(2)}) \text{ for at least one } i \in \{1,2,\ldots,m\} \qquad 4$$

Considering solution and objective space together, the following definitions adapted from [22] are presented:

1. Solution $\boldsymbol{x} \in X$ is said to be efficient when there does not exist $\boldsymbol{y} \in Y$ such that $f_i(\boldsymbol{y}) \leq f_i(\boldsymbol{x})$ for all objectives and $f_i(\boldsymbol{y}) < f_i(\boldsymbol{x})$ for at least one objective. Solution $\boldsymbol{x}$ is said to be inefficient, when such a $\boldsymbol{y}$ exists.
2. All the efficient solutions of the problem constitute the efficient set (frontier) of the problem.
3. *f(**x**)* is said to be nondominated or Pareto-optimal when $\boldsymbol{x}$ is efficient; and *f(**x**)* is said to be dominated when $\boldsymbol{x}$ is inefficient.

4. The set of all nondominated (Pareto-optimal) points constitute the nondominated (Pareto-optimal) set (frontier).

2.3.2. Multi-objective Evolutionary Algorithm

In general, computational requirements of solving multi-objective optimization problems are high due to their complexity. Solution spaces are often large and there are many local optimal solutions. Unlike single objective optimization problems that have a single optimal solution, there are many Pareto-optimal solutions that need to be discovered in multi-objective optimization problems. Some of the popular methods used in the literature to handle multiple-objectives are weighted sum, ϵ-constraint, weighted metric, Benson's, value function and goal programming methods. While these methods are useful, they also have some disadvantages. For example, they can find only one Pareto-optimal solution at a time. They struggle when finding Pareto-optimal solutions for nonconvex multi-objective optimization problems. Moreover, all these methods require extra efforts such as defining weights for the objectives or finding ϵ value.

As an alternative, MOEA algorithms have been popular solution approaches in the literature. They are population-based approaches mimicking the natural evolution process and able to approximate well-converged and diversified set of Pareto-optimal solutions in a single run. All objectives are simultaneously considered and dominance relation between solutions are used during the search process. These features give a substantial advantage for the use of MOEA since one of the fundamental goals of solving a multi-objective optimization problem is finding as many Pareto-optimal solutions as possible.

Various MOEA algorithms have been developed in the past three decades. In a broad sense, they are classified as non-elitist and elitist MOEAs depending on the usage of an elite-preserving operator that explicitly favours better solutions during generations. We direct those who are interested in the fundamental concepts and the literature of MOEA to, for example, Van Veldhuizen and Lamont [23], K. Deb [20], Glosh and Dehuri [24], Coello et al. [25]. Additionally, Coello et al. [26] presents a large collection of applications from various disciplines.

2.4. Travelling Salesperson Problem

TSP is a well-known combinatorial optimization problem of finding the shortest tour that visits all destinations (cities, nodes etc.) and returns to the initial point. TSP is known to be NP-hard [27].

Let a TSP be defined on a complete graph $G = (N, A)$ with the node set N and the arc set $A = \{(i,j), i \in N. j \in N, i \neq j\}$. Let each $(i,j) \in A$ has a scalar attribute d_{ij} indicating the cost of traveling from i to j. Introducing a binary variable x_{ij} for each $(i,j) \in A$ indicating that whether (i,j) is used or not, a TSP problem can be formulated as follows:

$$\text{Minimize} \quad \sum_{i\in N} \sum_{\substack{j\in N \\ i\neq j}} d_{ij} x_{ij} \qquad 5$$

$$\text{Subject to} \quad \sum_{\substack{j\in N \\ j\neq i}} x_{ij} = 1 \qquad \forall\, i \in N \qquad 6$$

$$\sum_{\substack{i\in N \\ i\neq j}} x_{ij} = 1 \qquad \forall\, j \in N \qquad 7$$

$$subtour\ elimination\ constraints \qquad 8$$

$$x_{ij} = 0\ or\ 1 \qquad \forall\, (i.j) \in A \qquad 9$$

Objective function (5) minimizes the summation of the travel costs of used arcs, constraint (6) and (7) are assignment constraints of nodes to ensure every node is visited once in a route, constraint (8) ensures that only one tour is allowed in a feasible solution and all other subtours are eliminated. There are subtour elimination constraints developed in the literature, and two popular ones are Miller-Tucker-Zemlin [28] and Dantzig-Fulkerson- Johnson [29] formulations.

In general, there are two types of TSPs, (a) symmetrical TSP (STSP) and (b) asymmetrical TSP (ASTP). STSP assumes that the distance between nodes i and j is the same from each direction ($d_{ij} = d_{ji}$). On the other hand, $d_{ij} \neq d_{ji}$ in ATSP.

TSP has various applications. For example, vehicle routing [30], time scheduling [31], integrated circuit design [32], physical mapping [33] and construction of optimal evolutionary trees [34] problems are formulated as a TSP in the literature. Many solution procedures are developed, and heuristic approaches are popular due to the complexity of the problem. GA is one of the popular heuristics used when solving TSPs. Although GA has been an effective method for TSP, the settings of its operators and parameters has been left to the user. In general, the literature either use settings proposed in other studies or determine their settings based on computational experiments. Braun [35] develops a GA model to solve TSP. The model can solve TSP with up to 229 nodes with less than 3 minutes and can solve the problems optimal with up to 442 nodes. Braun uses fixed settings for the parameters and operators and gives some personal suggestions about them. Moon et al. [36] study TSP with precedence constraints (TSPPC) and developed a new crossover operator for a GA to improve solution process. Their model gives better results than the traditional algorithms. They use other operator and parameter values from the settings constant which they think perform well. Schmitt and Amini [37] employs statistical experimental design to understand the effects of TSP problem class, TSP problem size, population size, population initialization, population evolution, population replacement and crossover operator on GAs use to solve TSP. They compare the impacts of the factors with separate observation strategies based on computational results (quality) and CPU time, but they do not use the performance measurements with multi objective search, and they analyse them separately. Tsai et al. [38] analyse the performances of some genetic operators by experimenting on 15 TSP instances to improve their efficiency and diversity. A new GA that can combine two genetic operators and a heterogeneous pairing selection to examine the factors is developed. They compare the operators based on the relative error. The GA settings when comparing different operator methods using genetic algorithm are arbitrarily determined. Takahashi [39] develops an approach for the crossover operator. The algorithm continues to work with another operator at any time when the first crossover operator of GA reaches fitness limit while working for the TSP. In this way, Takahashi makes experiments to show that the algorithm can reach a better maximum fitness value than it can achieve with only the first operator. In the study, Takahashi choses other operators and parameters from the ones suitable for the method and used them constantly. As a result, Takahashi is comparing two crossovers and shows that the experiment combined with the two gives the best fitness value and make arbitrary decisions on other GA settings.

3. METHODOLOGY AND ITS IMPLEMENTATION TO TSP

In this section, we first explain the methodology we use when searching best settings for the operators and parameters of GAs. Then we explain the details of the implementation of our methodology on TSP instances.

3.1. Searching Effective Settings for GA Using MOEA

We search effective settings for operators and parameters of GAs using a MOEA. MOEAs are population-based algorithms, where an initial population of solutions evolves towards better solutions through generations until termination. All solutions have same chromosome structure and solutions are evaluated according to their performances on the objective functions of the problem solved.

In our methodology, we have two objectives when searching the best settings for a GA: (1) maximizing the approximation quality, and (2) minimizing the approximation time. These are two substantial performance measures when evaluating the performance of a heuristic and there are trade-offs between them. Typically, improving approximation quality increases approximation time as well. On the other hand, shorter approximation times require sacrificing from approximation quality. Settings defined for operators and parameters of a GA often have a significant effect on the performance of the GA on these two objectives. We search the best settings for GAs considering both objectives simultaneously. Consideration of the objectives simultaneously results in a set of Pareto-optimal solutions, where each of which corresponds to an efficient setting for the GA considered. All efficient settings can be presented to decision maker (user of the algorithm) for his/her final decision. Any efficient setting is better than others in terms of one of the objectives. Depending on the requirements of the problem or the preferences of decision maker, final settings can be chosen accordingly. For example, one may prefer a setting having higher approximation quality, on the other hand, another may prefer a setting having lower approximation time.

In our methodology, we employ NSGA-II, a well-known MOEA, developed by Deb [19] to search the efficient settings of a GA used for an optimization problem. We let each solution in NSGA-II to represent a setting for the GA used. To evaluate a solution (setting) in a population of NSGA-II, the optimization problem is solved using the GA with the operators and parameter values defined in the chromosome of the solution. Then we let approximation quality and run time of the GA to form the fitness value of the solution. MOEA evolves through generations and at the end a set of nondominated solutions, each of which is a efficient setting, is returned.

3.2. NSGA-II

In this study, we employ nondominated sorting genetic algorithm II (NSGA-II) developed by Deb et al. [19] when searching efficient setting for a GA.

Let P_t , M_t , Q_t , and R_t , represent the parent, mating pool, offspring, and global populations at generation t, respectively. Steps of the NSGA-II are as follows:

Step 0. Generate an initial population P_0 of size *N* randomly. Each solution of the population corresponds to a setting of operators and parameters of the GA.

Step 1. Set generation counter *(t=0)*

Step 2. Apply nondominated and crowding-distance sorting to P_t.

Step 3. Apply tournament selection to P_t and let winners to constitute M_t of size *N*.

Step 4. Apply crossover and mutation to M_t, and let new solutions to constitute O_t of size *N*.

Step 5. Combine parent and offspring populations, $R_t = P_t \cup Q_t$.

Step 6. Form new population $P_{t+1}$of size *N* from R_t:

Step 6.1. Apply nondominated sorting to R_0 and let all solutions having rank *i* to form a front $F_i, i = 1,2,3$ ..., where F_1 is made of nondominated solutions found so far and smaller *i* indicates better ranks.

Step 6.2. Let $P_{t+1} = \emptyset$ and *i* =1.

Step 6.3.

While { $|P_{t+1}| + |F_i| \leq N$,

$P_{t+1} \leftarrow P_{t+1} \cup F_i$,

i = i + 1}

Step 6.4. Apply crowding-distance ranking F_i, and sort solutions of F_i according to their preference ranks in ascending order. Place the first $N - |P_{t+1}|$ solutions in P_{t+1}, $(P_{t+1} \leftarrow P_{t+1} \cup F_i[1{:}(N - |P_{t+1}|]\,)$

Step 7. Set $t = t + 1$. If $t = T$, terminate. Otherwise. Return to Step 2.

Steps 1-7 are illustrated in Figure 1. In each generation, genetic operations (selection, crossover, and mutation) are implemented to parent population in order to generate an offspring population. For the selection, tournament selection is used. Two solutions are selected from the parent population and the one with better nondominated and crowding-distance ranks wins the tournament. Each solution attends two tournaments to form an offspring population of size *N*. Then in Step 5, parent population and offspring population are combined for forming a global population of size *2N*. Finally, in Step 6, the parent population of next generation of size *N* is generated selecting the half of the solutions of the combined population. This selection is again based on the nondominated and crowding distance ranks of the solutions as explained in Step 6.1 – 6.4 and shown in Figure 1.

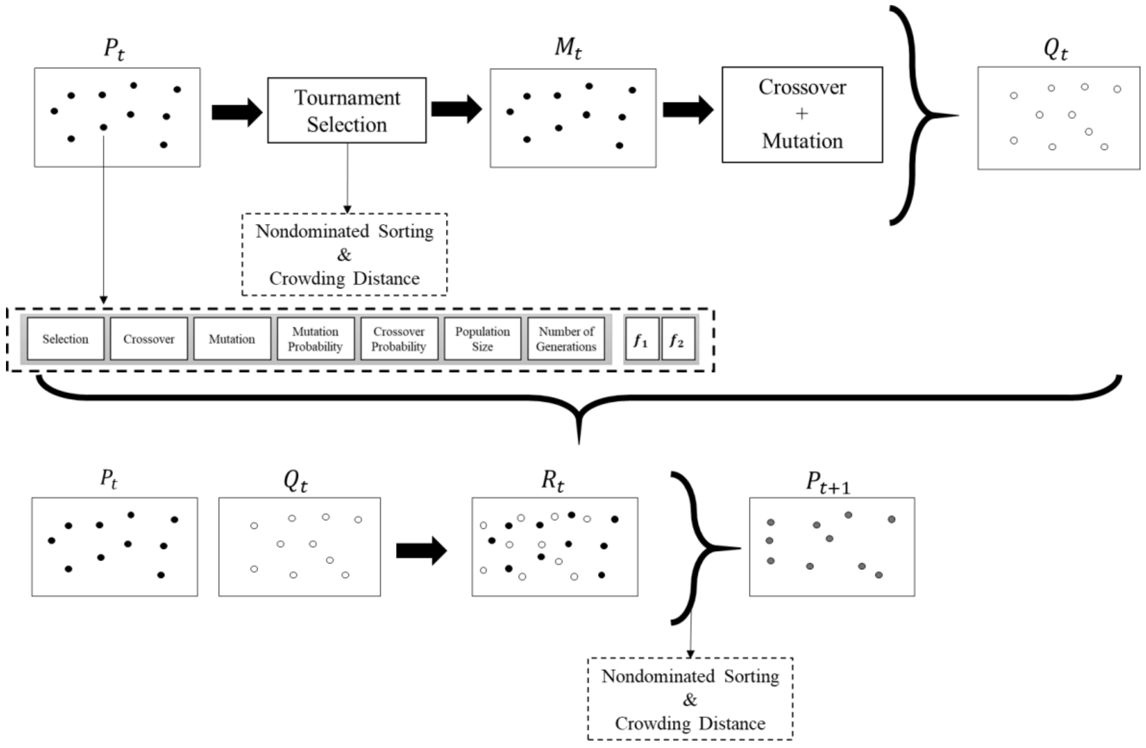

Figure 1 An illustration of the procedures applied in one generation of NSGA-II

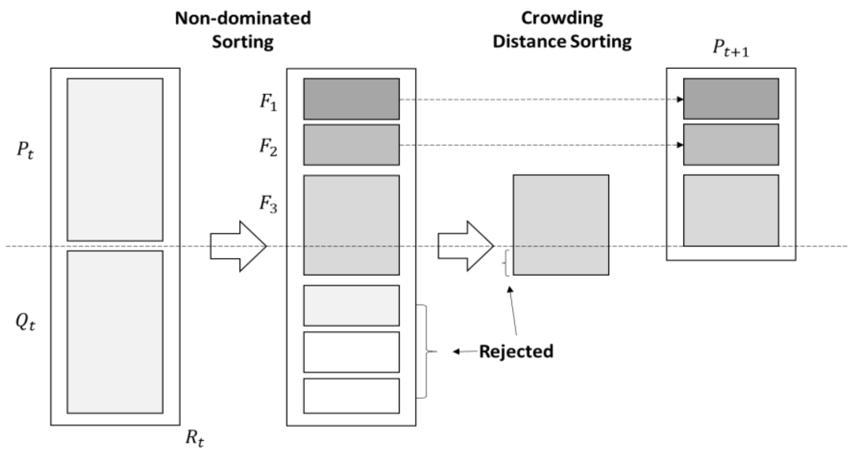

Figure 2 NSGA-II procedure (adapted from Deb et al. [19].)

3.3. Encoding in NSGA-II

Each population member in NSGA-II is a solution representing a setting for parameters and operators of the GA. Chromosome of a solution involves genes, each of which carry the information of the setting of an operator or parameter. Solutions are evaluated based on their objective function values f_1(approximation quality) and f_2 (approximation time) through nondominated and crowding distance sorting. Objective function values of a solution are found solving the GA using the settings in the genes of the solution.

In a chromosome structure, each gene corresponds to a parameter or operator, and values of genes indicate the setting for the corresponding parameter or operator. In this study, we consider following parameters and operators for the GA: selection method, crossover method, mutation method, population size, crossover probability, mutation probability and number of generations. Hence, there are seven genes in a chromosome as shown in in Figure 3. The operators and parameters that are considered for the GA are explained in the following.

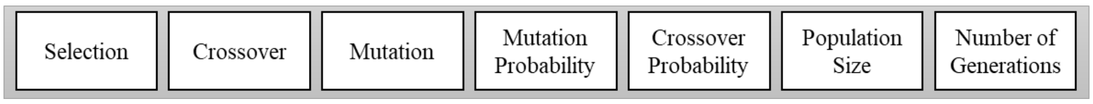

Figure 3 Chromosome structure of the solutions of NSGA-II

3.4. Operators and Parameters Considered

In this section, we explain the operators and parameters of GAs we consider in this study. These operators and parameters are selected since they are available in R's GA package [40]. For more details about the operators and parameters GA package, we refer readers to Bäck et al. [41], Yu and Gen [42] and Eiben and Smith [43], and some additional references are Holland [1], Mitchell [2], Larranaga et al. [44], Razali and Geraghty [45], Bäck and Hoffmeister [46], Whitley [47] and Hussain [48], Umbarkar and Sheth [49].

3.4.1. Selection Operator

Selection operator determines the approach used when selecting solutions from the population to match. This operator does not create new individuals, instead selects relatively good ones for a match. In general, selection is based on fitness values of solutions. It aims to give higher survival chance to the solutions with better fitness values by increasing the probability of matching individuals with good chromosomes. There are many selection methods developed in the literature. For instance, tournament selection, ranking selection, and roulette wheel selection methods [41] [42] [43]. The selection operator used often depends on the problem type. However, users often ignore the impact of using different selection operators on the performance of the GA they use.

3.4.2. Crossover Operator

Crossover operator produces new offspring/solutions. Using the solutions selected as parents, a new solution called as offspring is produced. The idea is based on the principle of gene exchange among individuals in nature. In a typical crossover operation, genes of the parents transferred to each other aiming to produce offspring that are better than the parents. There are many crossover methods developed in the literature. Some examples are partially mapped crossover, position based crossover, edge recombination crossover, alternating edge crossover, and sorted match crossover methods [41] [42] [43] [44].

3.4.3. Mutation Operator

While crossover operator is used to produce offspring from selected parents, mutation operator produces an offspring by changing the genes of the chromosome of a solution randomly. Typically, mutation operators are used to increase the diversity of new individuals. Although crossover operator provides diversity as well, mutation operator lets search process to jump to new search areas and avoid converging to local optimums. Despite the advantages of mutation, an imprecise selection for its setting it may slow down the convergence rate of the algorithm and increase computational times. There are many crossover methods developed in the literature. Some examples are displacement mutation, exchange mutation, insertion mutation, inversion mutation methods [41] [42] [43] [44].

3.4.4. Crossover and Mutation Probabilities

The crossover and mutation operations are implemented depending on the probability of crossover (*pc*) and the probability of mutation (*pm*), respectively. They are the parameters that show the probability of implementing the operations and are defined by the user of the algorithm at the beginning. In a typical GA, random values between 0 and 1 are generated using a uniform distribution for crossover and mutation operations. Whether the crossover and mutation operations will be applied is decided by these random values. When the generated value is less than or equal to the *pc*, crossover operator is applied. Similarly, when the generated value is less than or equal to the *pm*, mutation is applied. Setting *pc* and *pm* often have significant effects on the performance of a GA. Although larger selections may increase diversity and help avoiding local optimums, they may reduce the convergence power of the algorithm and hence increase the run time. Hence, a careful selection needs to be done.

3.4.5. Population Size

Population size refers to the number of individuals in the population of the GA considered. In general, a fixed size population is created at the beginning and the population size is maintained during generations. A population involves a set of alternative solutions for the problem. While weak individuals are eliminated from generation to generation, good solutions will survive and transfer their genes to future generations.

Since the population is a set with solutions, its size will affect the number of alternative solutions we have. GA will be able to search and produce new solutions based on the population size. Population size need to be set carefully since larger selections may increase run time while smaller selections may decrease approximation quality.

3.4.6. Number of Generations

Number of generations is a typical termination criterion used in GAs. It indicates how many generations an initial population will evolve until termination. This parameter needs to be carefully set as well. If it is not large enough, a GA can terminate before converging to the optimal solution. If it is too large, run time may take much longer than it is required.

3.5. Computing Fitness Value of a Solution

In this study, a genetic algorithm is evaluated based on two objectives: approximation quality and run time. Hence, a solution in the algorithm has two values, approximation quality (f_1) and approximation time (f_2), which form the fitness value of the solution. In MOEA, each solution represents a setting combination for the parameters and operators of the GA used. To find f_1 and f_2 of a solution, the GA used need to be run using the settings defined in the solution.

Approximation quality (f_1) measures the closeness of the approximation of the GA used to the optimal solution of the problem studied. We use optimality gap in Equation 10 to measure approximation quality of a solution x. This calculation is based on the relative optimality gap calculations of the famous commercial solvers CPLEX [50] and GUROBI [51]. In the following, $f_1(x)$ and $f_1(x^*)$ are the objective function values of solution x and the optimal solution x^*, respectively.

$$G(x) = \frac{|f_1(x^*) - f_1(x)\,|}{f_1(x)} \qquad 10$$

Run time (f_2) measures the wall-clock time spent by the GA until termination. In general, there are trade-offs among f_1 and f_2. Higher approximation qualities require higher run times as well. When MOEA terminates, it presents a set of final settings to decision maker. Each solution in this final set presents an advantage in terms of either f_1 or f_2. The final selection can be made by the used depending on his/her preferences.

3.6. Computational Settings of NSGA-II

NSGA-II is a MOEA and uses several operators and parameters. Similar to the problem we study in thesis, effective settings for its operators and parameters may be searched in a future work. However, in this study, we focus on developing a methodology to determine effective settings for GAs. We define the problem as a bi-objective optimization problem and employ NSGA-II to generate the Pareto-optimal frontier of the problem. We set the parameters and operators of NSGA-II based on Deb [19]. We use 100 as the population size of NSGA-II and limit the maximum number of generations to 200.

We use tournament-selection as the selection operator. Two solutions from the population are selected, tournaments are played between them, and the solution with better nondominated rank wins. If both solutions have the same nondominated rank, the solution with smaller crowding distance rank wins. Winners of the tournaments are aggregated in a mating pool. Each solution participates in the tournament twice.

As crossover operator, we use position-based crossover (POS) (see [52]). Two parent solutions are randomly selected from the mating pool. Crossover probability determines whether crossover operation will be implemented to the selected solutions or not. When crossover is implemented, we select three genes from each parent randomly. In the study, the crossover probability was chosen as 0.9. The values at the selected genes in *parent 1* are transferred directly to *offspring 2*, and the values at the selected genes in *parent 2* are transferred directly to *offspring 1*. The remaining positions in *offspring 1* and *2* are filled from *parents 1* and *2*, respectively, without changing the order of the genes in the parents. An example crossover operation is shown in Figure 4.

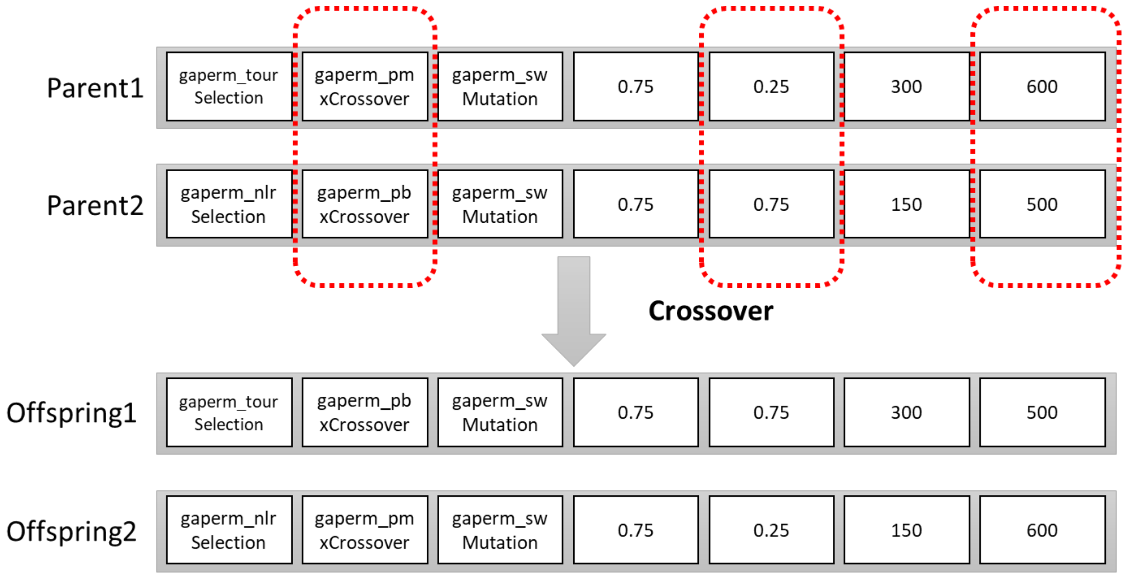

Figure 4 Crossover operator used in NSGA-II

For the mutation, we employ a greedy mutation operation. When mutation is implemented to a gene, the value of the gene is changed with another value from the available alternatives. Each gene is subject to a mutation probability such that the total probability of the chromosome is 0.2. In our chromosome structure, there are seven genes in total. Hence, mutation probability of each gene (P_{mut}) is selected such that $1 - (1 - P_{mut})^7 = 0.2$.

3.7. Demonstration on TSP

We select TSP as an example optimization problem to demonstrate our methodology. TSP is one of the well-known NP-hard problems. Heuristic approaches involving GA are employed in the literature to solve TSPs. Here, we experiment 31 TSP instances that are selected from the literature. Details of these instance are reported in Table 17 in Appendix 1. In Table 17, “Problem Size” shows which group the problem size belongs to, column “Name” presents the abbreviations of the instances, column “Number of Nodes” shows the number of nodes problem instances have and column “Data Type” reports the approach used to obtain the distances between node pairs. Column “$f^*(x)$” presents the optimal objective function values, and the column “References” reports the sources of the problem data.

There two types of datasets used in the literature for TSP instances. In MATRIX type datasets, distance information between node pairs in a matrix format is already available at the source. In EUC_2D type datasets, only the 2-dimensinonal (x, y) coordinates of the nodes are reported at the source. For those datasets, using these coordinates, we calculate the distances between node pairs by the Euclidean distance formula, as shown in Equation 11, where *i* and *j* are two nodes and x_i and y_i are the coordinates of node i.

$$d_{ij} = \sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2} \tag{11}$$

All problem instances we experimented have 200 or fewer nodes, and they can be classified according to the number of nodes they have:

- **Small-Sized Problems:** TSP instances having less than or equal to 50 nodes. Seven of 31 instances are in this class.
- **Medium-Sized Problems:** TSP instances with greater than 50 and less than or equal to 100 nodes. 10 of 31 instances are in this class.
- **Large-Sized Problems:** TSP instances having more than 100 and less than or equal to 200 nodes. 14 of 31 instances fall in this class.

We search the best settings of operators and parameters of a GA experimenting with these problem instances. The GA we use is the one available in the GA package of R software. For the operators and parameters, we use the alternative settings that we report in next sections. Among these alternatives, the best settings in terms of approximation quality and run time are searched.

3.7.1. Encoding of Solutions When Solving TSP

The permutation encoding is generally used for ordering problems like TSP [42]. A chromosome consists of genes representing nodes to be visited. Each chromosome illustrates a complete tour. Order of nodes in the chromosome shows the order of visits to the nodes. For example, the value of the second gene (which is 11) in Figure 5 indicates that the second visited node is the eleventh node.

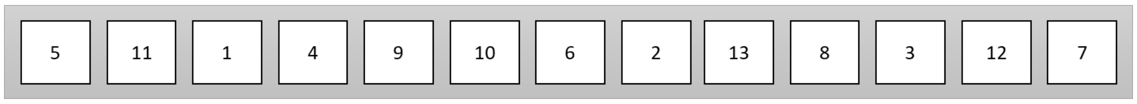

Figure 5 An example chromosome structure for a TSP instance having 13 nodes

3.7.2. Operators and Parameters of Application

As explained in the previous chapter, we consider three operators and four parameters when searching the best settings of a GA. The operators are selection, crossover, and mutation operators, and the parameters are crossover probability, mutation probability, population size and number of generations. For each operator and parameter, different alternatives are considered. The alternatives are reported in Table 1 and their details are explained in the subsequent sections. The labels used for the alternative operators reported in first three columns of Table 1 are the labels used in the GA package of R.

Each combination of the alternatives is a candidate setting and theoretically there are 414,720 combinations in total. Using our methodology, the best set of combinations of the alternatives in terms of approximation quality and time are searched.

Table 1 Selected alternatives for operator and parameter settings

Index	Selection Operator	Crossover Operator	Mutation Operator	Crossover Probability	Mutation Probability	Population Size	Number of generations
1	gaperm_lrSelection	gaperm_cxCrossover	gaperm_simMutation	0.1	0.1	50	50
2	gaperm_nlrSelection	gaperm_pmxCrossover	gaperm_ismMutation	0.25	0.25	100	100
3	gaperm_rwSelection	gaperm_oxCrossover	gaperm_swMutation	0.5	0.5	150	150
4	gaperm_tourSelection	gaperm_pbxCrossover	gaperm_dmMutation	0.75	0.75	200	200
5			gaperm_scrMutation	0.9	0.9	300	300
6				1	1	400	400
7						500	500
8						600	600
9						700	700
10						800	800
11						900	900
12						1000	1000

3.7.3. Alternatives Considered for the Selection Operator

We use linear-rank (gaperm_lrSelection), nonlinear-rank (gaperm_nlrSelection), roulette wheel (gaperm_rwSelection) and tournament selection (gaperm_tourSelection) operators as the alternative selection operators. These are the methods that can be used for permutation type encoding in R’s GA package [53].

Linear rank and nonlinear rank selection use rank-based fitness assignments. In rank-based fitness assignment methods, the population members are sorted based on their fitness values, and then their selection probabilities are computed based on their fitness values.

Linear rank selection operator assigns selection probability of a solution based on its rank in the population. Probabilities of solutions are set in linear proportion to the ranks of the solutions. In the linear rank selection, the rank of the least fit individual is equal to 0, and the rank of the fittest individual is equal to [*The number of individuals in the population - 1*]. On the other hand, nonlinear rank selection operator assigns selection probabilities based on each individual’s rank but using a nonlinear function. The probabilities are not linearly related with the ranks. For example, the selection probabilities can be in proportion to the square of rank, or based on a geometric distribution or exponential distribution [41] [42] [43].

Roulette wheel selection is a popular selection operator in the literature due to its ease of use. In this method, individuals are selected based on their selection probabilities. Selection probability of solutions are computed using objective function values of solutions. Let $P_s(x)$ and $f_1(x)$ represent selection probability and objective function value of solution x, and *ps* be population size, then $P_s(x)$ is calculated as in Equation 12, [41] [42] [43].

$$P_s(x) = \frac{f_1(x)}{\sum_{x=1}^{ps} f_1(x)} \qquad 12$$

Solutions are ordered descending according to their selection probabilities, probabilities of individuals are added cumulatively and a random number is generated to select individuals. This formula is for the maximization type problems since solutions having larger objective function values have higher selection probabilities. For minimization type problems, $1 - P_s(x)$ can be used as the selection probability.

Tournament selection is the last selection operator alternative considered in this study. In this method, some individuals are selected randomly from the population. The selected individuals are compared to each other and the individual with the highest fitness value wins the competition [41] [42] [43].

3.7.4. Alternatives Considered for the Crossover Operator

When chromosomes are structured using a permutation scheme, duplicate values in the chromosomes of offspring need to be avoided, which add an extra difficulty for crossover methods. The most common crossover methods used for TSP are cycle crossover (gaperm_cxCrossover), partially matched crossover (gaperm_pmxCrossover), order crossover (gaperm_oxCrossover), position-based crossover (gaperm_pbxCrossover). We use these four crossover operators. In the following paragraphs, we explain the details of the crossover operators, providing an example for each. Please note that all crossover operators produce two offspring but we demonstrate an example for the generation of only one offspring for some operators as the same procedure is followed for each offspring.

Cycle crossover, creates offspring according to the positions of the genes of parents [41] [42] [43] [44]. Based on the positions of the two parents' genes, a cycle is created between them. Then the genes of the cycle are selected from a parent and transferred to an offspring. Suppose P^1 and P^2 are the parents selected for crossover, and O^1 and O^2 are two resulting offspring. We demonstrate an example cycle crossover operation in Figure 6.

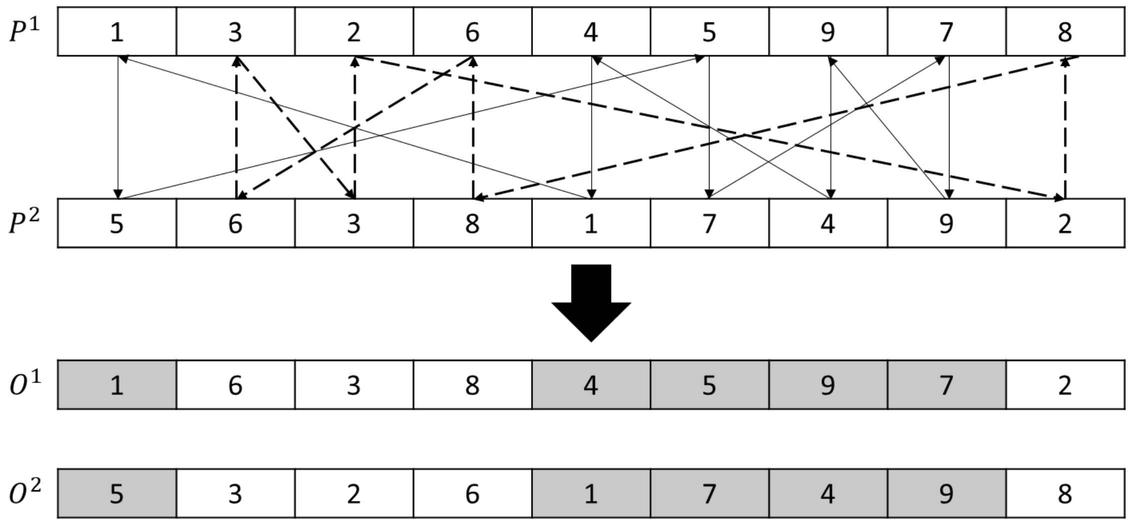

Figure 6 An example cycle crossover operation

To make a cycle starting from P^1, the following procedure is followed. The cycle starts with the value of the first gene of P^1 (which is 1) and then includes the value of the same gene (gene 1) of P^2 (which is 5). Then the cycle goes to the gene of P^1 whose value is 5 (which is gene six) and adds the value of the same gene of P^2 (which is 7, since gene six has 7). Then the cycle goes to the gene of P^1 that has 7 (which is gene 8) and adds the value of corresponding gene in P^2 (which is 9) from P^2. This procedure is repeated until the cycle reaches to the first gene. In the example, {1-5-7-9-4} is obtained when the procedure starts from the first gene of P^1. Then the genes of P^1 having 1, 5, 7, 9, 4 are maintained when forming O^1, and the remaining genes are filled from P^2 in the same order. Similarly, the genes of P^2 having 1, 5, 7, 9, 4 are maintained when forming O^2, and the remaining genes are filled from P^1 in the same order. {6-3-2-8} is obtained when the procedure starts from the first unselected gene from the P^2 and a new cycle created following the same procedure use when generating the first cycle.

Partially matched (mapped) crossover separates parents from two randomly selected cut points creating three subtours [41] [42] [43] [44]. The crossover operator transfers one of the subtours to the offspring and modifies the remaining subtours according to the partial maps. An example is illustrated in Figure 7. The middle subtour of the P^1 is transferred to the offspring O^1 directly in Step 1. The genes of the middle subtour of P^2 that are different than the genes of the middle subtour of $P^1$are transferred to the O^1 using the partial map procedure. In the examples, 8 and 1 are not included in the genes of the middle subtour of P^1. The genes in O^1 which 8 and 1 will be placed determined as shown Step 2 of Figure 7. In Step 3, the remaining genes of O^1 is filled from the genes of P^2 that are not transferred in Step 2 in the same order. To create O^2, the same procedure starts with the P^2.

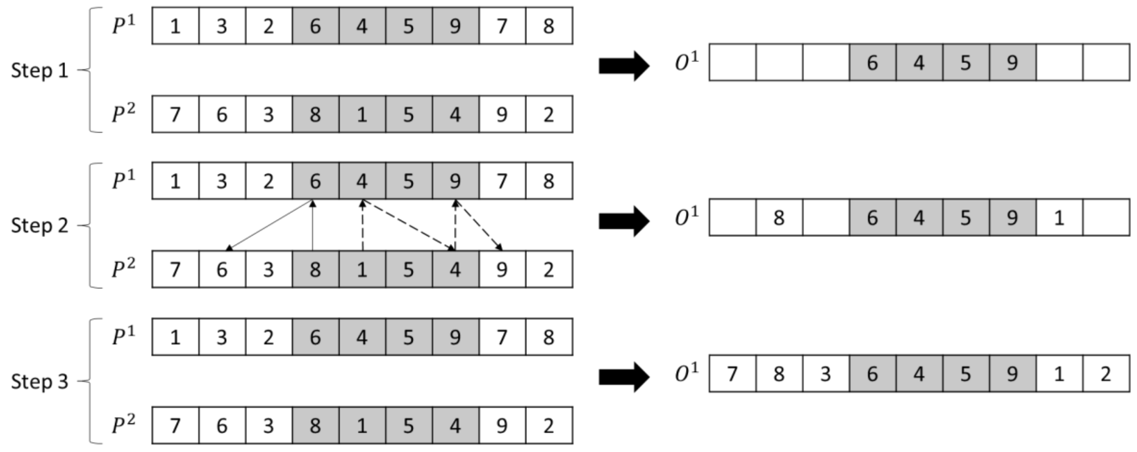

Figure 7 An example partially matched (mapped) crossover operation

In the order crossover, a subtour is randomly chosen and maintained when creating new offspring [41] [42] [43] [44]. An example is demonstrated in Figure 8. The values at the randomly selected genes (2, 6, 4, 5) of P^1 are transferred to the O^1 in the same order. The remaining positions in O^1 are filled from P^2 in the same order starting from the genes after the selected subtour. To generate O^2, same procedure is repeated starting from P^2.

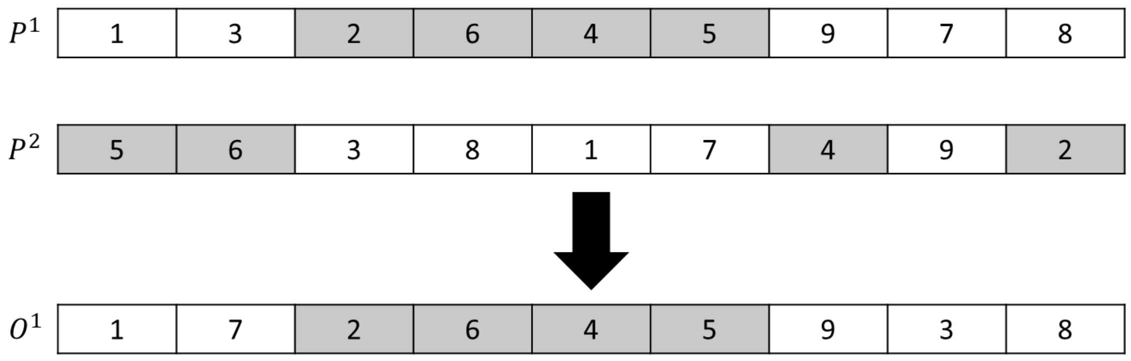

Figure 8 An example order crossover operation

Position-based crossover is based on changing randomly selected positions individually [41] [42] [43] [44]. An example is illustrated in Figure 9. Some genes are randomly selected from P^1 and the values at these genes are transferred directly to O^1 maintaining the order of the values. The remaining positions in O^1 filled from P^2 in the same order. To generate O^2, same procedure is repeated starting the procedure from P^2.

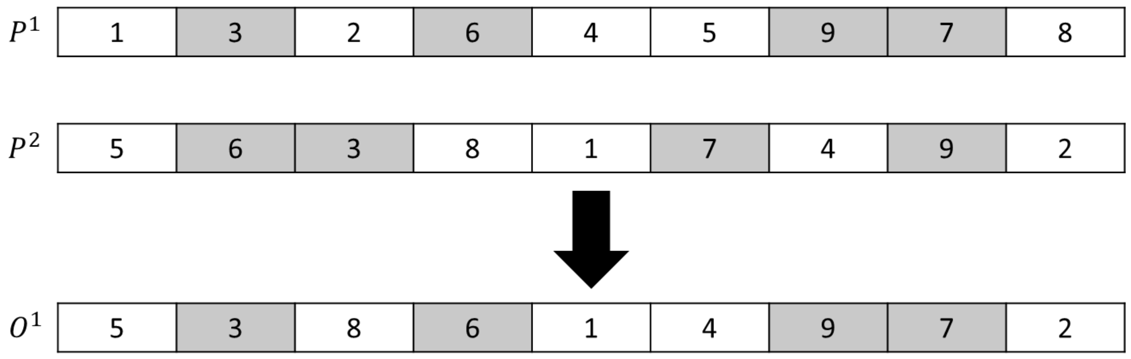

Figure 9 An example position-based crossover operation

3.7.5. Alternatives Considered for the Mutation Operator

In our study, we used five alternative mutation operators that are compatible with chromosomes having permutation scheme. These methods are simple inversion mutation (gaperm_simMutation), insertion mutation (gaperm_ismMutation), exchange mutation (swap mutation; gaperm_swMutation), displacement mutation (gaperm_dmMutation), and scramble mutation (gaperm_scrMutation).

Simple inversion mutation selects a subtour in the chromosome randomly, and reverses the order of the genes in this subtour [41] [42] [43] [44]. An example is illustrated in Figure 10.

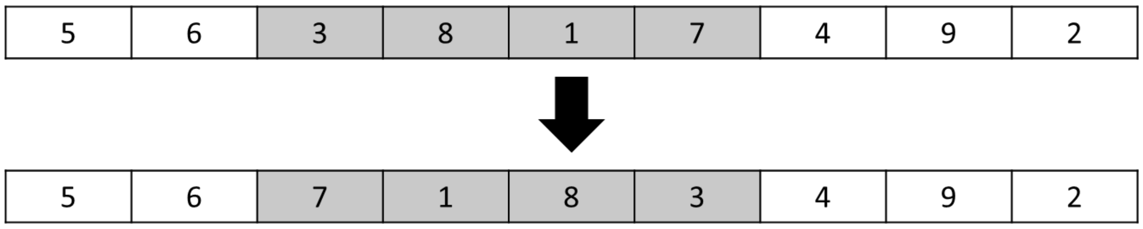

Figure 10 An example simple inversion mutation operation

In insertion mutation, a gene is selected randomly, and inserted after another randomly selected gene [41] [42] [43] [44]. An example is demonstrated in Figure 11.

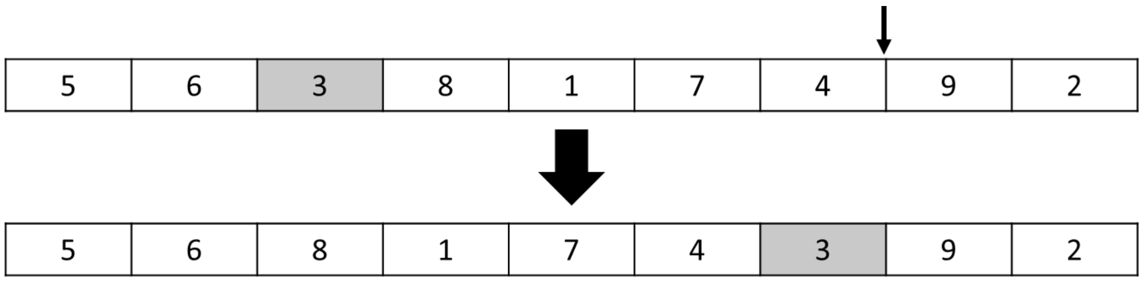

Figure 11 An example insertion mutation operation

Exchange mutation (swap mutation) randomly selects two genes and switches the values at the selected genes places [41] [42] [43] [44]. An example is demonstrated in in Figure 12.

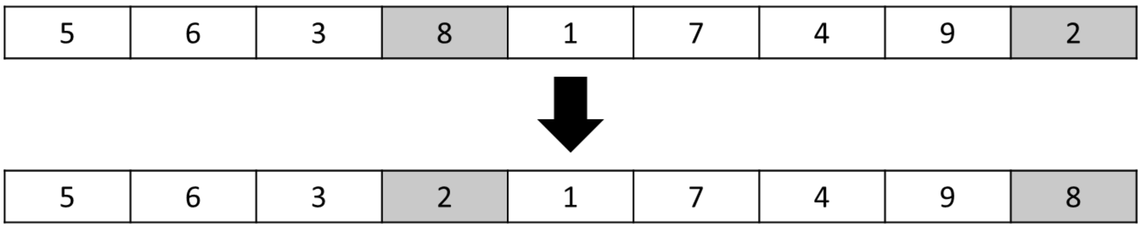

Figure 12 An example exchange mutation operation

Displacement mutation selects a subtour randomly and moves the subtour after another randomly selected gene [41] [42] [43] [44]. An example is demonstrated in Figure 13.

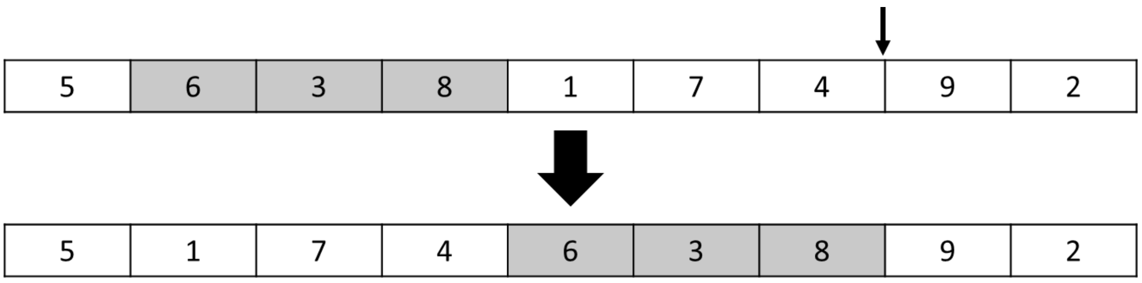

Figure 13 An example displacement mutation operation

Scramble mutation randomly selects a subtour of genes and scrambles the values at the selected genes [41] [42] [43] [44]. An example is demonstrated in Figure 14.

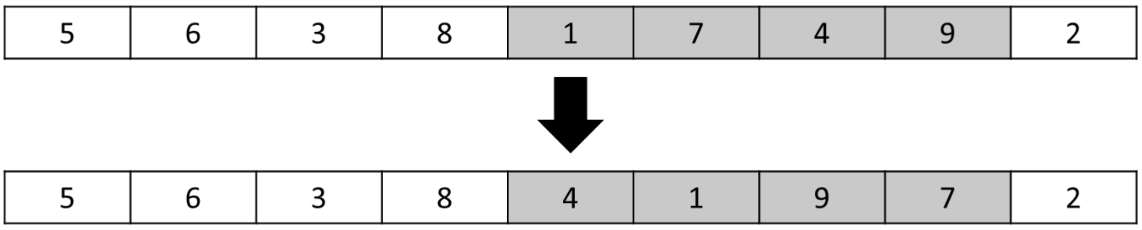

Figure 14 An example scramble mutation operation

3.7.6. Alternatives Considered for the Parameters: Crossover and Mutation Probability, Population Size and Number of Generations

In the following, we present the alternatives considered for the parameters of the GA when experimenting with TSP.

- **Crossover and Mutation Probabilities:** Six levels of probabilities (0.1, 0.25, 0.5, 0.75, 0.9, 1) are considered as the alternatives for the crossover and mutation probabilities.
- **Population Size and Number of Generations:** 12 levels are considered (50, 100, 150, 200, 300, 400, 500, 600, 700, 800, 900, 1000) as the alternatives for the population size and number of generations

Consideration of larger number of alternatives causes more combinations to be evaluated when finding the best parameter settings. Therefore, we select representative sets of alternatives for the parameters of the GA.

3.7.7. Computing Fitness Value of Solutions When Solving TSP

Remember that each solution in NSGA-II represent a setting for operators and parameters of the GA. When the GA is used to solve TSP under these settings, we calculate the values of two criteria (approximation quality and run time). The difference between GA's approximation and optimal solution is measured by optimality gap and we use optimality gap to measure approximation quality of a solution. We use wall-clock time spent until GA terminates as the run time.

To find optimality gap, we use Equation 10 presented before. In the equation, to calculate the gap, we need objective function values of the optimal solution of the problem solved and the solution found by the GA. For all 31 TSP instances, the optimal solutions are available at the sources given in Table 17. Objective function value of a solution is equal to the length of the tour found by the GA when the settings coded in the chromosome of the solution are used.

4. RESULTS AND DISCUSSION

We programmed our algorithm using R 3.5.3 programming language (R Core Team 2020 [54]) on a Dell 32-core computer running Linux Centos 7.5.x with a processor Intel Xeon Gold 6130 CPU, @32 x 2.10GHz and 192 GB usable RAM (CCR 2020 [55]). For genetic algorithms, we used the “GA” package developed for R [53]. For NSGA-II, we used “nsga2R” [56] and “mco” packages developed for R [57]. To compute wall-clock times, tic() and toc() functions are used from “tictoc” package developed for R [58]. We set the parameters and operators of NSGA-II based on Deb [19].

The “ga” function of the “GA” package [53] is used to solve single objective optimization problems using the genetic algorithm available at the package. To solve TSP problems, “*type”* argument of the “ga” function need to be set as “permutation”. Alternative operators that we used for the selection, crossover and mutation operators are available in the “GA” package. The names of the operators in the package are as given in Table 1.

4.1. Finding the Pareto-Optimal Sets of the TSP Instances

We approximate the Pareto-optimal set of settings of the GA for each of the 31 TSP instances. Each solution in a Pareto-optimal set corresponds to an efficient setting for the operators and parameters of the GA, and in the objective space each Pareto-optimal setting presents a trade-off between optimality gap (f_1) and run time (f_2) of the GA. When the efficient settings for 31 problem instances are gathered together, there are 2021 settings. However, an efficient setting found when experimenting with a problem instance may not be efficient for another problem and the number of efficient settings found for different problem instances may differ.

As explained before, we classify the problem instances into three categories (small-sized, medium-sized, and large-sized) based on the number of nodes they consist of. We experiment with each problem instance but due to the similarity of the results, in the next sections, we report the detailed results only for arbitrarily selected problem instances from each size class. Here, we provide some statistical results for each size class to report which operators and parameters are popular.

Table 2 reports how many times (in terms of percentage) each alternative of the operators and parameters appears in the combined set of the final sets of the seven smalls-sized problem instances belonging to the small-sized class (the combined set has 125 settings). For the selection and mutation operators, and the probability of crossover and the population size parameters, almost all settings use the same alternatives. For the crossover operator and number of generations parameter, fourth and first alternatives are popular, respectively. Here, we observe that many alternatives are never used and such alternatives can be classified as dominated alternatives for the small-sized problem class.

Table 2 Percentage of times operators and parameters appear in the 125 settings of small-sized problem class

Index	*s*	*c*	*m*	*pc*	*pm*	*ps*	*ng*
1	0%	0.80%	100.00%	98.40%	37.60%	100%	76.00%
2	99.20%	15.20%	0%	1.60%	36.80%	0%	13.60%
3	0.00%	27.20%	0%	0%	21.60%	0%	8.00%
4	0.80%	56.80%	0%	0%	3.20%	0%	2.40%
5			0%	0%	0%	0%	0%
6				0%	0.80%	0%	0%
7						0%	0%
8						0%	0%
9						0%	0%
10						0%	0%
11						0%	0%
12						0%	0%

Table 3 reports how many times (in terms of percentage) each alternative of the operators and parameters appears in the combined set of the final sets of the 10 medium-sized problem instances belonging to the medium-sized class (the combined set has 602 settings). Similar to the results of the small-sized class presented in Table 2, for the selection and mutation operators, and the probability of crossover and the population size parameters, almost all settings use the same alternatives, and for the crossover operator fourth alternative is popular. All alternatives of the probability of mutation and the number of generations appear in the settings. Some operators and paramaters are never used in medium-sized class as well and such ones can be classified as dominated alternatives for the medium-sized class.

Table 3 Percentage of times operators and parameters appear in the 602 efficient settings of medium-sized problem class

Index	*s*	*c*	*m*	*pc*	*pm*	*ps*	*ng*
1	0%	0%	100.00%	97.18%	6.98%	94.68%	24.42%
2	99.83%	1.33%	0%	1.66%	20.76%	2.49%	17.61%
3	0.17%	7.97%	0%	0.33%	54.65%	0.66%	13.95%
4	0%	90.70%	0%	0%	13.29%	0%	10.63%
5			0%	0.66%	2.16%	0%	11.30%
6				0.17%	2.16%	0%	6.48%
7						0%	1.50%
8						0.33%	3.99%
9						0.33%	4.49%
10						1.16%	1.00%
11						0.00%	2.16%
12						0.33%	2.49%

Table 4 reports how many times (in terms of percentage) each alternative of the operators and parameters appears in the combined set of the final sets of the 14 large-sized problem instances belonging to the large-sized class (the combined set has 1294 settings). Similar to the results of the small-sized and medium-sized class presented in Table 2 and for the selection and mutation operators, and the probability of crossover and the population size parameters, almost all settings use the same alternatives, and for the crossover operator fourth alternative is popular. All alternatives of the probability of mutation and the number of generations appear in the settings. Now, the number of alternatives for the operators and parameters that are never used (dominated) are less.

Table 4 Percentage of times operators and parameters appear in the 1294 efficient settings of large-sized problem class

Index	*s*	*c*	*m*	*pc*	*pm*	*ps*	*ng*
1	0%	0.00%	97.14%	99.07%	6.03%	87.87%	19.09%
2	99.92%	1.16%	0.00%	0.70%	19.55%	6.65%	14.99%
3	0.08%	12.98%	2.86%	0.08%	51.93%	0.85%	10.74%
4	0%	85.86%	0.00%	0.00%	19.78%	2.40%	8.58%
5			0.00%	0.15%	1.70%	0.46%	10.36%
6				0.00%	1.00%	0.62%	6.88%
7						0.00%	3.63%
8						0.15%	5.02%
9						0.23%	5.18%
10						0.31%	4.79%
11						0.15%	8.19%
12						0.31%	2.55%

To conclude, regardless of the problem size, almost all settings use second alternative of the selection operator, first alternative of the mutation operator, and first alternative of the probability of crossover parameter. For the crossover operator, the popularity of third and fourth operators increase when the problem size increases. For the probability of mutation, first four alternatives are shared by the settings and altough last two alternatives appear in the settings as well they are very rare. For the population size, the first alternative is common regardless of the problem size. Yet, its popularity decreases when the problem size increases. For the number of generations, the summation of the alternatives that have larger values increase when the problem size increase.

In the following, we provide more detailed results for one arbitrarily selected problem instances from each of three size classes. “dantzig42” (42 nodes), “st70” (70 nodes) and “u159” (159 nodes) are the arbitrarily selected instances for small-sized, medium-sized, and large-sized problem classes, respectively.

4.1.1. The Results of “dantzig42”

“dantzig42” has 42 nodes and is a small-sized problem. NSGA-II terminates with a set of 30 efficient GA settings for “dantzig42”. Each setting has two objective function values, i.e. the optimality gap (f_1) and run time (f_2) of the GA. The final solution set of NSGA-II (Pareto-optimal set approximation of the problem) is presented in Figure 15. As expected, two objectives are in conflict, i.e. decreasing f_1 increases f_2. The trade-off between the two objectives can be observed well from this figure.

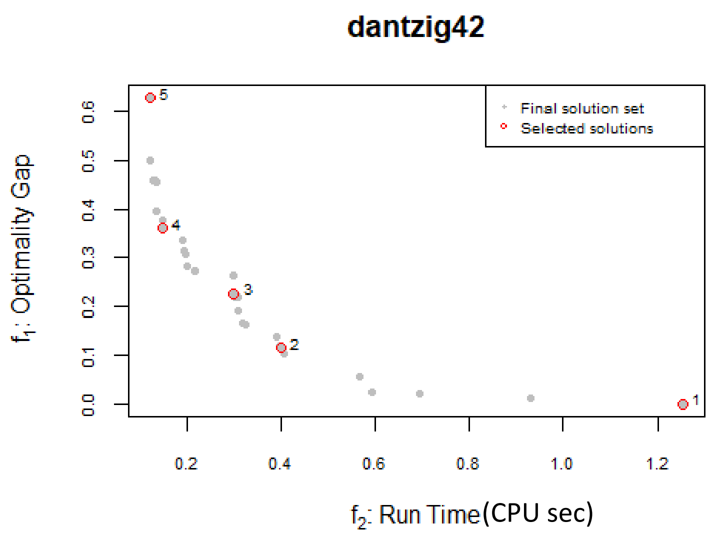

Figure 15 Final solution set of NSGA-II when GA is used to solve “dantzig42” and some selected efficient solutions for illustration

We select 5 efficient solutions arbitrarily to investigate the settings they consist of in their chromosomes. The selected solutions are labelled in Figure 15 and their details are reported in Table 5. Columns *s, c, m, pc, pm, ps*, and *ng* in Table 5 report selection operator, crossover operator, mutation operator, probability of mutation, probability of crossover, and number of generations, respectively, used in the chromosomes of solutions. Numbers reported for selection, crossover, and mutation operators are the indexes of the alternative operators. These indexes are reported in Table 1 before.

Table 5 Selected efficient solutions for illustration when GA is used to solve “dantzig42”

Selected Sols.	***s***	***c***	***m***	***pc***	***pm***	***ps***	***ng***	**f_1 (opt. gap)**	**f_2 (wall-clock sec.)**
1 (extreme)	2	4	1	0.1	0.5	50	200	0	1.26
2	2	4	1	0.1	0.25	50	100	0.12	0.40
3	2	2	1	0.1	0.5	50	50	0.23	0.30
4	2	4	1	0.1	0.1	50	50	0.36	0.15
5 (extreme)	4	4	1	0.1	0.1	50	50	0.63	0.12

Solutions 1 and 5 are extreme solutions. First extreme solution (Solution 1) has (0, 1.255) for (f_1, f_2) indicating that the settings defined for the operators and parameters allows GA to find the exact optimal solution. However, this setting causes the worst run time among all efficient solutions. Second extreme solution (Solution 5) has (0.628, 0.121) for (f_1, f_2) indicating that the settings defined for the operators and parameters let GA to run quickly but cause GA to find a solution that is far away from the optimal solution.

When the chromosome structures of five solutions are compared, it is observed that mutation operator (which is simple inversion mutation operator – gaperm_simMutation), the probability of crossover (which is 0.1) and the population size (which is 50) are exactly the same for all five solutions. For the selection operator, all solutions except Solution 5 uses nonlinear rank selection operator (gaperm_nlrSelection). Solution 5, on the other hand, uses tournament selection operator (gaperm_tourSelection). For the crossover, position-based crossover (gaperm_pbxCrossover) is used in four of five solutions. Partially matched crossover (gaperm_pmxCrossover) is used only in Solution 3.

When we move from the left extreme solution (Solution 5) to the right extreme solution (Solution 1), optimality gap decreases and run time increases. In general, moving from first extreme to the second extreme, probability of mutation and number of generations decrease. This makes sense since higher mutation probabilities let GA to jump to different locations of the search space, but at the same time reduce the convergence ability of the GA. Larger number of generations also increases the run time since the population need to be evaluated more times.

Table 6 reports how many times (in terms of percentage) each alternative of the operators and parameters appears in the efficient set of "dantzig42" (the set has 30 settings). For example, none of the 30 efficient settings use the first and third alternatives of the selection operator, and almost all of them uses the second alternative selection operator. For the crossover operator, fourth alternative is popular and for the mutation operator all solutions use the first alternative. All efficient solutions use the smallest probability of the crossover and the smallest population size. For probability of mutation, three alternatives are shared and for the number of generations four alternatives are shared. For the number of generations, the smallest alternative is popular but first four alternatives appear in the efficient solutions. We may interpret the alternatives of the operators and parameters that are used by none of the 30 efficient settings as the dominated alnternatives for the instance "dantzig42".

Table 6 Percentage of times operators and parameters appear in the 30 efficient settings of "dantzig42"

Index	*s*	*c*	*m*	*pc*	*pm*	*ps*	*ng*
1	0%	3%	100%	100%	27%	100%	67%
2	97%	17%	0%	0%	40%	0%	20%
3	0%	1%	0%	0%	33%	0%	10%
4	3%	70%	0%	0%	0%	0%	3%
5			0%	0%	0%	0%	0%
6				0%	0%	0%	0%
7						0%	0%
8						0%	0%
9						0%	0%
10						0%	0%
11						0%	0%
12						0%	0%

4.1.2. The Results of "st70"

The second selected instance is "st70", which has 70 nodes and is classified as a medium-sized problem instance. NSGA-II terminates with a set of 49 efficient GA settings for "st70". The final solution set of NSGA-II (Pareto-optimal set approximation of the problem) is presented in Figure 16. As expected, again the two objectives are in conflict, i.e. decreasing f_1 increases f_2.

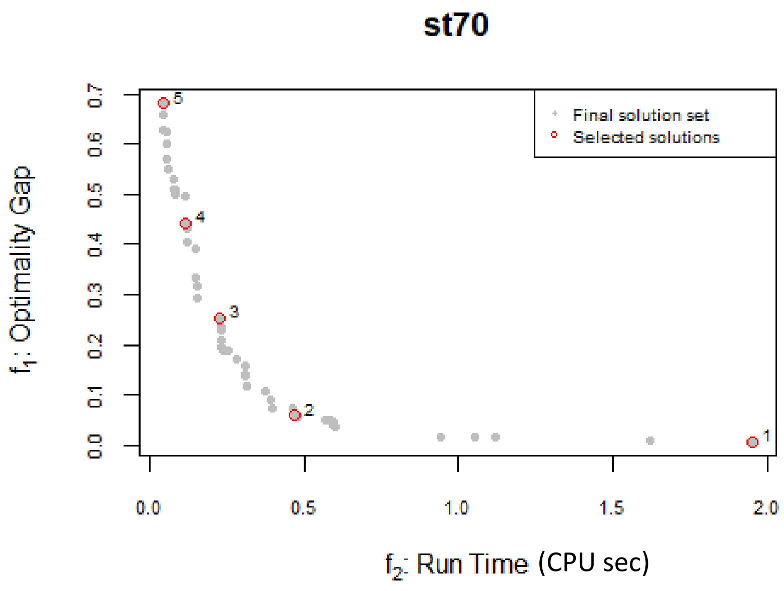

Figure 16 Final solution set of NSGA-II when GA is used to solve "st70" and some selected sample solutions for illustration

We again select 5 efficient solutions arbitrarily to investigate the settings they consist of in their chromosomes. The selected solutions are labelled in Figure 16, and their details are reported in Table 7. Table 7 has the same structure as the Table 5 presented in the previous section.

Table 7 Selected efficient solutions for illustration when GA is used to solve "st70"

Selected Sols.	s	c	m	pc	pm	ps	ng	f_1 (opt. gap)	f_2 (wall-clock sec.)
1 (extreme)	2	4	1	0.5	1	50	700	0.01	1.95
2	2	4	1	0.1	0.5	50	300	0.06	0.47
3	2	4	1	0.1	0.5	50	150	0.25	0.23
4	2	4	1	0.1	0.25	50	100	0.44	0.12
5 (extreme)	2	4	1	0.1	0.1	50	50	0.68	0.05

Solutions 1 and 5 are extreme solutions. Solution 1 indicates that the settings defined for the operators and parameters allows GA to approximate the optimal solution very well. However, this setting causes the worst run time (1.953 sec) among all efficient solutions. Second extreme solution indicates that there is a setting that let GA to run very fast (0.046 sec), however, causes GA to sacrifice from the approximation quality too much.

It is interesting to note in Table 7 that all five efficient solutions use same operators for selection (gaperm_nlrSelection), crossover (gaperm_pbxCrossover), and mutation operators (gaperm_simMutation), and same size (50) for population size parameter. This indicates that the operators do not affect the objective function values of the Pareto-optimal solutions significantly. Indeed, population size 50 is the alternative with the lowest size. It is expected that using a small population size decreases run time (see solution 5) but it is interesting to observe that using a small size but increasing the number of generations is enough for a high approximation quality (see solution 1).

When we move from first extreme (Solution 1) to the second extreme (Solution 2), optimality gap decreases, run time increases, and probability of mutation and number of generations decrease as well. This is expected because, although using higher mutation probabilities lets GA to discover more solutions and increases approximation quality, it reduces the convergence ability of the GA and increases the run time. Larger number of generations also increases the run time since the population need to be evaluated more times. In the previous section, all selected extreme solutions for “dantzig42” had 0.1 for crossover probability. Now, the results are similar except that Solution 1 uses a different probability (0.5) for the crossover.

Table 8 reports how many times (in terms of percentage) each alternative of the operators and parameters appears in the efficient set of “st70” (the set has 49 settings). Similar to the “dantzig42”, the second alternative of the selection operator is common in all efficient solutions. For crossover operator, fourth alternative is popular and for mutation operator all solutions use the first alternative. Almost all efficient solutions use the smallest alternative of the probability of crossover and all use the smallest population size. For probability of mutation, only the fifth alternative does not appear in any efficient solutions. Five alternatives of the number of generation parameter are not used by any efficient solutions, and the used alternatives are shared approximately equally by the first five alternatives. We may interpret the alternatives of the operators and parameters that are used by none of the 49 efficient settings as the dominated alternatives for the instance “st70”.

Table 8 Percentage of times operators and parameters appear in the 49 efficient settings of “st70”

Index	*s*	*c*	*m*	*pc*	*pm*	*ps*	*ng*
1	0%	0%	100%	98%	8%	100%	22%
2	100%	2%	0%	0%	14%	0%	16%
3	0%	6%	0%	2%	55%	0%	18%
4	0%	92%	0%	0%	18%	0%	14%
5			0%	0%	0%	0%	18%
6				0%	4%	0%	0%
7						0%	0%
8						0%	4%
9						0%	6%
10						0%	0%
11						0%	0%
12						0%	0%

4.1.3. The Results of “u159”

Finally, we illustrate the results of problem instance “u159”, which has 159 nodes and belongs to large-sized problem class. NSGA-II terminates with a set of 99 efficient GA settings for “u159”. The final solution set of NSGA-II (Pareto-optimal set approximation of the problem) is presented in Figure 17. As expected and similar to the previous two illustrations, two objectives of the GA are in conflict, i.e. solutions having lower optimality gaps have higher run times.

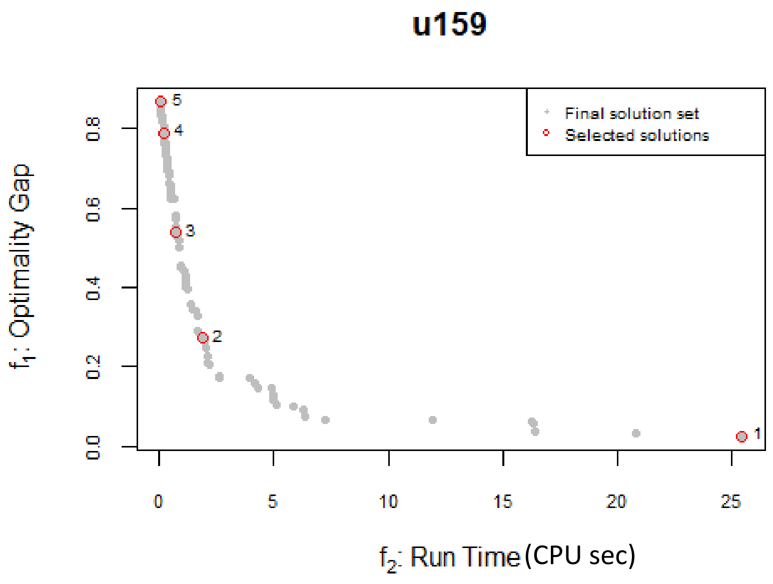

Figure 17 Final solution set of NSGA-II when GA is used to solve "u159" and some selected sample solutions for illustration

Similar to the previous problems, we investigate the details of 5 efficient solutions that are selected arbitrarily. The selected solutions are labelled in Figure 17, and their details are reported in Table 9. Table 9 has the same structure as the Table 5 and Table 7 presented in the previous sections.

Table 9 Selected efficient solutions for illustration when GA is used to solve "u159"

Selected Sols.	**s**	**c**	**m**	**pc**	**pm**	**ps**	**ng**	**f_1 (opt. gap)**	**f_2 (wall-clock sec.)**
1 (extreme)	2	4	1	0.25	0.9	200	900	0.03	25.45
2	2	3	1	0.1	0.5	50	700	0.27	1.87
3	2	4	1	0.1	0.5	50	300	0.54	0.71
4	2	3	1	0.1	0.25	50	100	0.79	0.20
5 (extreme)	2	4	3	0.1	0.1	50	50	0.87	0.07

Even for this large-sized problem, the first extreme solution consists of settings that let GA to approximate optimal solution very well. From solution 1 to the solution 5, approximation quality decreases; however, run time improves. When we look at the details of the settings, we see that solution 1 uses a very large setting for probability of mutation and number of generations. It also uses a higher value for the population size compared to the other solutions.

All five efficient solutions use the same selection operator (gaperm_nlrSelection). For crossover, 3rd (gaperm_oxCrossover) and 4th (gaperm_pbxCrossover) operators are preferred. For mutation, first operator (gaperm_simMutation) is common except for the solution 5. Except the first solution, all settings have 50 as the population size Probability of mutation, number of generations and population size decrease when we sacrifice from the optimality gap to improve the run time.

Table 10 reports how many times (in terms of percentage) each alternative of the operators and parameters appears in the efficient set of “u159” (the set has 99 settings). Similar to the previous two problem instances, the second alternative of the selection operator is common in all efficient solutions. For crossover and mutation operators, fourth and first alternatives are popular, respectively. Almost all efficient solutions use the smallest probability of crossover and most of them use the smallest population size. For the probability of mutation, only the fifth alternative does not appear in any efficient solutions. The alternatives of the number of generations are approximately distributed equally. We may interpret the alternatives of the operators and parameters that are used by none of the 99 efficient settings as the dominated alternatives for the instance “u159”.

Table 10 Percentage of times operators and parameters appear in the 99 efficient settings of “u159”

Index	*s*	*c*	*m*	*pc*	*pm*	*ps*	*ng*
1	0%	0%	97%	98%	7%	83%	15%
2	100%	3%	0%	2%	20%	11%	17%
3	0%	13%	3%	0%	56%	0%	9%
4	0%	84%	0%	0%	15%	6%	10%
5			0%	0%	2%	0%	10%
6				0%	0%	0%	5%
7						0%	3%
8						0%	2%
9						0%	9%
10						0%	6%
11						0%	13%
12						0%	0%

4.1.4. Discussion of the Results of “dantzig42”, “st70”, and “u159”

In this section, the results of the three problem instances are compared and discussed to understand the impacts of settings on the performance of the GA. First of all, as shown in Figure 15, Figure 16, Figure 17, there are trade-offs between f_1 and f_2. As the quality of the GA’s approximation increases, the time spent by the GA gets longer.

Problem instances with more nodes have more efficient solutions in their Pareto-optimal frontier. This is expected since more nodes lead a larger solution space and hence more trade-offs can be discovered. Increasing the size of the problem instances increases the ranges of the optimality gap and run time objectives. For “dantzig42”, optimality gaps and run times of the efficient solutions belong to intervals [0, 0.6284] and [0.121, 1.255], respectively. In the results of “st70”, the intervals of optimality gap and run time are [0.0073, 0.6817] and [0.046, 1.953], respectively. For “u159”, the intervals of optimality gap and run time are [0.026, 0.8684] and [0.069, 25.445], respectively. Both the minimum and maximum values of the results increased with the increase of the size of problems. According to the results of these three problems, both the minimum and maximum values of the optimality gap increase. On the other hand, the maximum value of the run time show a dramatic increase in the large-sized problem.

As it is reported in Table 6, Table 8 and

Table 10, some operators and parameters are approximately same regardless of the problem instance. In general, nonlinear rank selection operator is used for selection, position-based operator is used for crossover, and simple inversion operator is used for mutation operations. We observe that some operators and parameter values appear in none of the efficient sets of the “danzig42”, “st70” and “u159” and hence they can be classified as dominated operators and parameter values considering these three instances. For example, first and third selection operators; second, fourth and fifth mutation operators; fourth, fifth and sixth crossover probabilities; third and 5th to 12nd population sizes; and 12nd number of generations do not appear in the efficient sets of all instances and can be assumed as dominated.

In general, as the problem size increases from smaller to larger ones, efficient solutions require higher values for the population size, the probability of mutation and the number of generations. For example, when we look at the first extreme efficient solutions of three problem instances (these are the efficient solution with the best approximation quality), we observe that probability of mutation increases from 0.5 to 1, and number of generations increases from 200 to 900.

Considering the settings used by the selected efficient solutions, we can say that mutation probability and the number of generations are the two significant settings that determine the approximation quality and run time of the GA.

Finally, we see that there are many efficient settings for the GA that are found. Presenting all these solutions may confuse a user when determining which setting to use. The number of settings can be decreased using the preferences of decision makers and some structural approaches. For example, Pareto-optimal sets show that some efficient settings may be irrelevant for the decision maker. The approximation quality of the second extreme solutions of the problem instances (extreme solutions with the worst gap) are very large and unacceptable for decision makers. In general, approximations of a GA are expected to be within 10% optimality gap. Such preferences of decision makers can be used to reduce the number of efficient settings. Moreover, preference information of decision makers can be used to bias one of the objective functions (approximation quality and run time). Some may prefer better approximation gap, while some may prefer better run time. Using these preferences and some other structural approaches, a single setting that would work on all problem instances studied can be selected and presented to users. In the next section, we develop an algorithm and make some experiments to select the efficient settings that can work for all problem instances.

4.2. Reducing the Size of the Final Population of NSGA-II for the DM

NSGA-II approximates all Pareto-optimal settings. We showed in previous sections that Pareto-optimal sets are very diverse and large. However, it is better to provide a small set of settings to the practitioners to experiment when using their GA to solve their problems.

We use preferences of practitioners and develop a greedy algorithm to decrease the size of the Pareto-optimal sets and propose a final setting to the user when possible.

In general, there are two expectations from a heuristic algorithm. First, it should find a solution whose objective function value deviates from the optimal objective function value as less as possible. Second, it should approximate the optimal solution as fast as possible. There are trade-offs between these two expectations, and better approximations often require more solution time. One of the expectations can be favored based on the preferences of the user of the heuristic. Here, we arbitrarily define 10% as the maximum deviation from the optimal solution that the DM will accept. We assume that the DM wishes a well approximated solution, however he may slightly sacrifice from the approximation quality (up to 10%) in order to gain some advantage in solution time. 10% is a reasonable sacrifice, however, any other limit can also be determined depending on the preferences of the DM. We also experiment with 5% and 10% but due to the similarity of the results and the page considerations, here we only present the results for 10%.

We eliminate all efficient settings having more than 10% optimality gap. Then using remaining settings, we do some new experiments, and select the “best” setting based on some performance criteria we determine. We present this process as a greedy algorithm:

Step 0. Generate the Pareto-optimal frontier for each problem instance.

Step 1. Eliminate Pareto-optimal solutions based on the preferences of user.

Step 2. Combine the remaining Pareto-optimal solutions of the instances in each class (small, medium, large).

Step 3. For each problem size class, do the following:

Step 3.1. If there are solutions having exactly the same parameter settings, keep only one of them and eliminate the remaining.

Step 3.2. Each solution left corresponds to a setting for the GA. Using each of them, solve all instances belonging to the size class for *n* replications.

Step 3.2. For each solution, calculate in how many of the replications, the GA is able to find at least one efficient solution that satisfies the preferences of user for all problem instances belonging to the size class.

Step 3.3 Order solutions by the number of replications they solve all instances in ascending order and propose the "best" ones to the user as the final settings of the GA.

We use this algorithm as follows: After generating all Pareto-optimal sets in Step 0, we eliminate solutions having more than 10% optimality gap. Here, we assume that the only preference information we have is the 10% optimality gap. In Step 2, we combine the solutions within each problem class. When we eliminate the duplicate solutions in Step 3.1, there are 21, 65, and 70 solutions left for small, medium, and large-sized problem classes. In Step 3.2, we do 20 replications using 20 different seeds. These seeds are 26, 140, 164, 216, 346, 366, 410, 586, 622, 882, 916, 2340, 2590, 2945, 4892, 7588, 8716, 9014, 9520, and 9631. We report the results we find in Step 3.2 in Table 19, Table 20 and Table 21 in the Appendix for small, medium and large-sized problems, respectively. Finally, we determine a few numbers of "best" settings for the user in Step 3.3.

4.2.1. Results of Small-Sized Problems

In Figure 18, as an example, the Pareto-optimal set approximated using NSGA-II for the problem instance "dantzig42" and the preferred optimality gap tolerance is presented. The red line shows the 10% optimality gap tolerance. The values above the line are out of tolerance and are eliminated in Step 1 of the algorithm. These processes are conducted for all problem instances belonging to the small-sized problem class. Then remaining solutions of the problem instances belonging to the small-sized problem class are combined in Step 2.

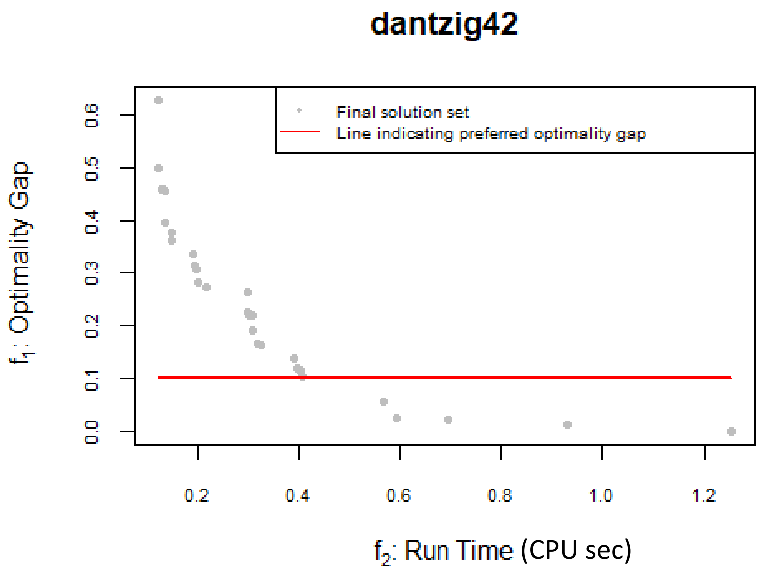

Figure 18 Preferred Pareto-optimal solutions of "dantzig42"

After we eliminate the duplicate solutions in Step 3.1, there are 21 solutions left. Each of these 21 solutions corresponds to a setting for the GA. Using each of them, we solve 7 problem instances (there are 7 problem instances classified as small-sized) for 20 replications is Step 3.2. The results of 21 settings are reported in Table 19 in Appendix 3. Here, in Table 11, we report only the best five settings, due to the page considerations, in terms of the number of runs they find at least one efficient solution that satisfies the %10 optimality gap limit for each of 7 problem instances. Index column shows the solution number given in Table 19. The statistic we are interested in is how many of seven instances are solved. Mean and Std. Dev. columns report the mean and standard deviation of the 20 replications. Min and max columns report the minimum and maximum number of problem instances solved when all replications are considered. The indexes of operators and parameters are given according to the index numbers in Table 1.

For example, the setting in row 1 is able to solve all 7 problem instances in 11 of 20 replications. The Min column shows that in the worst case, 5 of 7 instances are solved. Mean and standard deviation of 20 replications indicate that in the average 6.35 instances are solved and the standard deviation is less than 1. Hence, a practitioner solving a small-sized TSP with the GA, can use the settings presented in row 1. It has the highest mean and the highest number of successful solutions for all problems.

Table 11 Best five setting for small-sized TPS problems

Index	***s***	***c***	***m***	***pc***	***pm***	***ps***	***ng***	**Mean**	**Std. Dev.**	**Min**	**Max**	**Number of runs all solved**
10	2	4	1	1	3	1	4	6.35	0.81	5	7	11
18	2	2	1	1	6	1	3	6.10	0.72	5	7	6
19	2	3	1	1	3	1	3	5.95	0.83	4	7	5
20	2	2	1	1	2	1	4	5.20	1.06	3	7	2
14	2	4	1	1	4	1	2	5.05	0.76	4	7	1

4.2.2. Results of Medium-sized Problems

In Figure 19, as an example, the Pareto-optimal set approximated by using NSGA-II for the problem instance "st70" and the preferred optimality gap tolerance are presented. The red line shows the 10% optimality gap tolerance. The values above the line are out of tolerance and are eliminated in Step 1 of the algorithm. These processes are conducted for all problem instances belonging to the medium-sized problem class. Then remaining solutions of the problem instances belonging to the medium-sized problem class are combined in Step 2.

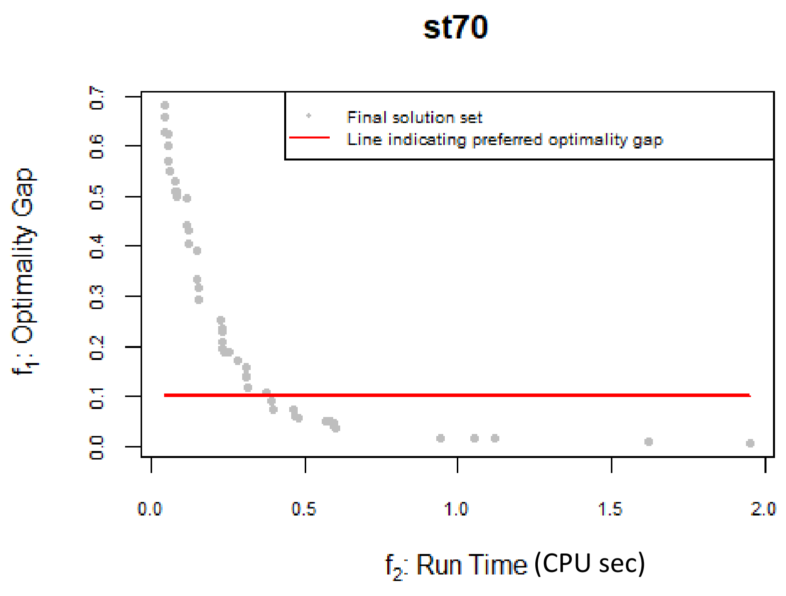

Figure 19 Preferred Pareto-optimal solutions of "st70"

After duplicate solutions are eliminated in Step 3.1, there are 65 solutions left. Each of these 65 solutions corresponds to a setting for the GA. Using each of them, we solve 10 problem instances (there are 10 problem instances classified as medium-sized) for 20 replications in Step 3.2. The results of 65 settings are reported in Table 20 in Appendix

4. Here, in Table 12, we report only the best five settings, due to the page considerations, in terms of the number of runs they find with at least one efficient solution that satisfies the %10 optimality gap limit for each of 10 problem instances. Index column shows the solution number given in Table 20. The structure of these tables is similar to the ones presented in the previous section for "dantzig42".

All 5 settings reported in Table 12 perform well in terms of mean, min, and number of runs all solved statistics. Setting 23 has the highest mean, which is very close to 10, and in the worst case it is able solve 9 out of 10 instances. In 12 out of 20 runs, it is able to solve all instances. Other four settings have similar results. Hence, any of these settings can be presented to the user.

Table 12 Best five setting for medium-sized TPS problems

Index	***s***	***c***	***m***	***pc***	***pm***	***ps***	***ng***	**Mean**	**Std**	**Min**	**Max**	**Number of runs all solved**
23	2	2	1	2	4	10	11	9.60	0.50	9	10	12
43	2	3	1	2	6	10	10	9.55	0.76	7	10	13
59	2	4	1	1	6	2	11	9.50	0.83	7	10	13
24	2	4	1	1	6	10	4	9.50	0.69	8	10	12
45	2	2	1	1	6	8	6	9.45	0.83	8	10	13

4.2.3. Results of Large-Sized Problems

The last class is the large-sized problem class. In Figure 20, as an example, the Pareto-optimal set approximated by using NSGA-II for the problem instance "u159" and the preferred optimality gap tolerance are presented. The red line shows the 10% optimality gap tolerance. The values above the line are out of tolerance and are eliminated in Step 1 of the algorithm. These processes are conducted for all problem instances belonging to the large-sized problem class. Then remaining solutions of the problem instances belonging to the large-sized problem class are combined in Step 2.

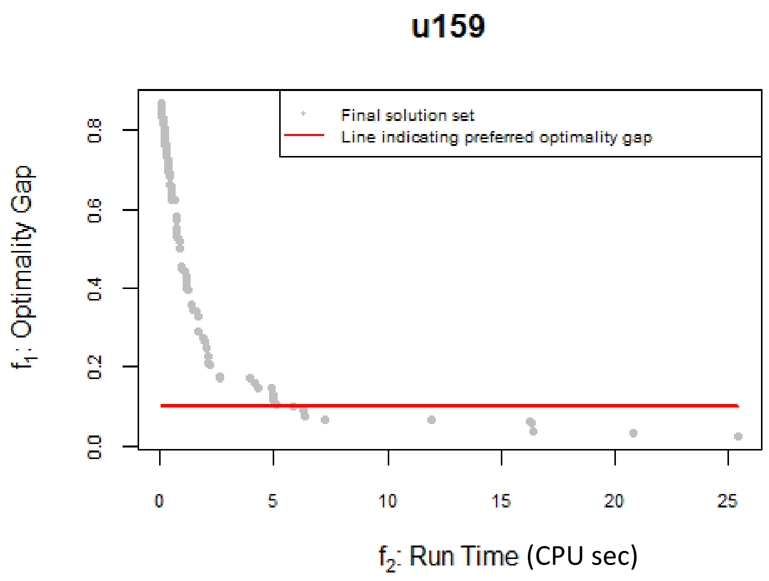

Figure 20 Preferred Pareto-optimal solutions of "u159"

After duplicate solutions are eliminated in Step 3.1, there are 70 solutions left. Each of these 70 solutions corresponds to a setting for the GA. Using each of them, we solve 14 problem instances (there are 14 problem instances classified as large-sized) for 20 replications in Step 3.2. The results of 70 settings are reported in Table 21 in Appendix 5. Here, in Table 13, we report only the best five settings, due to the page considerations, in terms of the number of runs they find with at least one efficient solution that satisfies the %10 optimality gap limit for each of 10 problem instances. The structure of the tables is similar to the ones presented in the previous sections for "dantzig42" and "st70".

In Table 13 and Table 21, the results are shared for the settings of large sized problems. It seems obvious among all the settings that the setting with the most stable performance is the setting 55. Its mean is close to 14, and in the worst case it is able solve 9 out of 14 instances. In 5 out of 20 runs, it is able to solve all instances. Although 5 is small, mean and min statistics shows that even all problem instances cannot be solved in a replication, at least 9 of them are solved.

Table 13 Best five setting for large-sized TPS problems

Index	***s***	***c***	***m***	***pc***	***pm***	***ps***	***ng***	**Mean**	**Std**	**Min**	**Max**	**Number of runs all solved**
55	2	3	1	1	6	5	11	12.3	1.38	9	14	5
43	2	3	1	1	5	9	11	12.05	1.32	9	14	1
27	2	4	1	2	5	12	9	11.75	1.16	10	14	1
36	2	4	1	1	6	6	8	11.5	1	10	13	0
25	2	4	1	1	5	5	11	10.8	1.67	8	13	0

4.2.4. Testing the Effective Settings of the Small-sized Instances on the Medium and Large-sized Instances

In this study we propose a greedy approach to reduce the size of the set of efficient settings, and demonstrate the approach on the TSP instances selected from the literature. When reducing the efficient settings, we classify the TSP instances into three classes (small, medium and large) based on the number of nodes they have. Each class is evaluated on its own and in each class, we select several efficient settings and show that these settings perform well in all problem instances belonging to the same class.

TSP is a well-studied problem and there are many problem instances with different sizes in the literature. Therefore, we are able to separate the instances based on their sizes and experiment in each class. Yet, heuristic algorithms are often employed problems that are very difficult to solve in the literature. For such problems, there are often not too many problem instances around and users have to experiment with the limited number of problem instances. The common approach in practice is experimenting on several small instances to determine the best settings of the GA or any other heuristic considered. Then employing these best settings, actual problems (which are often large-sized) are solved. However, the effectiveness of the settings found using a small problem instances on the larger problem instances is often not investigated.

To observe the performances of the effective settings of the small-sized TSP instances on larger problems, we experiment on all the medium-sized (10 instances) and large-sized (14 instances) instances using the five settings determined for small-sized problem instances in Table 11. Each problem instances of the medium and large-sized instances is solved 20 times with each effective setting of the small-sized instances. The averages of the optimality gaps of the 20 replications are reported in Table 14 and Table 15, respectively. In the tables, first eight columns report the details of the settings used, and the remaining columns report the average optimality gaps of the problem instances.

Table 14 and Table 15 reports that the effective settings found for the small problem instance class perform very poor on the medium and large-sized instances. These results show that searching the effective settings of a GA using small-problem instances (which is common in practice) may be misleading if the actual problem to be solved is large.

Table 14 The average optimality gaps of the five best settings of the small-sized problem class for the medium-sized problem instances

Index	***s***	***c***	***m***	***pc***	***pm***	***ps***	***ng***	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
10	2	4	1	1	3	1	4	13%	22%	24%	26%	40%	45%	46%	44%	43%	44%
18	2	2	1	1	6	1	3	15%	28%	29%	32%	47%	52%	52%	50%	49%	48%
19	2	3	1	1	3	1	3	15%	30%	28%	33%	47%	52%	52%	52%	51%	50%
20	2	2	1	1	2	1	4	19%	33%	33%	36%	50%	54%	55%	54%	54%	52%
14	2	4	1	1	4	1	2	19%	38%	37%	43%	57%	61%	62%	61%	60%	58%

Table 15 The average optimality gaps of the five best settings of the small-sized problem class for the large-sized problem instances

Index	***s***	***c***	***m***	***pc***	***pm***	***ps***	***ng***	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***
10	2	4	1	1	3	1	4	35%	49%	49%	57%	55%	57%	72%	64%	63%	72%	68%	70%	71%	75%
18	2	2	1	1	6	1	3	41%	53%	53%	60%	59%	62%	75%	67%	67%	74%	71%	72%	71%	76%
19	2	3	1	1	3	1	3	41%	56%	58%	61%	61%	62%	76%	69%	69%	77%	73%	74%	75%	78%
20	2	2	1	1	2	1	4	44%	56%	59%	62%	61%	64%	77%	69%	69%	77%	72%	74%	75%	78%
14	2	4	1	1	4	1	2	50%	63%	67%	68%	68%	70%	81%	74%	74%	81%	77%	78%	79%	81%

In order to understand the differences in the settings satisfying 10% optimality gap of different problem classes, we present Table 16. The best five settings of the small-sized problem class, uses second, first and first alternatives of the selection operator, mutation operator and the probability of crossover parameter. As it is seen Table 16. , For the selection and mutation operators, all problem classes use exactly the same alternatives. For the probability of crossover, most of the settings of medium and large-sized classes use the first alternative as well. Although the percentages of the alternatives of crossover operator slightly different in Table 16., when we look at Table 11, Table 12, and Table 13, we observe that similar alternatives are used for the crossover operator as well. However, the probability of mutation, the population size and the number of generation parameters differs from small-sized to medium and large-sized instances. Hence, we recommend a GA user who determines its effective settings using small-sized problem instances to increase these three parameters when experimenting on the actual (large-sized) problem he\she has. Please note that this reccomendation is empiric and based on the experiments we conducted on the 31 TSP problem instances. To secure the validty of the results, as a future study, more experiments can be conducted on different TSP instances and on different problems other than TSP.

Table 16 Percentage of times operators and parameters appear in the efficient solutions having less than 10% optimality gap of the small (sm), medium (md) and large-sized (lg) problem classes (values in the table are in terms of %)

Index	*s*			*c*			*m*			*pc*			*pm*			*ps*			*ng*		
	sm	***md***	***lg***	***sm***	***md***	***lg***	***sm***	***md***	***lg***	***sm***	***md***	***lg***	***sm***	***md***	***lg***	***sm***	***md***	***lg***	***sm***	***md***	***lg***
1	0	0	0	2.1	0	0	100	100	100	97.9	86.4	92.3	22.9	0	0.6	100	74.4	19.9	60.4	0	0
2	100	100	100	18.8	4.8	3.2	0	0	0	2.1	8.0	5.8	41.7	1.6	1.3	0	12.0	34.6	12.5	0.8	0
3	0	0	0	27.1	12.8	12.2	0	0	0	0	1.6	0.6	29.2	60.0	36.5	0	3.2	7.1	20.8	3.2	0
4	0	0	0	52.1	82.4	84.6	0	0	0	0	0	0	4.2	22.4	48.1	0	0	19.9	6.3	8.0	0.6
5							0	0	0	0	3.2	1.3	0	5.6	7.1	0	0	3.8	0	13.6	2.6
6										0	0.8	0	2.1	10.4	6.4	0	0.0	5.1	0	6.4	3.2
7																0	0	0	0	3.2	2.6
8																0	1.6	1.3	0	16.0	6.4
9																0	1.6	1.9	0	21.6	12.8
10																0	5.6	2.6	0	4.8	23.7
11																0	0	1.3	0	10.4	37.2
12																0	1.6	2.6	0	12.0	10.9

5. CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

In this work, we search for effective operator and parameter settings of GAs. It is known that settings of GAs have significant effects on the performance. Yet, determining the effective settings is often ignored in the literature and the settings are determined based on the previous experiences of the users or through trial-error experiments. In this thesis, we aimed to fill the gap in the literature and provided an approach that GA users can use when they set the operators and parameters of their GAs.

Typically, success of the heuristics is evaluated based on two performance criteria: approximation quality and run time. There are often trade-offs between these two criteria. In this study, we considered these two criteria and modelled the problem as a bi-objective optimization problem. We employed a MOEA to generate the Pareto-optimal frontier of the settings. A final best setting can be selected from this frontier depending on the requirements of the problem and preferences of the users.

When searching effective settings for GAs, we considered settings of the selection operator, crossover operator and mutation operator as GA operators, and the probability of crossover, probability of mutation, population size and number of generations as GA parameters.

We demonstrated our methodology by using 31 different TSP instances from the literature. We classified problem instances based on the number of nodes they had into small, medium and large problem classes. Our motivation is here to discover if different settings are successful for different sized problems. The GA we used is from the GA package of R and able to work with TSPs. For each operator and parameter, we considered different alternatives. In total there are 414,720 combinations of the alternatives and we searched the efficient of these combinations using the methodology developed. After finding Pareto-optimal settings, we developed and utilized a greedy algorithm that uses preferences of the user and some smart approaches to decrease the number of alternative settings. Thus, a few numbers of best settings can be presented to the users.

The results of the algorithm we created with MOEA have found successful settings for each problem. We found settings that could successfully solve all problems in less than 330 seconds with less than 10% gap. We examined the settings found again with a greedy heuristic that we proposed to solve all same size problems with a 10% gap. We evaluated the settings we found as a group according to the success of solving all the problems statistically. We obtained statistically strong results that we can recommend for each problem group. Please not that these results and discussions are based on the 31 TSP instances we experiment. Hence our reccomendations are empiric. However, we believe that the settings we propose can perform well for other problem instances having similar features since we conduct a wide range of experiments using many problem instances in different sizes. To secure the validty of the results, as a future study, more experiments can be conducted on different TSP instances.

In this study, we propose a general approach that can be used to determine the best settings of GAs and demonstrate the approach on the TSP instances. The results and recommendations provided in this thesis are based on the 31 TSP instances we experiment. However, the methodology can also be used to find the effective settings of GAs used to solve other problems. Additionally, the methodology can also be used to find the effective settings of the other heuristic algorithms as well. In practice, when a heuristic (let's say heuristic *H*) is employed to solve a problem, our approach can be employed following the steps below:

1. A MOEA needs to be chosen. We arbitrarily use NSGA-II in this study due to the its simple but effective structure, but any other algorithm can be employed as well.
2. In the selected MOEA, each solution in a population needs to represent a setting for the operators and parameters of heuristic *H*. In the chromosome structure, genes will represent the operators and parameters of heuristic *H*. Hence, each operator and parameter whose effective setting will be searched need to be encoded as a gene in the chromosome structure. Moreover, all alternatives for the operators and parameters considered need to be determined.

3. Depending on the type of operators and parameters (categorical or numerical), appropriate selection, crossover and mutation operators need to be determined for the MOEA used.
4. To evaluate a solution in the population, using the operator and parameter alternatives defined in the solution, heuristic *H* needs to be run on the optimization problem studied.
5. In this study we use optimality gap when measuring the approximation quality since we experiment on the problem instances whose optimal solutions are known. Yet, the optimal solution may not be known in practice. For such cases, objective function value can be used directly to measure the approximation quality. For example, if the problem is minimization type, a setting letting heuristic *H* to find an objective function value less then than that of another setting will be a better setting.
6. The problem instances to be solved using heuristic *H* need to be selected. If there are many problem instances around, or they can be easily generated, a similar approach that we used here for TSP can be employed. Problem instances can be classified based on the problem sizes, and the effective setting of each class can be search within itself. If the user is only able to experiment on limited number of small-sized instances, he\she should be aware that the effective settings he/she will find may not be that effective on larger problem instances. At least some trial-error experiments should be conducted changing some operators and parameters of the effective settings of small instances.
7. When the used MOEA terminates, a set of efficient settings will be found. Each setting will have a different trade-off in terms of the approximation quality and the run time. Depending on the preferences of the DM, the size of the set of efficient settings can be reduced and a final selection can be made.

In our experimental study to reduce the size of the efficient settings found for the TSP instance classes, we assume that the DM prefers solutions having optimality gap less than an upper limit. This choice eliminates many efficient settings that required effort to be found. In the multi-objective optimization literature, preference-based algorithms are developed for such cases. Preferences of DM's are elicited before or during algorithms, and only the efficient solutions that satisfy the preferences of DMs are searched. This approach saves computational costs and help algorithms to better approximate the focused search region. In our case, whenever there is a user whose preferences can be elicited, a preference-based algorithm can be employed and consideration of such an approach await future research. In this study, we have not considered such an approach since we aim to observe the all trade-offs between the approximation quality and the run time. Since the size of the efficient settings is too large in our experimental TSP study, we assume that the DM has an upper bound preference for the optimality gap and using this preference we reduce the number of the efficient settings. Although the MOEA we use spends extra computational efforts to explore the search regions that are not preferred by a DM, finding all efficient solutions may provide the DM more flexibility when defining his preferences. Moreover, the DM may change his/her preferences after the algorithm terminates. For example, he/she may sacrifice more from the approximation quality to gain from the run time. Sometimes, heuristic algorithms are used to find initial solutions and then these solutions are improved solving exact models through commercial solvers. Providing initial solutions to solvers may substantially improve their performances. Therefore, we present the complete trade-offs between the approximation quality and the run time in this thesis.

Another issue is that we have not consider fully the randomness in the nature of NSGA-II. When finding the efficient settings, we run NSGA-II for one replication. However, NSGA-II itself is a evolutionary algorithm and the efficient settings it finds may alter from one run to another. As a solution to this issue, we may run NSGA-II using different seeds, obtain a a final population for each seed, combine these final populations, eliminate dominated ones (if there are any), and select the remaining settings as the efficient settings. This may help eliminating the randomness in the nature of NSGA-II. Even more interesting, NSGA-II itself is a heuristic algorithm and the settings defined for its operators and parameters probably has an effect on the efficient solution set it finds. Consideration of such issues can be studied in the future to enhance the validity and the scope of the results of this study.

As an alternative to our approach, design of experiments (DOE) and response surfaces may be used to determine the effective settings. At the beginning of this study, we tried DOE to determine the effective settings. However, since there are too many factors with too many levels, the number of experiments need to be conducted is very large. Moreover, there are studies in the literature that consider DOE when searching for the effective settings of heuristics. We discuss the strong and weak properties of these studies in the literature review section. In general, the literature consider DOE to investigate the impacts of operators and parameters on the approximation quality. Yet, we consider two objectives (approximation quality and run time) and wish to develop a different approach than the literature. We defined the problem as a multi-objective optimization problem, and to so solve the multi-objective problem by employing a MOEA. The results of the MOEA presents a set of efficient setting each of which has a better value in terms of either the approximation quality or the run time. There is no study in the literature, to the best of our knowledge, that model the problem of finding the effective settings of GAs as a multi-objective optimization problem and follow a similar approach to ours.

This work can be extended in many ways in the feature. Settings can be searched with another MOEA instead of NSGA-II and impact of the used MOEA can be investigated. Different operators and parameters can be considered for GAs. More alternatives for each operator and parameter can be considered as well. The method should be demonstrated on other problem classes such as knap-sack, scheduling, and vehicle routing. Then results can be compared and better suggestions can be provided. Our methodology can be implemented to determine the effective settings of any heuristics. In addition to GAs, some other heuristics can be investigated.

6. REFERENCES

[1] J. H. Holland, Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence, MIT press, 1975 (reprinted 1992).

[2] M. Mitchell, An introduction to genetic algorithms, MIT press, 1998.

[3] J. R. Koza and R. Poli, "Genetic programming," in *Search Methodologies*, Boston, Springer, 2005, pp. 127-164.

[4] A. H. Wright, "Genetic algorithms for real parameter optimization," in *Foundations of genetic algorithms,* Elsevier, 1991, pp. 205-218.

[5] K. Deb, "An introduction to genetic algorithms," *Sadhana,* vol. 24, no. 4-5, pp. 293-315, 1999.

[6] D. A. Coley, An introduction to genetic algorithms for scientists and engineers, World Scientific Publishing Company, 1999.

[7] L. Davis, Handbook of genetic algorithms, CumInCAD, 1991.

[8] Á. E. Eiben, R. Hinterding and Z. Michalewicz, "Parameter control in evolutionary algorithms," *IEEE Transactions on evolutionary computation,* vol. 3, no. 2, pp. 124-141, 1999.

[9] S. P. Coy, B. L. Golden, G. C. Runger and E. A. Wasil, "Using experimental design to find effective parameter settings for heuristics," *Journal of Heuristics,* vol. 7, no. 1, pp. 77-97, 2001.

[10] A. Van Breedam, An analysis of the effect of local improvement operators in genetic algorithms and simulated annealing for the vehicle routing problem, RUCA, 1996.

[11] J. Xu and J. P. Kelly, "A network flow-based tabu search heuristic for the vehicle routing problem," *Transportation Science,* vol. 30, no. 4, pp. 379-393, 1996.

[12] B. L. Golden, E. A. Wasil, J. P. Kelly and I.-M. Chao, "The impact of metaheuristics on solving the vehicle routing problem: algorithms, problem

sets, and computational results," in *Fleet management and logistics*, Boston, MA: Kluwer Academic Publishers, 1998, pp. 33-56.

[13] J. J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Transactions on systems, man, and cybernetics,* vol. 16, no. 1, pp. 122-128, 1986.

[14] K. A. De Jong, "Analysis of the behavior of a class of genetic adaptive systems," 1975.

[15] T. Starkweather, S. McDaniel, K. E. Mathias, L. D. Whitley and C. Whitley, "A Comparison of Genetic Sequencing Operators," in *ICGA*, 1991.

[16] R. Parsons and M. E. Johnson, "A case study in experimental design applied to genetic algorithms with applications to DNA sequence assembly," *American Journal of Mathematical and Management Sciences,* vol. 17, no. 3-4, pp. 369-396, 1997.

[17] J. Magalhaes-Mendes, "A comparative study of crossover operators for genetic algorithms to solve the job shop scheduling problem," *WSEAS transactions on computers,* vol. 12, no. 4, pp. 164-173, 2013.

[18] E. K. Burke, G. Kendall ve et al., «Search methodologies,» %1 içinde *Multi-objective optimization,* New York, Springer Science+ Business Media, 2005, p. 273.

[19] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE transactions on evolutionary computation,* vol. 6, no. 2, pp. 182--197, 2002.

[20] K. Deb, Multi-objective optimization using evolutionary algorithms, Chichester: John Wiley & Sons, 2001.

[21] M. Ehrgott, Multicriteria Optimization, 2 ed., Springer-Verlag Berlin Heidelberg, 2005.

[22] E. Dasdemir, M. Köksalan and D. T. Öztürk, "A flexible reference point-based multi-objective evolutionary algorithm: An application to the UAV route planning problem," *Computers & Operations Research,* vol. 114, p. 104811, 2020.

[23] D. A. Van Veldhuizen and G. B. Lamont, "Multiobjective evolutionary algorithms: Analyzing the state-of-the-art," *Evolutionary computation,* vol. 8, no. 2, pp. 125-147, 2000.

[24] A. Ghosh and S. Dehuri, "Evolutionary algorithms for multi-criterion optimization: a survey," *International Journal of Computing & Information Sciences,* vol. 2, no. 1, pp. 38-57., 2004.

[25] C. A. C. Coello, G. B. Lamont and D. A. Van Veldhuizen, Evolutionary algorithms for solving multi-objective problems., 2nd ed., vol. 5, New York: Springer, 2007.

[26] C. A. C. Coello and G. B. Lamont, Applications of multi-objective evolutionary algorithms, World Scientific, 2004.

[27] J.-Y. Potvin, “Genetic algorithms for the traveling salesman problem,” *Annals of Operations Research,* vol. 63, no. 3, pp. 337-370, 1996.

[28] C. E. Miller, A. W. Tucker and Zemlin, “Integer Programming Formulation of Traveling Salesman Problems,” *Journal of the Association for Computing Machinery,* vol. 7, no. 4, pp. 326-329, 1960.

[29] G. Dantzig, R. Fulkerson and S. Johnson, “Solution of a large-scale traveling-salesman problem,” *Journal of the operations research society of America,* vol. 2, no. 4, pp. 393-410, 1954.

[30] G. Clarke and J. W. Wright, “Scheduling of vehicles from a central depot to a number of delivery points,” *Operations research,* vol. 12, no. 4, pp. 568-581, 1964.

[31] L. D. Whitley, T. Starkweather and F. D'Ann, “Scheduling problems and traveling salesmen: The genetic edge recombination operator,” *ICGA,* vol. 89, pp. 133-40, 1989.

[32] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, “Optimization by simulated annealing,” *science,* vol. 220, no. 4598, pp. 671-680, 1983.

[33] F. Alizadeh, R. M. Karp, L. A. Newberg and D. K. Weisser, “Physical mapping of chromosomes: A combinatorial problem in molecular biology,” *Algorithmica,* vol. 13, no. 1-2, pp. 52-76, 1995.

[34] C. Korostensky and G. H. Gonnet, “Using traveling salesman problem algorithms for evolutionary tree construction,” *Bioinformatics,* vol. 16, no. 7, pp. 619-627, 2000.

[35] H. Braun, “On solving travelling salesman problems by genetic algorithms,” in *International Conference on Parallel Problem Solving from Nature,* Springer, 1990.

[36] C. Moon, J. Kim, G. Choi and Y. Seo, “An efficient genetic algorithm for the traveling salesman problem with precedence constraints,” *European Journal of Operational Research,* vol. 140, no. 3, pp. 606-617, 2002.

[37] L. J. Schmitt and M. M. Amini, “Performance characteristics of alternative genetic algorithmic approaches to the traveling salesman problem using path representation: An empirical study,” *European Journal of Operational Research,* vol. 108, no. 3, pp. 551-570, 1998.

[38] H. K. Tsai, J. M. Yang, Y. F. Tsai and C. Y. Kao, “Some issues of designing genetic algorithms for traveling salesman problems,” *Soft Computing,* vol. 8, no. 10, pp. 689-697, 2004.

[39] R. Takahashi, «Solving the traveling salesman problem through genetic algorithms with changing crossover operators,» %1 içinde *Fourth International Conference on Machine Learning and Applications (ICMLA'05),* IEEE, 2005, pp. 6-pp.

[40] L. Scrucca, “GA: A Package for Genetic Algorithms in R,” *Journal of Statistical Software,* vol. 53, no. 4, pp. 1-37, April 2013.

[41] T. Bäck, D. Fogel and Z. Michalewicz, Evolutionary computation 1: Basic algorithms and operators, Bristol and Philadelphia: IOP Publishing, 2000.

[42] X. Yu and M. Gen, Introduction to evolutionary algorithms, Springer Science & Business Media, 2010.

[43] A. E. Eiben and J. E. Smith, Introduction to evolutionary computing, Berlin: Springer, 2003.

[44] P. Larranaga, C. M. H. Kuijpers, R. H. Murga, I. Inza and S. Dizdarevic, “Genetic algorithms for the travelling salesman problem: A review of

representations and operators," *Artificial Intelligence Review},* vol. 13, no. 2, pp. 129-170, 1999.

[45] N. M. Razali and J. Geraghty, "Genetic algorithm performance with different selection strategies in solving TSP," in *Proceedings of the world congress on engineering*, vol. 2, Hong Kong, International Association of Engineers, 2011, pp. 1-6.

[46] T. Bäck and F. Hoffmeister, "Extended selection mechanisms in genetic algorithms," 1991.

[47] L. D. Whitley, "The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best.," in *Icga*, Fairfax, VA, 1989, pp. 116-123.

[48] A. Hussain, Y. S. Muhammad, M. Nauman Sajid, I. Hussain, A. Mohamd Shoukry and S. Gani, "Genetic algorithm for traveling salesman problem with modified cycle crossover operator," *Computational intelligence and neuroscience,* vol. 2017, 2017.

[49] A. J. Umbarkar and P. D. Sheth, "Crossover operators in genetic algorithms: a review.," *ICTACT journal on soft computing,* vol. 6, no. 1, 2015.

[50] I. CPLEX, "Parameters of CPLEX: relative MIP gap tolerance," [Online]. Available: https://www.ibm.com/support/knowledgecenter/SSSA5P_12.9.0/ilog.odms.cplex.help/CPLEX/Parameters/topics/EpGap.html. [Accessed 24 August 2020].

[51] GUROBI, "Documentation: MIPGap," [Online]. Available: https://www.gurobi.com/documentation/9.0/refman/mipgap.html. [Accessed 2020 August 24].

[52] P. Larranaga, C. M. Kuijpers, R. H. Murga, I. Inza and S. Dizdarevic, "Genetic algorithms for the travelling salesman problem: A review of representations and operators.," *Artificial Intelligence Review,* vol. 13, no. 2, pp. 129-170, 1999.

[53] L. Scrucca and M. L. Scrucca, "Package 'GA'," 2019.

[54] R. C. Team, “R: A Language and Environment for Statistical Computing,” R Foundation for Statistical, [Online]. Available: http://www.R-project.org/. [Accessed 2020].

[55] Center for Computational Research (CCR), «CCR Facility Description,» University at Buffalo, 08 03 2019. [Çevrimiçi]. Available: http://hdl.handle.net/10477/79221. [Erişildi: 2020].

[56] C.-S. V. Tsou and M. M.-C. A. Lee, Package ‘nsga2R’, Citeseer, 2013.

[57] O. Mersmann, H. Trautmann, D. Steuer, B. Bischl and K. Deb, “Package “mco”: multiple criteria optimization algorithms and related functions,” 2014.

[58] I. Sergei , “tictoc: Functions for timing R scripts, as well as implementations of Stack and List structures,” *R package version,* vol. 1, 2016.

[59] J. Burkardt, «TSP - Data for the Traveling Salesperson Problem,» 23 July 2019. [Çevrimiçi]. Available: https://people.sc.fsu.edu/~jburkardt/datasets/tsp/tsp.html. [Erişildi: August 2019].

[60] G. Reinelt, “TSPLIB,” Universität Heidelberg, 1 June 1995. [Online]. Available: http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/index.html. [Accessed August 2019].

[61] G. Reinelt, “TSPLIB - A Travelling Salesman Problem Library,” *ORSA Journal on Computing,* vol. 3, no. 4, pp. 376 - 384, 1991.

[62] M. Grötschel and O. Holland, “Solution of large-scale symmetric travelling salesman problems,” *Mathematical Programming,* vol. 51, no. 1-3, pp. 141-202, 1991.

[63] M. Grötschel, «Optimierungsmethoden I, Lecture Notes,» Universität Ausburg, 1985.

[64] W. Cook, “Traveling Salesman Problem - Mathematics - University of Waterloo,” University of Waterloo, 10 March 2017. [Online]. Available: http://www.math.uwaterloo.ca/tsp/world/countries.html. [Accessed August 2019].

[65] M. Padberg and G. Rinaldi, “A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems,” IASI Reseach Report 247, 1988.

[66] P. Krolak, W. Felts and G. Marble, “A man-machine approach toward solving the traveling salesman problem,” *Communications of the ACM,* vol. 14, no. 5, pp. 327-334, 1971.

[67] B. Fruhwirth, «Untersuchung über genetisch motivierte Heuristiken für das Euklidische Rundreiseproblem,» Technischer Bericht 87-103, TU Graz, 1987.

[68] P. Van Laarhoven, “Theoretical and computational aspect of simulated annealing, Ph.D. Thesis,” Erasmus Universitiet Rotterdam, 1988.

[69] G. Reinelt, “Fast Heuristics for Large Geometric Travelling Salesman Problems, Report No.185,” Schwerpunktprogramm der Deutscher Forschungsgemeinschaft, Universität Augsburg, Augsburg, 1989.

[70] W. Cook, “Traveling Salesman Problem - Mathematics - University of Waterloo,” University of Waterloo, May 2013. [Online]. Available: http://www.math.uwaterloo.ca/tsp/vlsi/index.html. [Accessed August 2019].

APPENDICES

Appendix 1 – List of Problems

Details of the TSP instances are reported in Table 17 below. Problem size indicates the size class of the instances. Column "Name" presents the abbreviations of the instances, column "Number of Nodes" shows the number of nodes problem instances have and column "Data" reports the approach used to obtain the distances between node pairs. There two types of datasets used in the literature for TSP instances. MATRIX type datasets are data sets having the distance information between node pairs in a matrix format. EUC_2D type datasets are data sets having the (x, y) coordinates of the nodes. For those datasets, Euclidean distances between the coordinates are computed to find the distances between node pairs. Column "$f^*(x)$" presents the optimal objective function values, and the column "References" reports the sources of the problem data. Data type indicates whether the data is matrix or Euclidean type and $f^*(x)$ shows the optimal solution to the problem.

Table 17 List of problems

Problem Size	Index	Name	Number of Nodes	Data Type	$f^*(x)$	References
Small	1	P01	15	MATRIX	291	[59]
	2	gr17	17	MATRIX	2085	[60] [61] [62]
	3	fri26	26	MATRIX	937	[60]
	4	bays29	29	MATRIX	2020	[60] [63] [61]
	5	dj38	38	EUC_2D	6656	[64]
	6	dantzig42	42	MATRIX	699	[60] [61] [29] [62]
	7	swiss42	42	MATRIX	1273	[60]
Medium	8	berlin52	52	EUC_2D	7542	[60]
	9	st70	70	EUC_2D	675	[60] [61] [62] [65]
	10	eil76	76	EUC_2D	538	[60] [61] [65]
	11	pr76	76	EUC_2D	108159	[60] [61] [65]
	12	rat99	99	EUC_2D	1211	[60] [61]
	13	kroA100	100	EUC_2D	21282	[60] [61] [62] [65] [66] [67] [68]
	14	kroC100	100	EUC_2D	20749	[60] [61] [62] [65] [66] [67] [68]
	15	kroD100	100	EUC_2D	21294	[60] [61] [62] [65] [66] [67] [68]
	16	kroE100	100	EUC_2D	22068	[60] [61] [62] [65] [66] [67] [68]
	17	rd100	100	EUC_2D	7910	[60] [61] [69]
Large	18	eil101	101	EUC_2D	629	[60] [61] [65]

	19	lin105	105	EUC_2D	14379	[60] [61] [65]
	20	pr107	107	EUC_2D	44303	[60] [61] [65]
	21	ch130	130	EUC_2D	6110	[60]
	22	xqf131	131	EUC_2D	564	[70]
	23	pr136	136	EUC_2D	96772	[60] [61] [65]
	24	pr144	144	EUC_2D	58537	[60] [61] [65]
	25	kroA150	150	EUC_2D	26524	[60] [61] [65] [66]
	26	kroB150	150	EUC_2D	26130	[60] [61] [65] [66]
	27	pr152	152	EUC_2D	73682	[60] [61] [65]
	28	u159	159	EUC_2D	42080	[60] [61]
	29	qa194	194	EUC_2D	9352	[64]
	30	d198	198	EUC_2D	15780	[60] [61] [69]
	31	kroA200	200	EUC_2D	29368	[60] [61] [65] [66]

Appendix 2 – The final solution set of MOEA for TSP problems in the optimality gap tolerance (Seed=70646)

The results of the study with MOEA are given in the table below. *s*, *c*, *m*, *pc*, *pm*, *ps*, *ng*, f_1 and f_2 indicates selection operator, crossover operator, mutation operator, probability of crossover, probability of mutation, population size, number of generations, approximate quality, and approximate time, respectively. The results of operators are given according to index numbers in Table 1.

Table 18 The final solution set of MOEA for TSP problems

Name	*s*	*c*	*m*	*pc*	*pm*	*ps*	*ng*	f_1	f_2
bays29	2	3	1	0.1	0.75	50	50	0.00	0.32
bays29	2	3	1	0.1	0.5	50	50	0.00	0.25
bays29	2	2	1	0.1	0.5	50	50	0.01	0.24
bays29	2	4	1	0.1	0.5	50	50	0.01	0.23
bays29	2	2	1	0.25	0.25	50	50	0.03	0.20
bays29	2	3	1	0.1	0.25	50	50	0.03	0.17
bays29	2	2	1	0.1	0.25	50	50	0.04	0.16
bays29	2	4	1	0.1	0.25	50	50	0.08	0.15
bays29	2	4	1	0.1	0.1	50	50	0.08	0.11
berlin52	2	4	1	0.1	0.5	50	600	0.00	0.86
berlin52	2	4	1	0.25	0.5	50	300	0.01	0.48
berlin52	2	4	1	0.1	0.9	50	200	0.01	0.38
berlin52	2	4	1	0.1	0.5	50	200	0.03	0.31
berlin52	2	3	1	0.1	0.5	50	200	0.03	0.29
berlin52	2	4	1	0.1	0.25	50	200	0.04	0.23
berlin52	2	3	1	0.1	0.5	50	150	0.06	0.22
berlin52	2	4	1	0.1	0.5	50	150	0.06	0.21
berlin52	2	4	1	0.1	0.75	50	100	0.08	0.17
ch130	2	4	1	0.1	0.5	200	800	0.02	10.34
ch130	2	4	1	0.1	0.75	100	900	0.03	5.61
ch130	2	4	1	0.1	0.75	100	800	0.05	5.01
ch130	2	4	1	0.1	0.75	100	600	0.05	3.80
ch130	2	4	1	0.1	0.75	50	800	0.09	2.09
d198	2	4	1	0.1	0.75	300	900	0.02	45.29
d198	2	4	1	0.1	0.75	200	900	0.03	22.15
d198	2	3	1	0.1	0.75	150	900	0.04	15.03
d198	2	4	1	0.1	0.75	150	900	0.04	14.15
d198	2	4	1	0.1	0.75	150	800	0.05	12.56
d198	2	4	1	0.1	0.75	100	900	0.06	7.59
d198	2	4	1	0.1	0.75	100	800	0.09	6.72

dantzig42	2	4	1	0.1	0.5	50	200	0.00	1.26
dantzig42	2	4	1	0.1	0.5	50	150	0.01	0.93
dantzig42	2	1	1	0.1	0.5	50	100	0.02	0.69
dantzig42	2	4	1	0.1	0.25	50	150	0.03	0.59
dj38	2	4	1	0.1	0.75	50	100	0.00	0.16
dj38	2	4	1	0.1	0.5	50	100	0.00	0.13
dj38	2	4	1	0.1	0.25	50	100	0.04	0.10
eil101	2	4	1	0.1	1	1000	300	0.05	107.66
eil101	2	4	1	0.1	0.75	1000	300	0.05	78.57
eil101	2	3	1	0.1	1	1000	200	0.05	72.02
eil101	2	4	1	0.1	0.5	50	900	0.06	1.69
eil101	2	4	1	0.1	0.5	50	700	0.08	1.29
eil101	2	4	1	0.1	0.75	50	500	0.09	1.23
eil76	2	3	1	0.25	1	1000	700	0.03	193.46
eil76	2	3	1	0.25	0.5	1000	900	0.03	145.04
eil76	2	3	1	0.9	0.5	800	700	0.03	118.03
eil76	2	3	1	0.9	0.75	100	700	0.03	5.92
eil76	2	3	1	0.9	0.5	100	700	0.04	5.45
eil76	2	4	1	0.9	0.5	50	700	0.04	2.00
eil76	2	4	1	0.5	0.5	50	700	0.05	1.64
eil76	2	3	1	0.1	0.5	50	900	0.05	1.63
eil76	2	4	1	0.1	0.5	50	700	0.05	1.14
eil76	2	4	1	0.1	0.75	50	400	0.05	0.83
eil76	2	4	1	0.1	0.5	50	400	0.06	0.65
eil76	2	4	1	0.1	0.75	50	300	0.07	0.59
eil76	2	4	1	0.1	0.5	50	300	0.08	0.48
fri26	2	4	1	0.1	0.5	50	50	0.00	0.21
fri26	2	4	1	0.1	0.25	50	50	0.00	0.15
fri26	2	2	1	0.1	0.25	50	50	0.02	0.14
fri26	2	2	1	0.1	0.1	50	50	0.06	0.10
gr17	2	4	1	0.1	0.1	50	50	0.00	0.08
kroA100	2	2	1	0.25	0.75	800	900	0.00	176.84
kroA100	2	4	1	0.1	1	800	200	0.00	44.45
kroA100	2	4	1	1	0.75	150	900	0.01	13.90
kroA100	2	4	1	0.1	0.9	150	900	0.02	9.19
kroA100	2	4	1	0.1	0.9	150	500	0.02	5.19
kroA100	2	4	1	0.25	0.75	50	900	0.02	2.34
kroA100	2	3	1	0.1	0.75	50	900	0.03	2.14
kroA100	2	4	1	0.1	0.5	50	700	0.03	1.44
kroA100	2	4	1	0.1	0.5	50	600	0.04	1.11
kroA150	2	3	1	0.5	0.25	600	900	0.03	103.27
kroA150	2	4	1	0.1	0.75	400	1000	0.03	66.20
kroA150	2	4	1	0.1	0.5	400	800	0.04	37.68
kroA150	2	4	1	0.1	0.25	400	1000	0.05	28.13

kroA150	2	4	1	0.1	0.5	150	1000	0.05	9.37
kroA150	2	4	1	0.1	0.5	100	1000	0.06	5.49
kroA150	2	4	1	0.1	0.5	100	800	0.08	4.41
kroA200	2	3	1	0.1	0.5	700	900	0.04	164.81
kroA200	2	4	1	0.1	0.9	300	900	0.04	53.56
kroA200	2	4	1	0.1	0.75	200	900	0.05	22.80
kroA200	2	4	1	0.1	0.5	200	900	0.09	16.87
kroB150	2	4	1	0.25	0.9	1000	700	0.04	327.32
kroB150	2	4	1	0.1	0.75	200	800	0.04	15.85
kroB150	2	4	1	0.1	0.5	200	700	0.04	10.00
kroB150	2	4	1	0.1	0.75	100	800	0.05	6.10
kroB150	2	4	1	0.1	0.5	100	800	0.09	4.20
kroC100	2	4	1	0.1	0.5	100	1000	0.01	4.01
kroC100	2	4	1	0.1	0.75	50	1000	0.02	2.24
kroC100	2	4	1	0.1	0.5	50	1000	0.04	1.85
kroC100	2	4	1	0.1	0.5	50	700	0.05	1.29
kroC100	2	4	1	0.1	0.5	50	600	0.09	1.10
kroD100	2	3	1	0.1	0.9	100	1000	0.01	6.14
kroD100	2	3	1	0.1	0.9	100	500	0.02	3.11
kroD100	2	4	1	0.25	0.9	50	1000	0.03	2.76
kroD100	2	4	1	0.25	0.5	50	1000	0.03	2.17
kroD100	2	4	1	0.1	0.5	50	900	0.04	1.67
kroD100	2	4	1	0.1	0.5	50	700	0.06	1.34
kroD100	2	4	1	0.1	0.5	50	600	0.07	1.14
kroE100	2	4	1	0.1	0.9	700	200	0.01	31.49
kroE100	2	4	1	0.1	0.5	700	200	0.02	17.41
kroE100	2	4	1	0.1	1	100	1000	0.02	6.11
kroE100	2	4	1	0.1	0.75	100	1000	0.02	5.09
kroE100	2	4	1	0.1	0.5	50	1000	0.03	1.85
kroE100	2	4	1	0.1	0.75	50	700	0.03	1.62
kroE100	2	4	1	0.1	0.5	50	700	0.04	1.32
kroE100	2	4	1	0.1	0.5	50	600	0.07	1.11
lin105	2	3	1	0.1	1	150	800	0.01	9.55
lin105	2	4	1	0.1	0.5	150	800	0.01	5.78
lin105	2	4	1	0.1	0.5	100	700	0.03	2.94
lin105	2	4	1	0.1	0.75	50	800	0.04	1.85
lin105	2	4	1	0.1	0.5	50	800	0.06	1.59
lin105	2	3	1	0.1	0.5	50	700	0.06	1.46
lin105	2	4	1	0.1	0.5	50	700	0.08	1.36
lin105	2	4	1	0.1	0.75	50	500	0.08	1.19
P01	2	4	1	0.1	0.1	50	50	0.00	0.07
pr107	2	4	1	0.9	0.5	800	600	0.00	134.35
pr107	2	4	1	0.1	1	400	600	0.01	36.07
pr107	2	4	1	0.25	1	400	400	0.01	26.84

pr107	2	4	1	0.1	1	400	400	0.01	24.49
pr107	2	4	1	0.1	0.5	200	600	0.01	6.74
pr107	2	4	1	0.1	0.5	100	800	0.02	3.42
pr107	2	2	1	0.1	0.9	50	1000	0.02	2.88
pr107	2	4	1	0.1	0.9	50	1000	0.02	2.70
pr107	2	4	1	0.1	0.5	50	800	0.03	1.57
pr107	2	4	1	0.1	0.5	50	700	0.06	1.33
pr107	2	4	1	0.1	0.5	50	600	0.07	1.19
pr136	2	3	1	0.1	0.9	700	900	0.02	190.03
pr136	2	3	1	0.1	0.5	700	900	0.03	113.55
pr136	2	4	1	0.1	0.5	900	300	0.03	57.20
pr136	2	4	1	0.9	0.5	200	800	0.04	21.30
pr136	2	4	1	0.1	0.75	100	900	0.04	5.80
pr136	2	4	1	0.1	0.5	100	900	0.07	4.51
pr136	2	4	1	0.1	0.75	50	900	0.08	2.46
pr144	2	4	1	0.25	0.75	800	800	0.00	207.75
pr144	2	3	1	0.25	0.1	800	800	0.00	71.77
pr144	2	4	1	0.1	0.75	200	700	0.00	13.19
pr144	2	4	1	0.1	0.5	200	700	0.01	9.91
pr144	2	4	1	0.1	0.75	100	1000	0.02	6.63
pr144	2	4	1	0.1	0.5	100	1000	0.02	5.26
pr144	2	3	1	0.1	0.75	100	700	0.03	4.97
pr144	2	4	1	0.1	0.75	100	700	0.04	4.63
pr144	2	4	1	0.1	0.5	100	700	0.07	3.69
pr144	2	4	1	0.1	0.75	50	1000	0.09	2.85
pr152	2	3	1	0.1	1	300	900	0.01	46.07
pr152	2	3	1	0.1	0.75	300	900	0.01	37.35
pr152	2	3	1	0.1	0.5	150	900	0.02	9.12
pr152	2	4	1	0.1	0.75	100	900	0.04	6.14
pr152	2	3	1	0.1	0.75	100	700	0.05	5.16
pr152	2	4	1	0.1	0.5	100	900	0.06	4.93
pr152	2	4	1	0.1	0.75	100	700	0.06	4.89
pr152	2	4	1	0.1	0.5	100	700	0.09	4.14
pr152	2	3	1	0.1	0.5	100	700	0.10	4.04
pr76	2	3	1	0.25	1	800	800	0.00	145.81
pr76	2	3	1	0.1	1	800	800	0.00	132.06
pr76	2	2	1	0.1	1	600	400	0.01	38.49
pr76	2	4	1	0.25	1	800	150	0.01	26.21
pr76	2	4	1	0.1	0.5	600	400	0.01	20.45
pr76	2	2	1	0.1	0.75	100	700	0.01	3.33
pr76	2	2	1	0.1	0.5	100	500	0.01	1.86
pr76	2	4	1	0.1	1	50	800	0.01	1.83
pr76	2	3	1	0.1	0.5	50	800	0.01	1.38
pr76	2	4	1	0.1	0.5	100	400	0.02	1.36

pr76	2	3	1	0.1	0.25	50	800	0.02	1.13
pr76	2	4	1	0.1	0.75	50	400	0.03	0.79
pr76	2	2	1	0.1	0.5	50	400	0.03	0.74
pr76	2	4	1	0.1	0.5	50	400	0.04	0.66
pr76	2	4	1	0.1	0.75	50	300	0.04	0.59
pr76	2	4	1	0.1	0.5	50	300	0.06	0.48
qa194	2	4	1	0.1	0.5	600	900	0.05	116.03
qa194	2	2	1	0.1	0.75	200	900	0.05	24.19
qa194	2	3	1	0.1	0.75	200	900	0.06	23.08
qa194	2	4	1	0.1	0.75	200	900	0.07	21.78
qa194	2	4	1	0.1	0.5	200	1000	0.08	18.03
qa194	2	4	1	0.1	0.5	200	900	0.09	16.33
qa194	2	4	1	0.1	0.75	200	600	0.09	14.61
qa194	2	4	1	0.1	0.75	100	900	0.10	7.59
rat99	2	4	1	0.25	1	100	900	0.02	5.98
rat99	2	4	1	0.1	0.5	150	600	0.03	4.05
rat99	2	4	1	0.1	0.5	100	700	0.05	2.76
rat99	2	4	1	0.1	0.5	100	600	0.06	2.36
rat99	2	4	1	0.1	0.5	50	900	0.07	1.66
rat99	2	4	1	0.1	0.5	50	700	0.07	1.29
rat99	2	4	1	0.1	0.5	50	600	0.10	1.09
rd100	2	4	1	0.1	1	100	900	0.02	5.51
rd100	2	4	1	0.1	0.5	50	1000	0.03	1.86
rd100	2	4	1	0.1	0.5	50	900	0.06	1.68
rd100	2	4	1	0.1	0.75	50	600	0.07	1.42
rd100	2	4	1	0.1	0.75	50	500	0.09	1.12
st70	2	4	1	0.5	1	50	700	0.01	1.95
st70	2	4	1	0.1	1	50	700	0.01	1.62
st70	2	4	1	0.1	0.5	50	700	0.02	1.12
st70	2	2	1	0.1	0.5	50	600	0.02	1.06
st70	2	4	1	0.1	0.5	50	600	0.02	0.94
st70	2	4	1	0.1	0.75	50	300	0.04	0.60
st70	2	4	1	0.1	0.5	50	300	0.06	0.48
st70	2	4	1	0.1	0.75	50	200	0.07	0.40
swiss42	2	2	1	0.1	1	50	150	0.00	1.52
swiss42	2	3	1	0.1	0.5	50	150	0.00	0.92
swiss42	2	2	1	0.1	0.25	50	200	0.01	0.80
swiss42	2	3	1	0.1	0.25	50	150	0.03	0.60
swiss42	2	4	1	0.1	0.25	50	150	0.10	0.57
u159	2	4	1	0.25	0.9	200	900	0.03	25.45
u159	2	4	1	0.25	0.9	200	800	0.03	20.83
u159	2	4	1	0.1	0.75	200	800	0.04	16.44
u159	2	4	1	0.1	0.5	200	800	0.07	11.97
u159	2	2	1	0.1	0.75	100	900	0.07	7.25

u159	2	4	1	0.1	0.75	100	900	0.08	6.40
u159	2	2	1	0.1	0.5	100	900	0.10	5.87
xqf131	2	4	1	0.1	0.75	800	400	0.03	89.18
xqf131	2	4	1	0.1	0.5	900	300	0.03	56.15
xqf131	2	3	1	0.1	0.9	200	900	0.04	18.41
xqf131	2	4	1	0.1	0.75	100	900	0.04	5.66
xqf131	2	4	1	0.1	0.5	100	900	0.05	4.41
xqf131	2	2	1	0.25	0.75	50	1000	0.07	3.97
xqf131	2	4	1	0.1	0.75	50	1000	0.08	2.68
xqf131	2	4	1	0.1	0.75	50	900	0.08	2.43

Appendix 3 – Settings for TPS problems with 0-50 nodes (small-sized problems)

The results of the greedy heuristic are given for small-sized problems in the table below. *s*, *c*, *m*, *pc*, *pm*, *ps*, *ng* indicates selection operator, crossover operator, mutation operator, probability of crossover, probability of mutation, population size, number of generations, respectively. The results of operators and parameters are given according to index numbers in Table 1. Mean and Std show average and standard deviation according to the number of problems it solves in each trial. Min and Max show the minimum and maximum number of problems solved in the trials. Number of runs all solved shows how many experiments it solves all the problems.

Table 19 All final settings for TPS problems with 0-50 nodes

Index	***s***	***c***	***m***	***pc***	***pm***	***ps***	***ng***	**Mean**	**Std**	**Min**	**Max**	**Number of runs all solved**
1	2	3	1	1	4	1	1	3.3	0.73	2	4	0
2	2	3	1	1	3	1	1	3.2	0.70	2	4	0
3	2	2	1	1	3	1	1	3.2	0.77	1	4	0
4	2	4	1	1	3	1	1	3.15	0.75	1	4	0
5	2	2	1	2	2	1	1	2.1	0.64	1	3	0
6	2	3	1	1	2	1	1	1.95	0.60	1	3	0
7	2	2	1	1	2	1	1	1.9	0.45	1	3	0
8	2	4	1	1	2	1	1	2	0.56	1	3	0
9	2	4	1	1	1	1	1	1.35	0.59	0	2	0
10	2	4	1	1	3	1	4	6.35	0.81	5	7	11
11	2	4	1	1	3	1	3	5.5	0.69	4	6	0
12	2	1	1	1	3	1	2	4.3	0.47	4	5	0
13	2	4	1	1	2	1	3	4.05	0.22	4	5	0
14	2	4	1	1	4	1	2	5.05	0.76	4	7	1
15	2	4	1	1	3	1	2	4.5	0.76	4	6	0
16	2	4	1	1	2	1	2	3.65	0.49	3	4	0
17	2	2	1	1	1	1	1	1.25	0.64	0	2	0
18	2	2	1	1	6	1	3	6.1	0.72	5	7	6
19	2	3	1	1	3	1	3	5.95	0.83	4	7	5
20	2	2	1	1	2	1	4	5.2	1.06	3	7	2
21	2	3	1	1	2	1	3	4.05	0.60	3	5	0

Appendix 4 – Settings for TPS problems with 50-100 nodes (medium-sized problems)

The results of the greedy heuristic are given for medium sized- problems in Table 20 below. The structure of the table is same as the one given in Appendix 3.

Table 20 All final settings for TSP problems with 50-100 nodes

Index	*s*	*c*	*m*	*pc*	*pm*	*ps*	*ng*	Mean	Std	Min	Max	Number of runs all solved
1	2	4	1	1	3	1	8	2.7	0.98	1	4	0
2	2	4	1	2	3	1	5	0.8	0.41	0	1	0
3	2	4	1	1	5	1	4	0.35	0.49	0	1	0
4	2	4	1	1	3	1	4	0.35	0.49	0	1	0
5	2	3	1	1	3	1	4	0.35	0.49	0	1	0
6	2	4	1	1	2	1	4	0	0.00	0	0	0
7	2	3	1	1	3	1	3	0.15	0.37	0	1	0
8	2	4	1	1	3	1	3	0	0.00	0	0	0
9	2	4	1	1	4	1	2	0	0.00	0	0	0
10	2	3	1	2	6	12	9	9.35	0.88	7	10	11
11	2	3	1	2	3	12	11	8.95	1.19	6	10	9
12	2	3	1	5	3	10	9	8.5	1.05	7	10	4
13	2	3	1	5	4	2	9	7.85	1.04	5	9	0
14	2	3	1	5	3	2	9	7.15	1.35	5	9	0
15	2	4	1	5	3	1	9	3.35	1.31	0	6	0
16	2	4	1	3	3	1	9	3.35	1.14	1	5	0
17	2	3	1	1	3	1	11	5.1	1.37	3	9	0
18	2	4	1	1	3	1	9	3.05	1.23	1	6	0
19	2	4	1	1	4	1	6	2.3	0.86	1	4	0
20	2	4	1	1	3	1	6	1.2	0.89	0	3	0
21	2	4	1	1	4	1	5	1.35	0.81	0	3	0
22	2	4	1	1	3	1	5	0.75	0.79	0	3	0
23	2	2	1	2	4	10	11	9.6	0.50	9	10	12
24	2	4	1	1	6	10	4	9.5	0.69	8	10	12
25	2	4	1	6	4	3	11	7.55	1.15	5	10	1
26	2	4	1	1	5	3	11	8.65	1.18	7	10	7
27	2	4	1	1	5	3	7	8.5	1.00	6	10	3
28	2	4	1	2	4	1	11	6.65	1.23	5	10	1
29	2	3	1	1	4	1	11	6.85	1.63	4	9	0
30	2	4	1	1	3	2	12	6.9	1.48	4	10	1
31	2	4	1	1	4	1	12	7.1	1.17	5	9	0
32	2	4	1	1	3	1	12	5.65	1.46	4	9	0

33	2	3	1	1	5	2	12	8.5	1.10	7	10	5
34	2	3	1	1	5	2	7	7.25	1.12	5	10	1
35	2	4	1	2	5	1	12	6.75	1.33	4	9	0
36	2	4	1	2	3	1	12	5.35	1.46	3	8	0
37	2	4	1	1	3	1	11	5.05	1.32	3	8	0
38	2	4	1	1	5	9	4	8.2	1.15	6	10	2
39	2	4	1	1	3	9	4	7.3	1.49	5	10	1
40	2	4	1	1	6	2	12	9.1	0.91	7	10	7
41	2	4	1	1	4	2	12	7.75	1.68	4	10	3
42	2	4	1	1	4	1	9	5.65	1.09	4	8	0
43	2	3	1	2	6	10	10	9.55	0.76	7	10	13
44	2	3	1	1	6	10	10	9.45	0.76	7	10	11
45	2	2	1	1	6	8	6	9.45	0.83	8	10	13
46	2	4	1	2	6	10	3	7.4	1.10	6	9	0
47	2	4	1	1	3	8	6	7.55	1.05	6	9	0
48	2	2	1	1	4	2	9	7.5	1.43	4	9	0
49	2	2	1	1	3	2	7	4.5	1.28	2	7	0
50	2	4	1	1	6	1	10	4.25	0.97	2	6	0
51	2	3	1	1	3	1	10	3.85	1.14	1	6	0
52	2	4	1	1	3	2	6	3.95	1.05	2	6	0
53	2	3	1	1	2	1	10	1.8	0.89	0	3	0
54	2	2	1	1	3	1	6	1.75	1.07	0	4	0
55	2	4	1	2	6	2	11	8.95	0.89	7	10	6
56	2	4	1	1	3	3	8	7.1	1.25	5	9	0
57	2	4	1	1	3	2	9	6.7	1.45	4	10	1
58	2	4	1	1	3	2	8	5.7	1.26	4	8	0
59	2	4	1	1	6	2	11	9.5	0.83	7	10	13
60	2	4	1	1	4	1	8	3.65	1.18	1	5	0
61	2	4	1	1	4	1	7	2.95	0.69	2	4	0
62	2	4	1	3	6	1	9	2.75	0.85	1	4	0
63	2	4	1	1	6	1	9	3.6	0.88	1	5	0
64	2	2	1	1	3	1	8	3.15	0.81	1	4	0
65	2	4	1	1	4	1	4	0.4	0.50	0	1	0

Appendix 5 – Settings for TPS problems with 100-200 nodes (large-sized problems)

The results of the greedy heuristic are given for large-sized problems in Table 21 below. The structure of the table is same as the one given in Appendix 3.

Table 21 All final settings for TSP problems with 100-200 nodes

Index	***s***	***c***	***m***	***pc***	***pm***	***ps***	***ng***	**Mean**	**Std**	**Min**	**Max**	**Number of runs all solved**
1	2	4	1	1	3	4	10	7.55	1.76	5	11	0
2	2	4	1	1	4	2	11	6.55	1.67	3	9	0
3	2	4	1	1	4	2	10	5.6	1.05	4	7	0
4	2	4	1	1	4	2	8	3.5	1.05	2	6	0
5	2	4	1	1	4	1	10	1.4	0.88	0	3	0
6	2	4	1	1	4	5	11	10.3	1.69	6	13	0
7	2	4	1	1	4	4	11	9.3	1.66	7	12	0
8	2	3	1	1	4	3	11	8.75	1.12	6	10	0
9	2	4	1	1	4	3	11	8.9	1.41	7	12	0
10	2	4	1	1	4	3	10	8.45	1.79	6	12	0
11	2	4	1	1	6	12	5	10	1.34	7	13	0
12	2	4	1	1	4	12	5	9.8	1.61	7	12	0
13	2	3	1	1	6	12	4	5.95	1.15	4	8	0
14	2	4	1	1	3	1	11	0.85	0.88	0	2	0
15	2	4	1	1	3	1	9	0.25	0.44	0	1	0
16	2	4	1	1	4	1	7	0.1	0.31	0	1	0
17	2	3	1	3	2	8	11	9.9	1.45	8	12	0
18	2	4	1	1	4	6	12	10.45	1.23	8	13	0
19	2	4	1	1	3	6	10	9.7	1.42	7	12	0
20	2	4	1	1	2	6	12	8.1	1.45	5	11	0
21	2	4	1	1	3	3	12	8	1.41	6	11	0
22	2	4	1	1	3	2	12	5	1.38	2	8	0
23	2	4	1	1	3	2	10	2.7	1.34	1	5	0
24	2	3	1	1	3	9	11	10.7	1.45	7	14	1
25	2	4	1	1	5	5	11	10.8	1.67	8	13	0
26	2	4	1	1	3	4	11	8.25	1.94	4	11	0
27	2	4	1	2	5	12	9	11.75	1.16	10	14	1
28	2	4	1	1	4	4	10	9.15	1.60	6	13	0
29	2	4	1	1	3	4	9	6.75	1.29	4	9	0
30	2	3	1	1	6	3	10	9.25	1.41	5	11	0
31	2	4	1	1	3	3	10	6.2	1.79	1	8	0
32	2	4	1	1	3	2	9	2.15	0.81	1	4	0
33	2	4	1	1	3	1	10	0.6	0.68	0	2	0
34	2	3	1	1	3	1	9	0.25	0.55	0	2	0

35	2	4	1	5	3	10	8	10.1	1.77	6	13	0
36	2	4	1	1	6	6	8	11.5	1.00	10	13	0
37	2	4	1	2	6	6	6	7.85	1.60	5	11	0
38	2	4	1	1	6	6	6	8.25	1.62	5	11	0
39	2	4	1	1	3	4	8	4.9	1.77	2	8	0
40	2	2	1	1	5	1	12	1.9	1.07	0	4	0
41	2	4	1	1	5	1	12	1.8	0.77	1	3	0
42	2	4	1	1	3	1	8	0.05	0.22	0	1	0
43	2	3	1	1	5	9	11	12.05	1.32	9	14	1
44	2	4	1	1	3	11	5	7.2	1.70	3	10	0
45	2	4	1	5	3	4	10	7.55	1.15	5	10	0
46	2	4	1	1	3	2	11	3.65	1.31	1	7	0
47	2	4	1	1	4	1	11	1.85	0.93	0	3	0
48	2	4	1	2	4	10	10	10.6	1.64	7	14	1
49	2	3	1	2	1	10	10	7.7	1.63	4	10	0
50	2	4	1	1	4	4	9	9.55	1.57	6	12	0
51	2	4	1	1	4	2	12	7.65	1.60	5	10	0
52	2	3	1	1	4	2	9	4.85	1.14	3	7	0
53	2	4	1	1	4	2	9	4.15	1.14	2	6	0
54	2	4	1	1	4	1	12	2	1.08	1	4	0
55	2	3	1	1	6	5	11	12.3	1.38	9	14	5
56	2	3	1	1	4	5	11	10.2	1.20	8	12	0
57	2	3	1	1	3	3	11	6.9	1.83	4	10	0
58	2	3	1	1	3	2	9	2.2	0.89	0	4	0
59	2	4	1	1	3	8	11	10.45	1.73	7	13	0
60	2	2	1	1	4	4	11	9.9	1.37	7	12	0
61	2	3	1	1	4	4	11	9.55	1.57	6	12	0
62	2	4	1	1	3	4	12	8.45	1.61	5	11	0
63	2	4	1	1	4	4	8	7.45	1.67	5	11	0
64	2	4	1	2	5	4	11	10.6	1.27	9	13	0
65	2	4	1	2	5	4	10	10.35	0.99	8	12	0
66	2	2	1	1	4	2	11	6.75	1.07	5	9	0
67	2	2	1	1	3	2	11	3.7	1.38	2	6	0
68	2	4	1	1	4	10	6	10.35	1.46	8	13	0
69	2	3	1	1	5	4	11	10.4	1.64	7	14	1
70	2	2	1	2	4	1	12	1.9	1.02	1	4	0