Sequence Based Memetic Algorithms for Static and Dynamic Travelling Salesman Problems

by

Shakeel Arshad

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the

Department of Computer Science
University of Leicester

April 2012

Declaration of Authorship

The content of this submission was undertaken in the Department of Computer Science, University of Leicester, and supervised by Dr. Shengxiang Yang during the period of registration. I hereby declare that the materials of this submission have not previously been published for a degree or diploma at any other university or institute. All the materials submitted for assessment are from my own research, except the reference work in any format by other authors, which are properly acknowledged in the thesis.

Part of the research work presented in this submission has been published, or has been submitted for publication in the following papers:

1. A two-phase hybrid algorithm for the travelling salesman problem. To be submitted for journal publication.
2. A guided two-phase memetic algorithm for the dynamic travelling salesman problem. To be submitted for journal publication.
3. S. Arshad and S. Yang. A hybrid genetic algorithm and inver over approach for the travelling salesman problem. *Proceedings of the 2010 IEEE Congress on Evolutionary Computation*, pages 252–259, 2010. IEEE Press.
4. S. Arshad, S. Yang, and C. Li. A sequence based genetic algorithm with local search for the travelling salesman problem. *Proceedings of the 2009 UK Workshop on Computational Intelligence*, pages 98–105, 2009.

Sequence Based Memetic Algorithms for Static and Dynamic TSP

Shakeel Arshad

Abstract

Hybridization of genetic algorithms (GAs) with local search techniques has received significant attention in recent years and is being widely used to solve real-world problems. These hybrid GAs, also called memetic algorithms (MAs), are able to incorporate other powerful techniques within the framework of GAs, working as a single unit and counterbalancing each other's disadvantages.

In this thesis, we propose a hybrid GA, called Sequence Based Memetic Algorithm (SBMA) with Inver Over (IO), for solving the travelling salesman problem (TSP). This is a 2-phase MA. The first phase (SBMA) consists of traditional binary operators, and the second phase is based on a unary operator. In SBMA, a tour is split into equal sub-tours. Further, the shortest tour is selected among the sub-tours and then optimized locally. The sub-tours are stored in the memory and then used to guide the evolutionary process via a kind of embedded local search. Additionally, we apply some techniques to adapt the key parameters based on whether the best individual within the population improves or not while also maintaining the diversity. After the first phase, the hybrid algorithm enters the second phase which is the Inver Over with elite population. Here, the IO is directed to get a clue from the elite population by adding and preserving good edges.

We have also shown that the above approach can be extended to handle the dynamic TSP. The framework of our hybrid approach, along with the integration of the nearest neighbour list, applying 2-Opt local search on sub-tours and adaptive parameter control in the first phase, and the elite population with the rotating gene pool strategies in the second phase, works well for the DTSP. In order to test the performance of the proposed approach for the DTSP, experiments were carried out based on different DTSP test beds. From the study, it has been observed that the integrated heuristics or meta-heuristics are able to produce good-quality solutions for the DTSP. We also analysed the effect of the gene pool and immigrants generated with the nearest neighbour algorithm, which works well with all DTSP test instances, under different dynamics.

Acknowledgements

In the Name of ALLAH, the Most Gracious, the Most Merciful. May ALLAH shower His countless blessings and peace upon all His messengers sent from time to time for the guidance of humankind and, in particular, the last prophet Hazrat Muhammad (SAS!), who has always been and will remain, the fountain head of inspiration and guidance in all walks of life for all times and climes be it terrestrial or celestial.

I would like to pay thanks to all those who made this thesis, in particular, and my stay in Leicester, in general, a success. The foremost among these has been my thesis supervisor, Dr. Shengxiang Yang. It has been an immense privilege to work under the dynamic and consistent supervision of the scholar of his intellectual calibre and renowned repute. I must extend my gratitude to Dr. Shengxiang Yang for his continuing support, sympathetic patience, vibrant enthusiasm, and analytical thinking. His research insight and distrust for gratuitous abstractions have enormously influenced my mental growth and development, and made my creative ideas actualized. His caring, sharing and understanding nature helped me out to get through the circumstances, even when I was slightly discouraged pursuing the completion of my study. In spite of his several commitments, he has always been generous with his time for me.

I am greatly indebted to all the honourable staff members of the Department of Computer Science, University of Leicester, for their unconditional support and help. I would also like to mention the names of those whose rich contributions and willing generosity played a pivotal role in the achievement of my success. I wish to express my cordial appreciation to all the faculty members, especially, Prof. Rick Thomas, and Dr. Fer-Jan de Vries, for their moral and technical support, outcome-oriented advice, and ardent interest in assessing my yearly reports, presented to them, from time to time.

I wish to acknowledge Prof. Rick Thomas, the senior most pillar of the Computer Science Department, for his creative ideas, valuable suggestions and worthwhile discussions throughout my studies in Leicester. Once again my special thanks to Prof. Rick Thomas for his compassion and reassurance by putting me on the right track during my hard time of research. I cherish a desire that may Allah give me a strength to be an outstanding teacher like Rick Thomas. I also admit that initially,

I faced problems, during my stay in the Department of Computer Science, but the credit goes to my post graduate tutor Fer-Jan, who has been a mainstay across this period.

Certainly, this work would have rarely been possible without the help of my family. I put across my profound indebtedness to my loving parents for their candid reinforcement, incessant prayers and fervent dreams for my success, all my life. I would not forget to express my heartfelt thankfulness to my beloved wife and our charming son, Rayyan. I am thankful to my affectionate brother, caring sisters, and my precious friends (specially my G5 group) who have always been a pleasant push.

I am enormously obliged to my uncles and all those personalities who shouldered me to see this day. I am thankful to my cousins who have been helpful during my stay at Leicester in UK, and I did not find myself alone.

This thesis is blessedly dedicated to the poignant memory of my benevolent grandfather (Nanaji) and my both grandmothers. May Allah keep their souls in eternal rest and peace in Heaven! Ameen!

My heartiest contentment to acknowledge the University of Malakand, Chakdarda (Lower Dir), Khyber Pakhtunkhwa, and the Higher Education Commission (HEC) of Pakistan, for fully sponsoring me to pursue my Ph.D. studies at Leicester in UK.

I continue to pay my gratitude to all the administrative and supporting staff of the Computer Science Department at the University of Leicester, especially to John Landamore, Gavin Hornsey, Philip Shine and Richard Grant, for their unremitting technical help during this study span.

I committedly aspire to express my appreciation of all my revered teachers, from Primary to the Ph.D. levels of my education, for their indelible enlightened imprints they left on my person-continually-recurrently. I desire to carry on my gratefulness to all my fellow research scholars for their support and help. Let me reveal my priceless thanks to all my Ph.D. colleagues and friends, with whom, I spent precious time during my study at Leicester. Thanks to their open-mindedness and large-heartedness. Last but not the least, my sincere acknowledgements to Changhe Li, Sadaf Naseem Jat and the EA-Team for walking with me through all times.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

ABC	**A**rtificial **B**ee **C**olony
ACO	**A**nt **C**olony **O**ptimization
APX	**A**lternate **P**osition **X**over
ATSP	**A**symmetric **T**ravelling **S**alesman **P**roblem
B&B	**B**ranch & **B**ound
B&C	**B**ranch & **C**ut
BC	**B**ase **C**riteria
COP	**C**ombinatorial **O**ptimization **P**roblem
CH	**C**hristofides **H**euristics
CX	**C**yclic **X**over
DTSP	**D**ynamic **T**ravelling **S**alesman Problem
DM	**D**isplacement **M**utation
EA	**E**volutionary **A**lgorithm
EC	**E**volutionary **C**omputation
EP	**E**lite **P**opulation
GA	**G**enetic **A**lgorithm
GLS	**G**enetic **L**ocal **S**earch
GP	**G**ene **P**ool
HGA	**H**ybrid **G**enetic **A**lgorithm
IO	**I**nver **O**ver
LK	**L**in **K**ernighan
LS	**L**ocal **S**earch

MA	**M**emetic **A**lgorithm
MAOS	**M**ulti-**A**gent **O**ptimization **S**ystem
MST	**M**inimum **S**panning **T**ree
NN	**N**earest **N**eighbour
OX	**O**rder **X**over
PMX	**P**artially **M**apped **X**over
RI	**R**andom **I**mmigrants
SA	**S**imulated **A**nnealing
SBMA	**S**equence **B**ased **M**emetic **A**lgorithm
SBOX	**S**equence **B**ased **O**rder **X**over
SBIM	**S**equence **B**ased **I**nversion **M**utation
SBLS	**S**equence **B**ased **L**ocal **S**earch
SIM	**S**imple **I**nversion **M**utation
STSP	**S**ymmetric **T**ravelling **S**alesman **P**roblem
TS	**T**abu **S**earch
TSP	**T**ravelling **S**alesman **P**roblem.
VDNS	**V**ariable **D**epth **N**eighbourhood **S**earch
VNS	**V**ariable **N**eighbourhood **S**earch
VI	**V**ertex **I**nsertion

Symbols

i_a, i_b	*individual a, individual b.*
F_{insert}	Frequency of insertion in chromosome
S_{sel}	Selected sequence
N_{seq}	Number of sequences
p_c	Crossover probability
p_m	Mutation probability
s	Candidate Solution
n	Total number of nodes in the memory
π	Tour of an instance
d_{ij}	Distance matrix
$f()$	Fitness function
$\mathcal{S}$	Search space
$\mathcal{N}$	Neighbourhood structure
V	Vertex set
E	Edge set
Pop	Population
Pop_{tmp}	Temporary Population
$rand()$	random function
L_{inc}	Increase in length
L_{seq}	Length of the sequence
$popsize$	Population size (number of individuals in the population)
i_{temp}	Partial individual

$e-$	embedded.
Pop_{IO}	Population for Inver Over.
$Elite_{Pop}$	Elite Population.
MA_1	MA Phase-1
MA_2	MA Phase-2
p	probability for IO
t_{us}	Parameter for reducing the size of the sequence
α	Total number of individuals selected for sequence generation
β	Frequency of insertion
γ	Switching parameter for Phase-1 to Phase-2
f	Frequency of change
m	Severity of changes
$r-$	rotating

To
my sweet parents
and my beloved wife
who have been like cool shade
in the noontide of
life.

Chapter 1

Introduction

The Travelling Salesman Problem (TSP) is an example of a combinatorial optimization problem. In general, if a salesman starts a journey in which he visits each city in a given list exactly once, and then returns to the starting point, it is reasonable for him to select the order in which he visits the cities so that the total length of his tour is as small as possible (with respect to some measurement such as time or distance). It would be quite reasonable that, for each pair of cities, the distance between them would be known to him in advance, and so he can know the cost of moving from one city to another. In this case, he would have all the data necessary to find the minimum or maximum distance, but the difficulty for him is how to use this data in order to get the answer. The situation becomes more complex when the static problem changes to the dynamic one. In the Dynamic Travelling Salesman Problem (DTSP), the distances for the salesman change with the passage of time, and he has to update his route according to the change of the cities in the list or the change of the distances among them.

The importance of the TSP comes not only from the wealth of applications. The simple fact is that there are many salesmen clamouring for an algorithm, and some other cases where the mathematical model of the TSP is applicable to an engineering or scientific situation where we want to find an optimal path in a reasonable period of

time. However, the other issue is that TSP is a typical instance of a combinatorial optimisation problem. We are trying to minimize the total distance, and, if we consider the problem in this way, the task is a minimization problem.

As TSP is an NP-complete problem, no efficient algorithm is known, and so heuristics which give a good, but not necessarily optimal, solution are deemed to be sufficient. TSP heuristics typically fall into two groups: tour construction heuristics, which create tours from scratch, and tour improvement heuristics, which use the simple local search heuristics to improve existing tours. A more modern approach is to use a combination of the above two heuristics, which is called "compound", *i.e.*, an approach which first has a constructive phase and then has an improvement phase.

It is a general fact about the TSP that there are many approximate heuristic approaches, along with the exact ones, for solving it. The main aim of these approximate heuristics is to efficiently generate very good solutions. They do not necessarily find the optimal solution; however, these heuristics do have desirable characteristics, such as short running times, ease of implementation, flexibility and simplicity.

1.1 Background and Motivation

The TSP is an easy problem to state; one does not need to have a mathematical background to understand the problem. It is an interesting problem to work on, and it is continuously attracting the attention of researchers, practitioners and problem solvers. On the other hand, the TSP is resistant to all efforts to find a good algorithm which results in an optimal solution. So, the TSP has elements that have attracted mathematicians and computer scientists for centuries, that is, the simplicity of the statement and the difficulty of the solution. Moreover, the TSP remains an intriguing problem and there is no reason to believe that this attraction and curiosity will diminish in the future.

In general, combinatorial optimization problems have a finite number of candidate solutions. An exact approach would be to enumerate all the possible candidate solutions and compare them with each other. However, unfortunately, the search space becomes gigantic for some of the combinatorial optimization problems. For some standard cases, the search space is greater than 10^{9259}, which is huge when compared to the estimated number of 10^{80} atoms in the universe. For handling some such problems, some fast exact algorithms exist which are much faster than exhaustive search, and these are called algorithms with polynomial running time. At the same time, for some combinatorial problems, it is believed that no such algorithm exists. In recent years, a lot of effort has been put into investigating the performance of such exact algorithms. Even so, the computational complexity of such problems is still an open question. The only current answer to the question of tackling large instances is that approximate heuristic search techniques should be employed. Although these techniques deliver no guarantee of giving an optimal solution, a lot of effort has been made in developing such heuristics which aim to find good quality solutions in a short period of time.

In general, many modern heuristics are based on neighbourhood searching mechanisms, such as various types of local search, tabu search and simulated annealing, and nature-inspired algorithms, such as genetic algorithms, bee systems, ant colony optimization, and neural networks. However, there is another possibility, which is to combine two or more heuristics. These heuristics are termed as hybrids or memetic algorithms. Memetic algorithms are the combination of genetic algorithms and local search, or sometimes the combination of local search operators in which each local search works on a distinct neighbourhood structure. In recent years, researchers and practitioners have focused on the embedding of domain-specific knowledge into genetic algorithms in order to move the global search in a guided direction by avoiding moving randomly in the search space. These guided heuristics have been tested on problems such as TSPs in comparison with alternative approaches. They have proved to be more effective with respect to the solution quality and time.

In this thesis, we do not claim that we have found the best heuristics ever developed, because it is a common belief that there is no such algorithm, *i.e.*, one that is superior to all other heuristic approaches on the set of all problems. In the two-phase approach which we propose here, we try to answer the following important questions. Why is the performance of one approach stand-alone poor, but good when it works with another approach, even when the second approach also performs poorly when used by itself? What are the ways to bridge different heuristics which work on totally dissimilar working mechanisms and various environmental parameters? How do we cope with major issues such as premature convergence, maintaining diversity and keeping the candidate solution as flexible as possible for moving around in the search space?

This thesis is an attempt to investigate the above questions. We have chosen memetic algorithms as our object of study. The test bed we have chosen is the classical problem that is the Travelling Salesman Problem. The span of our study is further extended to the dynamic version of TSP, which is more realistic in modelling real-world problems. The main motivation for choosing memetic algorithms is that they are hybrids: domain-specific knowledge can be incorporated into the problem-independent algorithms. Additionally, memetic algorithms exploit the symbiotic effects of the combination of two or more memes (different search techniques).

In this thesis, we aim to combine the good properties of local and global search techniques for solving the static and dynamic TSPs. We have merged together the binary genetic algorithm with the unary genetic algorithm with an embedded LS technique. In addition, we have introduced some adaptive parameter control for bridging the two techniques. We have also extended our approaches for handling the static TSP to the dynamic TSP. For the DTSP, to respond swiftly to changes, we introduce a rotating gene pool, which quickly inserts good edges by directing the search space to take intelligent moves instead of random moves.

1.2 Aims and Objectives

This thesis investigates GAs to solve the static TSP and DTSP. The overall aim of this thesis is to develop and apply GAs to produce solutions of good quality for solving the static TSP as well as for solving the DTSP. In order to carry out this primary aim, some objectives are outlined as follows:

1. To design new local search operators or memetic algorithms for the TSP and DTSP.

2. To learn the concept of the local search techniques in general and the neighbourhood structure in particular, understand the challenges behind these problems and the issues which arise during solving these problems, and review relevant approaches that researchers have developed to solve these problems.

3. To carry out an experimental analysis of the effectiveness and efficiency of traditional GAs and memetic algorithms for solving the TSP and extending it to the DTSP.

4. To investigate local search techniques to enhance the exploitation and exploration ability of GAs for the TSP and DTSP and develop memetic algorithms for the TSP and DTSP which are composed of several heuristic approaches. One possibility is that one heuristic could tackle one issue, and the other may tackle another issue, which may arise during the searching process.

5. To develop multiple phase hybrid GAs or MAs that integrate GAs and LS strategies and other heuristics to solve the TSP and DTSP. Hybrid GAs or MAs should be efficient, *i.e.,* they should be capable of producing acceptable solutions in a short time, because hybrid GAs or MAs are composed of extra local search routines within the existing framework of GAs (most of the total CPU time is generally spent in the local search procedure).

6. To implement the above developed GAs and relevant algorithms for the TSP and DTSP, respectively, using a programming language, and carry out a systematic experimental study based on the implemented algorithms, such as checking the performance of the key parameters of memetic algorithms for improving the performance of GAs for the TSP and DTSP.

7. To present the efficiency of a solution methodology that is generic, robust and able to produce good solutions, when compared against the state-of-the-art methods.

1.3 Methodology

The TSP has a long history of attracting the interest of both researchers and practitioners. The TSP is, to put it simply, "easy to state" but "difficult to solve". In this study, we aim to develop an efficient approach to be merged together with traditional GAs to solve the TSP. Various step-by-step improvements have been proposed for increasing the performance, and getting good solutions in a short time. In general, most of the powerful GAs which have been proposed for tackling the TSP have used many robust heuristic and meta-heuristics. For example, the performance has been increased by utilizing intense LS operations after the crossover and mutation operations. The absence of strong LS techniques raises a question in the overall performance of GAs. In our proposed approach, we consider $\sqrt{n}$ nodes of a TSP instance, where n is the number of cities/nodes in the instance, by splitting the candidate solutions into equal parts and then storing them as a sequence and using it to guide GAs. Additionally, some procedures are applied to maintain the diversity by reducing the size of the selected sequences (by removing some nodes) into sub-tours if the best individual of the population does not improve.

Initially, for developing the solution approach, the basic order crossover operator with the inversion operator is used, and then the greedy local search technique is

used by inserting the stored sub-sequence into the partial individual resulting in a complete individual. During the experimental analysis, it has been observed that the proposed approach has a good convergence speed, but the technique is very expensive computationally. To avoid this disadvantage, some other approaches are merged to balance the convergence and computational time. For the same reason, the existing approach is merged with another operator or another MA; so the current approach is converted into the two-phase genetic algorithm or memetic algorithm. In this, the first phase handles the fast convergence and, if the first phase is unable to bring further gain in improvement, the algorithm is shifted to the second phase. So we will start our study by studying the basic MA, which is binary; this simply takes two individuals and performs the evolutionary process. Then we will add other refinement techniques to enhance the performance of the proposed MA for solving different TSP problems. The refinement techniques which are utilized in the first phase are: bringing the nodes near each other by using a Nearest Neighbour list, applying the simple 2-opt LS on the extracted sub-tours, and, after further breaking, *i.e.,* reducing the total size of a sequence, increasing and decreasing the intensity of local search and adaptively increasing the crossover and mutation probabilities. For the second phase, the performance has been enhanced by making the fast unary Inver Over operator guided. By default, the Inver Over operator is blind, getting the clue from an individual which might be better or worse in fitness (the edge which is added does not guarantee that the result will be an optimal one). The proposed enhancements for the second phase are the integration of the *Elite* population approach, forcing the IO to accept those inversions which contribute to a fitness gain and the rotating gene pool. We have extended the current two-phase MA to the DTSP. As our former approach works better in solving static TSP, and gives near optimal solutions regarding quality and time, we have just introduced the change operator, which changes the static TSP to the dynamic TSP. The change function, as mentioned above, introduces new nodes and deletes old nodes; this changes the existing edges in the problem at hand. The change function takes the optimal state of the problem to a non-optimal state. For tackling the dynamic TSP, the procedure

is the same as that mentioned above except that we introduced the new approach which is integrated with the main IO, namely the gene pool approach generated by using the Minimum Spanning Tree (MST). By addressing the issue of diversity and by making use of the strength of gene pool, it is rotated when the IO picks a city from the existing gene pool. So, for the next iteration, if the same node is under consideration, we would have a new city to be picked from the gene pool. Furthermore, the switching criterion from Phase-1 to Phase-2 is modified slightly, keeping with the view of the dynamic TSP.

In the field of evolutionary computation, it is very hard to give any analysis of the performance of GA/MA like convergence and optimality of the TSP because they are stochastic in nature. We therefore cannot give a formal analysis of a GA/MA for the TSP. In order to investigate the performance of the proposed hybrid approach for the TSP, experiments are carried out on benchmarks, to compare it with other relevant algorithms on a set of small and medium benchmark TSP instances. Experimental results show that proposed hybrid approach is superior to the IO algorithm and some other state of the art algorithms in terms of the convergence speed, as well as solution quality and time.

1.4 Scientific Contributions

Different enhancement techniques are investigated in this thesis for the improvement of the overall performance of GA/MA, such as maintaining diversity, speeding up the convergence, and reducing the computational time. Various experimental results raise many issues and new directions for producing better methods regarding the solution quality and computational time.

From our studies and analysis, the following scientific contributions are made in this thesis:

1. The experimental analysis shows that the standard Genetic Algorithm often suffers from slow convergence for solving combinatorial optimization problems. In this study, we introduce a sequence based memetic algorithm (SBMA) for the symmetric travelling salesman problem (TSP). In this method, a set of sequences are extracted from good individuals which are used to guide the search of SBMA. Additionally, some procedures are applied to maintain the diversity by breaking the selected sequences into sub-tours if the best individual of the population does not improve or if the MA gets stuck in local optima.

2. A new strategy is introduced, which proposes a two-phase hybrid approach for the travelling salesman problem (TSP). The first phase is based on a sequence based memetic algorithm (SBMA) which is mentioned above, but we integrated an embedded local search scheme. Within the SBMA, a memory is introduced to store better sequences (sub-tours) extracted from previous good solutions, and the stored sequences are used to guide the generation of offspring via LS during the evolution of the population. Additionally, we also apply some techniques to adapt the key parameters based on whether the best individual of the population improves or not and maintain the diversity. After SBMA finishes, the hybrid approach enters the second phase, where the inver over (IO) operator, which is a state-of-the-art algorithm for the TSP, is used to further improve the solution quality of the population. The experimental results show that the proposed hybrid approach is efficient in finding good-quality solutions for the test TSPs.

3. We describe a novel extended hybrid MA of our proposed two-phase approach. A number of new ideas are integrated in our MA framework. During the analysis, we have observed that the solution needs to be prevented from inserting expensive (long) edges, *i.e.,* a set of nodes, which are almost near to each other, should not include a long edge. To resolve this issue, initially sub tours are generated for some generations by utilizing the Nearest Neighbour list of a node and then left for the MA to work in a random manner for the rest

of the generations. Additionally, when the number of nodes of the sequence is reduced, it is further optimized by a 2-opt local search; so each time the inserted sequence of nodes is locally optimal. In phase two the IO is guided by getting the clue from the *elite* population which is a fraction of the whole population. Here, before entering to the IO operator, a kind of unique population which is best in fitness is extracted from the main population. Another enhancement is a kind of random immigrant scheme. In this RI, one third of the population, along with previous parents and children of the phase one, is added to maintain diversity for further increasing the exploration capability of phase two. The experimental results show that the proposed hybridization of the binary and unary operators and the enhancement strategies are efficient in finding optimal or near-optimal solutions for the test TSPs.

4. The above two-phase MA is extended to the framework of dynamic TSP. For the DTSP another heuristic approach which is integrated with the main IO is the gene pool approach generated by using the Minimum Spanning Tree. By keeping the issue of diversity, and to make use of the strength of the gene pool, it is rotated when the IO picks a city from the existing gene pool. So, for the next iteration, if the same node is under consideration we would be having a new city to be picked from the gene pool. Furthermore, the switching criteria from Phase-1 to Phase-2 is modified slightly by keeping the view of DTSP. The approach is tested on two different dynamic test beds, *i.e.,* main and spare pool, and by changing the positions of the nodes. In the main and spare pool the TSP instance is split into two equal parts. After the passage of time some new nodes come into the main pool, and some nodes go back out to the spare pool. In the changing node positions, the positions of the nodes are changed slightly. The experiments are performed on various change frequencies and severities.

1.5 Thesis Outline

This thesis consists of eight chapters. Chapter 1 gives the introduction, background and motivation, aims and objectives of the study, and, finally, the general methodology.

Chapter 2 presents the general introduction to the process of optimization, combinatorial optimization problems and natural evolution. We give common terms used in genetics and a brief history of evolutionary computation, and describe hybridization of EAs with other heuristics and various neighbourhood structures.

Chapter 3 contains an account of the history of the TSP. The mathematical treatment of the TSP is also discussed. In addition, various heuristics and meta-heuristics are briefly explained. A brief description about the dynamic TSP is presented as well.

In Chapter 4, a description of our proposed algorithm, that is the Sequence Based Memetic Algorithm, is given. The complete framework of the algorithm is also discussed. The experimental results suggest that SBMA has a good convergence power and can tackle small instances of the TSP.

Chapter 5 is about the extension of our SBMA along with the Inver Over approach. In this chapter we describe some of the deficiencies of SBMA; we also describe how we cope with the integration of another GA; this is the hybrid two-phase genetic algorithm. In this we have bridged together the two MAs with an adaptive parameter control. The span of TSP instances increases from small-scale to medium-scale instances. The algorithm is also compared with state-of-the art algorithms to show the comparative performance.

In Chapter 6, we give a Nearest Neighbour Sequence Based Memetic Algorithm with Elite Population Inver Over for the TSP, which is a further extension of the above approaches. In this study, we also investigate the performance of SBMA+IO

with LS techniques for solving the TSP. We describe some factors that improve the quality of solutions to TSP. This approach suffers from randomness, *i.e.,* sequences may contain long edges when the sequence size is large. To overcome the drawback of the above approach, initially sequences/sub-tours are constructed with the help of the nearest neighbour approach. In this approach $\sqrt{n}$ random nodes are first selected and then from the *NN-list*, $\sqrt{n}$ nodes are selected which are almost near to each other and which are then optimized with 2-Opt for a few generations. Another concept which was missing in the approach mentioned in Chapter 5 is that, after reducing the size of the sequence, no optimization of sequences was considered; here, after the sequence size changes, it is further optimized with 2-Opt. Secondly, we forced the IO to get the clue only from the elite population of the MA, because fit individuals are composed of gene fragments or good edges. We should mention that our approach gets better results than Iterated Local Search with advance 3-Opt, not only in quality but also in less time. When compared to ILS our technique only carries out optimization on less than $\sqrt{n}$ nodes. Experimental results of the sensitivity analysis of key parameters also reveal that reducing the size of sequences not only reduces time but also contributes in giving good quality tours. This technique is represented as "A Guided Two-Phase Hybrid Algorithm for the Travelling Salesman Problem (GTPHA)".

Moving away from static to dynamic TSP, in Chapter 7, we focus our attention on how to extend the SBMA+IO to the DTSP. In this chapter, we present how the current approach is suited to DTSP. The framework of the static algorithm is extended to the dynamic version. The dynamic framework is experimentally validated using several benchmarks created from TSP instances, with the number of nodes ranging from 150 to 724. The experimental analysis shows that the first phase achieves fast convergence and then the algorithm enters the second phase where the guided IO also explores the search space. The integration of the elite population and the rotating gene pool approaches makes the behaviour more intelligent.

Finally, Chapter 8 summarizes this thesis by describing the major outcomes of the

study, such as the contributions made from the research for solving the TSP and dynamic TSP. It also gives some ideas about possible directions for future research.

Chapter 2

Combinatorial Optimization and Evolutionary Algorithms

Optimization is the process of finding the best possible solution(s) to a problem. In mathematics, this often consists of maximizing or minimizing the value of a certain function, perhaps subject to some given constraints. In this world, the one and only primary effort is the search for a favorable and desirable state. Every individual who exists is striving for perfection in every span of life to get a maximum degree of benefits with less effort. In our routine life, everyone cherishes a desire in economy; that profit and sales must be maximized, and costs should be as low as possible. Therefore, optimization is one of the oldest sciences, and this reaches into our daily life.

The main goal of optimization is to find a solution to a given problem, which minimizes or maximizes some measure of *"goodness"*. In general, if we have a fitness function, $f : X \to \mathbb{R}$ over some closed domain X, the goal of optimization is to find a value of $x^* \in X$ which minimizes (or maximizes) the value of f. Such a value of x^* is called a global optimum of the entire search space. In a more general form, if we have an optimization problem at hand, then $x^* \in X$ is the minimum if and only if $f(x^*) \leqslant f(x)$, $\forall\ x \in X$; similarly, $x^* \in X$ is maximum if and only if $f(x^*) \geqslant f(x)$,

$\forall\ x \in X$. It is notable that there is no efficient algorithm to solve the problem of global optimization in the combinatorial domain [81].

In general, global optimization algorithms can be classified into two basic classes: *deterministic* and *probabilistic* algorithms, and further each class can be classified in subclasses as shown in Figure 2.1.

According to [144], deterministic algorithms are widely used if there is a clear relation between the characteristics of possible solutions and their effect on the objective value, or there is a simple way in which all solutions can be evaluated. Then the search space can be efficiently explored using, for example, a divide-and-conquer scheme.

However, it becomes very difficult for a deterministic algorithm when the relation between a candidate solution and its fitness is not clear, or the neighborhood and the dimensionality of the search space is complex. In such cases it turns out to be harder to solve a problem deterministically. Attempting a deterministic solution would often result in exhaustive enumeration of the search space, which is not feasible even for relatively small problems, and for large problems the size of the search space increases exponentially [144].

Probabilistic methods can be used when the search space is gigantic (having a huge number of local optima, or the fitness landscape is very complex) or the dimensionality of the search space is very high. These methods are used to obtain optimal or near-optimal solutions within reasonable time by not doing an exhaustive search like exact algorithms but making guesses that hopefully lead to better solutions. One of the drawbacks of probabilistic algorithms is that, while having a shorter runtime, they cannot guarantee finding the optimal solution. This does not mean that the result obtained by using a probabilistic algorithm is incorrect, but that it can be inferior or have cost slightly above the cost of the global optimum [144].

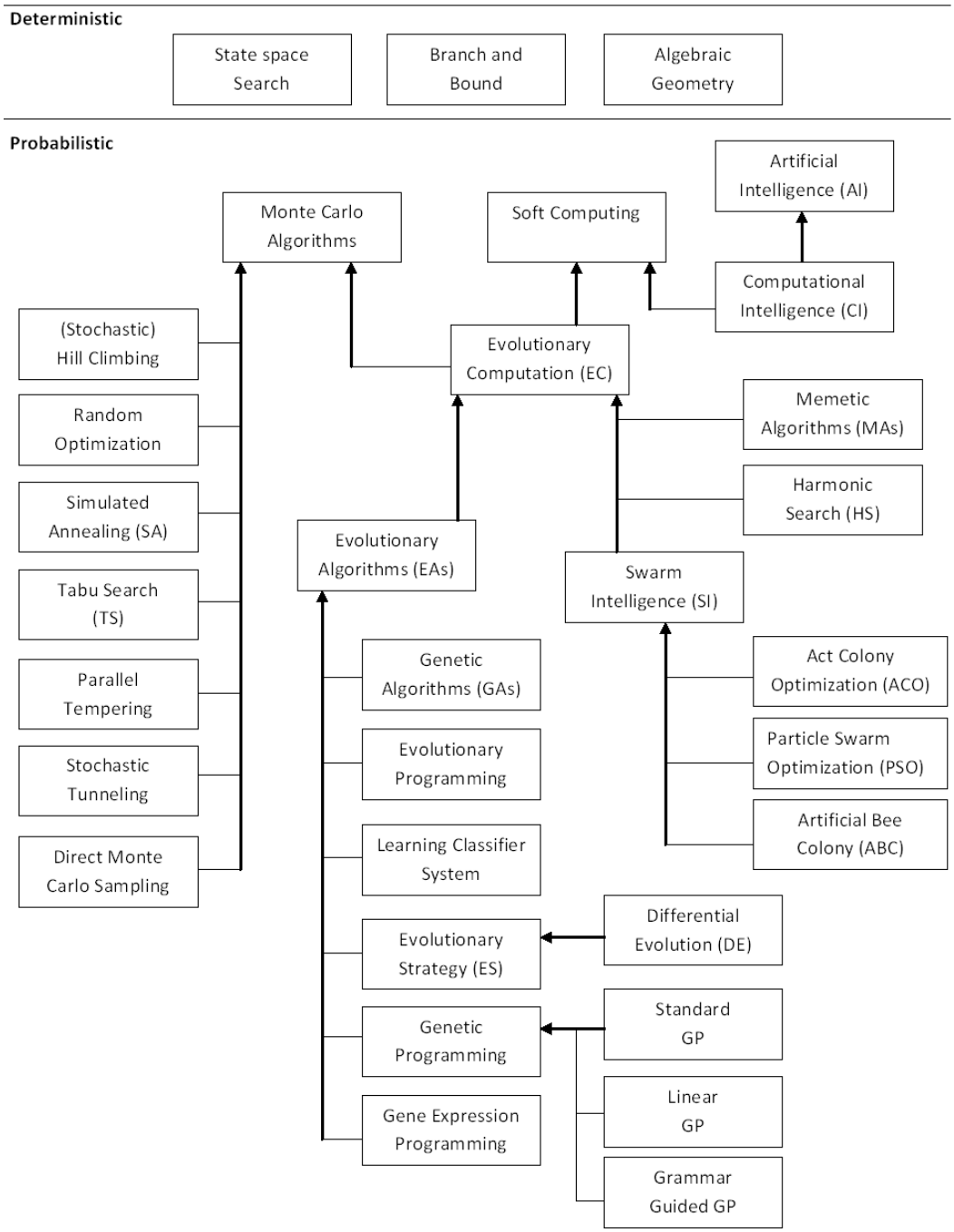


FIGURE 2.1: Taxonomy of global optimization algorithms, hierarchical diagram of various deterministic and probabilistic heuristic and meta-heuristic techniques [144].

In many situations, probabilistic algorithms have the upper hand over deterministic algorithms due to the properties of straightforwardness and ease of use for finding good solutions in a complicated search space [144].

In [99, 114], heuristics as used in global optimization are functions that aid in deciding which one of a set of possible solutions is to be examined next. Heuristics can be used by both deterministic as well as probabilistic algorithms. Deterministic algorithms normally utilize heuristics in order to define the processing order of the solution candidates. Probabilistic approaches, on the other hand, may only consider those elements of the search space in computations that have been favoured by the heuristic.

The heuristic comes within the category of optimization algorithms. It uses the information currently gathered by the algorithm and based on this knowledge base, it helps the algorithm to decide which solution candidate should be tested next or how the next individual can be produced. Heuristics are usually dependent on the problem class.

A meta-heuristic is a heuristic method for solving a very general class of problems. It integrates objective functions or heuristics in an expectantly productive way to yield better solutions in short time. Meta-heuristic often works on population-based techniques. These techniques use a pattern of some natural phenomenon or physical procedure as heuristic function. For example, simulated annealing determines which solution candidate is to be evaluated according to the Boltzmann probability factor of atom configurations of solidifying metal melts. Evolutionary algorithms mimic the behaviour of natural evolution and deal with solution candidates as individuals which strive in a virtual environment [144].

In principle, all the probabilistic optimization algorithms that we consider for the process of optimization as well as some of the deterministic ones come in the category of meta-heuristics. An important class of probabilistic algoriothms is the class of Monte Carlo methods, which are a class of computational algorithms that deal with random estimations, and most stochastic schemes are included in this approach. In addition to this, the role of heuristics and meta-heuristics is very prominent in almost all probabilistic algorithms, which on the one hand use the up-to-date information

or intelligent steps based on experience and learning for creating a new candidate solution which is hopefully better than the previous one, but on the other hand decide which decisions should be fruitful for the other candidate solutions. In general, the approaches which have been mentioned above can use feedback information achieved from samples in the search space. Usually, they use some abstract models from natural phenomena. The most important methods are simulated annealing (SA) and the evolutionary algorithms(EAs) [144].

2.1 Combinatorial Optimization Problems (COPs)

COPs exist in the domains of computer science and other disciplines in which computational techniques are applied [70, 81, 95]. Some of the practical applications are in the fields of management science, biology, chemistry, physics, engineering and (especially) computer science. These problems/fields involve finding groupings, ordering, and assignments of a discrete, finite set of objects that satisfy certain conditions or constraints. The majority of these problems are complex and very hard to solve. The search space of these problems is usually gigantic, and they do not reasonably have any helpful mathematical formulation. When solving a combinatorial optimization problem, the set of candidate solutions can grow exponentially as the problem size grows, so that a simple enumeration scheme or algorithm quickly becomes impractical. The simplest approach for finding the optimal solution to a COP is to evaluate all the possible solutions, which leads to a kind of exhaustive search procedure; but, as mentioned above, this is impractical.

There are two ways to find optimum solutions as mentioned above, one of which is guaranteed and the other not guaranteed. To get guaranteed optimum solutions, exact methods are used. Exact methods find an optimal solution but are often time-consuming or computationally expensive. Then other approaches which give (near) optimal solutions to the optimization problem in reasonable time are heuristic

methods. As compared to exact methods, heuristic methods do not guarantee to find an optimum solution, nor do they generally provide a guarantee of finding optimal values within a certain range.

Heuristic methods with no doubt are of great importance since they can give high-quality solutions in a reasonable time. These methods are applied on a wide range of problems. These heuristics can often be modified to account for changes in problems, and further constraints added to the problem. As exact methods are problematic by nature regarding the size of the combinatorial space, exact methods are not often used as an alternative to heuristics.

2.2 Classification and Examples of COPs

COPs can be classified into at least four classes. For each class, some well-known problems are given [147]:

Assignment Problems: The examples which come under this class include the linear and quadratic assignment problems and the timetabling problem in which the assignment of teachers to subjects, students and rooms is involved.

Ordering Problems: The travelling salesman problem (TSP), the linear ordering problem, the Chinese postman problem, and scheduling problems come in this category.

Partitioning Problem: The examples in this class include graph partitioning and the number partitioning problem.

Subset Selection Problem: The knapsack problem, the set partitioning problem, the set covering problem, the graph bi-partitioning problem and the maximum cut problem belong to this class of problems.

It is worth mentioning here that some of the problems fall in more than one class. One such example is the vehicle routing problem, which is the combination of ordering and partitioning problems.

2.3 Natural Evolution

The term *evolution* can better be defined as a gradual process in which something changes into a different and (usually) more complex or better form. From the biological point of view, evolution is the change in the genetic composition of a population during successive generations as a result of natural selection acting on the genetic variation among individuals and resulting in the development of new species [133].

The root of Evolutionary Computation (EC) is strongly linked with Darwin's theory of evolution, which is commonly called *"Darwinism"* [27]. In 1859, Charles Darwin presented his theory in his book *"Origin of Species"*. According to his theory, the concept of evolution is based on three fundamental concepts, which are:

- replication.
- variation.
- natural selection.

As new life is produced from life in the form of replicas by the process of replication, new offspring are produced which are identical. However, due to changes or mutation (sudden genetic change) during the replication process, a kind of variation is introduced that leads to the gradual development of new organisms. Another form of variation results from sexual recombination, in which different genetic materials from both the parents are participating [132]. Now the resources of the environment are limited; so, the replication process cannot go infinitely. Individuals of the same or other species have to compete with each other, and normally only the fittest will

survive. Thus natural evolution implicitly causes the adaptation of life forms to their environment since the fittest have a better chance to reproduce. This is called the *"survival of the fittest"*.

In [95] it has been mentioned that the natural evolutionary process is a stream of the huge optimization process as well as dynamic, in which the fitness of the individuals is maximized with the passage of time and also with their relation to the environment. In addition, in terms of transferring information from one generation to the next generation, natural evolution can be regarded as a huge information-processing system. Every organism possesses huge genetic information, which is referred as a genotype (the genetic make-up of an organism or group of organisms concerning a single trait, set of traits, or an entire complex set of traits). The organism's traits, which are developed with the passage of time as the organisms grow up, constitute the phenotype (the appearance of an organism resulting from the interaction of the genotype and the environment). The genetic information is passed from one generation to another generation, and the organism can be regarded as the mortal survival machines of the potentially immortal genetic information. During the phase of replication, it is combined with variation, which allows the improvement of genetic information. The natural selection process implicitly evaluates the fitness of each individual who is indirectly the result of the genetic set-up as well.

2.4 Genetics

Genetics is the branch of biology that deals with heredity, especially the mechanisms of hereditary transmission and the variation of inherited traits among similar or related organisms [66, 133]. The cell is the basic unit of life. All living things are composed of cells, inside which there are many substances such as the fluid, called *cytoplasm*, in which other cell bodies are contained, and cell centrioles, which controls spindle fibres during the cell division, *etc.* But the most important structure of

a cell is the nucleus which contains all the heredity materials, called chromosomes. Chromosomes are further composed of double spiral like structures, called Deoxyribo Nucleic Acid (DNA) [139, 143]. The chromosome serves as the mapping[1] information for an organism. It is also called the *"blueprint"* or *"construction plan"*. The chromosome is further divided into genes. Genes are the biological unit of heredity. Each gene is located at a particular location in the chromosomes which is called the *locus*. In general, we can think that gene is a kind of encoding trait such as colour, *etc.* The different (two or more) alternative forms of a gene at the same site in a chromosome, which determines alternative characters in inheritance, are called *alleles*. Majorities of complex organisms are composed of more than one chromosome in each cell, *e.g.*, in human beings the numbers of chromosomes are 46. The collection of chromosomes together is called the *genome*. A genome is all of a living thing's genetic material. It is the entire set of hereditary instructions for building, running, and maintaining an organism and passing life on to the next generation. In short, the genome is divided into chromosomes. Chromosomes contain genes and genes are made of DNA.

There are two types of reproductions or cell divisions found in nature, which are the asexual reproduction *"Mitosis"*[2] and sexual reproduction *"Meiosis"*[3] [130]. In Mitosis or asexual reproduction, an organism reproduces itself by cell division and the chromosomes are replicated. *Mutation* occurs during this process in which one or more alleles of genes are changed. Some genes are deleted or they are re-inserted in another loci within the chromosomes. The other form of cell division is sexual reproduction, in which genes are exchanged between the chromosomes of two parents to form a new set of chromosomes. This type of recombination can be thought of as crossing over of the chromosomes. The fitness of an organism is defined as the

[1]Mapping is the procedure of building a representative sketch cataloguing the genes and other features of a chromosome and showing their relative locations.

[2]Mitosis is a cellular process that replicates chromosomes and produces two identical nuclei in preparation for cell division. Generally, mitosis is immediately followed by the equal division of the cell nuclei and other cell contents into two daughter cells.

[3]Meiosis is the formation of egg and sperm cells. In sexually reproducing organisms, body cells are diploid, meaning they contain two sets of chromosomes (one set from each parent).

probability or the capability of living, developing, or germinating under favourable conditions to reproduce, called *viability*. Furthermore, the birth-rate of a population, defined as the number of offspring produced by an organism, is also termed as *fertility*.

The genetic changes which occur during mutation or recombination cannot be predicted due to the effects of gene interaction which is called *epistasis*. In simpler terms, epistasis is the interaction between non-allelic genes, especially an interaction in which one gene suppresses the expression of another. *Pleiotropy* occurs when a single gene influences multiple phenotypic traits. Consequently, a new mutation in the gene may affect some or all traits simultaneously. This can become a problem when selection on one trait favours one specific version of the gene (allele), while the selection on the other trait favours another allele. When a single phenotypic trait is influenced by the interaction of multiple genes, it is called *polygeny*. Table 2.1 presents a brief overview of some of the terminology borrowed from biology and used in EC.

Biological Terms	**EC meaning**
Chromosomes	String of symbols.
Population	A set of Chromosomes.
Deme	A local population of closely related chromosomes, a subset of the total population.
Gene	A feature, character or detector.
Allele	Feature value.
Locus	A position in a Chromosome.
Genotype	Structure.
Phenotype	A set of parameters, an alternative solution or a decoded structure.

TABLE 2.1: A summary of biological terms used within EC [130].

2.5 Brief History of Evolutionary Computation (EC)

Evolutionary algorithms are designed to mimic the performance of biological systems. The history of EC can be dated back to the 1940s [41]. In 1948, Turing proposed a kind of "genetical or evolutionary search" [36]. In 1962, Bremermann had actually performed some experiments on optimization by using the technique of evolution and recombination. The real birth of EC started in the 1960s at the Technical University of Berlin [95], where Rechenberg *et al.* invented Evolution Strategies (ESs) [122]. In this approach, they optimized the real-valued parameters for devices such as air-foils. The main focus of ESs was on the optimization of continuous parameters. The idea of ESs was further enhanced by Schwefel and is still one of the hot areas of research [128]. Originally, ES comprised recombination and mutation on a two-member population. Later, it was extended to allow more than two members in the population. The current ES includes multi-parent recombination and self-adaptation of parameters.

In the same era another technique, called Evolutionary Programming (EP), was developed by Fogel, Owens, and Walsh [39]. Initially, EP was proposed for sequence prediction and producing suitable responses in the light of a given goal. For EP, it was argued that mutation should be the sole breeding mechanism. EP is one of the old EC fields, and it was used to evolve individuals in the form of finite-state machines or finite-state automata (FSA). ES and EP have many things in common.

Genetic algorithms (GAs) were invented by Holland [68] at the University of Michigan in the 1970s. GAs are search-based methods that make use of processes found in natural and biological evolution. Holland provided a theoretical framework for adaptation under GAs. GAs search or operate on a given population (a set of candidate solutions) to find those that satisfy some predefined criteria. To accomplish this goal, the algorithm applies the principle of survival of the fittest to find better

and better approximations. At each step of the GA, a new set of approximations is created by the process of selecting individuals (potential solutions) according to some selection criteria and breeding them together using operators borrowed from Genetics. This process leads to the evolution of a population of individuals that is better suited to its environment than the individuals which had been created, *i.e.,* the previous population, just as in the natural adaptation process. GAs in general make use of three fundamental genetic operations: selection, crossover and mutation. The pseudo-code of the standard GA is shown in Algorithm 1 and the flowchart [63] of the standard GA is shown in Figure 2.2.

Algorithm 1 The Simple Genetic Algorithm

```
1: Initialize: Generate an initial population;
2: while ( Stopping conditions are not satisfied ) do
3:    Select set of individuals in the population;
4:    Do Crossover and Mutation;
5: end while
```

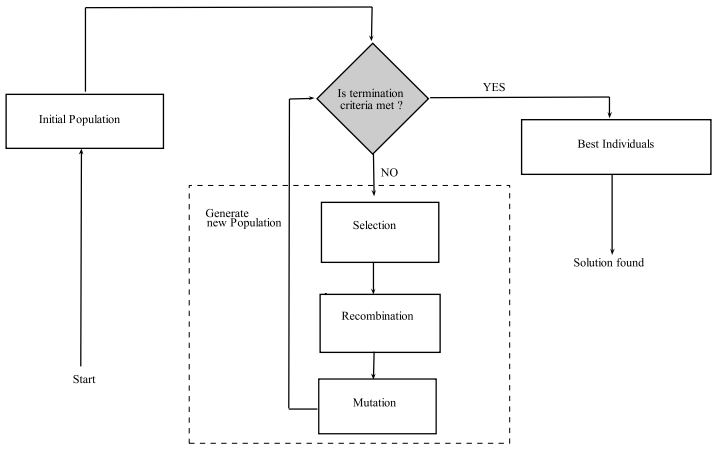


FIGURE 2.2: Flow chart of a simple genetic algorithm (SGA) [36, 63].

As mentioned above, GAs are a class of search and optimization techniques that work on the principles inspired by nature: *"Darwinian Evolution"*. It is now well established in the literature that pure GAs are not well suited for fine tuning search in complex combinatorial spaces and that hybridization, together with other techniques can greatly improve the search efficiency of GAs [80]. This technique of combining GAs with Local Search (LS) give rise to "Memetic Algorithms" (MAs). MAs are

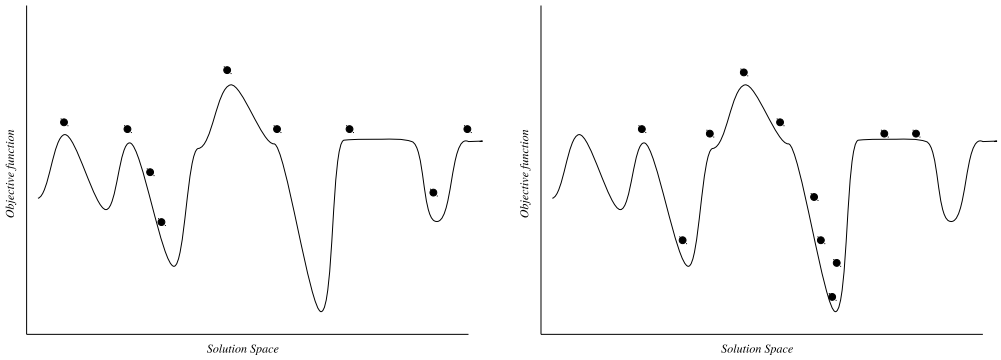


FIGURE 2.3: Working of a GA, showing how the population is moving towards the optimum.

extensions of GAs which apply separate processes that are additional local search techniques to refine individuals [80, 95]. According to Krasnoger and Smith, from the optimization point of view, MAs are more efficient, *i.e.,* they take fewer iterations and evaluations to find optima and are effective and give better solution quality than traditional GAs for some problem domains [80], as they embed the domain specific knowledge while (in contrast) GAs are totally blind techniques. As a result, MAs are gaining a wide acceptance, for solving known combinatorial optimization problems.

So, for a wide variety of issues, MAs can show better results than traditional GAs. On the other hand, the design of incompetent MAs raises important issues which need to be investigated in order to see what new insights can be gained [80].

It is worthwhile mentioning here that Moscato, inspired by both Darwinian principles of natural evolution and the Dawkins notion of a meme, first introduced the term *"Memetic Algorithm"* (MA) in his technical report [93] in 1989. The term *"meme"* was defined by Dawkins [28] in 1976 as *"the basic unit of cultural transmission, or imitation"*, as Dawkins said in [28]:

> *"Examples of memes are tunes, ideas, catch-phrases, clothes fashions, ways of making pots or of building arches. Just as genes propagate themselves in the gene pool by leaping from body to body via sperms or eggs,*

> *so memes propagate themselves in the meme pool by leaping from brain to brain via a process which, in the broad sense, can be called imitation."*

Memetic algorithms can be classified into three main categories. According to Merz [95], the first category is the combination of genetic algorithm and local search techniques; the second category is called multi-meme; according to Krasnoger [79] the memetic material is accompanied by genotype, Kendall *et al.* gives the idea of Hyper-heuristic [77] and Ong and Keane reported Meta-Lamarckian MA in [108], where the set of multiple candidate memes is considered and will compete based on the performance in previous generations or cycles. Here a kind of reward and penalty mechanisms is considered looking at which memes do well on which situation. Finally, in the third category, Smith in [131] gives the concept of a meta-heuristic search algorithm in which a rule-based representation of local search is co-adapted together with candidate solutions within the hybrid evolutionary system; this is also called Co-evolution [67, 112].
The pseudo code of a memetic algorithm is given in Algorithm 2, and a flowchart is given in Figure 2.4.

Algorithm 2 Procedure Memetic Algorithm

```
1: Initialize: Generate an initial population;
2: Apply LocalSearch(individual) to population to make it locally optimal;
3: while (Stopping conditions are not satisfied) do
4:    Select set of individuals in the population;
5:    Do Crossover and Mutation;
6:    Apply LocalSearch(individual) to population to make it locally optimal;
7: end while
```

2.6 The Framework of Genetic Algorithms (GAs)

As the history of this area shows, there are many different variants of evolutionary algorithms. The main idea behind all these techniques is a common one, however.

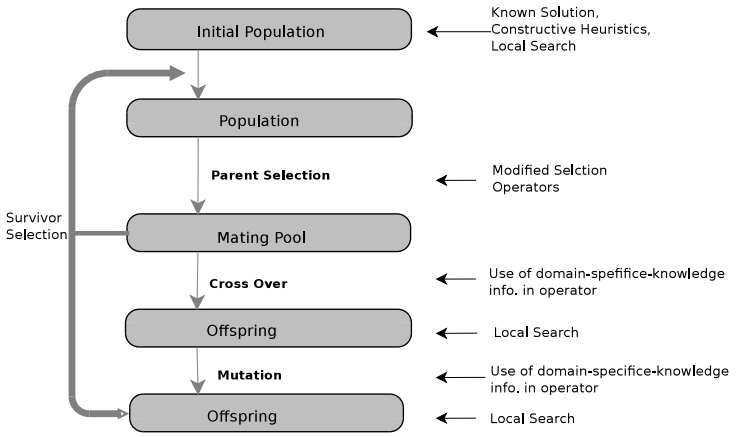


FIGURE 2.4: Flow Chart of Simple Memetic Algorithm [63]

First, a candidate set of solutions or individuals are provided to the EA or GA, which is called a population. The individuals in the population are decoded or evaluated according to a fitness function specified for the given problem. The fitness function is the main part of the EA; this is also called the objective function. For recombination and mutation, candidates are selected to compete for the next generation based on better fitness values. Recombination or crossover is a variation operator applied to two or more selected candidates (called parents) and results in newer candidates (called children). Further, another variation operator, mutation, is applied to one or more candidates and results in new candidates. Individuals are then selected for the new generation. This process is repeated until a candidate with sufficient quality is found or the process is terminated using the pre-defined criteria. The typical procedure of EAs is illustrated in Figure 2.3. The two driving forces of EAs are:

- Variation operators (Crossover and mutation) which produce the necessary diversity and therefore facilitate newness.
- Selection criteria which act as a kind of force pushing quality. There are two main selection criteria: one is the selection for variation, which selects individuals for variation operators, and the second one is selection for survival in which the EA decides which individuals will go for the new generation.

2.6.1 Fundamental Components of Evolutionary Algorithm

Modern EAs or GAs can be composed of many of components, procedures and operators. The most basic and important components which form the main framework of EAs are given below:

1. Chromosomal Representation (real-world problem to the EA world problem.)
2. Initialization.
3. Evaluation function (commonly known as the fitness function /objective function).
4. Population set of candidate solutions.
5. Selection.
   (a) Selection for variation.
   (b) Selection for survival.
6. Variation operators, recombination and mutation.
7. Termination condition.

2.6.2 Chromosomal Representation

Before a genetic algorithm can be put to work on any problem, a method is needed to encode potential solutions to that problem in a form that a computer can process. In this step, the EA links the *"real world"* to the *"EA world"*. This can encode the appearance, behaviour, and physical qualities of an individual. The following approaches are in use [130]. The representation of chromosome varies from problem to problem, but some of the common representations of chromosomes are given in Table.2.2.

1. Encoding solutions as binary string sequences of $1's$ and $0's$ where the digit at each position represents the value of some aspect of the solution.

2. Encoding solutions as arrays of integers or decimal numbers, allowing for greater precision and complexity than the comparatively restricted method of binary numbers.

3. Representing individuals in a GA as strings of letters where each letter again stands for a specific aspect of the solution.

Chromosomes could be:

Chromosomes	**Representation**
Bit strings	(0101, . . . ,1100)
Real numbers	(43.2, 33.1,. . . ,0.0 , 89.2)
Permutations of an element	(E11, E3, E7, . . . ,E1, E15)
Lists of rules	(R1, R2, R3,. . . ,R22, R23)
Tree	(Genetic Programming)
Any Data Structure	(Depending on the problem)

TABLE 2.2: Various representations of problems into EA chromosomes.

2.6.3 Evaluation Function

The evaluation function is the objective function which quantifies the optimality of a solution. This is commonly called the fitness function. The fitness function is the most vital part of an EA. The fitness function measures assigned quality or improvement to a genotype. So, for any problem, the fitness function has to be defined individually. The fitness function ranks the individuals against each other. On the basis of this fitness function, optimal chromosomes (or, at least chromosomes which are better) are selected for breeding in the hope that the new generated chromosome should be better.

2.6.4 Population Initialization

The initialization of population is a process which provides a set of chromosomes (potential solutions) to the EA cycle. There are two issues to be considered for the initialization process; one is population size and the other is the procedure for initializing the population. It is common that the length of time increases with the population size when the length of chromosome/problem increases. Recent literature shows that better results are largely dependent on the size of population. It has been observed that satisfactory results can be obtained with a small population size with less time, but the population suffers from premature convergence. On the other hand, a large population can be more useful, but it is computationally expensive. So, to keep the balance between time and the quality of solutions deciding the population size is a crucial point. The population can be initialized in two different ways. The first one is heuristic initialization and second one is random initialization. For providing a good start to the EA, problem specific heuristics are employed for higher fitness. One of the issues which is linked with heuristic initialization is that it explores a small part of the solution space, *i.e.,* individuals are almost identical and lack diversity. On the other hand, random individual initialization is more diverse as compared to the prior approach.

2.6.5 Population

The main role of the population is to represent the potential solutions to a problem. The population in this sense forms the unit of the evolutionary process. In EAs the population is the multi-set of genotypes or individuals. The individual itself is a static entity; it doesn't change or adapt but it interacts with the other members of population and goes on doing so. For EAs the size of population is established in advance. The other core operations of EAs such as selection and variation operators, work at the population level. In general, the EAs take the whole population into account and different choices are made relative to it. For example, the best (or better)

individuals are used to seed the next generation, in addition the worst individuals are replaced by new ones which are good in fitness quality. There are many issues linked with the population for the better performance of an EA, such as population size, how the EA would be affected by choosing large and small population and how the population is initialized, (*i.e.,* a random one or heuristic based). The key factor is that the population is diverse enough. There is no single measure for diversity in the literature; however, to answer this question briefly, people might refer to the variation in the fitness of the individuals within the population, or the number of genotypes, or number of different phenotypes present.

2.6.6 Selection

There are two types of selections, one is the selection for variation and the other is the selection for survival. The role of the first one is that parents are selected for mating in sets, which is called mating pool. In this scheme, potentially good parents are selected based on their fitness quality. It has been done in the sense that good parents should result in good children for the next generation. The latter one is sometime also called replacement. Survival selection mechanisms are responsible for pushing quality improvements. In this scheme newly generated offspring replace some (or all) the parents.

As mentioned above there are in general two forms of selections, one is for variation and one is for survival [95]. In the first one, individuals are selected for crossover and mutation; in the latter, the EA decides which individuals would go for the next generation. The latter is also called replacement, in which the EAs replace a few or all parents using by some pre-defined criteria. Some of the common ones are given below:

Fitness Proportionate Selection can be also known as the roulette wheel sampling [49]. This scheme works like spinning a roulette wheel of Casino, in which everyone

has a roulette wheel slot sized in proportion to its fitness, *i.e.,* the greater the fitness the larger the chance of selection (and vice-versa.)

Rank Based Selection is another form of sampling [12]. In this kind of sampling it first ranks individuals by their fitness and further uses that ranking to determine the selection probability. In *linear ranking*, individuals are initially sorted according to their fitness values; in this the top individual is the worst one and the last one is the best one. One of the drawbacks of the *fitness proportionate selection problem* is when the variance is decreased among the individuals of the population; the selection then becomes purely random. For this issue, rank based selection has been used for keeping the selection pressure constant and also independent of the variance of the fitness values. In this model, the probability of selecting an individual s_i is given by the following formula:

$$P(s_i) = p_{max} - (p_{max} - p_{min})(\frac{i-1}{n-1}) \tag{2.1}$$

Here n is the size of the total population and p_{max}, p_{min} are the maximum and minimum selection probability respectively.

Tournament selection is a sampling technique [16, 50] where in each step, k individuals from the population are selected randomly, (independent of the fitness of the individual) and the best out of the k is selected. The number k is called the tournament size. The process is repeated m times for selecting m individuals. This scheme is easily implemented. Other schemes for selection are *Stochastic Universal Sampling* [13], which provides zero bias and minimum spread, and *Truncation Selection* [16, 103], which is a kind of artificial selection method. which is used when the population size is very large (in this scheme individuals used for forming the next generation are formed by breeding only the best individuals in the population).

The strategies for the **Selection for Survival** are given below.

Generational Replacement: In this replacement scheme, all the parents are replaced by their offspring. This method has remained in common use for genetic algorithms (along with the fitness proportionate selection to enforce the selection pressure).

Steady State Selection: In this replacement scheme, not all of the parents are replaced unlike the above strategy [137]; only the worst (or old) parents are replaced.

(μ, λ) In this scheme μ parents are replaced by the best λ offspring. Using this criterion, the EA produces more children in each reproduction step if $\lambda \geq \mu$. The selection pressure can be increased by generating more offspring.

$(\mu + \lambda)$ In this scheme, μ members of the new population are selected from the sorted list of the temporary population composed of μ parents and λ children.

Some other selection strategies are *elitism* [71] based and *duplicate checking* [38], in which *identical* individuals are not selected common to their parent.

2.6.7 Variation Operators

Variation operators in GAs are responsible for producing new individuals from old ones. In the analogous phenotype space, the variation operators are used to generate new candidate solutions. In the perspective of generation and test, variation operators perform the generation step in EAs. It is not necessary that the newly generated individuals will be always better than the previous one in general. Variation operators in EAs can be classified based on the arity. A unary arity operator is commonly known as the mutation and a binary arity operator is a recombination operator.

2.6.7.1 Mutation

In biology, a mutation operation occurs when a DNA gene is damaged or changed in such a way so as to alter the genetic message carried by that gene. In the scenario of EAs, mutation is a procedure which is applied to one genotype and slightly modifies the mutant to generate a child or offspring. It is a stochastic operator. It results in a child who is the outcome of some random choice or a series of random choices. It is worth noting that an arbitrary unary operator is not necessarily seen as a mutation operator. Some problem-specific heuristic operator acting on one individual could also be termed as unary. However, the mutation operator can be considered as to simply cause random, un-biased change. The job of mutation is different in various approaches. For example, in genetic programming (GP), there is no concept of mutation, but, in the GA, mutation is considered as one of the key operators that gives a new direction to GA for exploring the more diverse region of the search space. An example of mutation is diagrammatically shown in Figure 2.5.

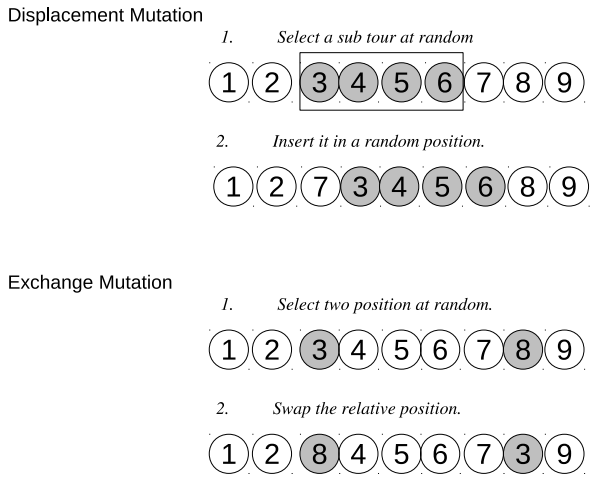


FIGURE 2.5: An illustration of a mutation operator (Displacement and Exchange).

2.6.7.2 Recombination

Recombination is a binary variation operator which is sometime also called crossover. As the name indicates, this operator merges information from two parents genotype into a single or two offspring genotypes. Like mutation, crossover is stochastic; the choice of which parts of each parent are merged and the way they are merged together is totally random. It is also worth mentioning that recombination operators with a higher arity are mathematically possible and easy to implement, but they are not common from a biological point of view. This is why they are not commonly used but research shows that they can have a fruitful impact on the evolution. The key idea behind the crossover is quite straight forward; by mating two individuals with different but desirable features, we can get both features. This principle has a very strong supportive case from the biological perspective, *i.e.,* for higher yields or other desirable features, selective individuals are bred together. The PMX crossover operator is illustrated in Figure 2.6. EAs in the evolutionary process create a number of offspring with different fitness quality, *i.e.,* better and worse. Biologists suggests that recombination is the most superior form of reproduction, and higher organisms reproduce sexually as compared to basic organisms, which reproduce asexually.

2.6.8 Termination Condition

EAs are stochastic and there is generally no guarantee that we will reach an optimum, hence this terminating condition is not satisfied at all and the algorithm suffers from an infinite execution of the evolutionary process. Some commonly used options for ensuring termination are the following:

1. The maximum allowed CPU time elapses.

2. The number of fitness evaluations reaches a given limit.

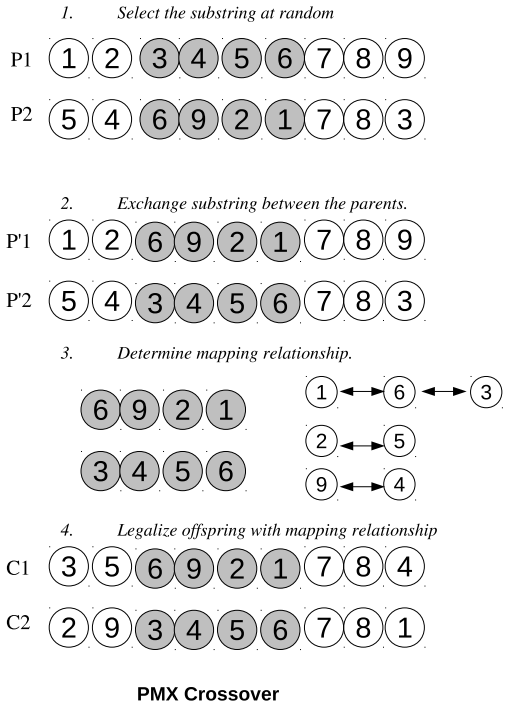


FIGURE 2.6: An illustration of Recombination/Crossover (PMX) and repairing.

3. The fitness of the individuals is not improving for some specified number of generations.

4. The population diversity drops under a given threshold.

5. The disjunction of the optimum value being hit or some other condition x being satisfied (although in case of no known optimum, there is no need for the disjunction).

2.7 Hybridizing EAs with Other Heuristics

The term hybridization is the extension of EAs towards the local-searchers [81]. As mentioned above local-search is an algorithm such as gradient descent or random hill-climbing which searches only for a local optimum. These methods are not meant to do any sort of global optimization, they leave that issue to the population and the conventional genetic operators. The only support which the local search provides

to the global optimization process is that, after the crossover and mutation, the local search further improves the fitness to the local optima, which are then used for the selection steps. The common view we take is that global optimization is complemented by local refinement operators. So we expect that the EA, with its diverse population and recombination operators, can perform global sampling over the entire search space in a better way.

The performance of a genetic algorithm on any global optimization algorithm depends on two conflicting objectives *exploration* and *exploitation*. The genetic algorithm can combine both exploration and exploitation in an optimal way. One of the advantages of the genetic algorithm is that it can swiftly find the regions in which the global optimum exists, but GAs take relatively longer time to locate the exact local optimum in the region of convergence. So, when GAs are hybridized with local search methods this can speed up the search process by locating the exact optimum. In this hybrid technique, local search is applied to the solutions that guide the genetic algorithm to the most promising regions and this accelerates the convergence of the GAs. GAs are random search-based techniques which do not employ or embed problem specific knowledge. Incorporating local search methods can also reduce the time needed to reach a global optimum by using domain-specific knowledge [63].

It is not expected in a local search that we search for a large portion of the search space. However, the immediate potential benefit which is offered by the local search is that it improves the quality of the candidate solutions, *i.e.,* making the population (to some extent) locally optimal. So an EA is likely better served by the values at the local optima than it is by values at random points. One other reason that a local search can help is that EAs are not efficient hill-climbers [81].

As mentioned above, there are many advantages that can be obtained by incorporating the global search of EAs into local searching or other approaches for progressing and refining the population. There are some factors that motivate the hybridization of EAs into other techniques [63, 80].

1. Domain-specific knowledge can be incorporated into the *"greedy and blind"* variation operators of EAs. It is also possible to use this knowledge to define local search operators or existing solution improvement techniques within an EA.

2. One common view is that there are no successful and efficient all-purpose *"Black-box"* optimization problem solvers, and empirical evidence and some theoretical studies such as the *No Free Lunch (NFL)* theorem [148] strongly support this view. From the perspective of Evolutionary Computation (EC), this shows that EAs are not the holy grail for a global search. Experience suggests that making EAs competent needs the incorporation of domain-specific-knowledge techniques.

3. In practice, EAs are very good for exploration by identifying good areas of the search space, but they are blunt for refining near optimal solutions,*i.e.,* exploitation. From an optimization point of view hybrid-GAs have shown that they are more accurate than traditional GAs for some problem domains such as continuous domains and combinatorial domain. It is also argued that the success of hybrid GAs is due to the trade-off between the exploration ability of GAs and the exploitation ability of local search. The only outcome is more fitness evaluation as compared to GAs and loss of diversity within the population [80].

4. In general, with constraints associated with problems, local search and other heuristics can be used as a means of repairing infeasible solutions generated by standard variation operators. Here designing constrained oriented local search is often considerably simpler and more advantageous as opposed to trying to find a specialized crossover and mutation operators, which ensures the feasibility of all offspring.

5. The base motivation for hybridization is Dawkins' concept of *"memes"* [28]. Memes can be regarded as units of *"Cultural transmission"* in analogy with

genes as the units of biological transmission. Examples of Memes are tunes, ideas, catch-phrases, clothing fashions, ways of making pots or of building arches. It is just like genes propagating themselves in the gene pool by transferring from one body to another body via heredity material in sperm and eggs. In the same way, Memes propagate, themselves in the Meme pool by passing on from brain to brain via a process which is called *Imitation*.

6. Genetic algorithms with local search can help to overcome most of the issues that arise as a result of finite population. Local search introduces new genes which can help to combat the genetic drift problem which is caused by the accumulation of stochastic errors due to finite population [37]. We can also speed up the search towards the global optimum by increasing the convergence rate to make it large enough to obstruct the genetic drift issue [8].

7. As discussed in [37], the Parallel Recombination Simulated Annealing (PRSA) [91] algorithm attempts to prevent the genetic drift problem by combining the idea of a cooling schedule of Simulated Annealing [78], Boltzman tournament selection [90] and the genetic operators, *e.g.*, crossover and mutation.

8. Stand-alone genetic algorithms produce low-quality solutions when the size of population is finite or limited. It is difficult for the operators of the GA to find the best solution in good regions, because of the inability to make small moves in the neighbourhood of current solutions. With local search, the GA can improve the exploiting ability of the search algorithm without affecting the exploring ability. While designing competent local search, the right balance between exploration and exploitation capabilities, which are two conflicting objectives, can be achieved and algorithm can produce high-quality solutions [58].

9. Genetic algorithm can locate the regions in which the global optima exist but take a long time to locate exact local optima in the region of convergence when the problem size is increased. A combination of local search can speed up

the search to locate the exact global optimum by guiding the GA to explore the good regions. This reduces the time needed for the GA to reach exact global optima; also, when local knowledge is used, it accelerates the process of locating the most promising search regions [119].

10. For solving real-world problems, stand-alone GAs are unable to *fine-tune* the control parameters. As discussed in [37], the search ability of GAs is strongly influenced by the control parameters in solving real-world problems due to the detrimental influence that wrong parameter settings can have on the trade-off between exploitation and exploration. Good parameter settings can succeed in finding a near optimal solution in an efficient way. The problem of choosing the correct parameters is a time-consuming task. In addition, the use of fixed parameters does not match the evolutionary spirit of GAs. For this reason, adaptive local search techniques can be utilized to set the values of these parameters whilst the search is progressing and the behaviour of the population is changing [10, 35].

11. The issue of premature convergence can further be controlled with hybridized GAs. Genetic algorithms may sample bad representatives of a good search region and good representatives of bad regions due to the limited size of the population. A local search can ensure balanced representation of the diverse and different search areas by sampling their local optima; this can reduce the premature convergence possibility up to some extent.

12. An additional approach which has been inspired by nature for addressing COPs apart from EC models is the Ant Colony System [30–32]. In Ant Colony Optimisation (ACO) a sort of ant system has been developed, which works on an effective way of finding the shortest path from their nest to a food source or any marked positions without using visual knowledge. In ACO systems, while seeking the food, ants deposit pheromones on the surface and follow, with high probability, the pheromones which have been deposited by earlier

ants (this means that paths are more likely to be followed if they have been used by many other ants as well). The ant system operates by ants guessing a path to a single food source. If there is more than one way to reach the food source, initially equal probability is given for an ant to choose any path. If more ants visit the shortest path on average, the maximum level of pheromone accumulates as they use paths with approximately the same speed. Now if a new ant arrives at that location and decides which path should be followed among different paths, it prefers to choose the shorter path with higher probability, *i.e.,* highly accumulated pheromone level. In ACO systems, a solution for solving a problem is made by a kind of agent (virtual ant), which uses the outcome variables by making up a feasible solution step by step. Every alternative is represented by a metaphor to real ants, which is a probabilistic choice proportional to a global variable representing the amount of pheromone. So, the ants communicate in an indirect fashion by way of a kind of global distributed memory. The distributed memory is a vector or matrix of pheromone variables. A local pheromone update-rule is applied for assigning a value to a component of a solution vector and eventually if one agent (ant) finds a feasible solution then a global pheromone update rule is applied that takes the objective function value of the produced solution into account. Ant colony systems share two significant objectives for solving the TSP. First of all, they have an effective mechanism for attaining a good balance between intensification and diversification of the search. Second, before updating the pheromone trials, a subsidiary local search procedure is applied to the tours constructed by the ants. For this reason, these algorithms are regarded as hybrid stochastic local search procedures, which integrate parablastic solution construction with standard LS procedures [33].

13. A new class of evolutionary optimization algorithms which are called Estimation of distribution algorithms (EDAs) [18, 83, 89, 101] were developed as

a natural substitute for genetic algorithms in the last few decades. Estimation of distribution algorithms are a kind of evolutionary algorithms that work with a multiset (or population) of candidate solutions (points). One main advantage of EDAs over genetic algorithms is the nonexistence of multiple parameters that require to be tuned (e.g., crossover and mutation probabilities). Furthermore, in EDA techniques the expressiveness and transparency of a probabilistic model is used to guide the search process. In addition, EDAs have been demonstrated to be more suitable for some applications than GAs, such as for solving challenging bio-informatics problems [5], while attaining competitive and robust results in the majority of the tackled problems. For any EDA algorithm, first of all, a random sample of points is generated. These points are evaluated using a fitness function. The fitness function evaluates the quality (e.g., accuracy or objective value) of each solution for the given problem. On the basis of this evaluation, a new subset of points is selected. In consequence, better points, with better fitness values, have a greater probability of being selected. Afterwards, a probabilistic model of the selected solutions is constructed, and a new set of points is sampled from the model. The process is repeated up to the time when the optimum has been found or a different termination criterion is satisfied.

2.7.1 Local Search (LS)

Local Search is a method concerned with finding a solution which is good or better than all the other solutions in its local *"neighbourhood"*. Such solution is called a local optimum; it will not necessarily be the global optimum. The basic way of working of LS is that it gets an initial solution and repeatedly making changes, the choice of small changes or large changes depend on how the LS explores the neighbourhood. The process continues until there are no further neighbours available to search. An

efficient local search can find the local optima more easily than a global search because for global optima, the whole search space should be examined. Local Search is a hill-climbing technique with the difference that the neighbourhood is searched systematically instead of randomly; Furthermore, the searching of the neighbourhood is repeated until an optimal solution is found. The accepting-criterion for such an improvement is called a *pivot rule*. The steepest ascent (or best improvement) pivot rule is a criterion for accepting in which the entire neighbourhood is searched [63, 81], and the greedy ascent (or first ascent/improvement) pivot rule accepts as soon as an improvement is found. The general working principle of local search is given in the following algorithm: Algorithm 3 and Figure 2.8 .

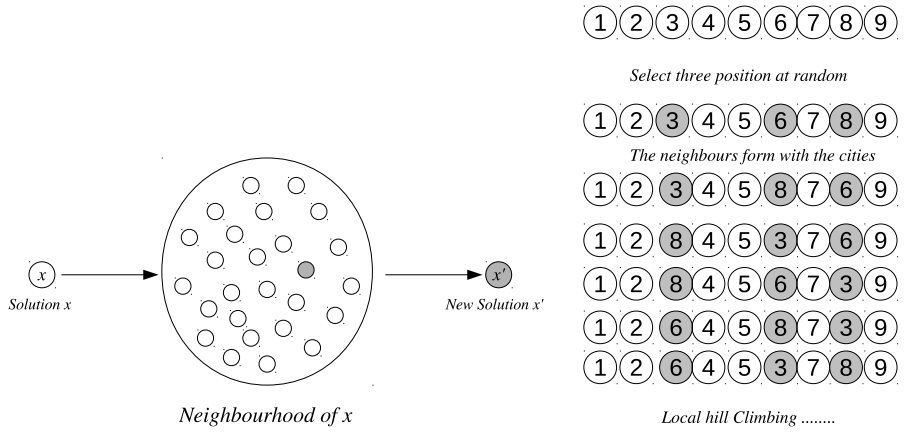


FIGURE 2.7: Simple Neighbourhoods, Neighbourhood Searching and local hill climbing.

Algorithm 3 Local Search $(s \in S) : S$

repeat
  Generate neighbour solutions $s' \in N(s)$;
  if $(f(s') < f(s))$ **then**
    $s = s'$;
  end if
until $\forall s' \in N(s) : f(s') \geq f(s)$;
return s;

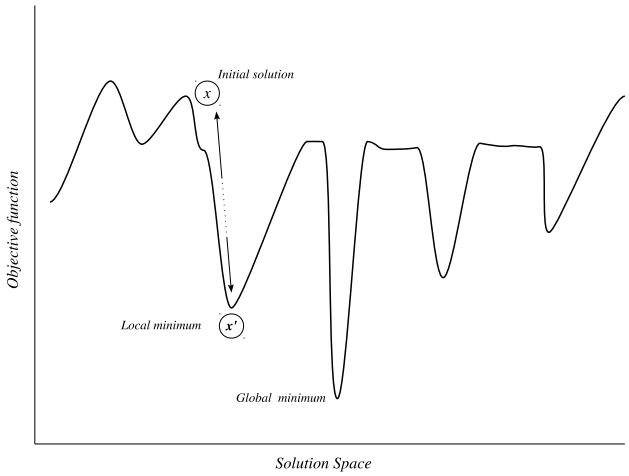


FIGURE 2.8: An Illustration of Local Search.

2.7.2 Reactive Search Optimization (RSO)

Reactive Search Optimization (RSO) [14] supports the combination of machine learning techniques with search heuristics for solving complex optimization problems. The word reactive suggests that the algorithm must be ready to give a response to any events. This mainly involves the self-tuning of critical parameters, etc. The main strength lies in the fact that the searching algorithm will work like the human brain, such as learning from past experience, learning on the job (e.g., how various activities are done during the job, recording any new activities which are not part of its knowledge-base), swift analysis of available alternatives and choices, ability to manage with deficient knowledge, and quick adaptation to new circumstances and occurrences.

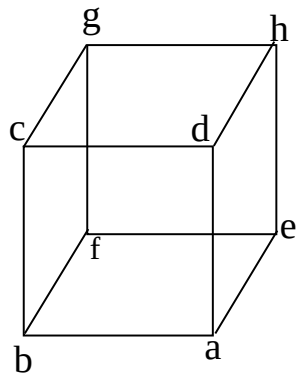
2.7.3 Guided Mutation

Guided mutation [152] can be considered as a composite of the traditional mutation operator and the EDA offspring generation scheme. Guided mutation simply combines global information with local information. Guided by a probability model, guided mutation alters a parent solution to produce a new solution. The resultant

solution which is generated by using the EDA scheme instead of a random scheme would be hopefully better and lie in a promising area which is distinguished by the probability model. Simultaneously, it clearly gets a user-specified percentage of elements from its parent, which is often a better solution found so far. In this way, the similarity between an offspring and its parent can be controlled to some extent in terms of maintaining diversity and exploration.

2.8 Neighbourhood Structure

Neighbourhood search is a recent meta-heuristic for solving combinatorial and global optimization problems whose basic idea is a systematic change of neighbourhood within a local search. Local search works on neighbourhood structures, which clearly define what step or move is going in favour of the optimization process. In general, a neighbourhood structure (or neighbourhood generating function), $\mathcal{N}$ on a search domain $\mathcal{S}$ is a function. $\mathcal{N} : S \rightarrow 2^S$ which shows the set of neighbours for each point in the search space. The neighbourhood and simple hill climbing is shown in Figure 2.7. Related to the landscape metaphor, $\mathcal{N}(x)$ is defined as the set of points that can be reached from x with one application of a move operator such as bit-flipping search on binary problems. In Figure 2.9 the solutions a, c and h can be accessed from d on single move. The success of the local search strongly depends on size and structure of the neighbourhood chosen. There are no well-defined rules to define the structure of the neighbourhood but, in general, the larger the neighbourhood the better the local optima that result. Further it means that any local optima under a given neighbourhood structure $\mathcal{N}$ would also be a local optima under any structure $\mathcal{N}'$ which is the subset of $\mathcal{N}$ in the sense that $\mathcal{N}'(x) \subset \mathcal{N}(x)$ for all x. The average local optima found under a given neighbourhood structure are also referred to as *"strength of the neighbourhood structure"*. The better the neighbourhood structure the better would be the solution quality, but this would be at the expense of computational time [63].



$$\mathcal{N}(d) = \{a, c, h\}$$

FIGURE 2.9: Simple neighbourhood structure, solution *a,c,h* could be reached from d on a single move.

The additional leading decision is the move operator; the choice of the move operator will have a impressive effect on the efficiency and the quality of the local search and hence of the resultant Memetic Algorithm. In the majority of cases, domain-specific knowledge can be used for designing the neighbourhood structure for a local search. Recent studies also confirm that the optimal choice of the operator may not be only instance specific within a class of problems, but, when incorporated in a Memetic algorithm, it can be dependent on the state of the evolutionary search. During the search process when the move operator is changed it may result in a means of progression in cases where the points are locally optimal for that specific given neighbourhood operator because a point may be locally optimal with respect to one neighbourhood structure but may not be optimal with respect to another neighbourhood structure (unless the point is globally optimal one). This observation gave rise to another well-known idea which is called variable neighbourhood search algorithms (VNS); for further details, readers are referred to [81, 116]. The illustration of VDNS and VNS is shown if Figure 2.10. VDNS is a neighbourhood technique, which searches the search space in a variable depth heuristically. For example, in the different neighbourhood structures, $N_1, N_2, N_3, \ldots, N_k$, the simple example of N_1 is *1-exchange* neighbourhood and N_2 swaps the values of two variables or position. In more general form, N_k would consider k exchanges, hence reducing the time of searching the neighbourhood. One well-known example of VDNS is

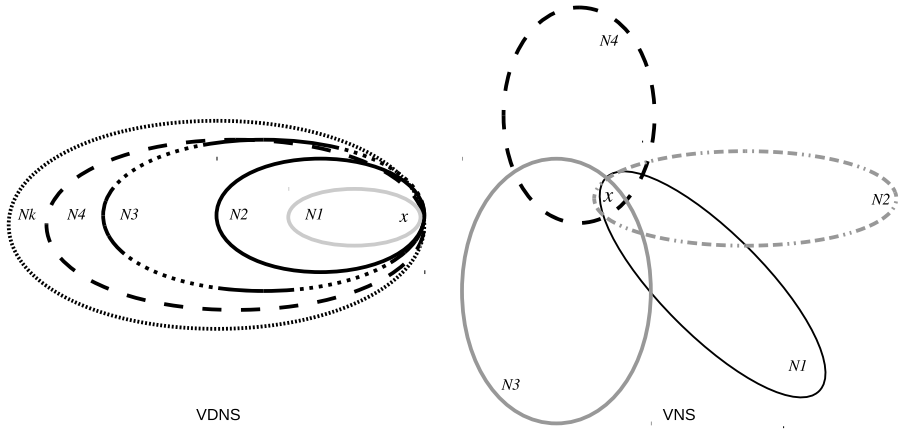


FIGURE 2.10: An illustration of the neighbourhood structure, the VDNS operates on one type of neighbourhood with variable depth, while VNS operates on different structures of neighbourhood. x is the current solution. While $N_1, N_2, N_3, \ldots, N_k$ represents various neighbourhoods.

Lin-Kernighan algorithm for solving the TSP. The base idea in the LK heuristic is that we replace n edges when moving a tour T_1 to T_2. On the other hand, the VNS operates on different $N_1, N_2, N_3, \ldots, N_k$, where N_1 is the structurally different neighbourhood from N_2.

2.9 Chapter Summary

In this chapter, a brief introduction to optimization and the detail of the techniques which are applied have been given. The historical background of Evolutionary Computation and the origin of the subject have been introduced. The process of natural evolution, and how the EC is linked with the natural process such as Genetics have been discussed. We have explained the notions of an evolutionary algorithm, genetic algorithm and many other evolution strategies and memetic algorithms. A comprehensive representation of real-world problems and how it would be transformed to the world of evolutionary computation processes has been given. The dissection of the genetic algorithm using initialization, selection, recombination and mutation operators has been explained. One of the weaknesses of the genetic algorithm (which is

by default the blind algorithm) could be addressed with the incorporation of domain-specific knowledge. The hybridization of GAs with local search and the impact of local search on GAs have been described. The main motivation behind local search is how to structure the neighbourhood and how to exploit useful knowledge in the minimum time.

In the next chapter, the details of a classic combinatorial problem, the Travelling Salesman Problem, are discussed and the past well-known heuristics and meta-heuristics that have been used to solve the problem are also described.

Chapter 3

The Static and Dynamic Travelling Salesman Problem

In this chapter we describe the problem we will be considering; we first look at the static Travelling Salesman Problem (TSP)[1] and then the Dynamic Travelling Salesman Problem (DTSP).

The origins of the travelling salesman problem are still a mystery. According to the handbook *"The Travelling Salesman Problem: a Computational Study"* [1], the origin of the name *"Travelling Salesman Problem"* is not known exactly; there is no authentic proof available about the creator of *TSP*. But one of the most prominent early TSP researchers was Merril Flood of Princeton University and the RAND Corporation referred to the name "Travelling Salesman Problem" [1].

The role of the TSP is very prominent in research also in a number of application areas. The study of this problem has been applied by many researchers from different fields, e.g., Mathematics, Operations Research, Physics, Biology and especially in the field of Artificial Intelligence, and there is a vast amount of literature available, which is very difficult to compile exhaustively. The main reason behind the

[1] "Official" website of TSP: http://www.tsp.gatech.edu/

popularity of TSP is the fact that it can be easily formulated and it also shows all the properties or aspects of combinatorial optimization. The TSP has been served and still continues to serve as the benchmark problem for designing and testing new algorithmic ideas like exact algorithms and probabilistic algorithms. There are many efficient algorithms that have been designed, which has provided a constant intellectual challenge, and many other important techniques for solving combinatorial optimization problems were also developed by keeping the TSP as an example test bed [123].

One of the interesting properties of TSP is that it is not attractive only from a theoretical point of view, but many applicable and real-world problems can be modeled as TSP or by variants of it. For example, in some of the practical applications, the number of nodes ranges from some dozens up to even millions, e.g., in case of VLSI design. Therefore, there is a tremendous need for designing efficient algorithms, which not only solve the problems but reduce the computational cost [123]. In the last few decades, outstanding milestones have been achieved for solving TSP to a satisfactory level of optimality. In 1954, Dantzig et al. solved a 48-city problem. In 1980, Grötschel [56] solved a 120-city problem, and in the same year Crowder and Padberg [21] solved a 318-city problem. A 532-city problem was solved by Padberg and Rinaldi [110] in the year 1987. In 1991, Grötschel and Holland [57] touched the 666-city problem, and also a 2392-city problem was solved by Padberg and Rinaldi [111]. In the same year, Applegate *et al.* [2] solved a 3038-city problem, a 4461-city problem in 1993, a 7397-city problem in 1994, and finally in 1998, a 13509-city problem [3].

In 1991, Reinelt made available a collection of TSP instances in the form of TSPLIB [124], which is a benchmark library for the TSP. The TSPLIB is composed of more than 100 instances with up to 85,000 cities. The majority of the TSPLIB instances stems from influential studies on the TSP, and they mainly originate from practical and real-world application, such as PCB manufacturing, X-ray crystallography or finding the shortest trips. In the TSPLIB, the majority of the instances are of

geographical nature, which means that the inter-vertex distances are derived from the distances between the cities and towns with given (x, y) coordinates. In addition, a set of TSP instances derived from problems in VLSI circuits ranging from 131 vertices to 744710 vertices is included [4].

In 1990, Applegate, Bixby, Chvatal and Cook developed a computer program, which is named "Concorde". The design of Concorde is mainly focused on the solution of larger, more difficult, instances of the problem, rather than, say, concentrating on the fastest solution of small examples. The full source code and documentation are available at:

http://www.tsp.gatech.edu/concorde/index.html

Even so, regardless of these achievements and improvements, the TSP is a long way from being solved. There are many angles of the problem which need consideration to be thought about, and many questions are still there which need acceptable solutions. First, the algorithms proposed for solving the TSP are able to solve large instances to optimality in some cases but are not stable in a general sense because the solution time can be very different for different problems with the same number of cities, and there is no generalized function depending on the number of cities which would give a clue for how much time is required to solve a particular problem. Second, many problems which are considered to be large enough are still unsuitable for being handled by the exact algorithms which are available nowadays. On the other hand, good heuristic algorithms may give better and improved solutions which are only a few percent above the optimal solution. However, these heuristic algorithms also need improvements to reduce the computational cost. Third, for certain problem instances the search space is very large, and not enough real time or CPU time is available to be able to feasibly apply a particular algorithm to solve them. Furthermore, there is not sufficient real time or man power available to code an ideal approach which can be applied to all problems given that some of the theoretical advances are not feasible for a practical implementation [123].

Among combinatorial optimization problems, the TSP has held a principal role for many reasons. Firstly, TSP is an NP-hard [46] problem which is very tough to solve. Secondly, the algorithms which are designed and evaluated for solving the TSP are not clouded by technicalities that originate from dealing with side constraints, which are often unyielding to handle. Thirdly, the TSP is a standard test bed for testing and analyzing new algorithmic ideas. Fourthly, it has received enormous attention from researcher and practitioner communities, which cannot be turned down. Finally, the TSP is applied in many real-world applications. Therefore, it has a significant applicable relevance [70].

Other variations of the TSP that are also common according to [1] include:

- Circuit Riders.
- Knight's Tour.
- The Grand Tour.
- Farmland Surveys.
- School Bus Routing / Vehicle Routing.
- Watchman.

The travelling salesman problem (TSP) is one of the most widely studied combinatorial optimization problems and has attracted a large number of researchers over the last five decades. For a TSP, a salesman needs to visit each of a set of cities exactly once, completing a tour by arriving at the city that is the start and by travelling the minimum distance. In graph theory terminology, the aim is to find a minimum weight Hamiltonian cycle in a graph.

3.1 Problem Statement for Static Travelling Salesman Problem

The problem statement of TSP is: Given a number of cities and the distance of travelling from any city to any other city, what is the least distance round-trip route that visits each city exactly once and then returns to the starting city?

Observation:

In TSP, a salesman will visit n cities, where the number of cities will be $n > 2$. The size of the complete search space would be $(n-1)!$ permutations.

Instances

A set N of n cities, and for each pair of cities c_i, $c_j \in N$ distance $d(c_i, c_j) \in \mathbb{R}$ and $d(c_i, c_j) > 0$.

Feasible solution:

A tour that visits each city exactly once (a permutation π of the n cities) .

Objective value: the total length of the tour, *i.e.,* the sum of the distances between consecutive cities on the tour.

$$\begin{aligned} d(\pi) &= d(c_{\pi(n)}, c_{\pi(1)}) + \sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) \\ &\textit{or} \\ f(\pi, D) &= \sum_{i=1}^{n} d(c_{\pi(i)}, c_{\pi((i+1) mod\ n)}) \end{aligned} \tag{3.1}$$

where $\pi(i)$ denotes the city at the i-th location in the tour, n is the given number of cities and the TSP requires to search for a permutation π, using a cost matrix $D = [d_{ij}]$, where d_{ij} denotes the distance (assumed to be known by the salesmen) of

travelling from city i to city j.

Goal: To find a permutation π^* within the set of all permutations whose total tour length is minimum:

$$\forall \pi \neq \pi^* \quad d(\pi) \geq d(\pi^*)$$

Thus an instance $I = \langle D \rangle$ of the problem is defined by a distance matrix $D = (d_{ij})$, and the solution is a tour represented by a permutation π with $j = \pi(i)$ denoting the city j to visit at the i^{th} step. One special case of TSP is the Euclidean TSP. Here the distance between two nodes or cities is defined by the Euclidean distance between two points in the plane. Furthermore, this assumption does not lead to a reduction of the complexity and hence the problem remains NP-hard [51].

If the distance matrix is symmetric and π_1 and π_2 are two tours in which the nodes are visited in reversed order, that is:

$$\pi_1 = \{i_1, i_2 \ldots, i_n, i_1\} \ , \ \pi_2 = \{i_1, i_n \ldots, i_2, i_1\}$$

then $d(\pi_1) = d(\pi_2)$.

In general there are $(n-1)!$ possible tours, but, for symmetric matrices only $(n-1)!/2$ tours need to be considered.

The equation (3.1) can also be defined as finding a shortest Hamiltonian cycle in a complete weighted graph $G = (V, E, d)$ where the set $V = \{1, 2 \ldots, n\}$ represents the cities and the edge set E the arcs between them. A weight that corresponds to the distance d_{ij} is assigned to each edge between incident cities. So a solution π is a subset of E with $\mid \pi \mid = \mid V \mid = n$.

For example, let the matrix $X = (x_{ij})$ be a Boolean matrix with $x_{ij} = 1$, if there is an arc from i to j in the tour and 0 if not. The TSP is then defined as:

$$min\ l(X) \quad = \quad \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij} \tag{3.2}$$

$$\text{subject to} \qquad \sum_{i=1,i\neq j}^{n} x_{ij} = 1 \quad \forall j \in V \tag{3.3}$$

$$\sum_{j=1,j\neq i}^{n} x_{ij} = 1 \quad \forall i \in V \tag{3.4}$$

$$\sum_{i\in Q}\sum_{j\in V-Q} x_{ij} \geq 1 \quad \forall Q \subset V \tag{3.5}$$

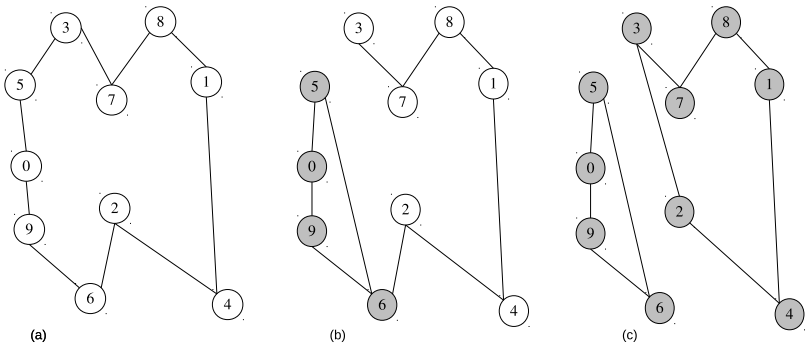


FIGURE 3.1: The Feasible and In-feasible TSP tours.

From the above equations the TSP can be formulated as a zero/one integer programming problem. In the first two constraints we ensure that the degree of each node is two (1 is the in-degree and 1 is the out-degree). Additionally, from the last constraint, we ensure that the solution does not consist of disjoint sub-tours. Feasible and infeasible tours are illustrated in Figure 3.1. Figure 3.1(a) represents a complete tour obeying all of the above constraints. However, the graph in Figure 3.1(b) does not represent a complete tour because node 3 and node 6 do not have degree 2, and hence the constraints (3.3) and (3.4) are violated. In Figure 3.1(c) the graph obeys the constraints (3.3) and (3.4) but violates the constraint (3.5), which leads to two disjoint sub-tours and not a complete tour. If one eliminates constraint (3.5) this would lead to a well-known assignment problem which can be solved by a polynomial-time method which is called the Hungarian method [127].

An ATSP and STSP example

The following examples of STSP and ATSP are borrowed from [59].

An instance of ATSP with distance matrix:

$$D_{ATSP} = \begin{pmatrix} 0 & 6 & 5 & 10 \\ 3 & 0 & 3 & 9 \\ 7 & 4 & 0 & 8 \\ 12 & 7 & 5 & 0 \end{pmatrix}$$

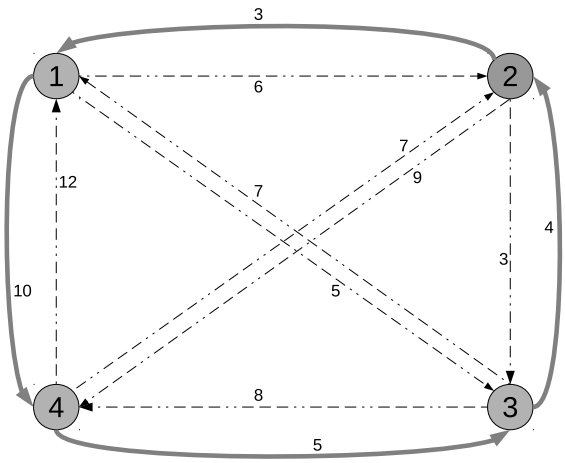


FIGURE 3.2: An instance of ATSP, showing all connected edges along with the shortest tour.

The connectivity of the cities are shown in the figure 3.2. The total number of possible tours are $(n-1)!$, here, $n = 4$, so the number of possible tours is $3! = 6$ having fitness /weight 29, 27, 30, 23, 27 and 22. The optimal tour with weight 22 is {1,4,3,2,1}.

An instance of STSP with distance matrix:

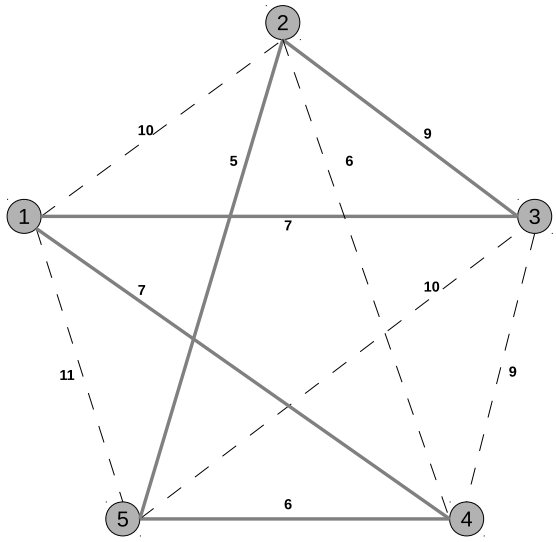


FIGURE 3.3: An instance of STSP, showing all connected edges along with the shortest tour.

$$D_{STSP} = \begin{pmatrix} 0 & 10 & 7 & 7 & 11 \\ 10 & 0 & 9 & 6 & 5 \\ 7 & 9 & 0 & 9 & 10 \\ 7 & 6 & 9 & 0 & 6 \\ 11 & 5 & 10 & 6 & 0 \end{pmatrix}$$

The connectivity of the cities are shown in the figure 3.3. The total number of possible tours for STSP is $(n-1)!/2$; the graph has 5 nodes, and so there are $4!/2 = 12$ tours. In this case the optimal tour is {1,3,2,5,4,1} of weight 34.

3.2 Application of TSP

Since the TSP has a variety of applications in a wide range of areas, such as, drilling of printed circuits boards (PCBs) of 1700 nodes and more [87, 123], robot control [123], VLSI-chip fabrication with as many as 1.2 million nodes [95], X-ray crystallography with up to 14000 nodes [15], over-hauling of gas turbine engines of air-crafts [117], computer wiring and clustering of data arrays [84], mask plotting in PCB

production [55, 95], scheduling and seriation in archaeology, *etc,* and in a typical combinatorial optimization problem.

Cox *et al.* in [22], used the TSP for the radiation hybrid mapping a somatic cell genetic method for constructing high-resolution maps of mammalian chromosomes. Grossman and Troitski in [54, 140] used the TSP to route the laser through the points to minimize the production time.

Carlson used TSP for aiming telescopes and X-rays [19]. Gilmore and Gomory [47] modelled a machine scheduling problem as a TSP. An interesting feature is that the application gives a case of the TSP that can be solved easily. Garfinkel in [45] used the TSP to model a problem in minimizing wall-paper waste. For further detailed applications we refer the reader to [1].

3.3 A Review of Methodologies for the TSP and DTSP

The purpose of this section is to highlight some previous research that has taken place in the area of TSP and to briefly describe some different methods that have been used for the solution of TSP and DTSP. In general the heuristics algorithms for TSP can be classified as [65, 107]:

1. Construction Heuristics.
2. Improvement Heuristics.
3. Population based Heuristics.

A large number of techniques and diverse methodologies have been proposed by many researchers and practitioners for solving TSP. These methods come from a number of scientific communities. In the handbook [98], Michalewicz mentioned

several construction algorithms for solving TSP which are nearest neighbour, greedy algorithm, nearest insertion, farthest insertion and double minimum spanning tree, as well as algorithms by Karp, Litke and Christofides, *etc.*[72, 107].

A well-known tour improvement heuristics was proposed by Lin [86] and further improved by Helsgaun [65]. The 2-opt, 3-opt, Lin-Kernighan(LK) were introduced and are still used for locally improvement by swapping the nodes *i.e.,* in 2–opt 2 edges are exchanged, in 3–opt three edges and so on.

In [65, 95, 98], we find some other well-known meta-heuristic such as Branch and Bound, Simulated Annealing and Genetic Algorithm/Memetic Algorithms which have been used up to near optimality. A brief account of the above is given in the following sections.

3.3.1 Heuristics for TSP

When solving TSP researchers usually use the following two methods for better solutions.

1. Construction Heuristics: These attempt to construct a *"good"* tour from scratch.
2. Improvement Heuristics: These attempt to improve an existing tour, by means of *"local"* improvements.

3.3.2 Construction Heuristics

Nearest Neighbour (NN)

One of the most common heuristics for constructing a tour is the nearest-neighbour. In [126], NN heuristics were discussed and a greedy technique was employed. In this technique, the greedy algorithm constructs a tour by choosing a random city and

then chooses the next node to visit,by taking the one that is nearest to the last one visited. The algorithm stops when all the cities are on the tour. A further extension to this algorithm is to repeat it with each city as a starting point and then return the best tour found so far. The pseudo-code Algorithm 4 and the diagrammatic representation is given below in Figure 3.4.

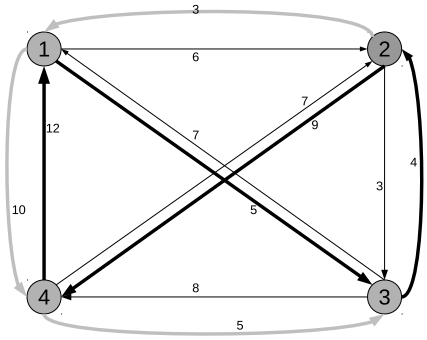


FIGURE 3.4: The Nearest Neighbour tour construction with an ATSP instance. Starting from node 1 moves to node 3 then to node 2, and finally moves to node 4. The total weight is 30 while the best tour is 22.

Algorithm 4 *Nearest Neighbour (NN)*

1: **Input :** $n \times n$ distance matrix d_{ij} and a fixed node i_1;
2: **Output:** $TSP\ tour = \{i_1, i_2 \ldots, i_n, i_1\}$;
3: Initialize $S := \{1, 2, 3, \ldots, n\} - \{i_1\}$;
4: **for** $k = 2$ **to** n **do**
5: choose i_k such that $d_{i_{k-1},i_k} = min_{s \in S}\{d_{i_{k-1},s}\}$;
6: $S = S - \{i_k\}$;
7: **end for**

Greedy (GR)

A second greedy algorithm works on the observation that a vertex-disjoint collection of paths in $\vec{K_n}(K_n)$ can be extended to a tour in $\vec{K_n}(K_n)$. The pseudo code of the greedy algorithm is given below in Algorithm 5.

Algorithm 5 *Greedy* (*GR*)

Input : $n \times n$ distance matrix d_{ij} and a fixed node i_1;
Output: $ATSP(STSP)$ tour as a set S of arcs (edges);
Initialize $S := \emptyset$ and $m = n(n-1)$ for (ATSP) or $m = n(n-1)/2$ for (STSP);
sort all the arcs $a_1, a_2, \ldots, a_m$ in non-decreasing order of weight;
for $i = 1$ **to** m **do**
  if ($S \cup a_i$ is the arc set of a collection of vertex- disjoint paths or is the arc set of a tour) **then**
    $S := S \cup \{a_i\}$;
  end if
end for

Insertion Heuristics

Rosenkrantz *et al.* in [126] proposed insertion algorithms. In these algorithms, a tour is started consisting of an arbitrary city and we then choose, at each step, a city k not yet in the tour. This city is inserted into the existing tour between two consecutive cities, say i and j, such that the insertion cost (the increase in the tour's length) is minimized. The insertion cost for inserting k between i and j is:

$$d(i,k) + d(k,j) - d(i,j)$$

The stopping criterion is when a tour contains all the n nodes. The pseudo-code is given in Algorithm 6.

Let $C = \{i_1, i_2, \cdots, i_m, i_1\}$ be the vertex sequence of a cycle in K_n such that the vertex v is not contained in C. Consider an arc (a,b) on the cycle C (the cycle represents a tour on a subset of cities). If the vertex v is inserted in the arc (a,b), this results in the arcs (a,v) and (v,b). An example is shown in Figure 3.5. The cycle resulting from C by inserting v in the arc (a,b) is denoted by $C(a,v,b)$. Thus, in general, if $(a,b) = (i_k, i_{k+1})$, where $1 \leq k \leq m-1$, then $C(a,v,b) = \{i_1, i_2, \cdots, a, v, b, \cdots i_m, i_1\}$, and if $(a,b) = (i_m, i_1)$, then $C(a,v,b) = \{i_1, i_2, \cdots, i_{m-1}, a, v, b\}$. Common rules which are used in Algorithm 6 in line 6 (*i.e.,* $rule[*]$) to choose the vertex i_s for insertion are given below [59]. For a vertex v and a cycle C in K_n that does not contain v, $d(v,C)$ denotes the distance from v to C, that is, $d(v,C) = min_{x \in V(C)} \{d_{vx}\}$.

Random or arbitrary insertion: The city i_s is chosen randomly from all cities not yet on the tour. This is also termed random vertex insertion (RVI).

Nearest insertion: The city i_s is chosen in each step in such a way that its distance to the cycle C is a minimum, that is, $d(i_s, C) = min_{i \notin V(C)}\ d(i, C)$.

Farthest insertion: In this kind of insertion the city i_s to be inserted is chosen in such a way that its distance to the cycle is maximum, that is, $d(i_s, C) = max_{i \notin V(C)}\ d(i, C)$.

According to [73], all the vertex insertion heuristics mentioned above gives better results for Euclidean TSP.

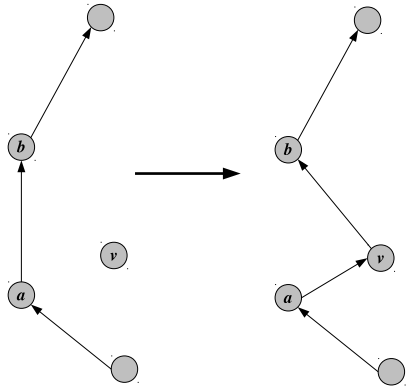


FIGURE 3.5: Insertion of vertex v in between the arc *(a,b)*

Algorithm 6 *Node/Vertex Insertion (VI)*

1: **Input :** $n \times n$ distance matrix d_{ij} and a fixed node i_1;
2: **Output:** $ATSP(STSP)$ tour $\{i_1, i_2, \cdots, i_n, i_1\}$;
3: Let i_1 and i_2 be two vertices of $\vec{K_n}$, selected by some pre-defined rule;
4: Initialize cycle $C = \{i_1, i_2, i_1\}$;
5: **for** $s = 3$ **to** n **do**
6: Let i_s be a vertex not on cycle C, chosen by some rule[$*$];
7: Insert vertex i_s at an arc (a, b) of cycle $C = \{i_1, i_2, \ldots, i_{s-1}, i_1\}$;
8: Such that the weight of $C(a, i_s, b)$ is minimum among the cycles $C(a, i_s, b)$ for all arcs (a, b) in C;
9: $C := C(a, i_s, b)$;
10: **end for**

Minimum Spanning tree tour construction heuristics

The construction heuristics based on MST are the Double Minimum Spanning Tree (DMST) and the Christofides heuristics (CH) [123]. However, the performance of CH is significantly better then DMST both theoretically and experimentally. In addition, the insertion technique using the nearest and cheapest corresponds to the MST algorithm proposed by Prim in 1957 [118]. For TSPs with distances obeying the triangle inequality, the theoretical upper bound is twice of the optimal tour while that of CH is 1.5 of the optimal tour. The Pseudo code of the Christofides heuristic algorithm is given below as Algorithm 7.

Algorithm 7 *Christofides Heuristics (CH)*

1: **Input :** $n \times n$ distance matrix d_{ij};
2: **Output:** $ATSP(STSP)$ tour $\{i_1, i_2, \ldots, i_n, i_1\}$;
3: Compute a minimum spanning tree;
4: Compute a minimum weight perfect matching on the odd-degree nodes of the tree and add it to the tree to obtain a Eulerian graph;
5: Compute an Eulerian tour in the graph;
6: And then finally construct a Hamiltonian tour;

3.4 Tour Improvement Heuristics

Tour improvement heuristics come under the category of local search or neighbourhood search. In these heuristics, the algorithms try to improve an initial tour (normally constructed by some construction heuristics) and iteratively improve it by changing some parts (in case of TSP nodes) at each iteration. These algorithms are also called *"edge exchange"* heuristics. In these algorithms, a tour is improved by replacing k edges (arcs) with k edges (arcs) not in the solution, *i.e.,* inserting k edges and deleting k edges, which are disjoint.

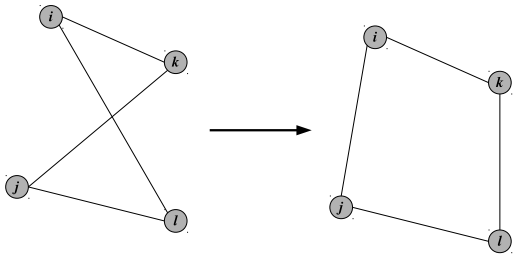


FIGURE 3.6: 2-opt, showing how edges (i, j) and (k, l) replace (i, l) and (j, k)

2-opt 3-opt, or r-opt:

For the TSP, when $k = 2$ the algorithm is 2-opt [23]. The 2-opt algorithm starts from an initial tour T and tries to improve the fitness of the tour $f(T)$ by replacing two of its non-adjacent edges with two other edges to form another tour T'. Once the improvement is obtained it becomes the original tour T. The procedure is repeated until no further improvement is made; this shows that the tour is locally optimal. A diagrammatic representation of 2-opt is given in Figure 3.6.

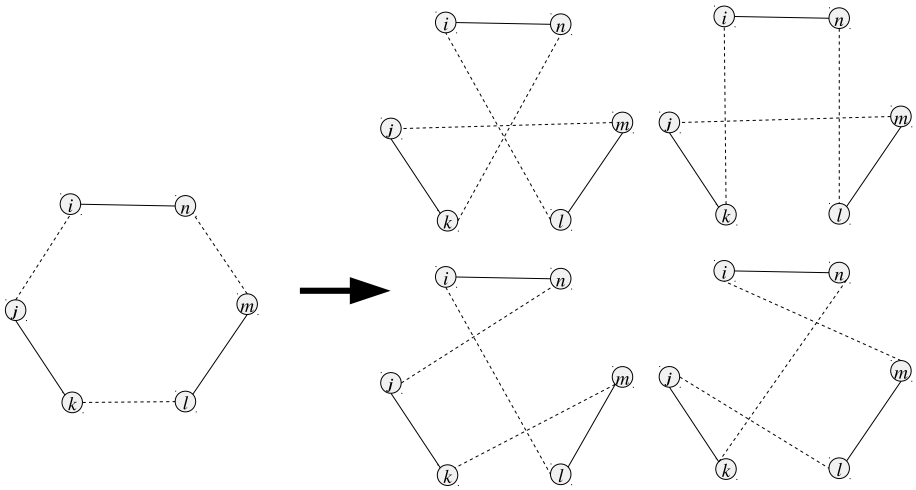


FIGURE 3.7: Examples of 3-opt local search, in which three edges are removed and reconnected.

For $k = 3$, the algorithm is 3-opt [86]. In this case, three edges are removed or exchanged. The rest of the procedure is the same as for 2-opt. When $k = 4$, then the procedure is called double-bridge. In this case, four edges are removed and four different edges are added. Examples of 3-opt are shown in Figure 3.7.

Lin-Kernighan

The Lin-Kernighan (LK) [86] algorithm belongs to the class of local optimization algorithms. The LK does not use fixed k for its $k-opt$ moves, but the choice of k is variable depending on the gain in fitness. The algorithm uses a sequence of $k-opt$ moves of 2-opt in a sequential order. It builds up a sequence of 2-opt moves, checking after each additional move, whether a stopping rule or criterion is met. After this the part of the sequence which gives the gain in fitness is used. This is almost equal to a choice of one k-opt move with variable k; this kind of move is used until the tour T is locally optimal. The other procedure which the LK uses is a kind of full backtracking through which an optimal solution could always be found, but this is computationally expensive. However, to deal with this issue, only limited backtracking is allowed; this helps in better fitness gain. The general working of LK is given below.

Let T be the current tour. At each iteration step the algorithm tries to find two sets of links, $X = \{x_1, x_2, \ldots, x_r\}$ and $Y = \{y_1, y_2, \ldots, y_r\}$, such that, if the links X are deleted from T and the links Y are added to T the result will be a better tour. This interchange of links is called an r-opt move. In order to get efficiency the links X and Y should fulfil one of the following criteria:

1. The sequential exchange criterion.
2. The feasibility criterion.
3. The positive gain criterion.
4. The disjunctive criterion.

The LK local search is still considered to be the state-of-the-art algorithm. LK produces solutions very close to the optimal tour. Although LK can be applied to STSP, it can also be applied to ATSP when transformed to STSP. For the transformation of an ATSP instance to STSP instance, for details refer to the source [70].

Branch and Bound

Eastman [34], Little *et al.* [88] and Shapiro [129] developed the branch and bound algorithm. Furthermore, Hatfield and Pierce [64] used the branch and bound algorithm for the job sequencing problem, which is closely related to the TSP, but more constraints are added because job deadlines have to be satisfied as well. The main difference between the work of Little *et al.* and Shapiro is that the method of Little *et al.* is a tour building algorithm while that of Shapiro is a tour elimination algorithm. The work of Eastman is also an example of a tour elimination algorithm.

The Eastman algorithm is further extended by Shapiro using a kind of search technique in which the set of tours is partitioned into subsets and which then calculate the lower bounds on the cost of all tours in a subset. The initial bound is found by solving the associated assignment problem (AP). The bound is considered as the value of the solution to AP. If the solution of the AP is not feasible to the TSP because of the sub-tours, then k further branches of the problem are made (sub-problems), where k is the number of arcs in one of the sub tours. For example, if the sub tour is $(i_1, i_2, \ldots, i_k, i_1)$ for sub-problem 1, let $c_{i_1,i_2} = \infty$, for sub-problem 2, let $c_{i_2,c_{i_3}} = \infty \ldots$ For sub-problem k, let $c_{i_k,i_1} = \infty$. The subsets then are the set of all tours in which arc (i_1, i_2) is prohibited, *etc.* The branch and bound technique is used successfully for the ATSP case. Shapiro found this approach very difficult for STSP; for this reason he adopted a new approach of integer programming formulation.

The algorithm of Little, *et al.* [88] uses a different approach for branching and bounding. Their approach is referred to as a matrix reduction.

Branch and Cut

Another well known exact algorithm for solving the TSP is based on *linear programming* [24]. The main idea is to find the relaxation in form of a linear program with the same optimal solution as the original problem [95]. For finding the optimum

solution to the (LP) an efficient algorithm, known as the simplex algorithm [24, 25] is used. The simplex algorithm always provides the optimum solution. For TSP, branch and cut showed enormous progress in the last decades. In 1954, Dantzig, Fulkerson and Johnson solved TSP instance up to 48 nodes for which the size of the search space S exceeded 10^{59} [26]. In 1998, Applegate, Bixby, Chvatal and Cook [3] solved TSP instances with up to 13509 nodes for which the size of the search space S exceeded 10^{49931}. The search space is composed of candidate solutions ($|S| = (N-1)!/2$).

Simulated Annealing

Simulated Annealing was first proposed by Metropolis *et al.* [97], and further utilized by Kirkpatrick *et al.* [78] for the process of optimization. The basic working principle of SA is adapted from the condensed physics processes where a low-energy state of a solid is sought by an annealing process. The analogy with combinatorial optimization arises when we consider that the optimal solution to the combinatorial optimization problem corresponds to the lowest energy of the solid, and the optimization process with that of annealing the solid. Thus, SA perturbed the solution of the problem and the neighbour solution is accepted with the probability according to Boltzman distribution, which is given below:

$$P(accept\ uphill\ moves) = exp(\tfrac{-\triangle E}{e^{k*t}})$$

Here $\triangle E$ represent the two solution states $f(s') - f(s)$, k is the scaling parameter and t is the temperature of the process. When t is high, the solution is more perturbed, *i.e.*, it allows moves resulting in solutions of worse quality than the current solution. When the temperature is low then the search process accepts the improving moves only. When the t is further decreased it possible to freeze the annealing schedule. The pseudo code is shown below in Algorithm 8

Algorithm 8 *Simulated Annealing* ($s \in S$)

1: $t = T(0), n = 1$;
2: Best solution $s_{best} = s$;
3: **repeat**
4: Generate $s' \in N(s)$;
5: $\triangle E = f(s) - f(s')$;
6: **if** $(\triangle E \leq 0)$ OR $(exp(\frac{-\triangle E}{e^{k*t}}) > random[0,1])$ **then**
7: $s = s'$;
8: **end if**
9: **if** $f(s) > f(s_{best})$ **then**
10: $s_{best} = s$;
11: **end if**
12: $t = T(n)$;
13: $n = n + 1$;
14: **until** (Termination Criteria fulfilled)
15: Return s_{best};

Tabu Search

The basic idea of Tabu Search method was described by Glover, Taillard and de Werra [48]. One of the drawbacks of the neighbourhood search algorithm is the greedy approach; not accepting the less greedy moves results in the optimization process getting stuck in local optima. The Tabu-Search will also allow moves with the negative gain criteria by not getting positive gain criteria. Thus, tabu-search escapes from locally optimal but not globally optimal solution, and, to prevent considering the moves which cause cycling, moves are stored in a list which is called Tabu list. The moves of Tabu search are based on the short-term and long-term memory of the sequence of the moves that were used to achieve the certain state of the search. In most of the cases, the whole solution is marked as tabu, but a tabu move can be overridden if it satisfies the criteria which are called *aspiration criteria*. By this criterion, we might include the case which, by removing a move from the tabu list, may lead to a solution which is the best obtained so far. A big problem with tabu search is that it is computationally expensive. The pseudo code of simple tabu search is given below in Algorithm 9.

Algorithm 9 *Tabu Search* ($s \in S$)

1: Tabu list $T = 0$;
2: Best solution $s_{best} = s$;
3: **repeat**
4: Find the best solution $s' \in N(s), s' \notin T$;
5: $\triangle f = f(s) - f(s')$;
6: $T = T \cup s$;
7: **if** $f(s) > f(s_{best})$ **then**
8: $s_{best} = s$;
9: **end if**
10: **until** (Termination Criteria fulfilled)
11: Return s_{best};

3.5 Population Based Heuristics

Researchers are often trying to imitate nature when solving complex problems. Two such methods are Ant Colony Optimization (ACO) and Genetic Algorithms.

ACO is an optimization methodology based on ant behaviours. The term ACO system was first introduced by Colorni *et al.* in 1992 [20]. Gambardella and Dorigo in [44] implemented ACO on the TSP, and also the quadratic assignment problem.

The TSP has become one of the standard test bed problems for the GAs community. In [82, 98], various encoding schemes for TSP were proposed such as binary, adjacency, matrix, ordinal and path. The path representation is the most natural one and it is widely used by researchers and practitioners. In GAs for the TSP the chromosome representing the route is a permutation of the N cities. The population size is known in advance. First the population is initialized, then some subset of individuals is selected from the population by using a selection scheme and then crossover and then mutation is performed on them.

It is now well established that pure GAs are not well suited for fine tuning the search. So the use of local search operators has been introduced into traditional operators of GAs; therefore the resulting algorithms are called hybrid GAs or simply Memetic

Algorithms which we have explained in previous sections. We can also conclude from the study of Grefenstette [52] that:

> *"Finally, it is widely recognized that GAs are not well suited to performing finely tuned local search Once the high performance regions of the search space are identified by the GA, it may be useful to invoke a local search routine to optimize the members of the final population."*

A wide range of representation schemes have been proposed for encoding the TSP tour. It is now well understood, in the GA community, that the binary representation of tours is not well suited for the TSP. The reason is that the binary representation would require special repair algorithms, since a change of a single bit may result in an illegal tour. As observed in [145]:

> *"Unfortunately, there is no practical way to encode a TSP as a binary string that does not have ordering dependencies or to which operators can be applied in a meaningful fashion. Simply crossing strings of cities produces duplicates and omissions. Thus, to solve this problem some variation on standard genetic crossover must be used, the ideal recombination operator should recombine critical information from the parent structures in a non-destructive, meaningful manner"*

Some genetic operators for solving TSPs are mentioned in [82, 98, 109].

3.5.1 Crossover Operators in GAs for TSPs

Partially mapped Crossover (PMX) was suggested by Goldberg and Lingle and explained in [82]. PMX falls into the category of blind operators. First it selects two cutting points in both parents. In PMX a portion of one parent is mapped to the corresponding portion of another parent.

Another variation of PMX is Maximal Preservative Crossover (MPX) proposed by Muhlenbein *et al.* in [102]. It first selects a portion of string whose length is at most the problem size divided by 2. This constraint on the length of the sub-string assures that there is enough information exchange between the parent strings without losing too much information from any of the parents. Further, all the nodes of the chosen string are removed from the other parent, *i.e.,* the second parent. After this the nodes chosen from the first parent are copied to the first part of the child and the end of the child is filled with cities in the same order as they appear in second parent [82], MPX preserves maximum distance from both the parents and loses a limited number of edges.

In [29], Order Crossover–1 (OX1) was proposed by Davis and Order Based Crossover–2 (OX2) by Syswerda [134] . The main idea behind order crossover is that it is the order of the cities which is important not their position. In OX1 we select a set of cities and preserve the relative order of the other parent, while, in OX2 random cities are selected. This operator is relatively fast and easy to implement.

In [105], Nagata and Kobayashi introduced a powerful Edge Assembly Crossover (EAX) which was able to solve large scale TSP instances.

In [138], Tao and Michalewicz introduced a unary operator called the Inver-over Operator (IO) which has the features of both mutation and crossover. Inver-Over is considered to be one of the fastest operators which has been introduced so far.

In [96], Merz and Freisleben introduced a Distance Preserve Operator (DPX) with local refinement by using LK search.

In [82], Larranaga *et al.* introduced an Alternate Position Crossover (AP) by selecting alternately, the next element of the first parent and the next element of the second parent into the child but not considering those nodes which are already present in a child.

Other crossover operators mentioned in [82] are Cycle Crossover (CX) proposed by Oliver in 1987, Smith and Holland in 1987, Voting Recombination Crossover (VR) proposed by Muhlenbein in 1989, and Heuristic Crossover (HEU) proposed by Grefenstette in 1987. For further details we refer the reader to [109].

Many hybrid GAs have been proposed for the TSP. A hybrid GA is also called a memetic algorithm or genetic local search (GLS). GLS can combine the global search ability of the GA with local search ability of heuristics. Tsai *et al.* [141] proposed a new hybrid GA for TSP, called the heterogeneous selection evolutionary algorithm (HeSEA) by integrating the edge assembly crossover (EAX) [104, 105] with Lin-Kernighan (LK) [86] through family competition and heterogeneous pairing selection.

Nguyen *et al.* proposed a hybrid GA which is based on a parallel implementation of a multi population steady-state GA involving variant of maximal preservative crossover, double bridge move mutation and Lin-Kernighan heuristics as a local search [106].

Xie and Liu in [149] proposed a nature-inspired method which is called the multi agent optimization system (MAOS), which supports cooperative search by the self-organization of a group of compact autonomous agents placed in an environment with certain public sharing of limited declarative knowledge.

3.5.2 Mutation Operators in GAs for TSPs

Mutation operators play a vital role in introducing diversity. In [68], Holland proposed Simple Inversion Mutation (SIM). In SIM two random points are selected with in a path and inverted. In [98], Michalewicz proposed Displacement Mutation (DM). In this operator a sub-path is selected and then inserted in another location.

In [40], Fogel introduced Insertion Mutation (ISM), which works like DM but reverses the selected path. In [134], Syswerda proposed Scramble Mutation (SM); this scheme selects a random tour and scrambles the nodes.

3.6 Evolutionary Approach for DTSP

If in the TSP problem, the cost between nodes, or the number of node changes with time then such a kind of TSP problem is termed as a dynamic TSP (DTSP). A *"dynamic"* TSP has some additional properties such as the fact that the number of nodes or cities n may change with time: some cities may appear and some old ones may disappear. Secondly, the city location changes geographically or the cost matrix may change with time as well. Mathematically for dynamic TSP the cost matrix could be formulated as:

$$D(t) = d_{ij}(t)_{n(t)\times n(t)}$$

where $d_{ij}(t)$ is the cost from city i to city j at time t, $n(t)$ is the number of cities at time t, and t is the real-world time. So, we want to find a tour $\pi(t) = \pi_1, \pi_2, \ldots, \pi_{n(t)}$ which minimizes

$$l(\pi)(t) = \sum_{i=1}^{n(t)} d_{\pi_i,\pi_{i+1}}(t)$$

where $l(\pi)(t)$ is a permutation over the set $\{1, 2, \ldots, n(t)\}$ and $\pi_{n(t)+1} = \pi_1$

The DTSP is a more challenging (and realistic) problem as compared with the static version.

Many real-world problems are dynamic in nature such as in the fields of communication, routing choice, robot control and vehicle routing. For example, a distributing salesman/vehicle wants to distribute some goods to different cities from one starting point. He has to visit all the cities in a suitable order regarding time and energy. For the same reason,the salesman must choose an optimal path, but it may happen

that, with the passage of time, the salesman must make some changes, *i.e.,* he may skip some cities or, due to traffic jams, may alter the pre-existing route; in other words, the salesman must re-route his plan.

In recent years, the DTSP is a hotspot of research in the field of dynamic optimization. One of the variants of DTSP, the dynamic vehicle routing problem, was first put forward by Psaraftis in 1988 [120]. His work was mainly focused on defining the problem, designing the algorithm, performance estimation and test-bed construction. The DTSP can be defined as the TSP problem whose distance matrix will change with time. For example, the change, in the distance matrix results from inserting a node, deleting a node or changing the node position.

A wide variety of algorithms have been proposed for the dynamic TSP, such as ant algorithms [17, 60–62], competitive algorithms (on-line algorithms) [9] and dynamic Inver-over evolutionary algorithms [153]. Yang *et al.* in [151] proposed the concept of multi-objective approach for solving DTSP based on Pareto optimality, partially mapped crossover (PMX) and IO. Mavrovouniotis and Yang in [94] proposed a memetic algorithm for DTSP which is based on population ACO framework and adaptive Inver-Over; they also utilized the concept of random immigrants to address the premature convergence.

In the following text we will discuss why EAs are effective for solving the DTSP. We will then focus on Memetic Algorithm because MAs can overcome some of the deficiencies of other EAs for the TSP and DTSP. We propose MAs with advanced and fine-tuned local search methods, promising results can be achieved for TSP and DTSP in terms of speed, quality, and diversity. MAs use their exploitive search ability to improve the explorative search ability of GAs.

3.6.1 Why Is the Evolutionary Approach Suitable for DTSP?

As mentioned in [153], the principal objective for a dynamic-problem-solving algorithm for problems including the DTSP is the speed, that is, we want to solve dynamic problems in real time. In [153] the drastic change and gradual change in time are mentioned. For the DTSP with a drastic changing cost matrix $D(t)$, $D(t_i)$ may be very (even completely) different from $D(t_{i-1})$. For such DTSP we can hardly use the information in the time window $[t_{i-1}, t_i]$ to speed up the optimization process in $[t_i, t_{i+1}]$. The problem may degenerate to a series of static optimization problems which have no relationship with each other if the change is drastic enough, (say random change). For a gradual changing of matrices, which is a very common situation in the real world, EAs are suitable for solving the DTSP with gradual changing matrices for the following two reason [153]:

1. EAs make use of knowledge of the current population and use it for the next generation. Simply, the next generation is dependent on the previous generation up to some extent (or completely).

2. The population policy has the potential for the individuals to retain diverse information.

The goal of this study is to overcome up to some extent the above two reasons for designing efficient evolutionary algorithms for DTSP. Most of evolutionary algorithms for the static TSP can be modified into DTSP algorithms by integrating the newly designed operators so that the static and dynamic approaches synergistically work together and perform well; such an approach has already been applied in Dynamic Inver-over evolutionary algorithm (DIOEA) [153]. DIOEA was further extended by Lishan and Li; for further details see [75, 85]. Obviously, the performance of the base static TSP algorithm is very important for the performance of the DTSP algorithm. In DIOEA the base Inver-Over operator is very efficiently used and it is converted into a dynamic one by introducing three dynamic operators INSERT, DELETE and

CHANGE and they are combined together into DIOEA. A brief description of these algorithms is given below, for more details see [138].

From the above description, we have indicated that evolutionary algorithms for static TSP can be modified into DTSP algorithm by integrating the dynamic operators. However, this research is still in an embryonic stage as to how to trade off the optimization speed and the qualities of the results (in most cases, the speed is probably more important than the quality); this is still an open question (and will be for real time speed).

1. It has been also observed from the above approach [138] that, in order to exploit the useful information of the previous population as fully as possible to accelerate the optimization process in the new environment, it is important to adopt MAs. A memetic recombination operator can exhibit several properties. One of the most important properties of a memetic algorithm is that it can incorporate domain-specific knowledge into EAs to make the search effective [52, 96]. There are possibilities to achieve this hybridization with other local search techniques. MAs with Baldwinian effect and Lamarckian evolution [42, 146] can better exploit the neighborhood. In the Baldwinian effect, the improvements or changes made by the parent are not saved in the individual, thus the learned traits during the lifetime of the parents are not inherited by their offspring. This property shows similarity with the natural evolution in which the acquired characteristics or traits of the individuals during the training in their lives have no effect on the genetic makeup. In conflict to this form of evolution, in Lamarckian evolution the acquired traits of an organism influence the genetic code of the organism [95]. However, it may be possible that the acquired traits which an individual obtained during its life time as a form of learning may affect the newly created individuals, which may be totally different from the previous generation.

2. In case of maintaining population diversity, robustness is a key element in the development of any meta-heuristic. Hence a method that shows an effective. behaviour, namely attaining very near optimal solutions in short CPU time regardless of the instance size (and the particular type of instance) being solved, is highly desirable. Compared with other GAs already proposed for the TSP, the population size in DIOEA [153] for DTSP is remarkably small, but this makes it possible to perform a much larger number of generations in the same time span. Since, however, the population is likely to lose diversity rapidly, the MA implementation here makes use of several specific procedures. It has been known that MA must overcome the issue of premature convergence [95]. This is usually prevented by two means, both of them termed disruptive [95, 100]: restarting the population and keeping high mutability rates. While both approaches can be used together, they implement fundamentally different ways in preventing convergence.

3. To trade off the optimizations speed and quality of the results, a kind of fast local search techniques can be designed which can speed up the process to some extent by designing Guided Local Search (GLS) [142] and Fast Local Search (FLS) [142]. As GLS sits on top of local search heuristics and has, as a main aim, to guide these procedures in exploring (efficiently and effectively) the vast search spaces of combinatorial optimization problems such as TSP. GLS can be combined with the neighbourhood reduction scheme of FLS which significantly speeds up the operations of the algorithm [142].

3.7 Conclusion of the Chapter

In this chapter, the historical background of the travelling salesman problem has been addressed. The general definitions of TSP and the mathematical treatment have been discussed. The most important issue is why TSP has remained a test bed

by the practitioners and researchers from different directions. There is a plethora of solving techniques for handling the TSP (and other variations of TSP); to discuss everything is beyond the scope of this text. Solving techniques such as exact and approximate ones including Branch and Bound, Branch and Cut, and many more were briefly discussed. With regard to approximate algorithms, various construction and improvement heuristics like 2-opt, 3-opt and the state-of-the-art Lin-Kernighan algorithm along with other well-known local search techniques such as Simulated Annealing and Tabu search were discussed as well.

Moreover, population-based heuristic which can now compete with exact techniques by giving near optimal solutions were also briefly discussed. Many crossover and mutation techniques which bring life to GAs were discussed. The transition of GA algorithm to MAs regarding the integration of local search techniques was put forward. The Dynamic Travelling Salesman Problem and the reasons why EAs are well suited for the treatment of various issues such as the use of previous knowledge and maintaining diversity are briefly explained.

Finally, the applications of TSP are described briefly. For further study, please refer to [1]. Interested readers can also refer to the TSP website (http://www.tsp.gatech.edu).

In the coming chapter, we will present an overview of our Sequence Based Memetic Algorithm. It is a sub-tours approach which is extracted from the population and stored in a memory. These sub-sequences or sub-tours are further used to guide the genetic search. We have also introduced a Local Search, which inserts the sequences into a proper location in the genome.

Chapter 4

A Sequence Based Memetic Algorithm for the Travelling Salesman Problem

4.1 Introduction

It is common that a standard genetic algorithm (GA) often suffers from slow convergence for solving combinatorial optimization problems. In this chapter, we present a sequence based memetic algorithm (SBMA) for solving the travelling salesman problem (TSP). SBMA uses a reverse approach of fragment assembly in DNA sequencing. In DNA sequencing, all possible base pairs of genome are put together in pieces that match and the sequence becomes bigger and bigger [113, 115]. In the proposed SBMA, the reverse approach occurs. In our proposed SBMA, a set of sub-tours, called *sequences*, are extracted from the top individuals, which are used to guide the search of SBMA. Some of the additional characteristics of SBMA are the following:

1. According to the building-block hypothesis, the current SBMA approach discovers (by generating the sub-tours or sequences) and puts together (by inserting the sequences back into the genome, and also in the second phase by combining the genes by considering the knowledge-base of the whole population) blocks which are in some sense part of global optima or near optima [36, 98].

2. The current approach also shows a crossover of multiple parents. The sequence which is extracted before the crossover (also in mutation as well) has been extracted from another parent which has come from the set of better individuals (having comparatively better fitness). In general, two parents are selected to mate, which is the binary approach, while the sequence which is inserted back is part of a third individual. So the approach is in a sense a ternary one.

In our proposed SBMA, first, a set of best individuals are selected from the population. The individuals are broken into sub-tours of equal size that have the same number of cities. The sub-tour with the shortest length is selected, further optimized by a 2-Opt improver [86], and then stored in a sequence set. This set of sequences are further used to guide the crossover, mutation, and local search operators. A random sequence is selected from the set of sequences for crossover, mutation, and local search in every iteration. Similar work has been done by Ray *et al.* in [121], where an individual is broken into parts and then reconnected in a random way. Additionally, some procedures are applied in SBMA to maintain the diversity by reducing the size (by removing some nodes) of the selected sequences into sub-tours if the best individual of the population does not improve.

The proposed SBMA is compared with the inver-over algorithm [138], a state-of-the-art algorithm for the TSP, on a set of benchmark TSP instances. Experimental results show that SBMA is superior to the inver-over algorithm in terms of the convergence speed and achieves a similar solution quality as the inver-over algorithm on the test TSP instances.

4.2 Sequence Based Memetic Algorithm (SBMA) for the TSP

In this section, the proposed SBMA for solving the TSP is described in detail. Several new operators, like the Sequence Based Local Search (SBLS), Sequence Based Order Crossover (SBOX), and Sequence Based Inversion Mutation (SBIM) are presented. The structure of the proposed SBMA is shown in Algorithm 10.

Algorithm 10 Sequence Based Memetic Algorithm (SBMA)

1: Initialize adaptive parameters (e.g., p_c, p_m, n_{inv}, s_{seq}, and $SBLS_{stepsize}$) to their default values;
2: Pop := Initialize a population of $popsize$ random individuals;
3: **for** each individual $ind_i \in Pop$ **do**
4: ind_i := 2-Opt(ind_i, n);
5: **end for**
6: $GenerateSequence(N_{seq})$;
7: **repeat**
8: $mating\_pool$:= perform tournament selection from Pop;
9: **//Crossover**
10: **for** $j := 0$ to $popsize$ **do**
11: Select two parents i_a and i_b from the $mating\_pool$;
12: **if** $rand(0,1) < p_c$ **then**
13: Create $child_a$ and $child_b$ by $SBOX(i_a, i_b)$;
14: Apply $SBLS(child_a)$ and $SBLS(child_b)$;
15: Add $child_a$ and $child_b$ to Pop_{tmp};
16: **end if**
17: **end for**
18: **//Mutation**
19: **for** each individual $ind_i \in Pop_{tmp}$ **do**
20: **if** $(rand(0,1) < p_m)$ **then**
21: $SBIM(ind_i, n_{inv})$;
22: **end if**
23: **end for**
24: $AdaptParameters()$;
25: $Pop := SelectNewPop(Pop + Pop_{tmp})$;
26: **until** $(Termination\ condition = true)$

The first step of SBMA is to initialize a population of $popsize$ random individuals. A simple 2-Opt improver [86] is applied to each initial individual ind_i for K iterations

to give a nice start to SBMA. The 2-Opt improver simply takes an individual and tries K random swaps in the hope to improve the individual. For each swap tried, we calculate the gain in fitness $Fitness_{Gain}$, *i.e.,* the difference of the fitness before and after the tried swap. If the fitness gain is greater than zero, *i.e.,* an improvement is found from swapping two nodes, the swap is accepted; otherwise, it is rejected. The pseudo-code of the 2-Opt improver is given in Algorithm 11. Note that the 2-Opt improver is called with different values of K in different places of SBMA. For example, K is set to the number of nodes of the individual (*i.e.,* the problem size) after the initialization of the population, while in the sequence-based local search (SBLS) scheme (to be described later on), K is the step size of SBLS, which is an adaptive variable adjusted by the diversity maintaining mechanism (to be described later on). The $Fitness_{Gain}$ can be calculated as:

$$\begin{aligned} remove &= d(i, i+1) + d(j, j+1) \\ add &= d(i, j) + d(i+1, j+1) \\ Fitness_{Gain} &= add - remove \end{aligned}$$

In the above statements for calculating the $Fitness_{Gain}$, some edges are removed and some are added. If $Fitness_{Gain} < 0$ then improvement has been found and nodes would be arranged accordingly. Here, $d(i, j)$ denotes the distance between node i and node j.

Algorithm 11 2-Opt(ind_i, K)

```
1: for i := 0 to K do
2:    Select two random edges within ind_i;
3:    Swap the positions of the two selected edges;
4:    Calculate the fitness gain FitnessGain from the swapping of the two edges;
5:    if FitnessGain > 0 then
6:       accept the swap;
7:    else
8:       reject the swap;
9:    end if
10: end for
```

After the initialization of the population, SBMA evolves the population generation by generation. For each generation, N_{seq} sequences are first generated from a set of best individuals that are selected from the population. Then, a mating pool is generated using the tournament selection with a certain tournament size. Thereafter, crossover is carried out with the guidance of the set of sequences, followed by sequence based local search and mutation operations. The details of each operation are given in the following subsections, respectively.

4.2.1 Sequence Generation

In each generation of SBMA, the first step is to generate a set of sequences to be stored in an archive, also called *memory*. The procedure of generating sequences is as shown in Algorithm 12. For the sequence generation, a number of the best individuals from the population are selected. For each of these selected best individuals, we generate a sequence as follows.

Algorithm 12 *GenerateSequence*(N_{seq})

1: Select N_{seq} best individuals from Pop and store them into a set $Best_{indi}[\,]$;
2: **for** $i := 0$ **to** $N_{seq} - 1$ **do**
3: Break $Best_{indi}[i]$ cyclically into n equal sub-tours, of which each sub-tour has s_{seq} nodes;
4: Calculate the length of each sub-tour;
5: Further optimize the sub-tour with the minimum length by a 2-Opt improver with K set to s_{seq};
6: Store the sub-tour into the set of sequences, i.e., the memory;
7: **end for**

First, a node in the individual solution is selected as the initial node. Then, the individual is broken into equal sequences (sub-tours) with the same number of nodes in each sequence as follows: starting from the initial node ($i = 0$), we slide a scale of size s_{seq}, one node a time to get a sub-tour, until we come back to the initial node. Here, s_{seq} represents the number of nodes in a sub-tour (*i.e.,* the size of the sequence). In this breaking procedure, one node comes into the scale and one goes

out cyclically. So, every node participates. This way, we will get n sub-tours in total, where n is the number of cities in the problem. Finally, we find the shortest sub-tour among the candidate sub-tours, apply the 2-Opt improver to further optimize it, and store it in the memory.

For example, given an individual ABCDEFGHIJKLMNOP and the number of nodes of a sequence $s_{seq} = 4$, then the candidate sub-tours in this individual are **ABCD**, **BCDE**, ..., **OPAB**, and **PABC**. Let's suppose the shortest sub-tour is **BCDE**. It is then further optimized by the 2-Opt improver to, say, $CDBE$. Finally, the sequence **CDBE** is stored in the memory for our future use.

According to the above description, the number of sequences generated equals the number of best individuals that are selected from the population to generate these sequences. If all individuals from the population are used, then the total number of sequences would be equal to the population size.

4.2.2 Sequence Based Order Crossover (SBOX)

The Order Crossover (OX) operator [29, 82] is a sexual reproduction operator. It is a variant of the "two-point crossover". It is a classical "blind" heuristic, which does not depend on the local city-to-city distance information, but only on the global "whole genome" fitness to achieve progress. It is observed to be one of the best in terms of quality and speed, and is simple to implement.

Our modified OX operator, SBOX, works as follows. First, a random sequence S_{sel} is selected from the set of sequences. Two individuals are selected from the *mating_pool*, which is created through the tournament selection as mentioned above. If the crossover condition is satisfied, then the nodes within the selected sequence are removed from both the parents. Then, the available number of nodes for crossover is $(n - s_{seq})$. The pseudo-code of SBOX is given in Algorithm 13.

Algorithm 13 $SBOX(i_a, i_b)$

1: S_{sel} := randomly select a sequence from the memory;
2: Remove the nodes of S_{sel} from individuals i_a and i_b, resulting in $i_{a_{tmp}}$ and $i_{b_{tmp}}$, respectively;
3: Perform Order Crossover on i_{atmp} and i_{btmp} to generate $child_a$ and $child_b$, respectively;
4: Re-insert S_{sel} into $child_a$ and $child_b$ at a random location, respectively;
5: Evaluate $child_a$ and $child_b$;

The following example shows how the above algorithm is implemented. Let $P1$ = ABCDEFGHIJKLMNOP and $P2$ = PONMLKJIHGFEDCBA represent two parents, *i.e.,* i_a and i_b, and $C1$ and $C2$ represent the two children, *i.e.,* $child_a$ and $child_b$, respectively, and assume the sequence S_{sel} to be selected is (CDBE). The crossover performs as follows:

Before crossover

$P1$ = ABCDEFGHIJKLMNOP

$P2$ = PONMLKJIHGFEDCBA

After removing the sequence (CDBE)

$P1_{temp}$ = AFG | H I J K | LMNOP

$P2_{temp}$ = PON | M L K J | IHGFA

After order crossover (OX)

$C1_{temp}$ = GFAHIJKPONML

$C2_{temp}$ = NOPMLKJAFGHI

After inserting (CDBE) at a random location

$C1$ = GFAHIJK**CDBE**PONML

$C2$ = NOPMLKJA**CDBE**FGHI

4.2.3 Sequence Based Inversion Mutation (SBIM)

After crossover, each offspring undergoes mutation with a small probability p_m. For the TSP, the Simple Inversion Mutation (SIM) operator is one of the best performers [82]. In our SBIM approach, we perform inversions on an offspring, after the removal of a selected sequence from the offspring, for a certain number of times, and preserve those inversions which have positive effect on the performance. This increases the convergence speed although involving an extra overhead on the mutation operator. The pseudo-code of SBIM is shown in Algorithm 14.

Algorithm 14 $SBIM(i_m, n_{inv})$

1: S_{sel} := randomly select a sequence from the memory;
2: S'_{sel} := reverse selected S_{sel};
3: i_{temp} := remove the cities in S_{sel} from individual i_m;
4: **for** $i := 0$ **to** n_{inv} **do**
5: Randomly select two points, denoted p_1 and p_2 such that $p_1 < p_2$;
6: i'_{temp} := invert cities in between points p_1 and p_2 of i_{temp};
7: **if** $f(i'_{temp}) < f(i_{temp})$ **then**
8: $i_{temp} := i'_{temp}$;
9: **end if**
10: **end for**
11: Insert S'_{sel} into i_{temp} at a random location;
12: Evaluate i_{temp};
13: $i_m := i_{temp}$;

The overall procedure of SBIM is similar to that of SBOX. The difference lies in that SBIM inverts the sequence before inserting it in the individual. Here, SBIM will be executed as follows. First, a random sequence S_{sel} is selected from the memory, which is inverted into S'_{sel}, and the nodes in S_{sel} are removed from the individual i_m to be mutated, resulting in an partial individual i_{temp}. Then, we carry out n_{inv} inversions on the partial individual i_{temp}. For each inversion operation, two random positions in i_{temp} are first selected and the sequence of those nodes in between these two selected positions is then inverted, which transfers i_{temp} into i'_{temp}. Then, the fitness $f(i'_{temp})$ of i'_{temp} is calculated. If it is better than the fitness $f(i_{temp})$ of the individual before inversion, the inversion is made permanent; otherwise, the inversion

is rejected. After the inversion iterations, S'_{sel} is re-inserted into i_{temp} at a random location. Our approach tries to produce fruitful individuals as the sequence S_{sel} to be inserted is optimized.

Below, we use a simple example to illustrate the mutation operation. We denote the individual before mutation as $P = ABCDEFGHIJKLMNOP$ (*i.e.,* i_m in SBIM), the randomly selected sequence as $ECDB$, and the child as C (*i.e.,* i_{temp}), and assume the number of inversions $n_{inv} = 1$. The mutation procedure is shown as follows:

Before mutation
P = ABCDEFGHIJKLMNOP
After removing (ECDB) and selecting two random positions
P_{temp} = AFG | HIJKLMN | OP
After inversion
C_{temp} = AFG | MNLKJIH |OP
After re-insertion of the inverted sequence
C = AFGMNLKJIHBDCEOP

4.2.4 Sequence Based Local Search (SBLS)

Local search algorithms are effective heuristic techniques for a lot of combinatorial optimization problems [96]. In our approach SBMA, information is gathered from various good individuals and stored in the set of sequences, which is used in the SBLS to guide the generation of children towards promising area of the search space. The pseudo-code of SBLS on a given individual X is shown in Algorithm 15.

In the algorithm, L_{inc} is calculated for the insertion of the sequence in a location, where the increase of length in the route is the minimum. The SBLS scheme simply takes the i-th individual and the sequence which is going to be inserted S_{sel}. In the above algorithm, the distance between the first and last node of the sequence is

Algorithm 15 $SBLS(X)$

1: S_{sel} := randomly select a sequence from the memory;
2: X' := remove the nodes of S_{sel} from individual X;
3: X' := 2-Opt$(X', SBLS_{stepsize})$;
4: Find the best position *best_position* of X' which gives the minimum length increase after inserting S_{sel}, according to the following equation:

$$L_{inc} = Min_{j=0}^{j=n-M}(dist[S_{sel}[0]][X'_j] + dist[S_{sel}[M-1]][X'_{j+1}] - dist[X'_j][X'_{j+1}]);$$

where n is the total number of cities of the test problem and M is the number of cities of S_{sel}
5: Insert S_{sel} into X' at position *best_position*;
6: $X := X'$;
7: Evaluate X;

calculated from the distance matrix relevant to the adjacent nodes of the individual where the sequence will be inserted. The SBLS procedure in the SBMA is quite simple, just like the crossover and mutation operators explained before. Similarly, the nodes of the selected sequence are removed from the individual and the 2-Opt improver is applied to the rest of nodes with the step size $SBLS_{stepsize}$ for possible gain in the fitness. Finally, the best location is searched and S_{sel} is inserted in that location. In SBLS, the step size $SBLS_{stepsize}$ shows how many random swaps SBLS will perform in the 2-Opt improver and hence controls the intensity of local search.

4.2.5 Adaptive Parameters

A number of researchers have recommended different constant values for key parameters in EAs in order to find good solutions for a particular fitness landscape [11, 53, 71]. These parameter settings are derived from experience or by trial-and-error, and are fixed before the execution of algorithms. However, it is very difficult, if not impossible, to find appropriate parameter settings for the optimal performance of EAs, and the approach of finding proper parameters is also time-consuming. Moreover, static values are eventually discouraged by the EA researchers because different values of parameters and different operators may be suitable at different stages of

the evolutionary process of an EA. So, no common optimal parameter setting can be found initially [148]. In order to address this deficiency, researchers have diverted their attention towards adapting the parameters during the evolutionary process for finding better solutions to the problem in hand.

Maintaining the diversity of the population throughout the run is a major approach to avoiding the premature convergence problem [125]. This section describes some techniques used in SBMA to adapt some key parameters to prevent the premature convergence to local optima. These considerations work by avoiding the loss of genetic diversity of the whole population, and in principle, will not damage the convergence process. The pseudo-code of adapting parameters in SBMA is shown in Algorithm 16. The relevant parameters are adapted as follows.

Algorithm 16 *AdaptParameters*()

if (the best fitness changes) **then**
  Reset $SBLS_{stepsize}$, n_{inv}, p_c, and p_m to their default values;
  Reset $MaxLS_{running} := MaxLS$;
  $s_{seq} := \lfloor \sqrt{n} \rfloor$;
  $GenerateSequence(N_{seq})$;
else
  $n_{inv} := min\{n_{inv} + 1, s_{seq}\}$;
  $p_c := min\{p_c + \delta_{p_c}, 1.0\}$;
  $p_m := min\{p_m + \delta_{p_m}, 1.0\}$;
  if $SBLS_{stepsize} > SBLS_{maxstepsize}$ **then**
    Reset $SBLS_{stepsize}$ to its default value;
    Decrease $MaxLS_{running}$ by 1;
  else
    $SBLS_{stepsize} := SBLS_{stepsize} + 5$;
  end if
  if $((0.5 \times MaxLS) < MaxLS_{running} \leq (0.75 \times MaxLS))$ **then**
    $s_{seq} := \lfloor 0.75 \times \sqrt{n} \rfloor$;
  else if $((0.25 \times MaxLS) < MaxLS_{running} \leq (0.5 \times MaxLS))$ **then**
    $s_{seq} := \lfloor 0.5 \times \sqrt{n} \rfloor$;
  else if $(0 < MaxLS_{running} \leq (0.25 \times MaxLS))$ **then**
    $s_{seq} := \lfloor 0.25 \times \sqrt{n} \rfloor$;
  else if $(MaxLS_{running} \leq 0)$ **then**
    $s_{seq} := 2$; // that is, a sequence becomes an edge
  end if
end if

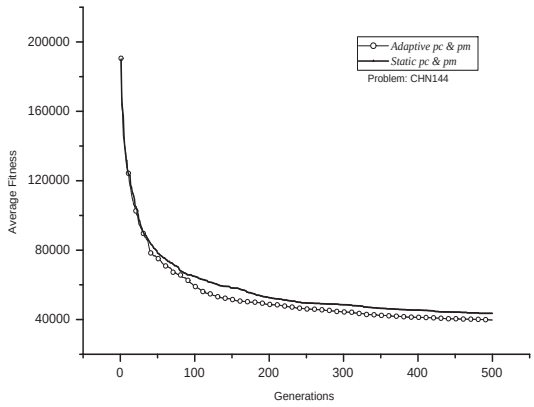


FIGURE 4.1: Showing the effect of static p_c and p_m against adaptive p_c and p_m.

Firstly, the crossover probability p_c and mutation probability p_m are adapted according to whether the best fitness of the population changes or not after a generation. If the best fitness changes, p_c and p_m are reset to their default values; otherwise, a small value δ_{p_c} and a small value δ_{p_m} will be added to p_c and p_m, respectively, which are both upper bounded to 1.0. The effect using adaptive p_c and p_m is shown in Figure 4.1.

Secondly, the number of inversions, *i.e.,* n_{inv}, within an SBIM operation is dependent on whether the best fitness changes. If the best fitness changes after each generation, then n_{inv} is reset to 0; otherwise, n_{inv} will be incremented by 1 until it reaches s_{seq}. So, the number of inversions comes in the range $[0, s_{seq}]$, *i.e.,* the maximum value of n_{inv} will not be more than the size of the selected sequence.

Thirdly, in SBLS, the step size $SBLS_{stepsize}$ is also an adaptive parameter, whose value changes according to whether the best fitness of the generation changes. If the best fitness changes, then $SBLS_{stepsize}$ is reset to its default value (which is 5 in this study); otherwise, $SBLS_{stepsize}$ is increased by an amount of 5 until it reaches the maximum step size $SBLS_{maxstepsize}$ (which is set to 20 in this study). If $SBLS_{stepsize}$ becomes greater than $SBLS_{maxstepsize}$ it is reset to the default value.

Finally, the sequence size s_{seq} is also an adaptive parameter in SBMA, which is defaulted to $\sqrt{n}$ and reduced with time according to whether the best fitness changes. To adapt s_{seq}, we introduce an associated iteration counting variable, $MaxLS_{running}$, which is defaulted to $MaxLS$. If the best fitness changes, $MaxLS_{running}$ is reset to $MaxLS$; otherwise, $MaxLS_{running}$ is decreased by 1 if $SBLS_{stepsize}$ becomes greater than $SBLS_{maxstepsize}$, and depending on situation, the sequence size s_{seq} is reduced based on the following criteria. If $(MaxLS_{running} \leq (MaxLS \times 75\%))$, then s_{seq} will be set to 75% of its default value ($\lfloor\sqrt{n}\rfloor$). When the sequence size is changed, in the next generation, all sequences will be generated with the new sequence size and stored in the memory. For example, suppose that the $CHN144$ TSP instance is studied (*i.e.,* the default sequence size is 12), $MaxLS = 20$, and $MaxLS_{running}$ is currently reduced to 15. Then, s_{seq} will become 9 at the next generation. If $MaxLS_{running}$ becomes equal to 0, the sequence size will become two, which means an edge and SBLS will search for the shortest edge for re-insertion at a proper location.

For selecting a new population from Pop and Pop_{tmp}, we use the social disaster technique (SDT), called *packing*. That is, among all the individuals that have the same fitness value, only one remains unchanged and the other individuals are fully randomized [125].

4.3 The Inver Over (IO) Algorithm

The Inver Over (IO) operator was proposed in [138] for solving the TSP. In this thesis, we will use the Inver Over (IO) operator [138] either as a comparative algorithm or as a component to be integrated into our approaches. Here, we give the description of the IO operator. The pseudo-code of the IO operator for the TSP is shown in Algorithm 17.

The IO operator is a smart operator based on simple inversions. IO is an unary operator since the inversion is applied to a segment of a single individual. However, the selection of a segment to be inverted is population-driven, *i.e.*, taking knowledge from other individuals in the population. Thus, the IO operator displays some characteristics of crossover/recombination. It works with only two parameters, the population size and the probability of random inversion p, which was set to $p = 0.02$ in [138].

Algorithm 17 The Inver Over Algorithm – $IO(Pop(t))$

```
1: for each individual route_i in Pop(t) do
2:    route* := route_i;
3:    Randomly select a city C from route*;
4:    while TRUE do
5:       if rand(0,1) < p then
6:          Select the city C* from the remaining cities in route*;
7:       else
8:          Select randomly a route from the population;
9:          Assign to C* the next city to C in the selected route;
10:      end if
11:      if (the next or previous city of city C is C* in route*) then
12:         Exit from the while loop;
13:      end if
14:      Inverse the section from the next city of city C to city C* in route*;
15:      C := C*;
16:   end while
17:   if (Length(route*) < Length(route_i)) then
18:      route_i := route*;
19:   end if
20: end for
```

In this chapter, we also study the combined approach of SBMA and IO, denoted SBMA+IO, by shifting the control from SBMA to IO under a certain condition. The pseudo-code of SBMA+IO is shown in Algorithm 18, where if the best fitness of SBMA does not change for consecutive T generations, the control is shifted from SBMA to the IO algorithm. In our study, the value of T is set to 100.

Algorithm 18 $SBMA + IO()$

```
1: no_change_count := 0;
2: while (no_change_count < T) do
3:    Perform SBMA for one generation;
4:    if the best fitness changes then
5:       no_change_count := 0;
6:    else
7:       no_change_count++;
8:    end if
9: end while
10: Perform IO for the remaining generations;
```

4.4 Experimental Study

4.4.1 Experimental Setting

In order to test the performance of our proposed SGBA, experiments are carried out to compare our proposed SBMA algorithm, the IO algorithm, and their combination in SBMA+IO, on a set of benchmark TSP instances. The compared algorithms have been implemented in C++.

All test TSP instances (except CHN144)[1] were chosen from TSPLIB [124]. The number of cities in these TSP instances varies from 51 to 144. The distance between two cities has been determined by calculating the Euclidean distance as a floating point number. Note that the optimal tours provided by TSPLIB (and their tour lengths) are given with respect to Euclidean distances that have been rounded to the nearest integer, whereas our experiments have been carried out with distances that have not been rounded to integers. This means that the value of the "global optimum" listed in our result tables is for instances with slightly different distances between cities, but the effect is small and does not affect the conclusions.

The parameters for the algorithms were set as follows. The population size was set to 50 for the first four TSP instances, and 20 for the remaining TSP instances.

[1] CHN144 is a Euclidean instance generated from the positions of 144 major cities in China. The positions of the 144 cities $(x_i, y_i), i = 1, 2, \cdots, 144$, are mentioned in [74] and listed in [75].

The tournament size in SBMA was set to a slightly high value 7 in order to give a high selection pressure. The default crossover probability was set to $p_c = 0.25$ and the increment value to p_c was $\delta_{p_c} = 0.05$, The default mutation probability $p_m = 0.0025$ and the increment value to p_m is $\delta_{p_m} = 0.0005$. The values for the parameters are arbitrarily chosen, as in SBMA the initial set of parameters are increasing for mutation and crossover by a small value to maximum value less than 1 when the fitness does not change. If the best fitness value changes, the values of these parameters are reset to their initial values. For local search, $MaxLS = 20$ and the step size was initially set to the default value 5 and the upper maximum limit for the step size was set to $SBLS_{maxstepsize} = 20$ for each TSP instance. The SBMA was executed for 5000 generations for each run on a TSP instance.

4.4.2 Experimental Results and Analysis

In Table 4.1, we present the results of IO, SBMA+IO, and SBMA over 20 independent runs. In this table, the results of "best" rows show the best tour found and "Avg" rows show the average result (*fitness*) found over 20 runs. The "Err" rows give the relative deviation to the global optimum (*fitness*), which is listed in the table after the instance name (recall that the global optimum is with respect to instances where the distances between cities have been rounded to the nearest integer). Table 4.2 shows the acceleration ratio of SBMA over the IO algorithm, which is the ratio of the number of generations needed to gain the specified fitness. The second column gives the arbitrarily chosen fitness value that has been used to evaluate for both approaches the number of generations that is necessary to reach that fitness value. The third column shows the number of generation taken by SBMA, while the fourth column shows the number of generations that IO takes to achieve the fitness value given in the second column. For example, in case of EIL51, the fitness 501 is achieved by SBMA in 153 generations while IO takes 512 generations. The fifth column shows the ratio of the number of generations of SBMA and IO.

TABLE 4.1: Comparison of results of Inver-Over, SBMA+IO, and SBMA

Instance	**Results**	**IO**	**SBMA+IO**	**SBMA**
EIL51 (426)	Best	429.53	439.184	**429.48**
	Err	0.0083	0.0309	**0.0082**
	Avg	430.66	439.84	437.80
	Err	0.0109	0.0325	0.0277
EIL76 (538)	Best	552.22	579.194	562.97
	Err	0.0264	0.0766	0.0464
	Avg	552.37	571.56	570.7
	Err	0.0267	0.0624	0.0608
EIL101 (629)	Best	654.26	675.2	**650.89**
	Err	0.0402	0.0741	**0.0348**
	Avg	656.78	685.7	666.6
	Err	0.0442	0.0902	0.0598
KROA100 (21282)	Best	21285.4	22193	**21282**
	Err	0.0002	0.0428	**0.00**
	Avg	22392.10	2239 2.10	**22382**
	Err	0.0522	0.0522	**0.0492**
KROC100 (20749)	Best	20820	21647.3	21008
	Err	0.0034	0.0433	0.0125
	Avg	20888.10	21942.10	21903.6
	Err	0.0067	0.0575	0.0560
KROD100 (21294)	Best	21517	22180	**21317**
	Err	0.0105	0.0416	**0.0011**
	Avg	21523.00	22504.30	22674.6
	Err	0.0108	0.0568	0.0648
LIN105 (14397)	Best	14432.6	14491.8	14782
	Err	0.0037	0.0078	0.0280
	Avg	14510.90	15531.20	15118.0
	Err	0.0092	0.0801	0.0514
CHN144 (30347)	Best	31253	32613.3	31782
	Err	0.0299	0.0747	0.0473
	Avg	31542.90	33268.20	32513.6
	Err	0.0394	0.0963	0.0714

A plot of the tour length against the number of generations is shown in Figure 4.2.

When comparing the experimental results between SBMA and the IO operator, from Table 4.1, it can be seen that SBMA achieves better solutions than IO on 5

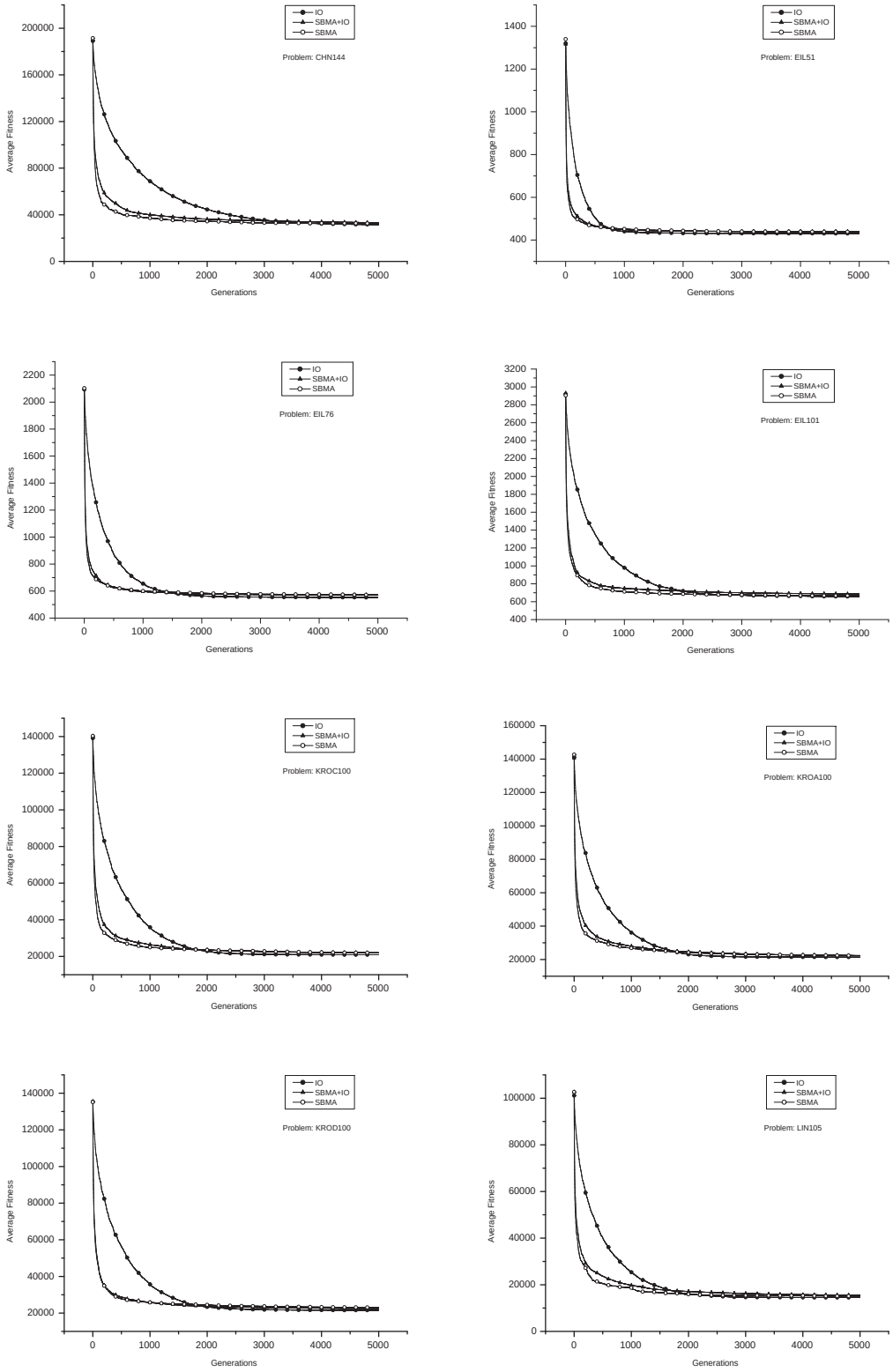


FIGURE 4.2: Experimental results of IO, SBMA+IO, and SBMA

TABLE 4.2: Experimental results regarding the acceleration ratio (AR) of SBMA over IO

Instance	**Fitness**	**SBMA**	**IO**	**AR**
EIL51	501	153	512	3.35
EIL76	627	450	1110	2.50
EIL101	745	600	1801	3.00
KROA100	30225	448	1277	2.85
KROC100	30310	298	1241	4.16
KROD100	26870	595	1521	2.55
LIN105	19818	456	1386	3.03
CHN144	49953	174	1687	9.7

test instances, while slightly worse solutions on the other instances. However, from the results of Table 4.2 and Figure 4.2, it can be seen that SBMA outperforms IO regarding the convergence speed. The acceleration ratios of SBMA over IO in the early stage of the evolutionary process on all test instances are from 2.5 to 9.7, respectively (*i.e.*, SBMA is from 2.5 to 9.7 times faster than IO regarding the convergence speed to achieve a certain fitness level).

It is also confirmed that SBMA can compete to the maximum level of optimality and most of the runs SBMA keep the control within itself. It has been shown that SBMA has almost the same characteristics on different benchmark TSP instances. The combined effect of SBMA+IO shows the same level of optimization with both the operators applied together one after another, which shows that if SBMA applied at the initial level of optimization process which do fast convergence and then control given to IO can thus enhance the performance and solution quality. Simply at the early stage of evolution, SBMA can drive the population to local optimum more rapidly and then IO is further applied so that the number of generations could be reduced as the IO has the ability to increase the population diversity, avoiding diversity loss in common crossover operators.

Further improvement of solutions requires greater computational efforts in terms of

the number of generations. For example, the best result SBMA got for Chn144 is 31086.4 ($Err = 0.0243648$) when we increase the number of generations to be more than 5000 generations. In Figure 4.2, this improvement is sensible for the instances of TSPLIB [124], respectively, which is a good indication of the convergence behaviour of SBMA.

4.5 Chapter Summary

In this chapter, we presented a sequence-based genetic algorithm (SBMA) with local search for solving the TSP. The basic idea behind SBMA is to make use of information extracted from the population to guide the crossover, mutation and local search operators. Some effective ideas are proposed for preserving the population diversity, preventing premature convergence, and enhancing the speed of convergence at the initial stage of SBMA. Our proposed mutation operator takes SBMA to promising areas of the search space as well as contributing in the fitness increase. The crossover operator exhibits a behaviour of displacement mutation, but here the extracted sub-tour is an optimized one. Our concept is totally dependent on the formation of the set of sequences, which is an area that should be further improved.

Experiments were carried out to compare the performance of SBMA with the Inver Over algorithm on a set of benchmark TSP instances. The results show that SBMA can converge very fast at the initial stage of evolution. However, one does not intend to claim that SBMA can compete with other state-of-the-art algorithms because in case of SBMA, its running time is still a big question, and improvement is needed in this direction.

The proposed SBMA works well for solving small and medium scale TSP instances. However, for larger scale instances, the computational cost is very high for SBMA and the speed of SBMA is comparatively slow. In the coming chapter, we will present some of the limitations of the SBMA approach which arose during the study. We

will argue some of the potential improvement that has been proposed to tackle the limitation, and also the current heuristics should be enhanced by integrating new heuristic approaches.

Chapter 5

A Two-Phase Hybrid Algorithm for the Travelling Salesman Problem

5.1 Introduction

In the previous chapter, we have observed that the Sequence Based Memetic Algorithm (SBMA) is good in obtaining good results for the Travelling Salesman Problem (TSP), but is computationally expensive. So, in this chapter, we hybridize the previous SBMA with the fast Inver Over (IO) operator to develop a two-phase hybrid approach (TPHA) to solving the TSP. In the first phase, the SBMA with an embedded local search scheme is applied to solve the TSP. Additionally, some procedures are applied to maintain the diversity by breaking the selected sequences into sub-tours if the best individual of the population does not improve. After SBMA finishes, the hybrid approach enters the second phase, where the IO operator [138], which is a state-of-the-art algorithm for the TSP, is used to further improve the solution quality of the population. In order to investigate the performance of the

proposed hybrid approach for the TSP, experiments are carried out to compare it with other relevant algorithms on a set of small and large benchmark TSP instances. Experimental results show that the proposed hybrid approach is superior to the IO algorithm in terms of the convergence speed as well as the solution quality.

5.2 Two-Phase Hybrid Algorithm (TPHA)

This chapter proposes a TPHA to solving the TSP. The framework of TPHA is shown in Algorithm 19. In the first phase, the SBMA, which was proposed in [7], is used. Within the SBMA, a memory is used to store good sequences (sub-tours) extracted from previous good solutions. The stored sequences are used to guide the generation of offspring during the crossover, mutation, and local search operations. In this chapter, we use an embedded local search scheme into SBMA, where the word "*embedded*" means that the local search technique is applied before the final evaluation of the individual within crossover and mutation. After each crossover and mutation operation, local search runs to improve the fitness of newly created child based on the set of sequences stored in the memory. Additionally, some procedures are applied to maintain the diversity by breaking the selected sequences into sub-tours if the best individual of the population does not improve.

After SBMA finishes, the hybrid approach enters the second phase. The criterion of switching from phase one to phase two is shown in Algorithm 20. In the second phase, the IO operator [138], which is a state-of-the-art algorithm for the TSP, is used to further improve the quality of solutions in the population. The IO operator belongs to a kind of blind operators. Along with its good adaptive power, IO suffers from random inversions which do not give better individuals finally. In this chapter, we modify the original IO operator by making the inversions restrictive: only those inversions that give better fitness scores will be retained and replace the original tour, *i.e.*, a kind of local refinement with IO is used here. In this study, we set

Algorithm 19 Two-Phase Hybrid Algorithm (TPHA)

1: Initialise adaptive parameters (e.g., p_c, p_m, n_{inv}, s_{seq}, and $SBLS_{stepsize}$) to their default values;
2: $no\_change\_count := 0$;
3: $Phase_1 := true$;
4: $Pop :=$ Initialise a population of $popsize$ random individuals;
5: **for** each individual $ind_i \in Pop$ **do**
6: $ind_i := 2\text{-}Opt(ind_i, n)$;
7: **end for**
8: $GenerateSequence(N_{seq})$;
9: **repeat**
10: **if** $Phase_1 = true$ **then**
11: $mating\_pool :=$ perform tournament selection from Pop;
12: // **Crossover**
13: **for** $j := 0$ to $popsize$ **do**
14: Select two parents i_a and i_b from the $mating\_pool$;
15: **if** $(rand(0,1) < p_c)$ **then**
16: Create $child_a$ and $child_b$ by $e\text{-}SBOX(i_a, i_b, F_{insert})$;
17: Add $child_a$ and $child_b$ to Pop_{tmp};
18: **end if**
19: **end for**
20: // **Mutation**
21: **for** each individual $ind_i \in Pop_{tmp}$ **do**
22: **if** $(rand(0,1) < p_m)$ **then**
23: $e\text{-}SBIM(ind_i, n_{inv}, F_{insert})$;
24: Add ind_i to Pop_{tmp};
25: **end if**
26: **end for**
27: $AdaptParameters()$;
28: $Pop := SelectNewPop(Pop + Pop_{tmp})$;
29: **if** $Switch\_Criteria(no\_change\_count) \geq \gamma$ **then**
30: $Phase_1 := false$;
31: $IO_{popsize} := 3 \times popsize$;
32: $Pop_{IO} :=$ Insert $popsize$ random individuals into $Pop + Pop_{tmp}$ for IO;
33: **end if**
34: **else**
35: **for** each individual $ind_i \in Pop_{IO}$ **do**
36: $RIO(ind_k)$;
37: **end for**
38: **end if**
39: **until** $(termination\ condition = true)$

Algorithm 20 *Switch_Criteria*(*no_change_count*)

```
1: if (best fitness changes) then
2:    no_change_count := 0;
3: else
4:    no_change_count + +;
5: end if
6: return(no_change_count);
```

$\gamma = 100$, which means that if the best fitness does not change for consecutive γ generations, then the shifting criterion would be satisfied and the control would be given to *RIO*.

It is worth noting that our approach could also be used in general for combining any two different optimisation methods. The approach is presented here with just two phases, but based on different parameter settings, it could also be extended to more than two phases. It may be desirable to construct a pool of crossover and mutation operators along with many other local search methods. A kind of intelligent shifting scheme could be employed which first analyses the fitness landscape and then takes action or calls the appropriate operators or executes a complete phase of a suitable type. Similar work has been done by Gabrys and Ruta [43] on classifier fusion of different classifier models, where each classifier system can handle different types of data at hand.

In the following sub-sections, we first describe the modified SBMA and then describe the modified IO operator used in the proposed two-phase hybrid approach respectively.

5.2.1 Phase I: Modified SBMA (SBMA-II)

In this hybrid approach, the working principle of the modified SBMA, denoted *SBMA-II*, is similar to that of the SBMA, which has been described in Chapter 4. In the previous SBMA, after the crossover and mutation, the selected sequence

is inserted at a random position, and then SBLS is further applied, within which a kind of exhaustive technique is used since it searches all the locations to find the best location for inserting a selected sequence, which is the main cause for our previous approach to be computationally expensive.

In SBMA-II, we do not apply SBLS after crossover and mutation. Instead, we apply the SBLS operation within crossover and mutation that deal with partial individuals. By making these partial individuals into a complete one, we check F_{insert} of random locations instead of all the locations as in SBMA and select the best location among these locations and insert the sub-tour and then evaluate the individual. Thus, local search is the embedded part of crossover and mutation. The parameter F_{insert} supplied to the crossover and mutation operator represents the percentage of positions (to the size of the individual we want to insert a sequence) that are selected as the candidate positions for the insertion of a selected sequence S_{sel}. The value of F_{insert} varies from 5% to 15% to the size of a TSP instance. For example, for the $eil101$ TSP instance used in the experimental study, 5 random locations will be checked if $F_{insert} = 5\%$ since there are 101 cities in the $eil101$ TSP instance.

5.2.1.1 Sequence Based Order Crossover with Embedded Local Search (e-SBOX)

The e-SBOX crossover operator is based on the SBOX operator, which has been described in the previous chapter. However, the working principle is slightly different. The pseudo-code of e-SBOX is shown in Algorithm 21. In e-SBOX, after crossing over the partial parent solutions, local search operations are carried out on them directly to generate complete children solutions.

Algorithm 21 *e-SBOX*(i_a, i_b, F_{insert})

1: S_{sel} := randomly select a sequence from the memory;
2: Remove the nodes of S_{sel} from individuals i_a and i_b, resulting in i'_a and i'_b, respectively;
3: Perform Order Crossover on i'_a and i'_b to generate children $child_a$ and $child_b$, respectively;
4: Apply *SBLS-II*$(child_a, S_{sel}, F_{insert})$ and *SBLS-II*$(child_b, S_{sel}, F_{insert})$;
5: Evaluate $child_a$ and $child_b$;

5.2.1.2 Sequence Based Inversion Mutation with Embedded Local Search (e-SBIM)

Similarly, the mutation operator *e-SBIM* is based on the SBIM mutation operator described in Chapter 4. The pseudo-code of e-SBIM is shown in Algorithm 22. In e-SBIM, after mutating the partial parent solution, local search is carried out on the partial solution directly to generate a complete child solution.

Algorithm 22 *e-SBIM*$(i_m, n_{inv}, F_{insert})$

1: S_{sel} := randomly select a sequence from the memory;
2: S'_{sel} := reverse selected S_{sel};
3: i_{temp} := remove the cities in S_{sel} from individual i_m;
4: **for** $i := 0$ **to** n_{inv} **do**
5: Randomly select two positions p_1 and p_2 such that $0 < p_1 < p_2 < |i_{temp}|$, where $|i_{temp}|$ denotes the number of cities in i_{temp};
6: i'_{temp} := inverse cities in between positions p_1 and p_2 of i_{temp};
7: **if** $f(i'_{temp}) < f(i_{temp})$ **then**
8: $i_{temp} := i'_{temp}$;
9: **end if**
10: **end for**
11: Apply *SBLS-II*$(i_{temp}, S'_{sel}, F_{insert})$;
12: Evaluate i_{temp};
13: $i_m := i_{temp}$;

5.2.1.3 Modified Sequence Based Local Search (SBLS-II)

Local search is an efficient heuristics for combinatorial optimization problems [96]. In SBMA, the set of sequences stored in the memory is applied in the LS to guide

the generation of children towards promising area of the search space.

As we have mentioned above, in SBMA-II, the working principle of the SBLS is changed. The only difference lies in that SBLS is a kind of exhaustive search technique, which searches all locations to insert a selected sequence into an individual and hence is computationally expensive. In the modified SBLS-II, we only check a limited subset of locations, which reduces the computational time a lot.

Algorithm 23 $SBLS\text{-}II(ind_i, S_{sel}, F_{insert})$

1: $X :=$ 2-$Opt(ind_i, SBLS_{stepsize})$;
2: Create a set of $|X| \times F_{insert}$ random locations, where $|X|$ denotes the number of cities in X;
3: Find the best position *best_position* among the selected locations in X which gives the minimum length increase after inserting S_{sel}, according to the following equation:

$$L_{inc} = Min_{j=0}^{j=n-M}(dist[S_{sel}[0]][X_j] + dist[S_{sel}[M-1]][X_{j+1}] - dist[X_j][X_{j+1}]);$$

where n is the total number of cities in the TSP and M is the number of cities in S_{sel}
4: $ind_i :=$ insert S_{sel} into X at position *best_position*;
5: Evaluate ind_i;

The pseudo code of SBLS-II is shown in Algorithm 23. SBLS-II takes an individual ind_i and first performs 2-Opt improver for K times. Then, SBLS-II finds the best position from a set of randomly selected positions to insert a selected sequence into ind_i. The distance between the first and last nodes of the sequence S_{sel} is calculated according to the distance matrix relevant to the adjacent nodes of the individual ind_i where the sequence may be inserted. The position corresponding to the minimum length increase value L_{inc} is used to insert the sequence S_{sel} to ind_i.

5.2.1.4 Adapting Parameters and Maintaining the Diversity

As in the SBMA, we use adaptive techniques in SBMA-II to adapt several key parameters, including the step size $SBLS_{stepsize}$ for the 2-Opt improver used in SBLS, the size of sequences s_{seq}, and the crossover and mutation probabilities.

For the first parameter, $SBLS_{stepsize}$ is initially set to 5. When the best fitness of the population does not improve, the value of $SBLS_{stepsize}$ is increased by 5 until it reaches 20. If the best fitness of the generation improves, $SBLS_{stepsize}$ is reset to 5.

The size of sequences s_{seq} stored in the memory is adapted also according to whether the best fitness of the generation has been improved. Initially, s_{seq} is set to the value of $\lfloor\sqrt{n}\rfloor$, where n is the total number of cities in the TSP and $\lfloor x\rfloor$ returns an integer nearest or equal to x. We use a variable t_{us} to denote when to update s_{seq}, which is initialized to 20. When the best fitness of the population does not improve, t_{us} is decreased by one. When $t_{us} = 15$, we set $s_{seq} := \lfloor 0.75 \times \sqrt{n}\rfloor$. When t_{us} is further reduced to 10, we set $s_{seq} := \lfloor 0.5 \times \sqrt{n}\rfloor$. When t_{us} is further reduced to 5, we set $S_{seq} := \lfloor 0.25 \times \sqrt{n}\rfloor$. When t_{us} is further reduced to 0, $s_{seq} := 2$, which means a sequence will become an edge (i, j) and SBMA searches for the shortest edge and re-inserts it in a proper position of an individual in SBLS. If the best fitness of the population improves, t_{us} is reset to 20 and s_{seq} is reset to $\lfloor\sqrt{n}\rfloor$. The algorithm of adapting parameters has been given in the previous Chapter 4 as Algorithm 16, in this pseudo code $MaxLS_{running}$ refers to t_{us} of the above text.

We also adapt the crossover probability p_c and mutation probability p_m as follows. Initially, we set $p_c = 0.55$ and $p_m = 0.05$. If the best fitness of the population does not improve, we increase p_c with a step size $\delta_{p_c} = 0.05$ until it reaches 0.8 and p_m with a step size $\delta_{p_m} = 0.005$ until it reaches 0.5. If the best fitness of the population improves, p_c and p_m are reset to their initial values.

5.2.2 Phase 2: Modified Inver Over (IO) Algorithms

We propose two simple modifications to the original IO algorithm [138], which has been described in Chapter 4 as Algorithm 17. The modifications are described as follows.

Algorithm 24 $RIO(route)$

```
1: p = 0.02
2: route* := route;
3: select randomly a city C from route*;
4: while (TRUE) do
5:    if (rand(0,1) < p) then
6:       select the city C* from the remaining cities in route*;
7:    else
8:       select randomly a route from the population;
9:       assign to C* the next city to C in the selected route;
10:   end if
11:   if (the next or previous city of city C is C* in route*) then
12:      exit from the while loop;
13:   end if
14:   inverse the section from the next city of city C to city C* in route*;
15:   C := C*;
16:   if (Length(route*) < Length(route)) then
17:      route := route*;
18:   end if
19: end while
```

5.2.2.1 Restricted IO Algorithm

It is clear from Algorithm 17 that the main loop terminates and only if the next or previous city of city C is C^* in $route^*$, then it exits from the main loop. It does not consider which inversion contributes in fitness gain or not. We have made the inversion restricted by shifting the evaluating part of Algorithm 17 into the main *while()* loop before the assignment of $C := C^*$. The pseudo-code of the modified IO algorithm, called the *restricted inver over (RIO)* algorithm, is shown in Algorithm 24.

At the first glance, it seems to be an extra overhead on the IO and this would increase the execution time. But, it is interesting to see in the experimental results that comparing with the original IO, the proposed restricted IO not only decreases the computational time but also contributes to obtain a good solution quality.

5.2.2.2 Restricted IO with Partial Random Initialization

As mentioned above, SBMA performs well at the early stage of evolution and has fast convergence, which can get good fitness but reduces the diversity as well. So, when IO is employed in the second phase of our hybrid approach, it not only brings diversity but also contributes better in obtaining good results in terms of fitness.

Since the diversity of the population affects the performance of IO greatly, in our hybrid approach, before giving control to IO, along with previous parent and child populations, some percentage of random individuals are injected into the population. If the *popsize* is 30, then the total size of the population for IO in the second phase will be 90 (30 parents, 30 children, and 30 random individuals respectively) and for large benchmark problems, the population size is 60 (20 parents, 20 children, and 20 random individuals respectively). This approach is denoted by TPHA+RI in this thesis.

5.3 Experimental Study

5.3.1 Experimental Setting

In this section, we present the experimental results of the proposed hybrid approach TPHA+RI in comparison with other three relevant algorithms, which are the IO algorithm [138], the original SBMA proposed in Chapter 4, and TPHA that is the proposed hybrid approach but without adding random individuals into the population when the second phase (*i.e.,* IO) is started.

The proposed approach was implemented in C++ on a 2.66 GHz PC under the Windows Visual Studio environment. All TSP problem instances (except CHN144) are obtained from TSPLIB[1] for the symmetric TSP. The number of cities in these

[1]Available from http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

cases varies from 51 to 144 for small problems and from 318 to 1173 for large problems. The SBMA is not performed for large problems, because it takes longer time which can be seen from Table: 5.1. The parameters for the algorithms were set as follows. The population size was set to 30 for the first eight TSP instances in the four techniques. For IO and TPHA the population size is 60 in large problems. And for TPHA+RI algorithm, initially the population size was set to 20 for the first phase and for second phase the population size was increased to 60 for making the population size consistent for the three techniques. The crossover probability and mutation probability were initially set to $p_c = 0.55$ and $p_m = 0.055$, respectively, which are adapted by a small value to a maximum value when the best fitness of the generation does not change. The percentage of random locations to insert a sequence, F_{insert}, was set to 5% for these experiments. The number of generations for small instances such as eil76, eil101, lin105, kroA100, kroC100 and chn144 is 10000 and for medium instances such as lin318, pcb442, rat783, u724, vm1084 and pcb1173 is 50000.

5.3.2 Experimental Results and Analysis

In Table 5.1, we present the averaged results of IO, SBMA, TPHA, and TPHA+RI over 30 independent runs. In this table, "Best" denotes the best tour found and "Avg" denotes the average fitness over 30 runs. The "Err" rows give relative deviation to the global optimal fitness listed in the table after the instance name. Finally, "Avg Time" is the average time used by algorithms in seconds. Figure 5.1 and 5.2 show the dynamic performance of algorithms regarding the average fitness against the number of generations.

From Figure 5.1 and Figure 5.2, it can be seen that TPHA and TPHA+RI gives a better convergence speed at the initial stage of the solving progress, which is the SBMA part of our hybrid approach. The convergence speed is also visible from Figure 5.3, which is plotted from 10,000 evaluations of both the algorithms for

Instance	Measure	IO	SBMA	TPHA	TPHA+RI
EIL76 (538)	Best	544.369	553.097	546.886	544.369
	Err	0.0118	0.0280	0.0165	0.0118
	Avg	550.304	557.2756	556.401	**548.294**
	Err	0.0228	0.0358	0.0342	**0.0191**
	Avg Time	6.50	19.00	4.80	5.7
EIL101 (629)	Best	644.275	663.867	645.919	645.205
	Err	0.0242	0.0554	0.0268	0.0257
	Avg	652.851	677.73	653.879	**651.444**
	Err	0.0379	0.0774	0.0395	**0.0356**
	Avg Time	6.80	23.00	5.60	6.3
KROA100 (21282)	Best	21285.4	21890.66	21285.4	21285.4
	Err	0.0001	0.0285	0.0001	0.0001
	Avg	21328.8	21896.66	21430.5	**21321.7**
	Err	0.0021	0.0288	0.0069	**0.0018**
	Avg Time	6.80	23.09	5.60	6.16
KROC100 (20749)	Best	20769.9	21732.1	20750.8	20750.8
	Err	0.0010	0.0473	0.00001	0.00001
	Avg	20879.1	21884.96	20921.7	**20822.1**
	Err	0.0062	0.0547	0.0083	**0.0035**
	Avg Time	6.70	23.00	5.40	6.2
LIN105 (14397)	Best	14397	14755	14397	14397
	Err	0	0.0248	0	0
	Avg	14446.5	15276.7	14505.2	**14426.4**
	Err	0.0034	0.0611	0.0075	**0.0020**
	Avg Time	6.90	24.50	5.60	5.7
CHN144 (30347)	Best	31388.1	32169	30661.1	**30353.9**
	Err	0.0343	0.0600	0.0103	**0.0002**
	Avg	31681.6	33470.9	30953.9	**30698.7**
	Err	0.0439	0.1029	0.0199	**0.0115**
	Avg Time	6.96	17.00	5.00	7.00
LIN318 (42029)	Best	43045.5	– –	42831.6	**42964.4**
	Err	0.02419	– –	0.01909	**0.0222**
	Avg	43174.8	– –	42955.8	**43070**
	Err	0.0272	– –	0.0220	**0.0247**
	Avg Time	47.35	– –	47.84	37.36
PCB442 (50778)	Best	55625.7	– –	51868.3	**51866.9**
	Err	0.0954	– –	0.0214	**0.0214**
	Avg	55868.9	– –	52013.46	**52236.8**
	Err	0.1002	– –	0.0243	**0.0287**
	Avg Time	74.26	– –	61.78	55.36
RAT575 (6773)	Best	12096	– –	7018.98	**7031.91**
	Err	0.7859	– –	0.0363	**0.03822**
	Avg	12721.4	– –	7031.45	**7048.58**
	Err	0.8782	– –	0.0381	**0.0406**
	Avg Time	110.36	– –	98.08	76.29
RAT783 (8806)	Best	34093.5	– –	9526.48	**9218.27**
	Err	2.8716	– –	0.05115	**0.0468**
	Avg	34946.3	– –	9267.84	**9244.28**
	Err	2.9684	– –	0.05244	**0.0497**
	Avg Time	190.57	– –	72.33	106.06
U724 (41910)	Best	140569	– –	43304.2	**43441**
	Err	2.3540	– –	0.0332	**0.0365**
	Avg	145847	– –	43519.3	**43485.8**
	Err	2.4800	– –	0.03839	**0.0376**
	Avg Time	157.80	– –	139.94	92.84
V1084 (239297)	Best	2.26e+06	– –	250757	**252305**
	Err	8.4344	– –	0.0478	**0.0543**
	Avg	2.29e+06	– –	251469	**252955**
	Err	8.5698	– –	0.0508	**0.0570**
	Avg Time	332.80	– –	196.98	159.20
PCB1173 (56982)	Best	432382	– –	60223.4	**60055.9**
	Err	6.5880	– –	0.0568	**0.0539**
	Avg	440485	– –	60476	**60481**
	Err	6.7302	– –	0.0613	**0.0614**
	Avg Time	347.17	– –	217.36	172.37

TABLE 5.1: The experimental results of IO, SBMA, TPHA, and TPHA+RI

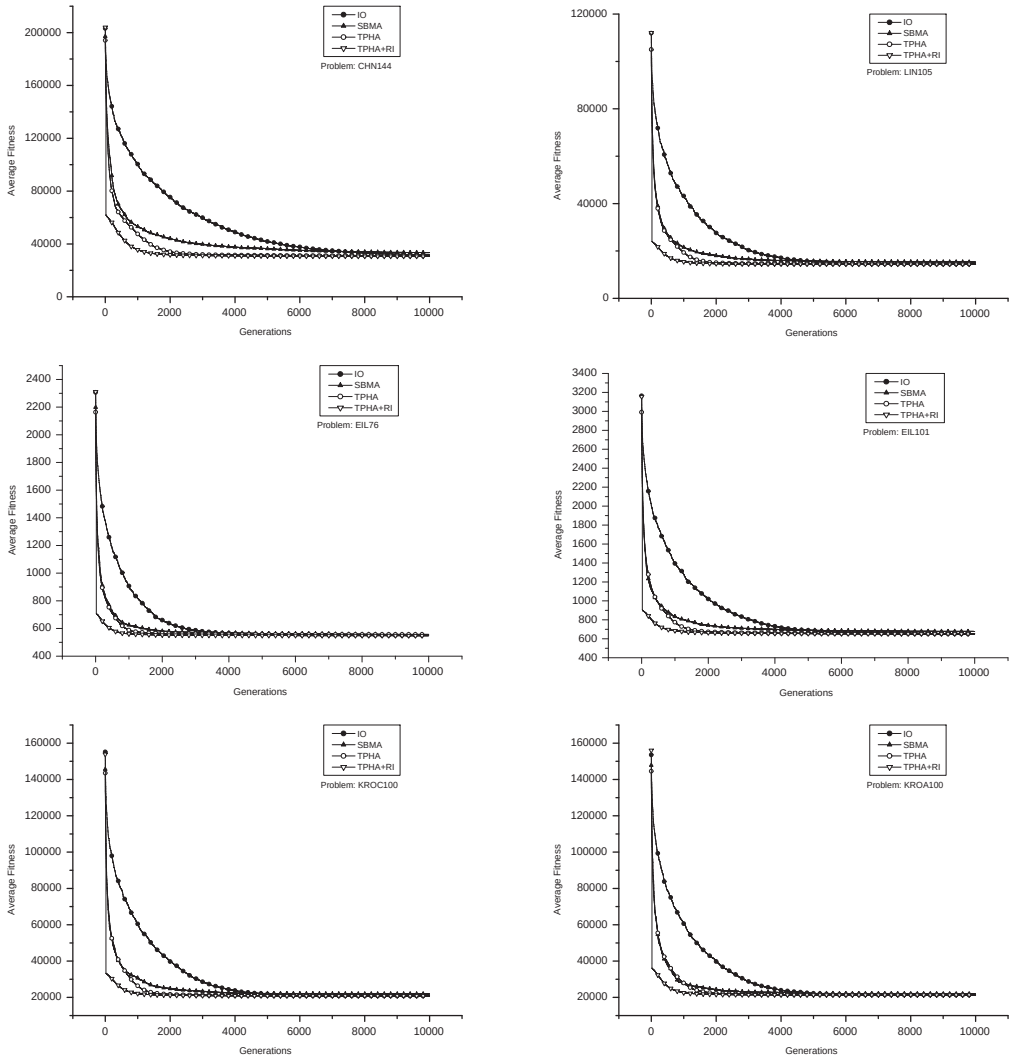


FIGURE 5.1: Experimental results of IO, SBMA,TPHA, and TPHA+RI. The effect of our approach has more additive improvements over the original IO.

chn144 and *pcb1173*. Then, it behaves similar to the IO operator. In terms of the number of evaluations, the ratio between IO and SBMA is 1:3, as SBMA uses the traditional binary operator with an additional embedded SBLS. But, due to the adaptive behaviour of IO, it gives a better solution quality at the later stage of the hybrid approach. The hybrid approach combines both the features of SBMA and IO. First, SBMA brings the fitness to a near-optimal level in a few generations and then IO further works and improves the fitness to give a better solution quality.

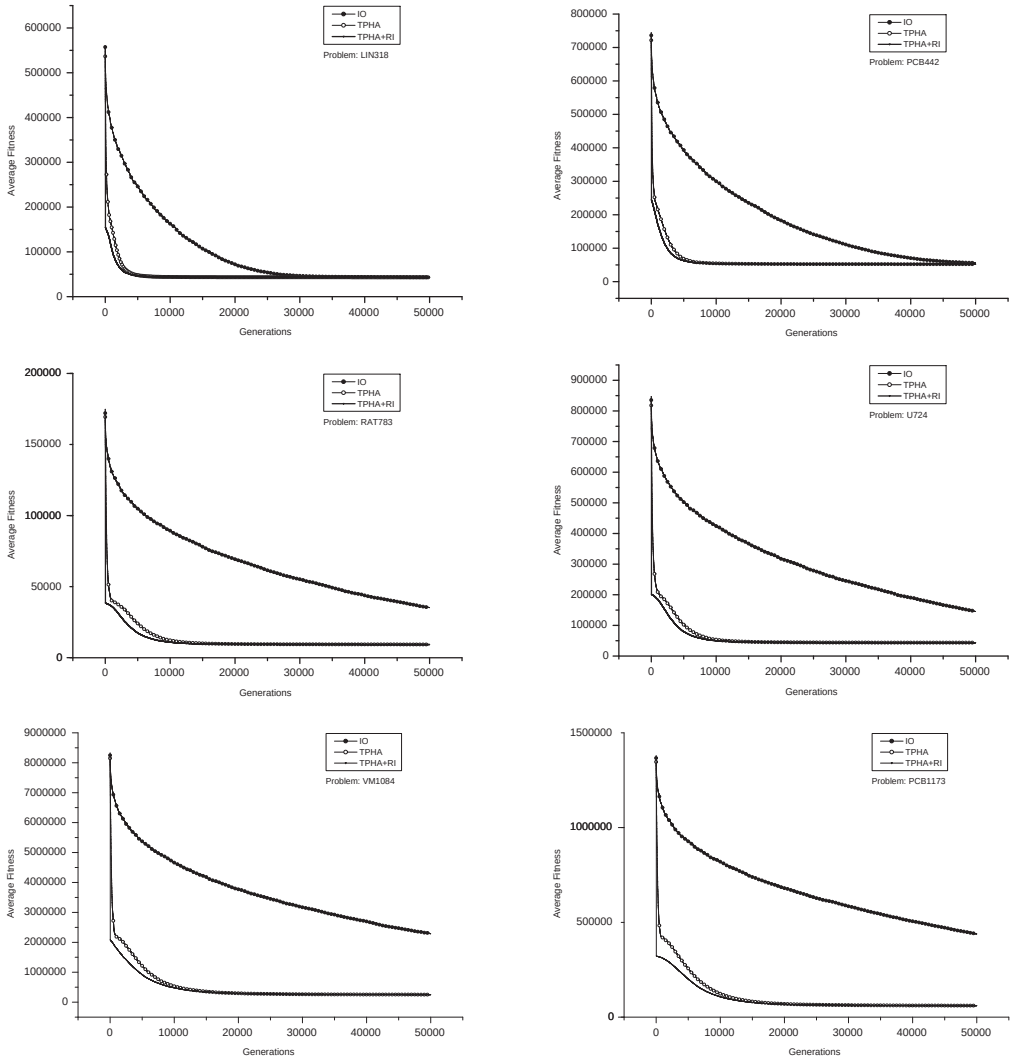


FIGURE 5.2: Experimental results of IO,TPHA, and TPHA+RI. The effect of our approach is more prominent in larger problems.

It can also be observed that when the new population is selected from parent and child, lots of information is lost. So, by keeping parent and child population and introducing some random individuals,the TPHA+RI outperforms SBMA and TPHA. From Table 5.1, it can be seen that TPHA+RI achieves better solutions than IO on all test instances, while TPHA is slightly worse (first eight small and medium instances), but comparatively better then SBMA. However, from the results of Table 5.1 and Figures 5.1 and 5.2, we can see that our hybrid approach TPHA+RI

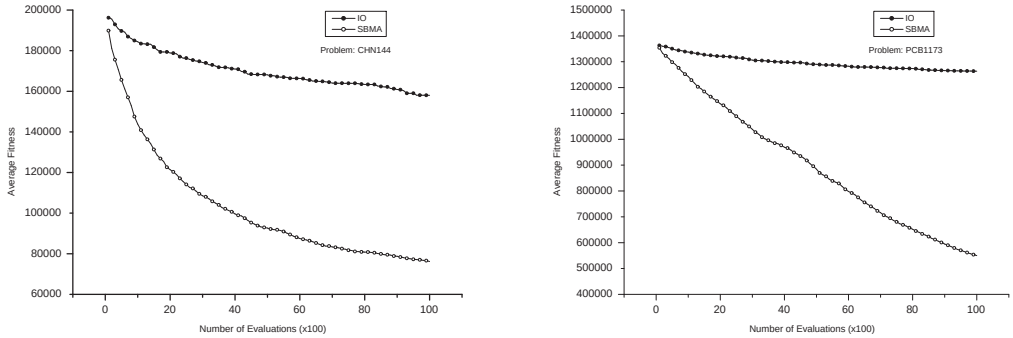


FIGURE 5.3: Evaluation results of IO,SBMA. The which best fitness evaluation within 100 fitness evaluations is recorded. The total evaluation for each algorithm are 10000, which shows the convergence of SBMA *vs* IO.

outperforms IO regarding both the convergence speed and solution quality and time.

It should be interesting to compare the original IO operator and our modified IO approaches which have been described before. Figures 5.1 and 5.2 plot the average tour length against the number of generations for 13 TSP instances. The three types of refinement can enhance the performance, both converged more rapidly than original IO. The use of SBMA, restrictive IO and RI also have additive effects on the performance gain and the contribution is dominating. From these preliminary results, one may speculate that our approach is more effective and increases the *"Adaptive Power"* of the IO which are not fully contributed by the original IO in case of small as well as in large TSPs instances.

In terms of the computational time, it is obvious that performing extra steps in our proposed approaches increases the execution time. But, from Table 5.1, the characteristics are totally opposite. The IO takes longer time on all the instances, either small or large. In cases of $rat575$, $rat783$, $u724$, $v1084$, and $pcb1173$, the IO operator is unable to achieve acceptable fitness, but due to the additive effect of our approaches with IO, it not only gets better fitness but also reduces the execution time remarkably. We may speculate that most of the time is wasted in inversions which are not fruitful regarding the fitness gain.

From Figure 5.1 and Figure 5.2, various algorithms are shown in different line styles. It is obvious from the plots that our proposed approaches are not overlapping. This means that each refinement can contribute to additional performance gain. These contributions are more effective for large problems. However, these refinements not only decrease the error rate but also reduce the CPU time on almost all test instances.

5.4 Chapter Summary

This chapter proposed a two-phase hybrid approach for the TSP based on a sequence based genetic algorithm (SBMA) and the inver over (IO) operator. In the first phase, SBMA is used with an embedded local search scheme to solve the TSP. Within the SBMA, a memory is introduced to store good sequences extracted from previous good solutions. The stored sequences are used to guide the generation of offspring during the crossover and mutation operations. After each crossover and mutation operation, a sequence based local search scheme runs to improve the fitness of newly created child. Some effective ideas are proposed for adapting the key parameters and maintaining the population diversity. After SBMA finishes, the second phase uses the IO algorithm [138], which is a state-of-the-art algorithm for the TSP, with some extra refinements, *i.e.,* restrictive inversions and a random immigrants like scheme, to further improve the quality of solutions of the population.

In order to investigate the performance of the proposed hybrid approach for the TSP, experiments were carried out to compare it with three relevant algorithms on a set of benchmark TSP instances. Experimental results show that the proposed hybrid approach is superior to the IO algorithm and the original SBMA regarding both the convergence speed and solution quality on most test TSP instances.

The next chapter would be based on further refining some issues which arise during this study. The first one is to consider how to prevent the sequences from inserting

long edges. Secondly, we need to consider how the inver over, which takes blind decisions of inversions, should be made more intelligent or guided. Finally, the sensitivity analysis of key parameters will be carried out.

Chapter 6

A Guided Two-Phase Hybrid Algorithm for the Travelling Salesman Problem

6.1 Introduction

In this chapter, we move forward toward a bit deeper analysis of our previous approaches, which have been described in the previous two chapters. We further increase the convergence speed of the first phase of our sequence based memetic algorithm (SBMA) by integrating some heuristics, e.g., the nearest neighbours of a node, which ultimately brings the nodes near to each other by preventing long edges. This work leads to a guided two-phase hybrid algorithm (GTPHA), which is the combination of the Nearest Neighbour Sequence Based Memetic Algorithm (NN-SBMA) and the Restricted Elite Population Inver Over (REIO) operator, for solving the travelling salesman problem (TSP).

Similarly, in the first phase of GTPHA, a set of sequences/sub-tours are generated and stored in a memory and are used to guide the evolutionary process via local

search. Additionally, we apply some techniques to adapt the key parameters based on whether the best individual of the population improves or not and maintains the diversity. After the first phase, GTHPA enters the second phase, which is the REIO operator. Here, REIO is directed to get the clue from the elite (best) population by adding and preserving good edges. The GTPHA includes some other refinement approaches for employing the local search information and various mechanisms for achieving the balance between the two phases.

Different experiments are carried out to investigate the performance of the hybrid approach in comparison with several relevant state-of-the-art algorithms on a set of benchmark TSP instances. The experimental results show the proposed hybridization of various binary and unary operators are efficient in finding optimal or near-optimal solutions for the TSP test instances. Experimental results of the sensitivity analysis of key parameters also reveals that adjusting the size of sequences not only reduces the computational time but also contributes to obtaining good quality tours.

6.2 The Guided Two-Phase Hybrid Algorithm

The framework of the GTPHA is similar to the two-phase hybrid algorithm (TPHA) described in Chapter 5. The pseudo-code of GTPHA is shown in Algorithm 25.

The TPHA approach suffered from including expensive edges as sequences were generated in a totally random manner. In the current guided TPHA, we have made the process more intelligent by incorporating the behavior of the nearest neighbor algorithm by generating the sequences using a nearest neighbor list with the hope that it would decrease the computational time. The mutation operator is replaced by a double bridge move operator which has been addressed by many other practitioners [65, 96]. Additionally, we introduced a complete 2-opt local search instead of a partial 2-opt, which was doing only K swaps while here it is doing as many swaps as possible until no further improvement is impossible. In Phase-2, we have tried to force the

Algorithm 25 Guided Two-Phase Hybrid Algorithm (GTPHA)

```
1: Create Distance_Matrix() according to the problem instance;
2: nnList_size := ⌊√n⌋;
3: nnList_{n×⌊√n⌋} := GenerateNearestNeighbourList(nnList_size);
4: Set parameters, e.g., p_c, p_m, n_inv, s_seq, α, F_insert, γ, and t_us, to their default values;
5: no_change_count := 0;
6: Phase_1 := true
7: Pop := Initialise a population of popsize random individuals;
8: GenerateNearestNeighbourSequence(N_seq);
9: repeat
10:    if (Phase_1 = true) then
11:       mating_pool := TournamentSelect(Pop);
12:       // Crossover
13:       for j := 0 to popsize do
14:          Select two parents i_a and i_b from the mating_pool;
15:          if ((rand(0,1) < p_c)) then
16:             Create child_a and child_b by e-SBOX(i_a, i_b, F_insert);
17:             Add child_a and child_b to Pop_tmp;
18:          end if
19:       end for
20:       // Mutation
21:       for each individual ind_i ∈ Pop_tmp do
22:          if ((rand(0,1) < p_m)) then
23:             e-SBDB(ind_i, n_inv, F_insert);
24:             Add ind_i to Pop_tmp;
25:          end if
26:       end for
27:       AdaptParameters(α, t_us, Gen_running);
28:       Pop := SelectNewPop(Pop + Pop_tmp);
29:       if (Switch_Criteria(no_change_count) ≥ γ) then
30:          Phase_1 := false;
31:          IO_popsize = 3 × popsize;
32:          Pop_IO := add nearest neighbour based random individuals into Pop + Pop_tmp;
33:       end if
34:    else
35:       ElitePop[ ] := GenerateElitePop(Pop_IO);
36:       for each individual ind_i ∈ Pop_IO do
37:          REIO(ind_i, ElitePop[ ]);
38:       end for
39:    end if
40:    Gen_running++;
41: until (Gen_running ≥ MaxGen)
```

IO operator to add the segment which yields better results, and the concept of an elite population introduced.

The first phase of the GTPHA is the NN-SBMA. In NN-SBMA, a population of solutions is first randomly initialised. Then, a set of N_{seq} sequences are generated as follows. A set of best individuals are selected from the population. In TPHA, each selected individual is then broken into sub-tours, and the sub-tour with the shortest length is selected, further optimized by a 2-Opt improver [86], and then stored in the memory. In GTPHA, sequences are generated in two different ways, which will be further explained below. After the construction of sequences, a mating pool is generated using the tournament selection with the tournament size 3. Then, crossover and mutation are performed based on the sequences stored in the memory to generate offspring. Here, *e-SBOX*, as described in Algorithm 21, is used as the crossover operator, and the mutation operator is a new one, which will be further described in Section 6.2.2. In GTPHA, we integrate the local search operator as a part within crossover and mutation as in our previous study for TPHA. Simply, local search runs after each crossover and mutation operation to improve the fitness of a newly created child. In GTPHA, when NN-SBMA does not give any further improvement for a number of generations, GTPHA enters the second phase, which is the REIO part. The switching criterion is the same as the one used in the TPHA, see Algorithm 20.

In the following sub-sections, we mainly focus on describing the different aspects of GTPHA from TPHA.

6.2.1 Sequence Generation

The purpose of constructing sequences from the population is to guide the SBMA. But, our previous sequence generation approach, see Algorithm 12, suffers from randomness, *i.e.*, sometimes a sequence may contain long edges, especially when the size of the sequence is large, as illustrated in Figure 6.1. To overcome this drawback, in the GTPHA, we use two different ways to generate sequences/sub-tours, which are denoted *nearest neighbour sequence generation* and *random sequence generation*,

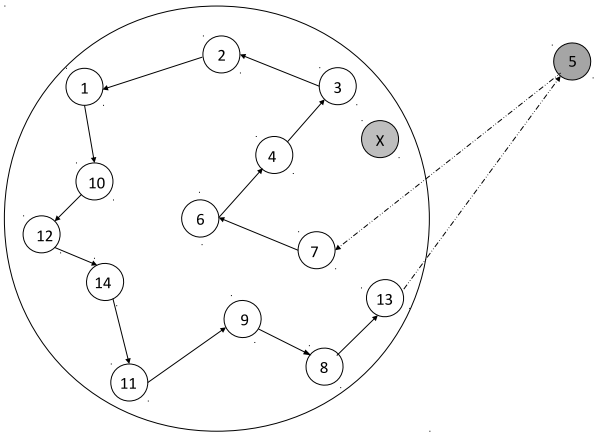


FIGURE 6.1: A sequence generated by Algorithm 12 may contain long edges, especially when the sequence size is large. The feasible node would be the x one but node 5 is connected which is expensive.

respectively. The first approach helps in generating sequences that contain nodes near to each other while the second approach helps in creating random optimal sequences by making the sequences/sub-tours a little bit more diverse.

In the GTPHA, for a few initial generations, *e.g.*, 0.05% of the total number of generations for a run, sequences are generated based on the nearest neighbour concept, and for the rest of generations, sequences are generated based on random sequences. After the set of sequences are initialized, we update the set of sequences when the fitness of the best individual of the population improves. For the generation or update of the set of sequences, a certain percentage of the best individuals from the population are selected. The construction of sequences using both approaches are described as follows.

6.2.1.1 Nearest Neighbour Sequence Generation

As mentioned above, during the early generations, sequences/sub-tours are constructed with the help of the nearest neighbour approach. In the nearest neighbour

sequence generation approach, we need first to construct a two-dimensional nearest neighbour list, denoted NN_List, as shown in Algorithm 26.

Algorithm 26 $GenerateNearestNeighbourList(nnList_{size})$

```
1: for node := 0 to n − 1 do
2:    for i := 0 to n − 1 do
3:       DistToNode[i] := dist_Matrix[node][i] // Copy the distances from node to
         remaining n nodes;
4:       DistToNode_index[i] := i;
5:    end for
6:    DistToNode[node] := ∞; // a city is not nearest neighbour to itself;
7:    Sort the distances DistToNode[ ] in an ascending order and arrange
      DistToNode_index[ ] accordingly;
8:    for k := 0 to nnList_size do
9:       nnList[node][k] := DistToNode_index[k];
10:   end for
11: end for
12: return(nnList);
```

First, the algorithm receives a parameter $nnList_{size}$, which is the size of the second level list in NN_List. In our experiments, we have kept the value of $nnList_{size}$ equal to the size of sequences, *i.e.*, s_{seq}. In the next step, the distances from the node to all other nodes are stored in a vector $DistToNode[\ ]$ and the indexes of the nodes are also stored in another vector $DistToNode_{index}[\ ]$. Then, the vector of distances is sorted in the ascending order and also the vector of indexes is set accordingly. The distance from a node to itself is set to a maximum distance such as ∞. So, a node is not considered to be the nearest neighbour of itself. Finally, the number of nodes equal to the size of the sequence are stored in a two-dimensional list, where the first dimension represents the total number of nodes of the instance and the second one goes from 0 to the size of sequences s_{seq}. The generated nearest neighbour list is returned to the main program where it is further used for the generation of nearest neighbour sequences.

When the NN-list is constructed, it can be used to construct sequences as follows. First, N_{seq} random nodes are selected from the TSP problem instance. For each of the N_{seq} selected nodes, we construct one sequence as follows. We take the node

Algorithm 27 *GenerateNearestNeighborSequence*($N_{seq}, nnList$)

```
1: Select N_seq random nodes to Random_nodes[ ] from the whole nodes of the instance;
2: for i := 0 to N_seq do
3:     for j := 1 to s_seq − 1 do
4:         Seq_i.block_j = nnList[Random_nodes[i]][j]]; // copy the s_seq − 1 nearest nodes from nnList;
5:     end for
6:     Seq_i.block_{j=0} := Random_nodes[i]; // copy the node Random_nodes[i] to Seq_i;
7:     Calculate the length of Seq_i;
8:     Seq_i := 2-Opt(Seq_i); // Further optimize the sequence by the 2-Opt improver
9:     Store the sequence Seq_i into the memory S_seq;
10: end for
```

as the initial one and the remaining $s_{seq} - 1$ nodes are selected from the NN-list, which are the nearest neighbours of the node. The obtained sequence is further optimized with the 2-Opt improver. The nearest neighbour sequence generation is performed for some percentage of the total number of generations. The main aim of this approach is to bring the nodes nearer to each other up to some extent. The process is shown diagrammatically in Figure 6.2.

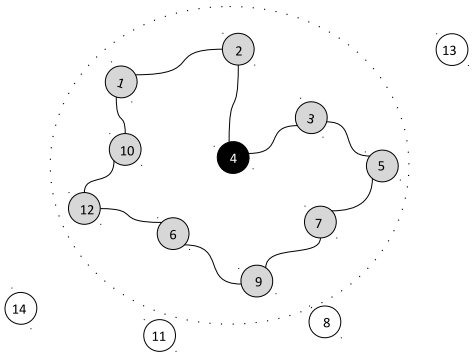


FIGURE 6.2: A sequence generated with the nearest neighbour technique does not contain long edge(s).

6.2.1.2 Random Sequence Generation

For random sequence generation, the process is similar to the sequence generation process in Chapter 5. We also select N_{seq} best individuals from the current population to create N_{seq} sequences. But, here, in order to generate a new sequence from

Algorithm 28 *GenerateRandomSequence*(N_{seq})

```
1: Select the N_seq best individuals from the population and store them in Best_indi[ ];
2: for i := 0 to N_seq do
3:    Random_nodes[ ] := Select √n random nodes as starting points within Best_indi[i];
4:    for j := 0 to √n do
5:       Seq_j := Copy s_seq consecutive nodes starting from node Random_nodes[j] within
         Best_indi[i];
6:       Calculate the length of Seq_j;
7:    end for
8:    Select the sequence Seq_sel with the minimum length among the set of √n sequences;
9:    Seq_sel := 2-Opt(Seq_sel) // Further optimize the sequence by the 2-Opt improver;
10:   Store Seq_sel into the memory S_seq;
11: end for
```

each selected individual $Best_{indi}[i]$, we first select $\sqrt{n}$ random nodes as the starting nodes to extract $\sqrt{n}$ sub-tours from $Best_{indi}[i]$. Each sub-tour starts from one selected random node and contains $s_{seq} - 1$ consecutive nodes within the same individual $Best_{indi}[i]$. Among these $\sqrt{n}$ sub-tours, the one with the minimum length is selected as a candidate sequence, and is then further optimized via a 2-Opt improver before stored in the memory.

In the current NN-SBMA approach, we have kept the crossover operator the same as in our previous approach, i.e., the *e*-SBOX in Algorithm 21 is used. However, we have replaced the mutation operator e-SBIM in the TPHA with an *embedded*-Sequence Based Double Bridge (e-SBDB) mutation operator. The detail of this operator is explained below. The SBLS operator is the same as the previous approach in Chapter 5.

6.2.2 Embedded Sequence Based Double Bridge (e-SBDB) Mutation

In the current approach, we replace the embedded sequence based inversion mutation (e-SBIM) with the simplest non-sequential move, which is a 4-opt move, so called "*Double Bridge*" [86], on an individual for some iterations, and preserve those

Algorithm 29 *e-SBDB*$(ind_i, n_{inv}, F_{insert})$

1: S_{sel} := randomly select a sequence from the memory;
2: i_{temp} := remove the cities in S_{sel} from individual $indi_i$;
3: **for** $i := 0$ **to** n_{inv} **do**
4: Randomly remove four different edges from i_{temp} such as (g_k, g_{k+1}), (h_k, h_{k+1}), (i_k, i_{k+1}), and (j_k, j_{k+1});
5: i'_{temp} := re-connect the four broken sub-tours into a complete tour;
6: **if** $(f(i'_{temp}) < f(i_{temp}))$ **then**
7: $i_{temp} := i'_{temp}$;
8: **end if**
9: **end for**
10: Apply *SBLS-II*$(i_{temp}, S'_{sel}, F_{insert})$;
11: Evaluate i_{temp};
12: $indi_i := i_{temp}$;

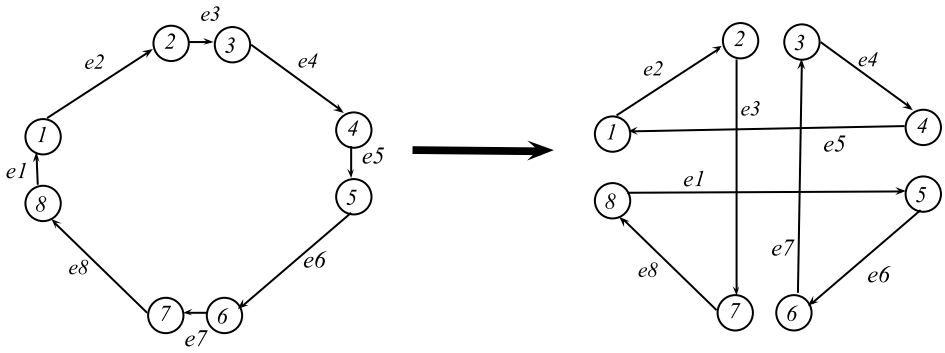


FIGURE 6.3: Illustration of the double-bridge move operator.

moves which have a positive effect on the performance. This increases the convergence speed although involving an extra overhead on the mutation operator. The number of iterations of moves in e-SBDB depends on whether the best fitness of the population changes. If the best fitness changes for each generation, e-SBDB will not execute. The details of e-SBDB are shown in Algorithm 29. Figure 6.3 illustrates a double bridge move. In Figure 6.3, let $e_1 = (g_k, g_{k+1})$, $e_3 = (h_k, h_{k+1})$, $e_5 = (i_k, i_{k+1})$, and $e_7 = (j_k, j_{k+1})$, respectively. If these edges are broken, e_1, e_3, e_5, and e_7 are rewired as an 4-opt move, the resulting rewired edges would become $e_1 = (g_k, i_{k+1})$, $e_3 = (h_k, j_{k+1})$, $e_5 = (i_k, g_{k+1})$, and $e_7 = (j_k, h_{k+1})$.

6.2.3 Adapting Parameters and Maintaining the Diversity

The mechanism of adapting parameters for maintaining the diversity remains the same as the previous approach, see Algorithm 30. The change made here is that we adapt the crossover probability p_c and mutation probability p_m as follows. Initially, we set $p_c = 0.65$ and $p_m = 0.025$. If the best fitness of the population does not improve, we increase p_c with a step size $\delta_{p_c} = 0.05$ until it reaches 0.75 and p_m with a step size $\delta_{p_m} = 0.005$ until it reaches 0.05. If the best fitness of the population improves, p_c and p_m are reset to their initial values.

The nearest neighbour sequence generation and random sequence generation are based on setting the parameter $\delta_{NN_{seq}}\%$, which works as follows: initially for some generations we generate nearest neighbour sequences while for the rest of generations we generate random sequences as in our previous approaches. So, the condition $(Gen_{running} \leq (Max_{generation} \times \delta_{NN_{seq}}\%))$ would enforce that when and which type of sequences would be generated. For example, if $Gen_{running}$ is less than or equal to $(Max_{generation} \times \delta_{NN_{seq}}\%)$, then the sequences would be generated based on the nearest neighbour approach. For example, if $Max_{generation} = 5000$ and $\delta_{NN_{seq}} = 0.5\%$, then for the initial 25 generations, the sequences would be generated based on the nearest neighbour approach as mentioned earlier.

The size of sequences s_{seq} stored in the memory is also adapted according to whether the best fitness of generation improves. Initially, s_{seq} is set to the value of $\lfloor\sqrt{n}\rfloor$, where n is the total number of cities in the TSP and $\lfloor x\rfloor$ returns an integer nearest or equal to x. We use a variable t_{us} to denote when to update s_{seq}, which is for example initialized to 20. When the best fitness of the population does not improve, t_{us} is decreased by one. When $t_{us} = 15$, we set $s_{seq} := \lfloor 0.75 \times \sqrt{n}\rfloor$. When t_{us} is further reduced to 10, we set $s_{seq} := \lfloor 0.5 \times \sqrt{n}\rfloor$. When t_{us} is further reduced to 5, we set $s_{seq} := \lfloor 0.25 \times \sqrt{n}\rfloor$. When t_{us} is further reduced to 0, $s_{seq} := 2$, which means a sequence will become an edge (i, j) and SBMA searches for the shortest edge and re-inserts it in a proper position of an individual in SBLS. If the best

Algorithm 30 $AdaptParameters(N_{seq}, MaxLS, Gen_{running})$

if (the best fitness changes) **then**
  Reset n_{inv}, p_c, and p_m to their default values;
  Reset $t_{us} := MaxLS$;
  $s_{seq} := \lfloor \sqrt{n} \rfloor$;
  if $(Gen_{running} \leq (Max_{generation} \times \delta_{NN_{seq}}\%))$ **then**
    $GenerateNearestNeighborSequence(N_{seq})$;
  else
    $GenerateRandomSequence(N_{seq})$;
  end if
else
  $n_{inv} := min\{n_{inv} + 1, s_{seq}\}$;
  $p_c := min\{p_c + \delta_{p_c}, 0.75\}$;
  $p_m := min\{p_m + \delta_{p_m}, 0.05\}$;
  Decrease t_{us} by 1;
  if $(0.5 \times MaxLS) < t_{us} \leq (0.75 \times MaxLS)$ **then**
    $s_{seq} := \lfloor 0.75 \times \sqrt{n} \rfloor$;
    $Apply_{2-opt} := TRUE$;
  else if $((0.25 \times MaxLS) < t_{us} \leq (0.5 \times MaxLS))$ **then**
    $s_{seq} := \lfloor 0.5 \times \sqrt{n} \rfloor$;
    $Apply_{2-opt} := TRUE$;
  else if $(0 < t_{us} \leq (0.25 \times MaxLS))$ **then**
    $s_{seq} := \lfloor 0.25 \times \sqrt{n} \rfloor$;
    $Apply_{2-opt} := TRUE$;
  else if $(t_{us} \leq 0)$ **then**
    $s_{seq} := 2$; // that is, a sequence becomes an edge;
  end if
  if $(Apply_{2-opt} = TRUE)$ **then**
    for $i := 0$ to N_{seq} **do**
      $Seq_i := 2\text{-}Opt(Seq_i, s_{seq})$;
    end for
    $Apply_{2-opt} := FALSE$;
  end if
end if

fitness of the population improves, t_{us} is reset to default value and s_{seq} is reset to $\lfloor\sqrt{n}\rfloor$. We have experimented different parameters for t_{us} which will be explained in the following sections. The procedure on how to adapt different parameters is given in Algorithm 30. Finally, if the size of sequences s_{seq} is reduced, then the variable $Apply_{2-opt}$ is set to true and the sequences in the memory are truncated (*i.e.,* removing some nodes from the sequences) and further optimized via the 2-Opt improver. This is a new feature in the GTPHA, which is different from the TPHA in Chapter 5, where no optimization of sequences was considered.

In this chapter, we apply 2-opt[1] local search, as shown in Algorithm 31 [70], when the size of sequences changes. In this study for the iterative improvement, we use a 2-opt local search method for making the sequences optimal when created and also when the algorithm adapts the size of sequences. The 2-opt is based on the k-exchange neighbourhood relation, in which candidate solutions s and s' are direct neighbours if and only if s' can be obtained from s by deleting a set of k edges and rewiring the resulting sub-tours into a complete sequence by inserting a different set of k edges. For this iterative improvement algorithm, we use a fixed k-exchange neighbourhood relation with $k = 2$. The implementation of a 2-exchange iterative improvement algorithm considers in each step all possible combinations of the k edges to be removed and added. After removing k edges from a given candidate sequence s, the number of ways in which the resulting sub-tours can be reconnected into a candidate sequence different from s depends on k, which is 2 in this case. After removing the two edges (u_i, u_j) and (u_k, u_l), the only way to re-connect the two partial tours into a different complete sequence is by introducing the edges (u_i, u_k) and (u_l, u_j). If an exchange yields gain in fitness, the exchange is made permanent. The procedure is repeated until no improvement is possible.

[1]The 2-Opt implementation used for these experiments is available from (http://www.sls-book.net/implementations.html)

Algorithm 31 2-Opt($tour, s_{seq}$)

```
1: improvement := TRUE;
2: dim := s_seq;
3: while (improvement) do
4:    improvement := FALSE;
5:    Max_gain := 0;
6:    for i := 0 to dim − 2 do
7:       if (i > 0) then
8:          inner_loop := dim;
9:       else
10:         inner_loop := dim − 1;
11:      end if
12:      for j := i + 2 to inner_loop do
13:         remove := dist[tour[i]][tour[i + 1]] + dist[tour[j]][tour[(j + 1)%dim]]
14:         add := dist[tour[i]][tour[j]] + dist[tour[i + 1]][tour[(j + 1)%dim]]
15:         gain := add − remove
16:         if (gain < Max_gain) then
17:            improvement := TRUE;
18:            Max_gain := gain;
19:            h := i+1;
20:            l := j
21:         end if
22:      end for
23:   end for
24:   if (improvement) then
25:      while (l ≥ h) do
26:         temp := tour[h];
27:         tour[h] := tour[l];
28:         tour[l] := temp
29:         h++;
30:         l − −;
31:      end while
32:   end if
33: end while
34: return (tour);
```

6.2.4 Phase 2: Restricted Elite Population Inver Over (REIO) Algorithm

In GTPHA, we enforce the IO to get clue only from the elite population of the SBMA because fit individuals are usually composed of good gene fragments or edges. In this regard, the key parameter for IO is reduced from $p = 0.02$ to $p = 0.005$ in order to reduce the mutation/inversion and get more and more clue from elite population. In Chapter 5, we have added two simple extra modifications to the original IO algorithm. In this chapter, we further improve the restricted IO (RIO) described in

Chapter 5 in two ways: restricted IO with partial nearest neighbour initialization, and IO with elite population. The criteria for switching the control from phase 1 to phase 2 is shown in Algorithm 20, where the control is shifted to phase 2 depending on the basis of parameter γ. For example, if $\gamma = 20$ and the best fitness does not change for consecutive 20 generations, then the control should be given to REIO. We have also experimented different values of γ which mainly affect the execution time, *i.e.*, how to keep the balance between the two phases.

Algorithm 32 $REIO(route, ElitePop[\,])$

```
1: p = 0.005;
2: route* := route;
3: select randomly a city C from route*;
4: while (TRUE) do
5:    p_tmp := rand(0, 1.0);
6:    if (p_tmp < p) then
7:       select the city C* from the remaining cities in route*;
8:    else
9:       select randomly an individual route_select from the elite population ElitePop[ ];
10:      while (route_select = route*) do
11:         select randomly an individual route_select from the elite population ElitePop[ ];
12:      end while
13:      assign to C* the next city to C in the selected route_select;
14:   end if
15:   if (the next or previous city of city C is C* in route*) then
16:      exit from the while loop;
17:   end if
18:   invert the section from the next city of city C to city C* in route*;
19:   C := C*;
20:   if (Length(route*) < Length(route)) then
21:      route := route*;
22:   end if
23: end while
```

The working of REIO is straightforward, as shown in Algorithm 32. It receives an individual, denoted *route*, and copies all its nodes to a temporary route $route^*$. From $route^*$, a random city C is selected. In the main loop, REIO is divided into two main procedures: one is mutation and the other one exhibits the behaviour of crossover. The decision is based on a random number $p_{tmp} \in [0, 1]$. If p_{tmp} is less than p, then mutation is performed; otherwise, a random individual is selected from the *elite* population instead of the plain population. Within that individual, the location of previously selected city C is identified. Then, the city that is next to C,

denoted C^*, is identified. Again in $route^*$, the location of city C^* is identified and C^* is brought next to C if and only if that C is not the same city as C^*. If C and C^* are the same, the main loop is terminated.

Let's demonstrate the procedure of the REIO algorithm by a simple example which has been adapted from the original source [138]. The main procedure of REIO is divided into two parts, depending on whether p is greater than the random value or less than the random value. When it is greater, then an inversion mutation is carried out, else a kind of inversion that mirrors crossover as part of a pattern of two cities of the second individual appears in the offspring. Here, the other individual is selected from an *elite* population. Now assume we have a tour

$$route^* = \{2, \underline{\mathbf{3}}, 9, 4, 1, 5, 8, 6, 7\}$$

and the current city C is 3.

If $(p_{tmp} < p)$, another city C^* from the same individual $route^*$ is selected, for example $C^* = 8$, and the segment is inverted. The resulting tour $route^*$ becomes:
$route^* = \{2, 3, \mathbf{8,5,1,4,9,}6, 7\}$.

If $(p_{tmp} \geq p)$, then another individual is selected from the population randomly, say:
$route_{selected} = \{1, 6, 4, 3, 5, 7, 9, 2, 8\}$.
This individual is searched for C^* "next" to city 3, which is 5 in this case. So, 5 would be searched in $route^*$ and the corresponding segment would be inverted, giving:
$route^* = \{2, 3, \mathbf{5,1,4,9,}8, 6, 7\}$, where the substring (3–5) arrived from $route_{selected}$.

Suppose that after many inversions this process terminates when the next city C^* (to the current city C) in the randomly selected individual is also the "next city" in the original individual. For example, assume that after some inversions, the existing

individual $route^*$ becomes

$$route^* = \{9, 3, 6, 8, 5, 1, 4, 2, 7\}$$

and the current C is 3. If ($p_{tmp} \geq p$), a 'next' city is recovered from a randomly selected individual from the population, say it is city 6 (if ($p_{tmp} < p$), a random city is selected, so it may also happen that city 6 was chosen). So we have that C is 3 and C^* is 6. Since city 6 already follows city 3, according to the condition whether the next city or the previous city of C in $route^*$ is C^*, the loop of inversions would be terminated, and the fitness of $route^*$ will be evaluated and compared with the fitness of $route$. If the fitness of the individual $route^*$ resulting from the IO process is better, it will replace the original $route$.

The main difference between the original IO operator and our REIO operator is that after inversion, we find the fitness of newly generated individual from $route$. If any improvement is found, then the original $route$ is replaced by $route^*$. However, this procedure is performed after the termination of the main loop in the original IO. On the one hand, this procedure missed all those inversions which bring gain in fitness. Additionally, it does not guarantee that $route^*$ which is the result of many inversions may or may not be better than $route$. So, our REIO seems to be greedier than the original IO and catches those inversions which bring in gain in fitness. Another difference between the original IO and REIO is that the original IO gets the clue from a random individual, but REIO gets the clue from an elite population which mainly contains good edges. In REIO, we also guarantee that candidate individuals selected from the elite population should be different from $route$.

6.2.4.1 Restricted Inver Over with Partial Nearest Neighbour Initialization

As mentioned above, NN-SBMA performs well at the early stage of evolution and converges quickly, which can get good fitness but reduces the diversity as well. So,

when IO is employed in the second phase of our hybrid approach, it not only brings diversity but also contributes better in obtaining good results in terms of fitness.

Since the diversity of the population affects the performance of IO greatly, in our hybrid approach, before giving control to IO, along with previous parent and child populations, some percentage of nearest neighbour individuals are injected into the population. For example, if the *popsize* is 30, then the total size of the population for REIO in the second phase will be 90 (30 parents, 30 children, and 30 random individuals constructed based on nearest neighbours, respectively) and for large benchmark the population size is *60* (*20* parents, *20* children, and *20* random individuals respectively). This approach is denoted by GTPHA+NNRI in this study.

6.2.4.2 Inver Over with Elite Population

The concept of elite population comes from nature where a population is divided into sub-populations, *e.g.*, a small elite and a large plain, based on their fitness difference [92]. To keep balance between the quality of tour and the computational time, we introduce the concept of *Elite Population* to the REIO operator. The original IO by [138] gives better performance when the number of nodes is under 1000.

Through observations, it has been found in [6] that IO is good in exploring but the convergence ability is weaker although it has better edge recombination ability. The fitness of a tour strongly depends on good edges. IO gets clue from the whole population which may be composed of both good and bad individuals. Good individuals contain good edges which are the part of global optimum. If IO is enforced to get clue from good individuals instead of bad one, it can transfer good traits to offspring instead of bad traits. Many approaches [65, 86, 95, 96] have strongly focused on "*preserving good edges and adding good edges*". So, we integrate the concept of elite population into REIO in order to explore the search space in a better way with less computational time. The elite population for REIO is selected from Pop_{IO}. Only 1/3 of individuals which are different from each other in fitness are selected as

the elite population. For duplicate individuals only a single one is selected into the elite population. The procedure of generating the elite population is illustrated in Figure 6.4 and the pseudo-code is given in Algorithm 33.

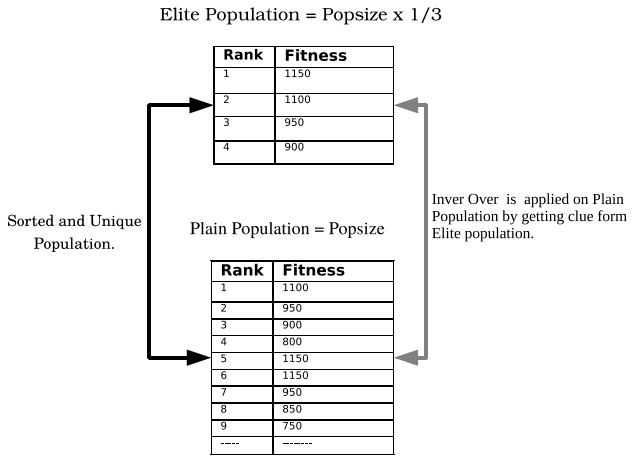


FIGURE 6.4: Elite population is extracted from the plain population and then REIO evolves the plain population by getting clue from the elite population.

Algorithm 33 *GenerateElitePop*(Pop_{IO})

1: Sort the individuals in the population Pop_{IO};
2: **for** $i := 0$ to $popsize/3$ **do**
3: $ElitePop[i]$:= assign the top unique individual among the remaining individuals of the sorted population;
4: **end for**
5: return ($ElitePop[\,]$);

6.3 Experimental Study

6.3.1 Experimental Settings

The proposed GTPHA+NNRI approach was experimented on 14 TSP benchmark instances, in which the number of cities varies from 100 to 1300. The method was implemented in C++ on a 2.66 GHz PC under the Visual Studio Environment. All the TSP benchmark instances were obtained from the TSPLIB except *CHN144*.

Each algorithm on a problem instance was run 20 times and the results were averaged over the 20 runs. The population size was set to 20 for all problem instances initially for Phase 1 and when switched to Phase 2 the population size was increased to 60 individuals including 20 parents, 20 children, and 20 newly created individuals based on the Nearest Neighbour algorithm. The default crossover probability was set to $p_c = 0.65$ and the default mutation probability $p_m = 0.025$. The crossover and mutation probabilities were increased to 0.75 and 0.5 when the best fitness does not show any gain. The switching criteria from Phase 1 to Phase 2 was set to 50 (in Chapters 4 and 5 it was 100). This means that if the best fitness does not change for 50 consecutive generations, then the algorithm is switched to Phase 2. For Phase 2, the key parameter $p = 0.005$ as in the IO operator. The total number of generations was set to 1000 to 3000 for small problem instances with fewer than 200 nodes and to 20000 to 30000 for other problem instances with 300 to 1291 nodes.

The GTPHA+NNRI was compared to five different approaches, including MAX-MIN ACO [135], Iterative Local Search with fixed radius and candidate list (ILS 3-opt-fl-cl), LK-Helsgaun [65], Inver Over, and our previous Random SBMA approach. These methods are considered as state of the art for solving the TSP. The ILS-3-opt uses the fast 3-opt local search [69], MAX-MIN ACO uses the ant colony with additional local search, LK-Helsgaun gives the excellent performance and uses one variant of LK using the iterated local search (ILS), alpha candidate list, and double bridge moves.

Two sets of experiments were performed in this study along with some other experiments to show the effect of various approaches like nearest neighbour sequence construction, applying 2-opt on sequence when created, and the concept of getting clue from the elite population. The first set of experiments is devoted to analysing the sensitivity of parameters of NN-SBGA. In the second set of experiments, GTPHA+NNRI was compared with the afore-mentioned state of the art algorithms for the TSP.

6.3.2 Sensitivity Analysis of Key Parameters of NN-SBMA

The performance of our two phase approach (GTPHA+NNRI) depends on the parameters of Phase 1 internally, the parameters of Phase 2, and when to switch the control from Phase 1 to Phase 2. The key parameters for phase one are as follows: how many generations the Nearest Neighbour technique should use; by which criteria Phase 1 should be switched to Phase 2. For this reason, for the first set of experiments, we use the nearest neighbour sequence generation approach for the initial 0.5% of the total number of generations and use the random sequence generation approach for the rest of generations to maintain the diversity.

Table 6.1 and Figure 6.5 show the fitness and time of the algorithm for 500 generations, averaged over 20 runs. The observation shows that when the sequences are created totally randomly, the average error is 7.40 above the optimum for the TSP instance *pcb1173*. But, when sequences are generated with the nearest neighbour approach for the first 0.5% to 80% of generations, the fitness goes from 7.00 to 0.93 with a little increase in time. From the observation, we can clearly say that the concept of generating sequences based on the nearest neighbour approach works. But, when we go from low to high values for $\delta_{NN_{seq}}$, the individuals suffer from likeliness and may be prematurely converged. So, we set $\delta_{NN_{seq}}$ to the low value, i.e., $\delta_{NN_{seq}} = 0.5\%$, in our experiments.

TABLE 6.1: Comparison results of applying nearest neighbour sequence generation and random sequence generation with different $\delta_{NN_{seq}}$ (% of generations using the nearest neighbour sequence generation approach for *pcb1173*.)

$\delta_{NN_{seq}}$	**Avg-Error**	**Avg-Time**
0.0 (Random)	7.40	$6.74sec$
0.5	7.00	$6.87sec$
10	3.61	$7.14sec$
30	1.45	$7.23sec$
50	1.05	$7.20sec$
80	0.93	$7.15sec$

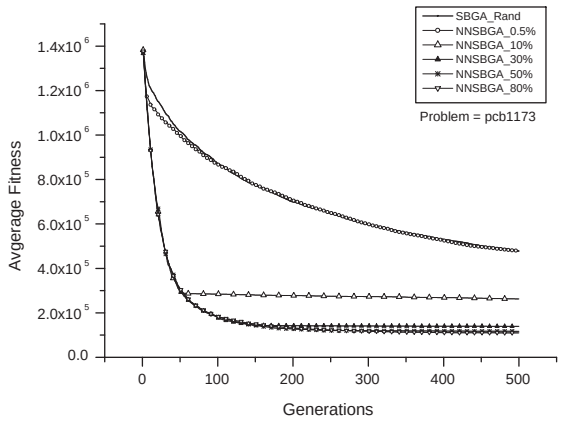


FIGURE 6.5: The effect of sequence generation methods, i.e., SBMA_Rand (sequences generated in a random way), $NNSBMA\_0.5\%$ (sequences generated with $\delta_{NN_{seq}} = 0.5\%$ of total generations), and $NNSBMA$ with $\delta_{NN_{seq}} = 10\%$, 30%, 50%, and 80%.

The efficiency of Phase 1 is totally dependent on several key parameters regarding the solution quality and computational time. Table 6.2 shows the setting of key parameters of Phase 1 in our experiments on the sensitivity analysis of the effect of key parameters on the performance of our GTPHA. The parameter α is the percentage of individuals that are selected from the current population for sequence generation. Here, the value of α was set to 50% and 100%, respectively. β is called the frequency of insertion for the sequence, i.e., (F_{insert}), which determines how many locations are checked for sequence insertion. The value of β was set to 5%, 30%, and

TABLE 6.2: Parameter settings in NN_SBMA

Parameter	**Values**		
α	50	100	–
β	5	30	60
γ	20	50	100
t_{us}	10	20	30

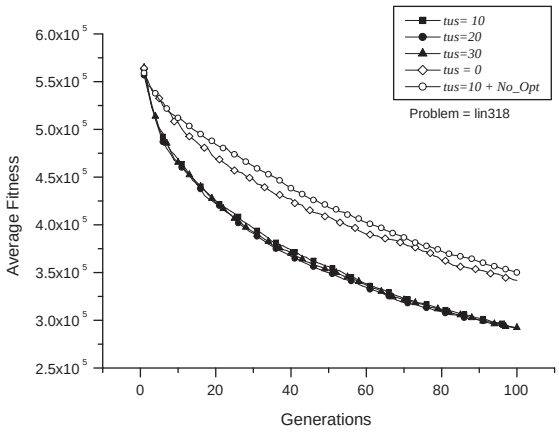


FIGURE 6.6: Comparison of reducing the size of sequences, without reducing size of sequences, and reducing the size of sequences but applying no 2-opt.

60%, respectively. The parameter γ is the switching parameter, which controls when to switch from Phase 1 to Phase 2. Here, γ was set to 20, 50, and 100, respectively. And the parameter t_{us} decides when to reduce the size of sequences. The value of t_{us} was set to 10, 20, and 30, respectively.

In order to help understand the experimental results, Figure 6.6 shows the dynamic performance of the algorithm regarding the fitness gain with and without the 2-Opt improver, and with and without reducing the size of sequences. From Figure 6.6, it can be seen that when we set $t_{us} = 10$, 20, and 30 with the 2-Opt improver, the fitness obtained by the algorithm drops down towards the optimum. Furthermore, the difference between the algorithm with reducing the size of sequences and the algorithm without reducing the size of sequences is quite obvious.

Figure 6.6 shows the effect of reducing the size of the sequences and applying 2-Opt or not applying 2-Opt. The experiments are performed on three different non-zero parameter values for the initial value of t_{us}, which are 10, 20 and 30. These values affect the intensity of reducing the size of the sequences, e.g., if $MaxLS$ is 10 and assigned to the t_{us} variable. During the course of the execution, if the fitness does not improve then t_{us} is decremented after each generation and will be checked against $MaxLS$. If the condition is satisfied for one of the different ranges, then the size of the sequences would be reduced accordingly. In Figure 6.6, if reducing the size of the sequences along with applying 2-Opt is considered, then the performance is better in the cases when t_{us} is 10, 20 and 30 with 2-Opt. However, when t_{us} is 0 (the size of the sequences will not be reduced and will remain equal to $\sqrt{n}$ throughout the run), which means that the set of sequences is generated and 2-Opt is applied, the performance is not good. In another case, when t_{us} is 10 (meaning that the size of the sequences is reduced over time), but the sequences that have been reduced in size are not optimised (without applying 2-Opt), the performance is slightly worse than in the former case (*i.e.*, t_{us}=0). From this we can observe that not only does reducing the size of the sequences helps in gaining fitness, but also optimising the sequences that have been reduced in size gives better results.

Tables 6.4 and 6.5 show the experimental results of our algorithm with different parameter settings. In these tables, "A-Fitness" denotes the average fitness over 20 runs, "Avg-Err" denotes the relative deviation to the global optimal fitness, which is listed in the tables after the instance name, and "Avg -Time" is the average time in seconds used by the algorithm. In order to find out which parameter settings have a great effect on the performance of GTPHA+NNRI, we run the algorithm for different possible 26 combinations based on the settings in Table 6.2, which are named from configuration 1 to configuration 26, respectively. Although many combinations are possible, here we focus on some key parameter settings for the experiments.

We analyse the results using two criteria: one is the average best fitness and the other is the average computational time. In Tables 6.4 and 6.5, the best result

TABLE 6.3: Number of generation for each instance used for the sensitive analysis of key parameters of NN-SBMA experiments in Table 6.4 and 6.5.

Instance	**Global Optima**	**Alloted Generations**
eil101.tsp	629	5000
kroA100.tsp	21282	5000
kroB200.tsp	29437	5000
lin105.tsp	14397	5000
chn144.tsp	30347	5000
d198.tsp	15780	5000
lin318.tsp	42029	25000
pcb442.tsp	50778	25000
d493.tsp	35002	25000
rat783.tsp	8806	25000
u724.tsp	41910	25000
vm1084.tsp	239297	40000
d1291.tsp	50801	40000
pcb1173.tsp	56892	40000

regarding the average best fitness and the average computational time achieved by the algorithm with different configurations for each TSP instance are marked using the symbol "●" and the symbol "■", respectively. For example, for the problem instance *lin318*, the algorithm under configuration 2 (i.e., $\alpha = 50$, $\beta = 5$, $t_{us} = 10$, and $\gamma = 50$) achieved the average error 0.0092 and the average best fitness 42416 with the average computational time 23.6 seconds, which is the best average best fitness achieved by the algorithm with different configurations. This result shows that when the size of sequences is reduced more often and the frequency of insertion is less, the algorithm is more efficient regarding the fitness performance measure. Table 6.3 shows the number of generations used for each instance in the experiments to show the sensitivity analysis of key parameters of NN-SBMA.

For the second criterion of the computational time, the parameter configuration scenario 19, i.e., $\alpha = 100$, $\beta = 5$, $\gamma = 20$, and $t_{us} = 30$, gives the best performance. For this set, the algorithm takes only 18.5 seconds with the fitness error 0.0168. From this configuration, we can easily observe that there is more balance of breaking as $t_{us} = 30$, *i.e.*, this will sustain the same size of sequences for a long time before reduction occurs and the algorithm can have more chance to insert more diverse sequences in proper locations as $\alpha = 100$. The other configurations that give similar

TABLE 6.4: Averaged best value of GTPHA+NNRI with different parameter settings on the TSP instances with 100 to 442 nodes

Set	Parameter				Measure	chn144	d198	eil101	kroA100	kroB200	lin105	lin318	pcb442
	α	β	t_{us}	γ									
					A-Fitness	30742.3	15912.5	629.5	21282	30092.9	14410	42846.9	51426.1
1	50	5	10	20	Avg-Err	0.013	0.0083	● 0.0007	● 0	0.0222	0.0009	0.0194	0.0127
					A-Time	3.5	3.8	2.7	■2.8	4	3	18.7	30.6
					A-Fitness	30800.2	15985.4	631.2	21282	30069	14504.6	42416	51502.2
2	50	5	10	50	Avg-Err	0.0149	0.013	0.0034	0	0.0214	0.0074	● 0.0092	0.0142
					A-Time	5	6.5	3.5	4.5	5.4	3.7	23.6	25.7
					A-Fitness	30892.5	15914	630.3	21286.6	30123.4	14476.3	43008.2	51495.2
3	50	5	10	100	Avg-Err	0.0179	0.0084	0.002	0.0002	0.0233	0.0055	0.0232	0.0141
					A-Time	6.3	8.4	4.6	4.9	7.3	4.3	24	35.3
					A-Fitness	30833.4	16017.2	630.8	21282	30155.1	14458.2	42771	51483
4	50	30	10	20	Avg-Err	0.016	0.015	0.0028	0	0.0243	0.0042	0.0176	0.0138
					A-Time	3.5	4.8	2.9	3.1	4.3	2.9	20.5	33.1
					A-Fitness	31045.4	15984.6	631.9	21284.3	30139.3	14440.7	43085.9	51434.6
5	50	30	10	50	Avg-Err	0.023	0.0129	0.0046	0.0001	0.0238	0.003	0.0251	0.0129
					A-Time	3.9	5.2	2.8	3	5.2	3.1	21.4	29.9
					A-Fitness	31002	15944.5	631.8	21296	30168.9	14498.1	43131.1	51534.7
6	50	30	10	100	Avg-Err	0.0215	0.0104	0.0044	0.0006	0.0248	0.007	0.0262	0.0149
					A-Time	4.4	5.7	3.3	3.7	5.6	3.7	23.1	34.9
					A-Fitness	30959.4	15943	629.5	21330.5	30095.4	14495.8	42897.1	51484.3
7	50	60	10	20	Avg-Err	0.0201	0.0103	0.0007	0.0022	0.0223	0.0068	0.0206	0.0139
					A-Time	4	4.4	2.7	3.2	5.3	3.3	21.7	28.3
					A-Fitness	30954.8	15938.6	631	21284	30080.8	14430.6	42893	51408
8	50	60	10	50	Avg-Err	0.02	0.01	0.0031	0	0.0218	0.0023	0.0205	0.0124
					A-Time	3.6	4.9	3	3.1	4.7	3.7	23.4	35
					A-Fitness	31121	16037	634	21330.5	29973	14577	42887.8	51473
9	50	60	10	100	Avg-Err	0.0255	0.0162	0.0079	0.0022	0.0182	0.0125	0.0204	0.0136
					A-Time	4.3	5.6	3.5	3.3	5.5	4.9	23.1	31.6
					A-Fitness	30710.2	15930.4	633.5	21282	30184	14397	42691.2	51447
10	50	60	20	20	Avg-Err	0.0119	0.0095	0.0071	0	0.0253	0	0.0157	0.0131
					A-Time	3.4	4.5	3.4	3.1	4	3.2	19.1	28.9
					A-Fitness	30971	16125	631	21288.6	30135	14486	43118.6	51481
11	50	60	20	50	Avg-Err	0.0205	0.0218	0.0031	0.0003	0.0237	0.0061	0.0259	0.0138
					A-Time	4.2	7	3.5	3.7	7.2	3.6	24.4	42.6
					A-Fitness	30983	16042	630	21288	30567	14533.2	42895.4	51312
12	50	60	20	100	Avg-Err	0.0209	0.0166	0.0015	0.0002	0.0383	0.0094	0.0206	0.0105
					A-Time	5.1	6.6	3.7	4	5.8	4	25.7	40.8
					A-Fitness	30792.3	15896.7	630	21284	30222	14397.3	42754.3	51501
13	50	60	30	20	Avg-Err	0.0146	0.0073	0.0015	0	0.0266	0	0.0172	0.0142
					A-Time	3.6	■3.7	3.2	3.1	3.4	2.6	19.1	21.8
					A-Fitness	30990	16036	631	21282	29999	14426	43035	51599.3
14	50	60	30	50	Avg-Err	0.0211	0.0162	0.0031	0	0.019	0.002	0.0239	0.0161
					A-Time	5	7.6	4.2	3.8	6.2	4.3	27.1	36.3
					A-Fitness	31063	16135	631	21305	30459	14442	42979	51688
15	50	60	30	100	Avg-Err	0.0235	0.0224	0.0031	0.001	0.0347	0.0031	0.0226	0.0179
					A-Time	7.8	6.7	4.6	5	9	4.8	21.2	32.7
					A-Fitness	31048	15985	630	21282	30245	14412	42498	51268
16	100	30	30	20	Avg-Err	0.023	0.0129	0.0015	0	0.0274	0.001	0.0111	● 0.0096
					A-Time	3.2	4	2.8	3.6	4.2	2.8	19.3	■20.4
					A-Fitness	31140.6	16044.4	634.4	21301.4	30399.6	14469.2	43541	51455
17	100	30	30	50	Avg-Err	0.0261	0.0167	0.0085	0.0009	0.0327	0.005	0.0359	0.0133
					A-Time	6	8.8	3.9	4.8	8.5	4.8	34.3	38.2
					A-Fitness	30983.8	15974.2	632.6	21294.2	30362	14469.4	43236.6	51573
18	100	30	30	100	Avg-Err	0.0209	0.0123	0.0057	0.0005	0.0314	0.005	0.0287	0.0156
					A-Time	6.1	9.5	4.2	5.1	9	5.5	29	34.9
					A-Fitness	30665.6	15894.4	629.6	21282	30170.4	14397.4	42739	51510.6
19	100	5	30	20	Avg-Err	● 0.0104	0.0072	0.0009	0	0.0249	0	0.0168	0.0144
					A-Time	■3.1	3.9	2.7	2.8	■3.3	■2.6	■18.5	21
					A-Fitness	30949.2	16013	631.8	21288.6	30235.8	14433.8	43195.8	51395.8
20	100	5	30	50	Avg-Err	0.0198	0.0147	0.0044	0.0003	0.0271	0.0025	0.0277	0.0121
					A-Time	5.6	7.1	3.7	4.3	6.8	3.7	26.5	29.9
					A-Fitness	31014.2	15953.2	630.6	21319.6	30142.6	14464.4	43039.2	51414.2
21	100	5	30	100	Avg-Err	0.0219	0.0109	0.0025	0.0017	0.0239	0.0046	0.024	0.0125
					A-Time	9.1	12.1	5.4	6.5	10.8	6.2	34.6	48.3
					A-Fitness	30745.8	15882.4	630.6	21282	30018	14397	42703.6	51509.6
22	100	5	20	20	Avg-Err	0.0131	0.0064	0.0025	0	0.0197	● 0	0.016	0.0144
					A-Time	3.4	4.1	■2.6	2.9	3.8	3	18.8	22.5
					A-Fitness	30826.8	15910.8	630	21286.6	30064.2	14450.8	42853	51500.8
23	100	5	20	50	Avg-Err	0.0158	0.0082	0.0015	0.0002	0.0213	0.0037	0.0196	0.0142
					A-Time	6.8	6.9	4	4.8	6.5	3.8	25.5	31.6
					A-Fitness	31078.4	15972	630	21306	30207	14397	43133.8	51527.2
24	100	5	20	100	Avg-Err	0.0241	0.0121	0.0015	0.0011	0.0261	0	0.0262	0.0147
					A-Time	9.2	8.6	5.6	5.6	8	5.8	31.8	40.2
					A-Fitness	30768.4	15863.4	630.4	21282	29799.6	14404.2	42759	51494.8
25	100	5	10	20	Avg-Err	0.0138	● 0.0052	0.0022	0	● 0.0123	0.0005	0.0173	0.0141
					A-Time	3.4	3.9	2.9	2.9	4.1	2.9	19.2	21.8
					A-Fitness	30875.6	15975.8	630.4	21282	30206.8	14554.2	42912.4	51473.2
26	100	5	10	50	Avg-Err	0.0174	0.0124	0.0022	0	0.0261	0.0109	0.021	0.0136
					A-Time	5.5	7	3.3	4.1	6.5	4.4	22.9	38.9

TABLE 6.5: Averaged best value of GTPHA+NNRI with different parameter settings on the TSP instances with 500 to 1291 nodes

Set	Parameter				Measure	d493	u724	rat783	vm1084	pcb1173	d1291
	α	β	t_{us}	γ							
1	50	5	10	20	A-Fitness	35569	43450.3	9108.2	248284	59958.5	53440.3
					Avg-Err	0.0161	0.0367	0.0343	0.0375	0.0539	0.0519
					A-Time	31.8	42.7	39	59.3	67.9	73.9
2	50	5	10	50	A-Fitness	35737.4	43371.6	9107.4	248073	59633	53443
					Avg-Err	0.021	0.0348	0.0342	0.0366	● 0.0481	0.052
					A-Time	43.7	42.5	52.5	71.1	78.6	82.3
3	50	5	10	100	A-Fitness	35692.1	43265.6	9084.5	247385	59886.9	53283.6
					Avg-Err	0.0197	0.0323	● 0.0316	0.0337	0.0526	0.0488
					A-Time	37.3	43.3	57.9	75.1	90.7	90.4
4	50	30	10	20	A-Fitness	35644.6	43286.8	9095.5	248379	59828.5	53490.2
					Avg-Err	0.0183	0.0328	0.0328	0.0379	0.0516	0.0529
					A-Time	39.6	53.5	54.3	83.1	101.3	96.1
5	50	30	10	50	A-Fitness	35714.4	43419.8	9103.1	247201	59806.9	53455.3
					Avg-Err	0.0203	0.036	0.0337	0.033	0.0512	0.0522
					A-Time	46.8	53.3	62.8	79.9	106.4	102.4
6	50	30	10	100	A-Fitness	35660.7	43363.2	9107.9	248188	59845	53480.9
					Avg-Err	0.0188	0.0346	0.0342	0.0371	0.0519	0.0527
					A-Time	48.6	56.9	63.7	84.3	103.6	103.8
7	50	60	10	20	A-Fitness	35680.7	43139	9095	247315	59832.5	53472.1
					Avg-Err	0.0193	● 0.0293	0.0328	0.0335	0.0516	0.0525
					A-Time	41.1	63.4	64	111.1	115.7	114.7
8	50	60	10	50	A-Fitness	35730.4	43369	9120.6	247111	59744.4	53381
					Avg-Err	0.0208	0.0348	0.0357	● 0.0326	0.0501	0.0507
					A-Time	41.9	53.1	89.1	98.9	159.1	117.9
9	50	60	10	100	A-Fitness	35759	43233	9099.6	248276	59698.4	53339
					Avg-Err	0.0216	0.0315	0.0333	0.0375	0.0493	0.0499
					A-Time	59.5	54.1	85	122.3	173.4	182.7
10	50	60	20	20	A-Fitness	35663.4	43348.4	9106.8	248964	60020.4	53509.4
					Avg-Err	0.0188	0.0343	0.0341	0.0403	0.0549	0.0533
					A-Time	29.7	45.1	38.4	74.3	71.2	90.7
11	50	60	20	50	A-Fitness	35695.8	43678	9127	248264	59794.2	53698
					Avg-Err	0.0198	0.0421	0.0364	0.0374	0.051	0.057
					A-Time	48.8	84	73.7	104.8	178.4	132.5
12	50	60	20	100	A-Fitness	35559.5	43364.8	9101.4	247997	59867	53475
					Avg-Err	● 0.0159	0.0347	0.0335	0.0363	0.0522	0.0526
					A-Time	77.6	70	87.6	172.7	165.2	180.2
13	50	60	30	20	A-Fitness	35717.3	43307.7	9101.33	247818	60256.8	53212
					Avg-Err	0.0204	0.0333	0.0335	0.0356	0.0591	0.0474
					A-Time	■29.6	43.2	■36.1	67.3	68.7	86.3
14	50	60	30	50	A-Fitness	35703	43259.3	9124	247704	60333	53766.5
					Avg-Err	0.02	0.0321	0.0361	0.0351	0.0604	0.0583
					A-Time	72.9	63.8	105.4	163.7	125.5	184
15	50	60	30	100	A-Fitness	35670	43194	9104.33	248020	59996	53287
					Avg-Err	0.019	0.0306	0.0338	0.0364	0.0545	0.0489
					A-Time	81.2	67.2	105.2	200.5	234.9	187
16	100	30	30	20	A-Fitness	35764	43674	9138.67	248300	60456	53367.5
					Avg-Err	0.0217	0.042	0.0377	0.0376	0.0626	0.0505
					A-Time	31.2	44	36.6	67.4	67.6	94.6
17	100	30	30	50	A-Fitness	35877	43864	9170	248475	60113	53582
					Avg-Err	0.0249	0.0466	0.0413	0.0383	0.0566	0.0547
					A-Time	82.3	92.1	115	161.4	215.4	188.6
18	100	30	30	100	A-Fitness	35893	43687	9183.67	248596	60222	53213.5
					Avg-Err	0.0254	0.0424	0.0428	0.0388	0.0585	0.0474
					A-Time	69.6	113.6	141.2	174.7	305.7	277.8
19	100	5	30	20	A-Fitness	35711	43462	9115.67	247737	60439	53492.5
					Avg-Err	0.0202	0.037	0.0351	0.0352	0.0623	0.0529
					A-Time	31.9	■37.4	36.6	60.7	62.4	■73.1
20	100	5	30	50	A-Fitness	35646.2	43239.3	9116	248285	59643.8	53251.2
					Avg-Err	0.0184	0.0317	0.0352	0.0375	0.0483	0.0482
					A-Time	50	79.8	63.3	87.4	113	112.6
21	100	5	30	100	A-Fitness	35864.2	43472	9144.8	247486	60027	53203
					Avg-Err	0.0246	0.0372	0.0384	0.0342	0.0551	0.0472
					A-Time	75.1	102.8	100.3	126.8	163.7	164.6
22	100	5	20	20	A-Fitness	35684.8	43499.3	9087.6	249218	60167.8	53854.8
					Avg-Err	0.0195	0.0379	0.0319	0.0414	0.0575	0.0601
					A-Time	34	42.9	37	■58.7	■61.5	76.2
23	100	5	20	50	A-Fitness	35684.4	43359.7	9159.8	247770	59751	53158
					Avg-Err	0.0194	0.0345	0.0401	0.0354	0.0502	● 0.0463
					A-Time	52	64.3	70.6	74	102.7	97.1
24	100	5	20	100	A-Fitness	35659.4	43298.7	9121.6	247849	59886.8	53443.4
					Avg-Err	0.0187	0.0331	0.0358	0.0357	0.0526	0.052
					A-Time	60.3	98.7	95.2	137	148.2	173.1
25	100	5	10	20	A-Fitness	35724.2	43450.7	9101	249618	60103	53442.4
					Avg-Err	0.0206	0.0367	0.0334	0.0431	0.0564	0.0519
					A-Time	35.6	42.2	38.5	60.3	66.9	75.6
26	100	5	10	50	A-Fitness	35709.2	43290.7	9101	248142	59717	53582.8
					Avg-Err	0.0202	0.0329	0.0334	0.0369	0.0496	0.0547
					A-Time	50.9	66.1	62.3	74.9	87.9	98

TABLE 6.6: Comparison results of the original IO with $p = 0.02$, RIO with $p = 0.02$, REIO with $p = 0.005$, and REIO with $p = 0.7$. Each algorithm runs for 5000 generations and the results are averaged over 20 runs

Inver Over Variations	**Avg-Error**	**Avg-Time**
Inver Over *with p=0.02*	13.32	8.73*sec*
Inver Over + LS (RIO) *with p=0.02*	3.32	23.32*sec*
Inver Over+LS+Elite (REIO) *with p=0.005*	2.58	13.31*sec*
Inver Over+LS+Elite (REIO) *with p=0.7*	6.28	118.49*sec*

results are configuration 1 and configuration 22, which have the similar parameter settings as configuration 19 with the only difference in t_{us}. The configurations 1, 22 and 19 take around 18 seconds but the fitness of configuration 22 is slightly better among the three configurations. From this, we can observe that reducing the sequence size can affect the fitness a lot, by introducing or exposing the first and last node of sequences and thus making sequences more diverse, because when the size of a sequence is reduced then the first and last node also change, making them different from the first and last node before reducing the size of the sequence. For a given problem instance, the solution quality achieved by the algorithm mainly depends on the parameters t_{us} and α, while the other two parameters β and γ can affect the running time of NN-SBMA via inserting the sequences and switching to Phase 2.

Table 6.6 shows the experimental results on the refinements of the original IO operator, which further enhance the performance regarding the solution quality and the computational time under the TSP instance *pcb1173*. Figure 6.7 shows the dynamic performance of relevant algorithms regarding the average fitness against the number of generations. For Inver Over (IO) and Restricted IO (RIO), the fitness difference was 13.32 to 3.32, which shows that via restricting the operation of IO, which is random and blind, the solution quality increases but the computational time also increases from 8.73 seconds to 23.32 seconds. By the introduction of elite population into the former two approaches, REIO not only increases the solution quality but also enormously reduces the time to 13.31 seconds. And changing the value of p

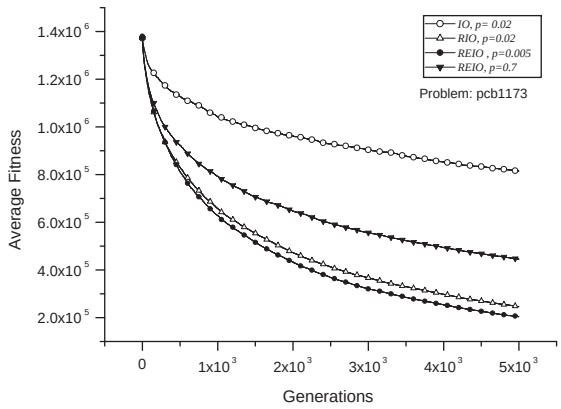


FIGURE 6.7: Comparison of IO with $p = 0.02$, RIO with $p = 0.02$, REIO with $p = 0.005$, and REIO with $p = 0.7$.

from 0.005 to 0.7, which allows IO more chances to do mutation and keeps it away from getting guidance from the elite population, not only worsens the fitness but also extremely lengthens the computational time.

6.3.3 Comparative Experiments

The GTPHA+NNRI was compared to five different approaches, including MAX-MIN ACO [135], ILS 3-opt-fl-cl [69], LK-Helsgaun [65], Inver Over [138], and our previous TPHA approaches. These methods are considered as state of the art for solving TSP instances [70]. The source code of ILS 3-opt-fl-cl for this experimental comparison has been downloaded from the website [70], and the source code for MAX-MIN ACO with additional local search has been downloaded from [136].

Figure 6.8 presents the comparison of GTPHA+NNRI with other algorithms on different TSP benchmark instances. Table 6.7 compares all algorithms in terms of the average best fitness, the average error, and the average computational time over 20 runs. From Figure 6.8 and Table 6.7, it can be seen that GTPHA+NNRI gives a better convergence speed at the initial stage of the solving progress. In terms of the number of evaluations, the ratio between IO and SBMA is 1:3, as GTPHA+NNRI

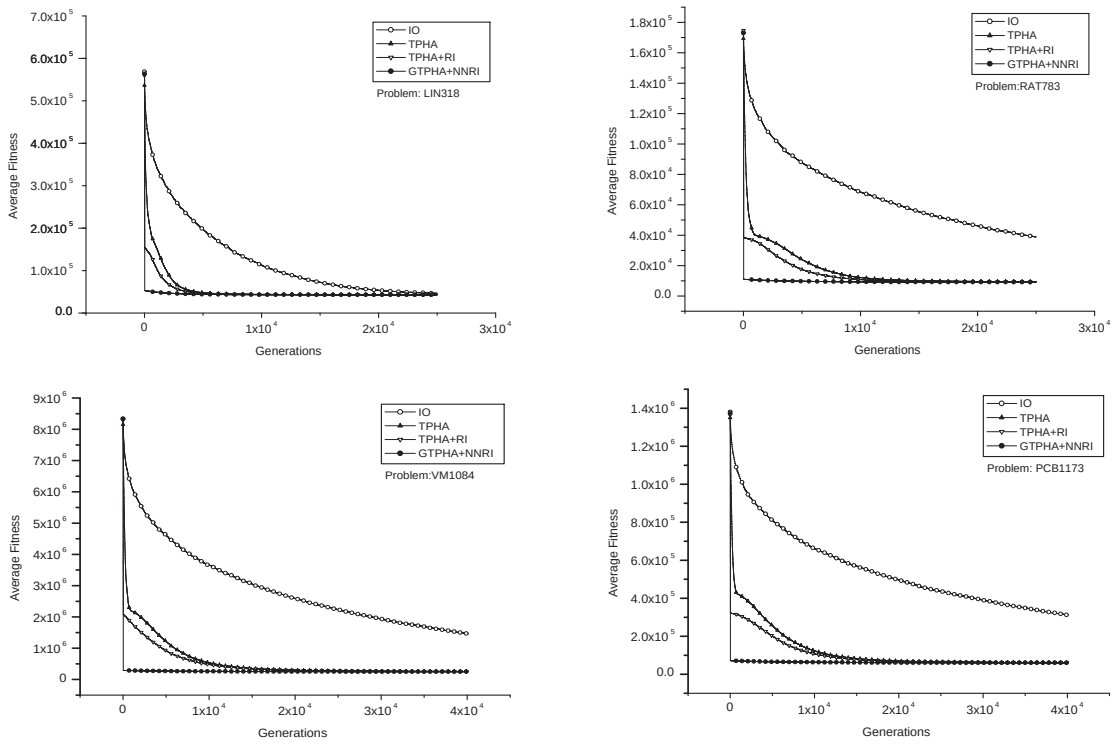


FIGURE 6.8: Experimental results of IO, TPHA, TPHA+RI, and GTPHA+NNRI. The efficiency of our approach GTPHA+NNRI is more prominent for larger problems.

uses the traditional binary operator with an additional embedded SBLS. But due to adaptive behaviour of IO, it gives a better solution quality at the later stage of the hybrid approach. The hybrid approach combines both the features of NN-SBMA and IO. First, NN-SBMA brings the fitness to a near-optimal level in a few generations and then REIO further works and shows progress in the fitness to give a better solution quality.

However, from the results in Table 6.7, GTPHA+NNRI outperforms IO, MAX-MIN ACO and ILS-3-opt-fl-cl on some TSP instances regarding both the convergence speed and the solution quality. The five types of refinements can enhance the performance; both converged more rapidly than original IO. The use of NN-SBMA, restrictive IO, RI, and elite population also has additive effects on the performance gain and the contribution is dominating. From these results, one may speculate that

TABLE 6.7: Comparison results of GPTHA with other well-known algorithms.

Instance	Measure	ILS-3-opt-fr-cl	Max-Min ACO	LK-H	Inver Over	TPHA+RI	GTPHA+NNRI
CHN144	A-Fitness	31083	30773	30347	50401.6	30645	30634
(30347)	Avg-Err	0.0242	0.014	0	0.6608	0.0098	0.0094
	A-Time	6	1.09	0.2	1.49	2.4	2.3
EIL101	A-Fitness	648	641	629	1351.6	637	633.8
(629)	Avg-Err	0.0302	0.019	0	1.1488	0.0127	0.0076
	A-Time	2.7	0.5	0.05	0.37	0.7	0.5
KROA100	A-Fitness	21767	21360	21282	47544.6	21324.8	21282
(21282)	Avg-Err	0.0227	0.0036	0	1.234	0.002	0
	A-Time	4	0.9	0.1	0.59	0.9	0.9
LIN318	A-Fitness	43414	42035	42029	46579.8	42980.2	42781.5
(42029)	Avg-Err	0.0329	0.0001	0	0.1082	0.0226	0.0179
	A-Time	26	18	3.7	11.8	14.9	15
PCB442	A-Fitness	52597	50875	50778	69664.2	51586.3	51214
(50778)	Avg-Err	0.0358	0.0019	0	0.3719	0.0159	0.0085
	A-Time	17	31	3.5	16.22	19.6	20
RAT575	A-Fitness	7025	6974	6773	14250	7102.3	6953
(6773)	Avg-Err	0.0372	0.0296	0	1.1039	0.0486	0.0265
	A-Time	38	0.05	5.5	20.77	25.9	26
RAT783	A-Fitness	9144	9072	8806	31935	9270.4	9035
(8806)	Avg-Err	0.0383	0.0302	0	2.6265	0.0527	0.026
	A-Time	58	0.07	18.3	24.91	36.1	35
U724	A-Fitness	43395	42019	41910	141619	43786.4	42902
(41910)	Avg-Err	0.0354	0.0026	0	2.3791	0.0447	0.0236
	A-Time	53	2.5	20.3	23.72	49.4	38
V1084	A-Fitness	249134	239384	239359	1501480	251821	246350
(239297)	Avg-Err	0.0411	0.0003	0.0002	5.2745	0.0523	0.0294
	A-Time	103	43	113	50.8	93.1	69
PCB1173	A-Fitness	59697	59697	59697	311256	61641.7	59616
(56892)	Avg-Err	0.0493	0.0493	0.0493	4.4709	0.0834	0.0478
	A-Time	90	37	15	48.61	91.3	66

our approach is more effective and increases the "Adaptive Power" of the IO which is not fully contributed by the original IO in case of small as well as in large TSPs instances. In terms of the computational time, it is obvious that GTPHA+NNRI does not perform better than LK-H.

In Figure 6.8, algorithms are shown in different line styles. It is obvious from the plots that our proposed approaches are not overlapping. This means that each and every refinement can contribute to some additional performance gain. These contributions are more effective for large problems. These refinements not only decrease the error rate but also reduce the CPU time used in almost all problem instances.

6.4 Chapter Summary

It is well-known that it is very hard for a pure genetic algorithm to fine tune the search in complex spaces, especially in a combinatorial search space, and hybridization with other techniques can greatly improve the performance of GAs. In this regard, we have proposed the idea of nearest neighbour sequence based memetic algorithm into a two phase hybrid algorithm, denoted GTPHA+NNRI, to solve the TSP. An evolutionary algorithm for solving the TSP should include mechanisms for fast convergence, reducing the computational time, and maintaining the diversity of population. The above mentioned approach integrates the binary order crossover operator, the double bridge mutation operator, the embedded local search, and the techniques which can increase the adaptive power of the unary Inver Over operator. The integration of binary and unary approaches improves the overall search capability.

In order to test the performance of GTPHA+NNRI for the TSP, experiments were performed to analyse the sensitivity of parameters and the effect of various refinement strategies for the performance of GTPHA+NNRI based on a number of TSP instances. The experimental results of GTPHA+NNRI were also compared with several state-of-the-art algorithms from the literature on the test TSP instances. The experimental results also confirmed that the proposed approach is competitive and shows reasonable efficiency. In general, with the hybridization of various operators and local search strategies, GTPHA+NNRI is able to find optimal or near optimal results for the tested TSP instances.

In the next chapter, we will extend the current GTPHA to solve the dynamic TSP.

Chapter 7

A Guided Two-Phase Hybrid Algorithm for the Dynamic Travelling Salesman Problem

7.1 Introduction

Memetic algorithms (MAs) have attracted an expressive curiosity in recent years and are being progressively used to solve real-world problems. Many real-world optimization problems are time-dependent and can be modelled as dynamic optimization problems (DOPs). One typical example is the travelling salesman problem (TSP) which could be modelled as a dynamic TSP (DTSP). In the DTSP, the number of cities may change over time and the cost matrix may also change over time. So, the DTSP becomes one of the harder versions of the static TSP, which is an NP-hard problem.

For DOPs, an MA needs to promptly adapt to the environment. One important factor is to provide some diversity schemes to address the problem of premature convergence. It is also quite beneficial for an MA to get good-quality solutions

quickly. This requirement may be achieved by integrating some domain-specific knowledge in the evolutionary process to track the dynamic optima in a short time and further guide an MA to explore the search space.

To address the above issues or considerations, we investigate a two-phase MA (TPMA) to improve the performance of MAs for solving the DTSP. In the first phase, the TPMA contributes in fitness gain by adding good edges. When phase one is unable to improve the performance, the control is shifted to the second phase with the introduction of some random immigrants scheme and some additional structures, which further guides the MA to take the knowledgeable decisions instead of the random one. We also investigate in this study how to bridge the two phases in a better way to get the good-quality solution in the minimum time. The experimental results show that our proposed approach is effective in solving the DTSP and has the potential to be extended for other dynamic combinatorial optimization problems as well.

7.2 Modelling the DTSP

The TSP is a well-known classic *Combinatorial Optimization Problem* (COP). The static TSP can be shortly stated as: given a set of n cities and the geographical distances between them, the travelling salesman has to find the cheapest tour of visiting all the cities exactly once and returning to the starting city. In the mathematical form, the length of a tour is calculated as follows:

$$l(\pi) = \sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)} + d_{\pi(n),\pi(1)}$$

where d_{ij} is the distance between city i and city j, and π is a permutation of $1, 2, \ldots, n$. Thus, an instance I is defined by a distance matrix $D = (d_{ij})$ and a solution to the TSP is a vector π with $j = \pi(i)$ denoting city j to visit at the i-th step.

If the cost between cities (or nodes), or the number of cities in the TSP problem changes with time, then the TSP becomes a DTSP. A DTSP has some additional properties like the number of nodes or cities n may change with time: some cities may appear and some old ones may disappear. Secondly, the city location may geographically change or the cost matrix may change with time as well. Mathematically, for the DTSP, the cost matrix can be formulated as follows:

$$D(t) = d_{ij}(t)_{n(t)\times n(t)}$$

where $d_{ij}(t)$ is the cost from city i to city j at time t, $n(t)$ is the number of cities at time t, and t is the real-world time. So, the aim of the DTSP is to find a tour $\pi(t) = \pi_1, \pi_2, \ldots, \pi_{n(t)}$ in order to minimize

$$l(\pi)(t) = \sum_{i=1}^{n(t)} d_{\pi_i,\pi_{i+1}}(t),$$

where $l(\pi)(t)$ is the length of the tour $\pi(t)$, which is a permutation over the set $\{1, 2, \ldots, n(t)\}$, and $\pi_{n(t)+1} = \pi_1$.

The DTSP is more challenging and realistic as compared to its static version. Many real-world TSPs are dynamic in nature. For example, a distributing salesman/vehicle wants to distribute some goods to some different cities from one starting point. He has to visit all the cities in a proper order regarding the time and energy. For the same reason, the salesman must choose an optimal path, but it may happen that with a passage of time the salesman must face some changes, *i.e.*, he may skip some cities, or due to traffic jams the existing route may be affected and the salesman must re-route his plan.

For our experimental analysis, we have designed two test beds. The first one uses a main pool of cities and a spare pool of cities. The second one is via changing the positions of nodes. The description of these test beds is given as follows:

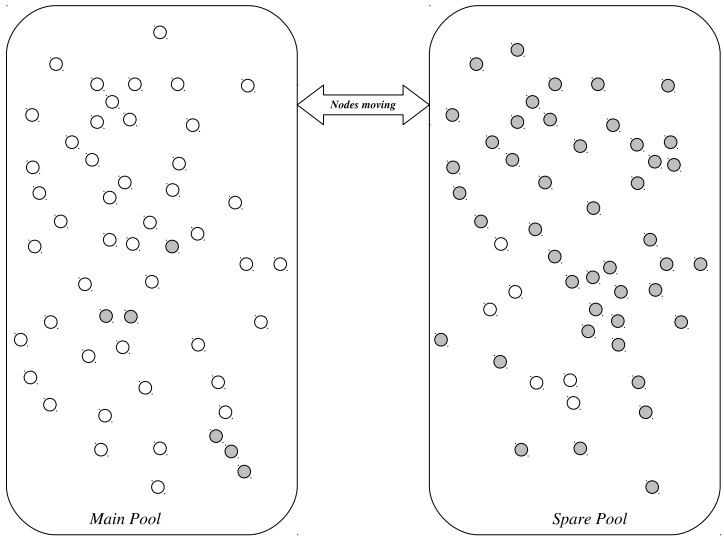


FIGURE 7.1: Illustration of constructing the DTSP using a Main Pool and a Spare Pool.

Main Pool and Spare Pool (MS): We divide the number of cities of the original TSP problem into two equal sets: a *Main Pool* and a *Spare Pool*. The algorithm works on the nodes of the Main Pool. However, when a change occurs, some percentage of nodes from the Main Pool are swapped with the same number of nodes from the Spare Pool. This way, the number of nodes remains the same throughout the process. This DTSP test bed is generated as follows: Every f generations m percentage of random nodes are swapped between the Main Pool and the Spare Pool. Here, f denotes the *frequency of changes* and m denotes the *severity of changes*. Figure 7.1 illustrates the basic idea of constructing the DTSP using the Main Pool and Spare Pool.

Changing the Positions of Nodes (PN): In this test bed, the total number of nodes remains the same. Every f generations, m percentage of random nodes are selected and their geographical positions are changed. For example, suppose that $node_1$ is located at (x_1, y_1) before a change. After the change, the coordinates of $node_1$ are changed by a random function, which simply moves the node to a new location (x'_1, y'_1). Figure 7.2 illustrates the basic idea of constructing the DTSP based on moving the nodes.

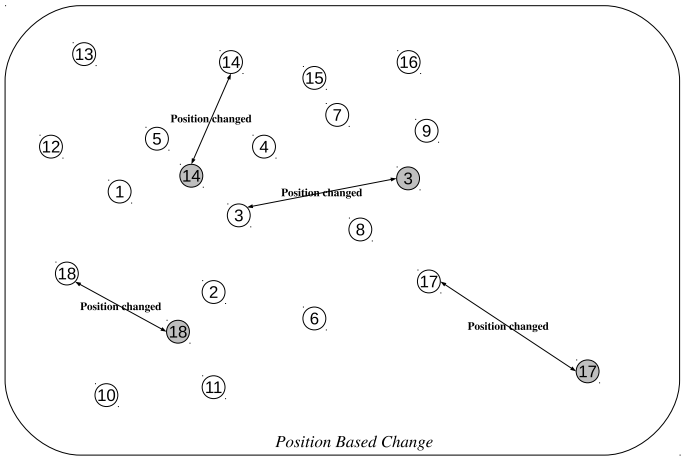


FIGURE 7.2: Illustration of constructing the DTSP via changing the positions of nodes.

For handling the dynamic TSP we have kept the number of cities equal before and after the change. In the real-world situation, for example in the wireless sensor network, the number of nodes may not remain the same after a change for some reasons like power issues, etc. By keeping this view in mind it is desirable that the current approach should be capable enough to handle situations in which the number of nodes may vary from one change to another change. In that situation, a kind of repair algorithm would be required to remove or add the nodes within the existing chromosomes of the population.

7.3 Proposed Approach for Solving the DTSP

In this chapter, we extend our guided two phase hybrid approach (GTPHA), described in Chapter 6, to solve the DTSP. The extended algorithm is called *dynamic GTPHA* (DGTPHA), of which the pseudo-code is given in Algorithm 34. Similar to GTPHA, DGTPHA also consists of two phases: the first phase is based on the Nearest Neighbour Sequence Based Memetic Algorithm (NNSBMA) and the second phase is based on the Restricted Elite Population Inver Over (REIO) operator. In the first phase, the NNSBMA brings gain in fitness. When the first phase is unable

to give any further improvement in fitness, then control is shifted to the second phase, which is mainly based on the Inver Over (IO) operator.

Here, the traditional IO operator is strengthened by integrating some techniques. The first one is the elite population approach, which means that, instead of getting the clue from random individuals from the population, the IO operator gets the clue from an elite population, of which the individuals are extracted from the main population and have good fitness. The second technique which is integrated with the main IO operator is the gene pool, which is generated based on the concept of the *Minimum Spanning Tree* (MST).

Furthermore, the switching criterion from Phase-1 to Phase-2 in DGTPHA is modified a little in the view of the DTSP. The following considerations are made:

1. Base Criterion (BC): If Phase-1 is not able to improve the best fitness of the population after $\gamma = 10$ consecutive iterations, then Phase-1 will be shifted to Phase-2.

2. Assuming that the change frequency is f, *i.e.,* the problem changes every f generations, then Phase-1 is allowed to run up to $\rho_{phase\_1} \times f$ generations, where ρ_{phase_1} is the percentage of the maximum number of generations Phase-1 can run to the total number of generations between two changes. This shifting criterion aims to keep the balance between Phase-1 and Phase-2. For example, if $f = 100$ and $\rho_{phase\_1} = 50\%$, when the number of generations of Phase-1 reaches 50, the control will be shifted to Phase-2.

3. When a change occurs, the key parameters are reset to their default settings, and the control is given to Phase-1 again.

In the following sections, the refinements within the DGTPHA over the GTPHA to solve the DTSP are described in details. We first present the overall picture of the NNSBMA, and then present the integration of elite population and rotating gene pool within the second phase.

Algorithm 34 Dynamic Guided Two-Phase Hybrid Algorithm (DGTPHA)

Set parameters, e.g., p_c, p_m, n_{inv}, s_{seq}, α, β, γ, and t_{us} to their default values;
if $(Change\_Type = MS)$ **then**
  copy randomly half of all n nodes to the Main Pool and Spare Pool;
  $n := n/2$;
end if
Create $Distance\_Matrix$ according to the problem instance;
$Gene\_Pool := GenerateGenePool()$;
$nnList_{size} := \lfloor \sqrt{n} \rfloor$;
$nnList_{n \times \lfloor \sqrt{n} \rfloor} := GenerateNearestNeighborList(nnList_{size})$;
$no\_change\_count := 0$;
$Change_{problem} := 0$;
$F_{insert} := \beta$;
$Phase_1 := true$;
$Pop :=$ Initialise a population of $popsize$ individuals;
$GenerateNearestNeighborSequence(N_{seq}, nnList)$;
repeat
  if $(Phase_1 = true)$ **then**
    call $NNSBMA()$;
    if $(Switch\_Criteria(no\_change\_count) \geq \gamma)$ **OR** $(Change_{problem} \geq (\rho_{phase\_1} \times f))$ **then**
      $Phase_1 := false$;
      $IO_{popsize} = 3 \times popsize$;
      $Pop_{IO} :=$ add nearest neighbor based random individuals into $Pop + Pop_{tmp}$;
    end if
  else
    $ElitePop[\,] := GenerateElitePop(PopIO)$; // generate the elite pop for r-REIO
    for each individual $ind_i \in Pop_{IO}$ **do**
      $r\text{-}REIO(ind_i, ElitePop[\,], Gene\_Pool)$;
    end for
  end if
  if $(Change_{problem} \geq f)$ **then**
    if $(Change\_Type = MS)$ **then**
      $ChangeProblem\_MS(change\_severity)$;
    else
      $ChangeProblem\_PN(change\_severity)$;
    end if
    $Change_{problem} := 0$;
    $Phase_1 := true$;
    Reset parameters, e.g., p_c, p_m, n_{inv}, and s_{seq} to their default values;
  end if
  $Gen_{running} + +$;
  $Change_{problem} + +$;
until $(Gen_{running} \geq MaxGen)$

Algorithm 35 NNSBMA()

```
1: mating_pool := TournamentSelect(Pop);
2: // Crossover
3: for j := 0 to popsize do
4:    Select two parents i_a and i_b from the mating_pool;
5:    if (rand(0,1) < p_c) then
6:       Create child_a and child_b by e-SBOX(i_a, i_b, F_insert);
7:       Add child_a and child_b to Pop_tmp;
8:    end if
9: end for
10: // Mutation
11: for each individual ind_i ∈ Pop_tmp do
12:    if (rand(0,1) < p_m) then
13:       e-SBDB(ind_i, n_inv, F_insert);
14:       Add ind_i to Pop_tmp;
15:    end if
16: end for
17: AdaptParameters(α, t_us, Gen_running);
18: Pop := SelectNewPop(Pop + Pop_tmp);
```

7.3.1 Phase-1: NNSBMA

As mentioned above, we extend our previous approach in Chapter 6 to solve the DTSP. The procedures of Phase-1 and Phase 2 remain almost the same. The only modification that we have made lies in Phase 2, where the rotating gene pool approach is introduced in order for the adaptive IO to be able to add and preserve more and more good edges.

The pseudo-code of one generation of NNSBMA is given in Algorithm 35. The crossover (e-SBOX), mutation (e-SBDB), local search (SBLS) operators, and the procedure for adapting the parameters for maintaining the diversity are the same as those we proposed for tackling the static TSP in Chapter 6.

7.3.2 Phase-2: REIO with Rotated Gene Pool (r-REIO)

For Phase-2, the following changes are made in the framework of DGTPHA over GTPHA in Chapter 6. Integrating domain-specific knowledge through an efficient

heuristic can affect the performance of algorithms for the DTSP. To make the IO operator more efficient, we introduce the concept of gene pool. Similar concept has also been employed and recommended by other researchers [85, 150, 151]. We have adapted the idea of gene pool proposed by Yang *et al.* in [150]. This idea indicates that some good gene fragments can be obtained from the minimum spanning tree. But in our approach, we rotate the set of edges associated with a node to avoid repetition in order for the IO operator to have higher chances to explore more and more options for inversion. The pseudo-code of the r-REIO algorithm is given in Algorithm 36.

Algorithm 36 *r-REIO*$(route, ElitePop[\,], Gene\_Pool)$

```
1: p := 0.02;
2: route* := route;
3: Select randomly a city C from route*;
4: while (TRUE) do
5:    p_tmp := rand(0,1);
6:    if p_tmp < p then
7:       Select the city C* from the remaining cities in route*;
8:    else
9:       if (p_tmp > p) and (p_tmp <= 0.33) then
10:         Select the city C* from the gene pool Gene_Pool of C and rotate Gene_Pool accordingly;
11:      else
12:         Select randomly an individual route_select from the elite population ElitePop[ ];
13:         while (route_select = route*) do
14:            Select randomly an individual route_select from the elite population ElitePop[ ];
15:         end while
16:         Assign to C* the next city to C in the selected route;
17:      end if
18:   end if
19:   if (the next or previous city of city C is C* in route*) then
20:      Exit from the while loop;
21:   end if
22:   Invert the section from the next city of city C to city C* in route*;
23:   C := C*;
24:   if (Length(route*) < Length(route)) then
25:      route := route*;
26:   end if
27: end while
```

The procedure of generating the gene pool is shown in Algorithm 37. First, a minimum spanning tree is generated over the working set of nodes. Then, the nodes which are linked with any specific *node* in the minimum spanning tree are stored in

Algorithm 37 *GenerateGenePool*()

1: Generate a Minimum Spanning Tree T;
2: **for** $node := 0$ to n **do**
3: Find the edges in T which are connected to $node$;
4: Copy the edges connected to $node$ to $Gene\_Pool[node][\,]$;
5: **end for**
6: return($Gene\_Pool$);

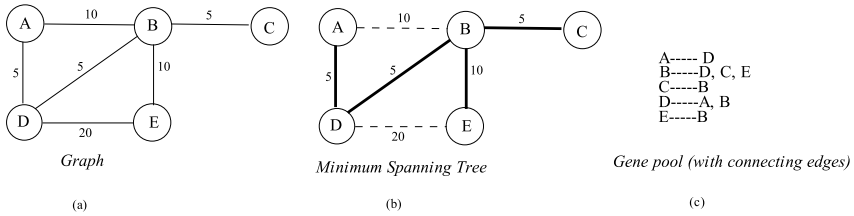


FIGURE 7.3: (a) Graph; (b) the Minimum Spanning Tree (MST); (c) Gene pool showing connected edges to each node, e.g., $Gene\_Pool[B] \rightarrow \{D, C, E\}$, which means that nodes D, C, and E are connected to node B in the MST.

a gene pool. The structure of this gene pool is similar to the nearest neighbor list $nnList$. The only difference is that, in the gene pool, the number of nodes which are linked to a node is variable, i.e., there may be one or more of them, while in $nnList$, each node has a fixed number of neighbors, i.e., $\sqrt{n}$ neighbors. For example, Figure 7.3 illustrates the generation of the gene pool from a given graph.

In r-REIO, we slightly modify the gene pool concept. In order to maintain the diversity and make use of the strength of the gene pool, when the IO operator picks a city from the existing gene pool, the corresponding list is rotated. So, for the next iteration, if the same node is under consideration, a new city would be picked from the gene pool. For example, Figure 7.3 (c) represents an edge list. If in an iteration, $city1 = B$, $city2 = D$ would be selected for the next city within the individual. Then, the gene pool list $B \rightarrow D, C, E$ would be rotated to $B \rightarrow C, E, D$ for the next iteration.

7.3.3 The Change Functions

The dynamic changes are also represented in Algorithm 34, where $Change\_Type = MS$ means that the change function is based on the main and spare pools and $Change\_Type = PN$ means that the change function is based on positions of nodes. Every f generations, a change occurs: m percentage of nodes are changed and the distance matrix, the nearest neighbor list, and the gene pool are also updated. Details of the change functions $ChangeProblem\_MS(m)$ and $ChangeProblem\_PN(m)$ are given in Algorithm 38 and Algorithm 39, respectively. In these two algorithms, $GenerateNearestNeighborList(nnList_{size})$ has been defined in Algorithm 26 and $GenerateGenePool()$ has been defined in Algorithm 37.

Algorithm 38 $ChangeProblem\_MS(m)$

1: $m' := n \times m\%$;
2: Select randomly a set of m' nodes from the main pool;
3: Swap the m' selected nodes with m' random nodes from the spare pool;
4: Update $Distance\_Matrix$;
5: $Gene_{pool} := GenerateGenePool()$;
6: $nnList_{n\times\lfloor\sqrt{n}\rfloor} := GenerateNearestNeighborList(nnList_{size})$;
7: Evaluate the population;

The main difference between the two functions lies in that, in $ChangeProblem\_MS(m)$, $m\%$ nodes are swapped between the spare pool and the main pool. However, in $ChangeProblem\_PN(m)$, $m\%$ nodes are first randomly selected. Then, the position of each selected node is moved slightly. In the Algorithm 39, first the random function generates a random value to x and y then it is checked that the newly generated co-ordinates lies in between the x_{upr} and x_{lwr}. The procedure is the same for y *i.e.,* $x_{lwr} \leq x \leq x_{upr}$, if not then the else part is executed $node_i.x = rand(x_{lwr} + 1, x_{upr} - 1)$. The procedure is the same for moving the y co-ordinates as well. We have used the τ which mainly effect the movement of (x, y) co-ordinates, if τ is low the movement or jump of the node is short, but if it is high the movement or jump of the node is high. We kept the value $\tau = 0.9$ for these experiments.

Algorithm 39 *ChangeProblem_PN*(m)

$m' := n \times m\%$;
Select randomly a set of m' nodes;
for $i := 0$ to m' **do**
  $x := node_i.x + (\tau \times (x_{upr} - x_{lwr}) \times rand(-1, 1))$;
  $y := node_i.y + (\tau \times (y_{upr} - y_{lwr}) \times rand(-1, 1))$;
  if $((x \leq x_{upr})$ AND $(x \geq x_{lwr}))$ **then**
    $node_i.x := x$;
  else
    $node_i.x = rand(x_{lwr} + 1, x_{upr} - 1)$;
  end if
  if $((y \leq y_{upr})$ AND $(y \geq y_{lwr}))$ **then**
    $node_i.y := y$;
  else
    $node_i.y := rand(y_{lwr} + 1, y_{upr} - 1)$;
  end if
end for
Update $Distance\_Matrix$;
$Gene_{pool} := GenerateGenePool()$;
$nnList_{n \times \lfloor \sqrt{n} \rfloor} := GenerateNearestNeighborList(nnList_{size})$;
Evaluate the population;

7.4 Experimental Study

7.4.1 Experimental Setting

To examine the performance, all the algorithms were tested on the DTSP instances that are constructed from *chn*144, *krob*200, *lin*318, *pcb*442 and *u*724, which are taken from the TSPLIB except the first one. The experiments were performed with two values of f and 4 values of m, which gives overall 8 combinations of f and m. The value of f was set to 50 and 100, while the value of m was set to 10, 25, 50, and 75, respectively.

Each algorithm on a problem was run 30 times, and the results are averaged over 30 runs. The population size was set to 20 for all problems initially for Phase-1 and when switched to Phase-2, the population size was increased to 60 individuals including 20 parent, 20 children, and 20 newly created immigrant individuals (RI:

generated randomly, or NNRI: generated with nearest neighbour algorithm). The crossover probability was set to $p_c = 0.65$ and the mutation probability $p_m = 0.025$. The crossover and mutation probabilities were increased (by Adapting Parameters as in Algorithm 30) until they reach 0.85 and 0.075, if the best fitness does not show any improvement. The switching criterion from Phase-1 to Phase-2 was kept $\gamma = 10$; this means that if the best fitness does not change for 10 consecutive generations while handling DTSP, then the algorithm is switched to Phase-2. For Phase-2, the key parameter $p = 0.02$ was kept the same as in the original IO operator. The number of generations was kept 500 for all problems.

The performance of our DGTPHA depends on the parameters of Phase-1 internally, the parameters of Phase-2 and when to switch the control from Phase-1 to Phase-2. The key parameters for Phase-1 are: how many generations the nearest neighbour technique should be used, and under which criteria Phase-1 should be switched to Phase-2. For this reason, we set the following parameters. For the first set of experiments, we set 10% of the total generations for the nearest neighbour sequence generation and the rest of generations for random sequence generation to maintain the diversity.

Table 7.1 shows the internal setting of parameters of Phase-1 and the efficiency of the first phase is totally dependent on these parameters regarding the fitness quality and computational time. The parameter α is the percentage of individuals that are selected from the current population for sequence generation. Here, the value of α was set to 100%. The parameter β (i.e., (F_{insert})) determines how many locations are checked for sequence insertion. The value of β was set to 5% of the total nodes of the individual. The parameter γ controls when to switch from Phase-1 to Phase-2. Here, γ was set to 10, which means that the algorithm would switch to Phase-2 when the best fitness does not change for consecutive 10 generations. In addition, to keep the balance between Phase-1 and Phase –2, when Phase-1 reaches 50% of the change frequency in between two changes, control is shifted to Phase-2 automatically. And

the parameter t_{us} decides when to reduce the size of sequences. The value of t_{us} was set to 10.

TABLE 7.1: Parameter Settings in NNSBMA

Parameter	Setting
α	100
β	5
γ	10
t_{us}	10

The internal parameters of Phase-2 were set as follows. The key parameter $p = 0.02$ was kept the same as that of the original IO operator. A random number p_{tmp} is generated randomly in the range $[0, 1]$. If $(p_{tmp} < p)$, the mutation part of IO would be used, *e.g.*, *city*2 would be selected at random. Otherwise, if $(p < p_{tmp} < 0.33)$, then *city*2 would be selected from the gene pool. If $(p_{tmp} > 0.33)$, *city*2 would be selected from any elite individual.

In the following experimental studies, we will show the effect of various heuristic techniques and key parameters on the performance of our proposed approach for the DTSP.

7.4.2 Effect of Sequence Generation Methods

Figure 7.4 shows the comparison between the DGTPHA with sequences generated with the nearest neighbor list and the DGTPHA with sequences generated without the nearest neighbor list on *chn*144 with $f = 100$ and $m = 10\%$. It can be seen that using the nearest neighbor list, which ultimately brings the nodes nearer to each other during the first phase of our MA, contributes a lot to the overall better results.

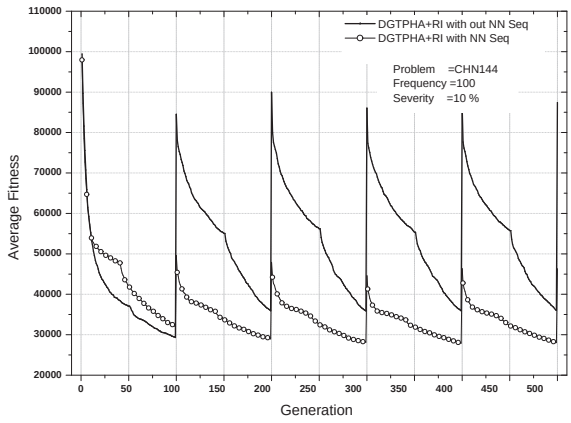


FIGURE 7.4: The effect of sequences generated with and without the nearest neighbor list within the first phase of DGTPHA.

7.4.3 Effect of Immigrants Schemes in Phase-2

In case of tackling the DTSP, the time between two changes is very limited. To get a better solution quality, we analyse the effect of two different immigrants schemes when the second phase is started. When the control is given to the second phase, we add immigrants generated by the nearest neighbor algorithm or immigrants generated randomly. Figure 7.5 shows the effect of immigrants generated randomly and generated by the nearest neighbor algorithm, where a change occurs every 100 generations, during which in DGTPHA the Phase-1 will run for the first 50 generations at most (i.e., $\rho_{phase\_1} = 50\%$), while the Phase-2 runs for the remaining 50 generations at least. In Figure 7.5, "BC+50" means that the base criterion with $\gamma = 10$ and $\rho_{phase\_1} = 50\%$ are used for shifting Phase-1 to Phase-2, but Phase-2 starts without immigrants. "BC+50+RI" means that Phase-2 starts with random immigrants. Moreover, "BC+50+NNRI" means that Phase-2 starts with immigrants generated based on the nearest neighbor algorithm.

From Figure 7.5, it can be seen that when immigrants are generated through the

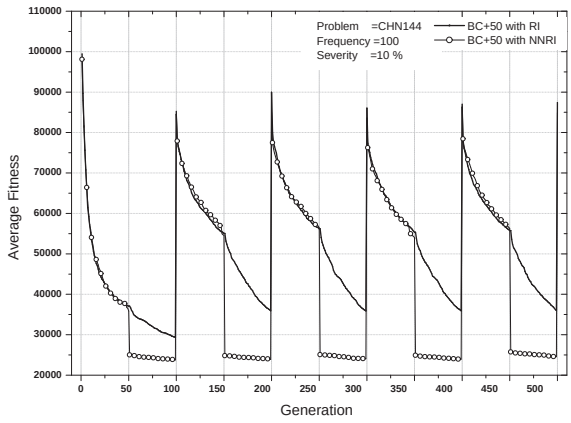


FIGURE 7.5: The comparison of the algorithms with random immigrants and nearest neighbour based immigrants schemes.

Nearest Neighbour algorithm it can give better performance, but it makes the individuals rigid for further improvement. On the contrast, if the immigrants are generated in a random way, it may take a longer time to converge but may guarantee to give better solutions at the end.

7.4.4 Effect of Shifting Parameter to Phase-2

In general, key parameters affect the performance of algorithms. The performance of our approach strongly depends on the shifting parameter from Phase-1 to Phase-2. In this experiment, we analyse how many generations should be given to Phase-1 (SBMA with nearest neighbour sequences) and how many generations should be given to Phase-2 (REIO) since the first phase is clearly time consuming and it would be better to keep the balance between the two phases. The first phase in general collects useful information and knowledge for the second phase and the second phase integrates the knowledge extracted by the SBMA algorithm.

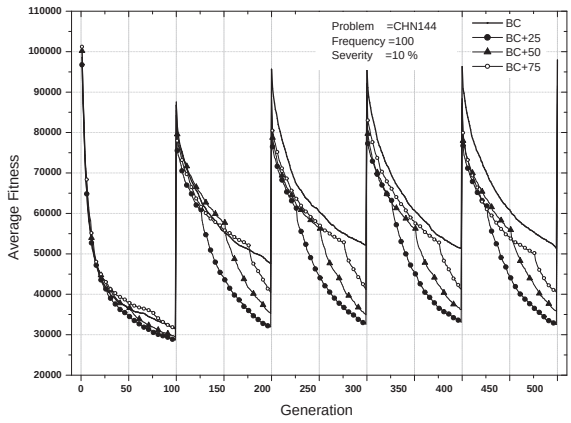


FIGURE 7.6: The effect of when to shift from Phase-1 to Phase-2.

In order to show how much the shifting criterion affects the overall performance of our algorithm, we carry out experiments with different shifting criteria in our algorithm. Figure 7.6 shows the performance of the algorithm under various conditions of shifting over to Phase-2. In the figure, BC means that only the base criterion of shifting is used, *i.e.,* if Phase-1 is un-able to change the fitness for consecutive $\gamma = 10$ generations, control is shifted to Phase-2. Here, there is no restriction regarding how many generations Phase-1 will run. For example, if in the first phase the fitness is continuously changing and the base criterion is never fulfilled, then the control will not be shifted to Phase-2. In Figure 7.6, "BC+X" has the similar meaning as we explained before for "BC+50". For example, "BC+25" means that the first phase will run for up to 25 generations if the BC is not fulfilled, given that $f = 100$.

From Figure 7.6, it can be seen that when less time is given to Phase-1 instead of Phase 2, the overall performance of the algorithm increases. This is because the second phase has more time to get better solutions before the next environmental change takes place. From the experimental analysis, it is obvious that it is not necessary that more time should be given to Phase 1 and less time should be given to the Phase-2.

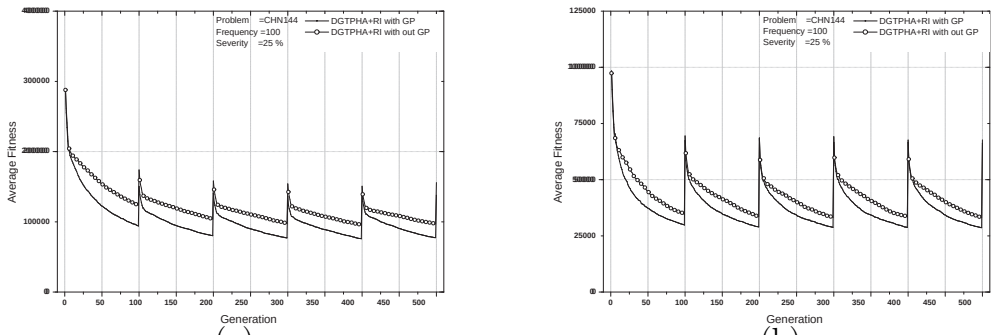


FIGURE 7.7: The comparison of algorithms with and without the gene pool on the DTSP test bed based on (a) the change of positions of nodes, and (b) the main pool and spare pool.

7.4.5 Effect of the Gene Pool

The purpose of the gene pool is to preserve the edges that may be promising edges of optimal solutions and may further guides the evolution of the population toward better fitness. So, the gene pool stores useful information from time to time from the problem when a change occurs. The useful information is extracted by creating the edge list from the minimum spanning tree (MST). In most of the cases, over 70% edges of the MST are part of the global optima [150]. The previous gene pool is further updated with the new edge list and further guide the second phase along with the elite population. The basic idea behind the gene pool is that it prevents the individuals from adding an edge which results in bad fragments up to some extent.

Figure 7.7 shows the comparison of algorithms with and without the gene pool on the DTSP test beds. It can be seen that the concept of gene pool contributes to the fitness gain for the second phase due to reducing the randomness of IO and that the effect of the gene pool on both test beds is prominent.

7.4.6 Experiments on Comparing Different Algorithms

The experimental results of comparing different algorithms on the test bed using main and spare pools are given in Table 7.2 and Figures 7.8, 7.9, and 7.10. The experimental results of comparing different algorithms on the test bed with changing positions of nodes are given in Table 7.3 and Figures 7.11, 7.12, and 7.13. In these tables and figures, the number of nodes in the TSP instances varies from 144 to 724. The overall best average performance before the change is given for each combination of f and m.

From the Tables 7.2 and 7.3, it can be observed that the execution time of the DGTPHA algorithm is very high as compared to the other three algorithms. Actually, in the DGTPHA algorithm, the control is not given to the second phase, so this is only the first phase (NNSBMA) of our hybrid algorithm. However, for the other algorithms, along with the base criteria after 50% of the generations the control is given to the second phase. Furthermore, among the DGTPHA+RI and the DGTPHA+NNRI, the execution time of DGTPHA+NNRI is slightly higher. This is due to the generation of random individuals by the nearest neighbor algorithm.

From the experimental results, several observations can be made by comparing the behaviour of the algorithms. The proposed DGTPHA+NNRI outperforms other algorithms on all test cases regarding small and drastic changes in terms of the change frequency f and change severity m. In small changes, the performance of both DGTPHA and DGTPHA+RI is almost the same, but with RI the performance is increased. However, when the change happens, the performance of Phase-1 (SBMA with nearest neighbour sequences generation) degrades while the approach with RI is consistent. But when the change is gradual where f is 100 the performance of DGTPHA+RI is better as compared to when f is 50. Which shows that the Phase-2 has more time to take the fitness into the optimal range. One of the features of Phase-1 in DGTPHAs is that it converges very rapidly but is unable to show any further performance. Simply, Phase-1 gets stuck in the local optima and cannot

escape further but when the MA is shifted to the Phase-2 it escapes the algorithm from local optima and explores the search space because the IO by default is an inversion operator.

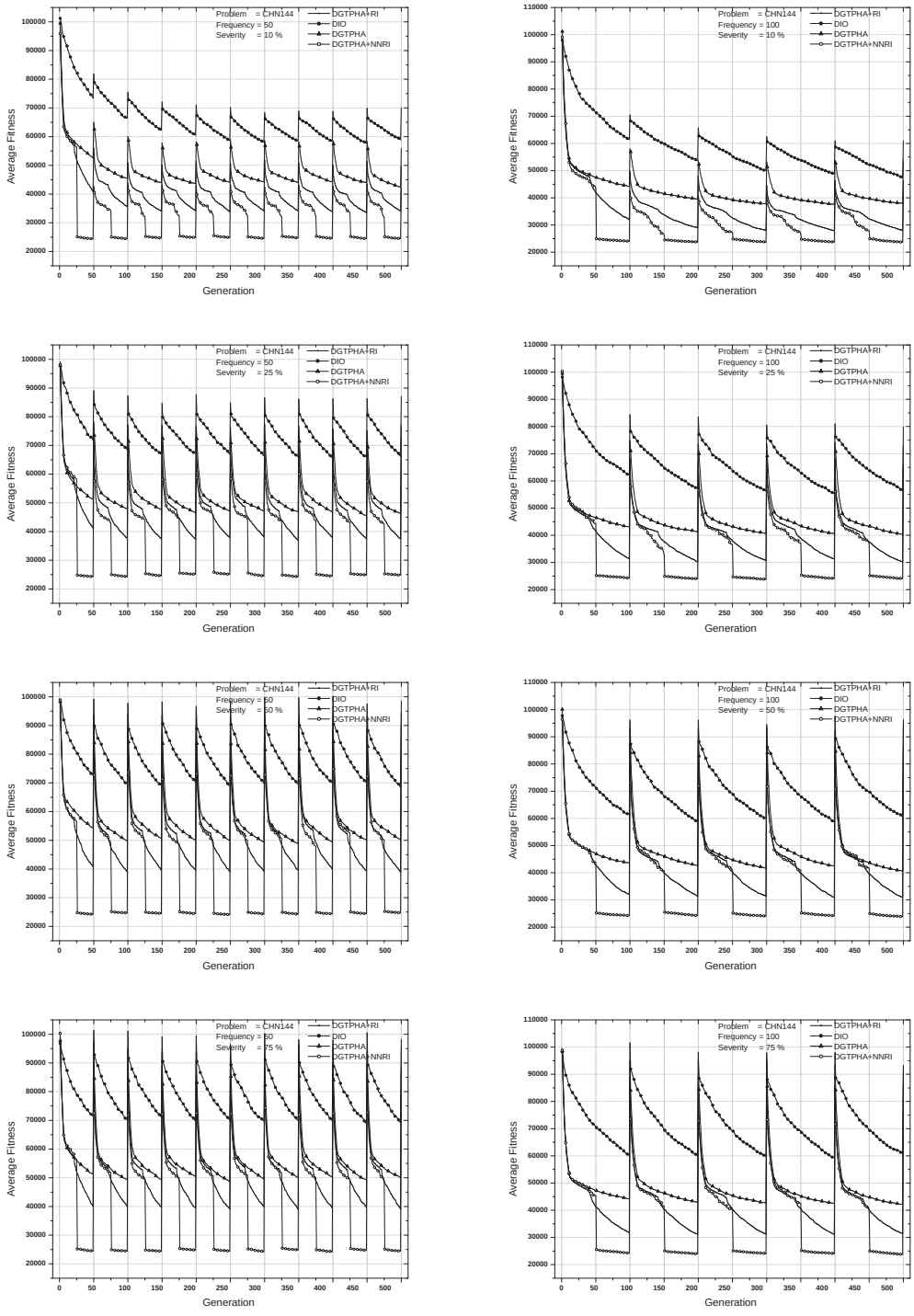


FIGURE 7.8: Experimental results of DGTPHA+RI (DGTPHA with random immigrants), DIO (Dynamic IO operator), DGTPHA, and DGTPHA+NNRI (DGTPHA with nearest neighbor based random immigrants) on the test bed using main and spare pools (MS), where f was set to 50 and 100, and m was set to 10%, 25%, 50%, and 75% for the TSP instance *chn*144 with 144 nodes.

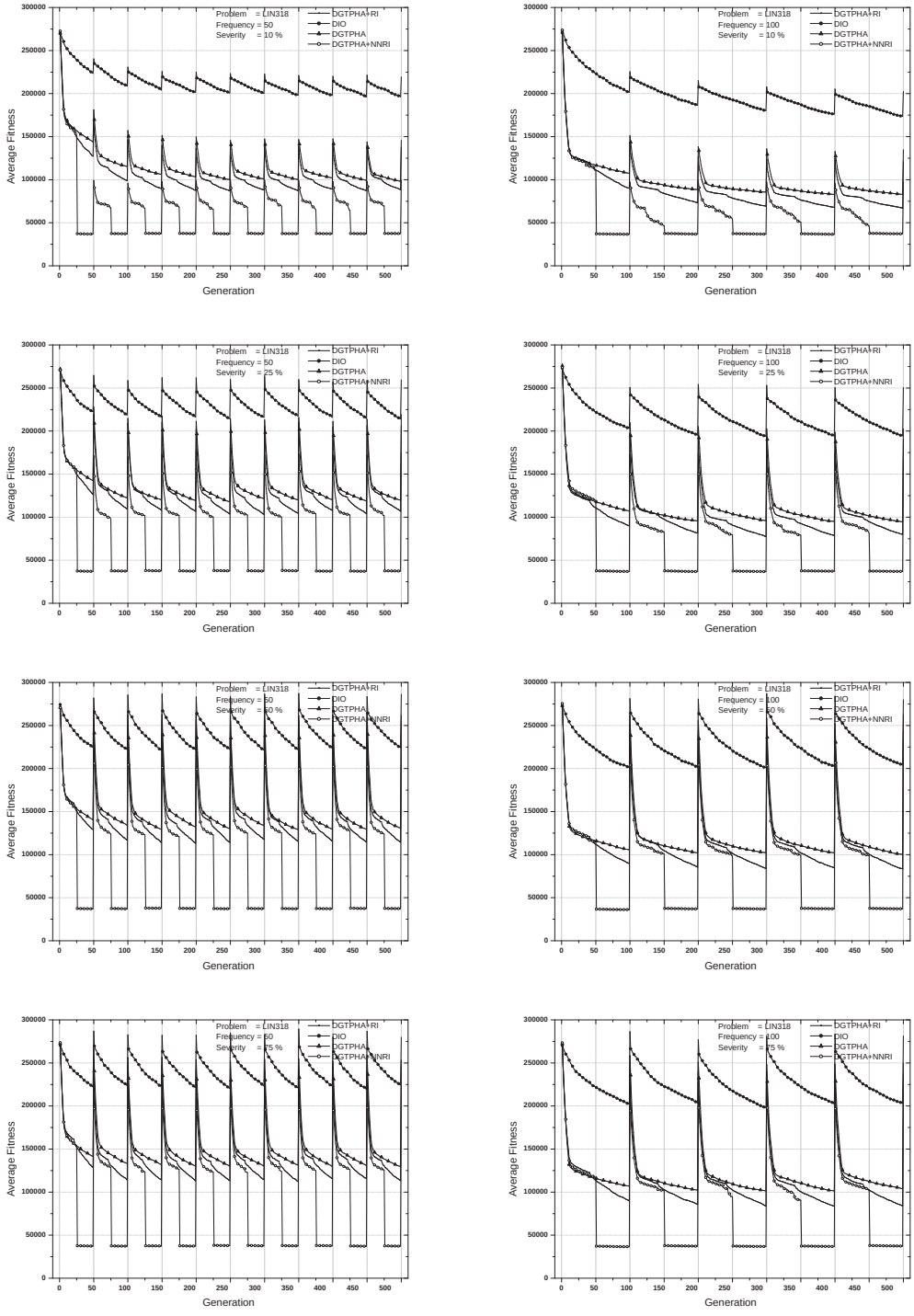


FIGURE 7.9: Experimental results of DGTPHA+RI (DGTPHA with random immigrants), DIO (Dynamic IO operator), DGTPHA, and DGTPHA+NNRI (DGTPHA with nearest neighbor based random immigrants) on the test bed using main and spare pools (MS), where f was set to 50 and 100, and m was set to 10%, 25%, 50%, and 75% for the TSP instance $lin318$ with 318 nodes.

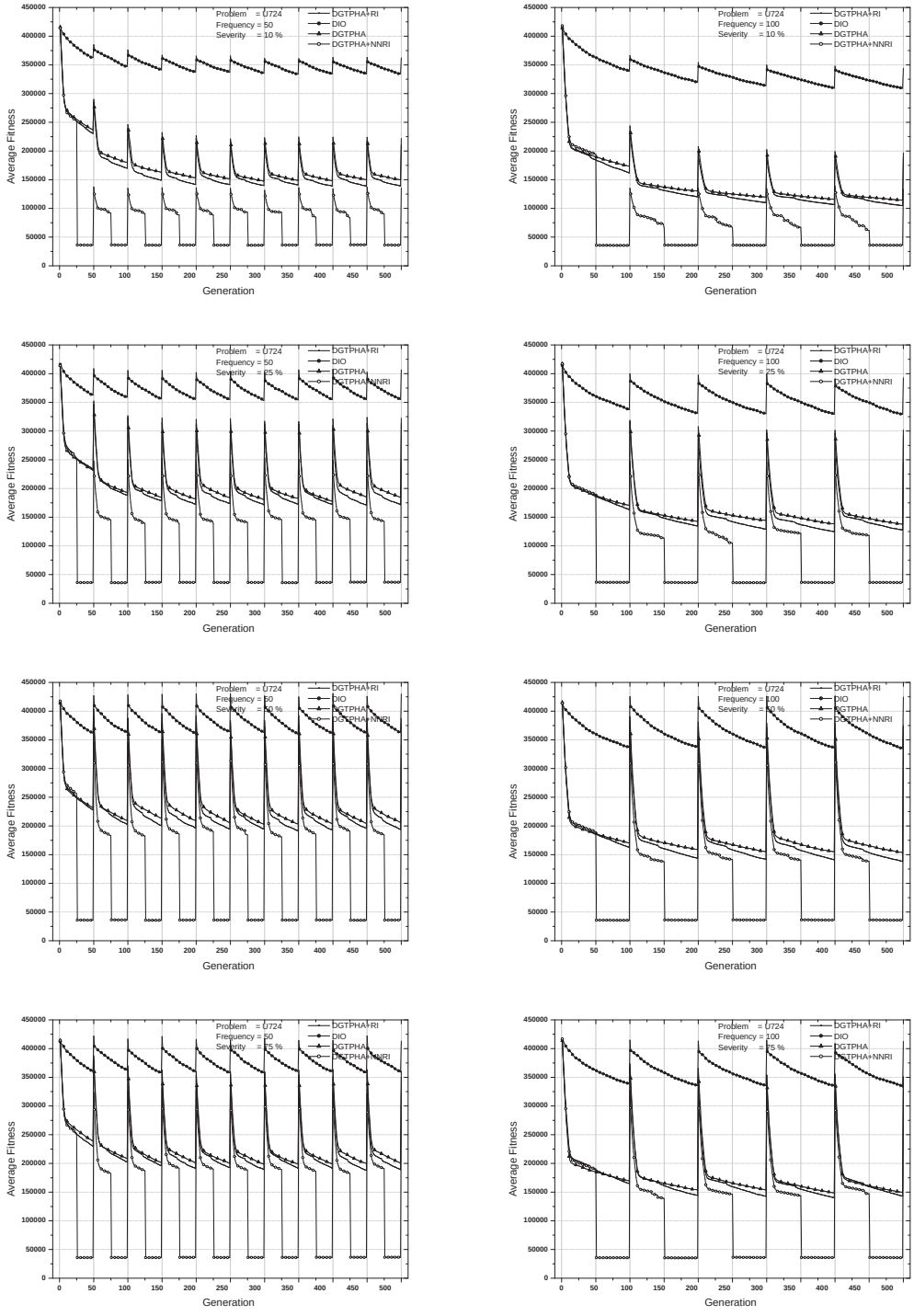


FIGURE 7.10: Experimental results of DGTPHA+RI (DGTPHA with random immigrants), DIO (Dynamic IO operator), DGTPHA, and DGTPHA+NNRI (DGTPHA with nearest neighbor based random immigrants) on the test bed using main and spare pools (MS), where f was set to 50 and 100, and m was set to 10%, 25%, 50%, and 75% for the TSP instance $u724$ with 724 nodes.

TABLE 7.2: Experimental results on the test bed with the main and spare pools, regarding the best average performance of DGTPHA (with RI and NNRI) and other MAs with the traditional Inver Over.

Test Bed Based on Main Pool and Spare Pool																				
Algorithm	**chn144**				**kroB200**				**lin318**				**pcb442**				**u724**			
***f*=50, *m* ⇒**	**10%**	**25%**	**50%**	**75%**	**10%**	**25%**	**50%**	**75%**	**10%**	**25%**	**50%**	**75%**	**10%**	**25%**	**50%**	**75%**	**10%**	**25%**	**50%**	**75%**
DGTPHA+RI	34694	37746	39198	39289	44944	50646	54715	56227	92784	107610	116000	114921	128248	150049	165720	165749	152714	180089	198480	195487
Time in *seconds*	0.37	0.38	0.4	0.4	0.41	0.44	0.43	0.45	0.51	0.53	0.55	0.53	0.66	0.66	0.66	0.68	1.09	1.04	1.05	1.04
DIO	61130	66932	69662	69694	95858	105475	110969	112235	202320	216175	222200	221610	284367	299952	307094	307660	339378	355417	361508	358470
Time in *seconds*	0.17	0.18	0.18	0.18	0.19	0.22	0.22	0.24	0.27	0.26	0.26	0.27	0.33	0.32	0.35	0.38	0.52	0.51	0.52	0.5
DGTPHA	44797	47235	50159	49692	55018	62380	67156	68171	106581	121926	131481	131523	140753	164998	180698	182154	162432	187931	208391	204072
Time in *seconds*	0.65	0.62	0.66	0.66	0.67	0.61	0.62	0.6	0.73	0.71	0.71	0.7	0.97	0.87	0.89	0.87	1.39	1.41	1.43	1.29
DGTPHA+NNRI	24564	24608	24361	24459	24716	24557	24791	24693	37119	37265	37142	37302	43334	43168	43080	43080	35957	36043	35825	35963
Time in *seconds*	0.41	0.43	0.45	0.43	0.43	0.55	0.48	0.54	0.63	0.56	0.56	0.66	0.68	0.69	0.71	0.72	1.16	1.19	1.12	1.14
***f*=100, *m* ⇒**	**10%**	**25%**	**50%**	**75%**	**10%**	**25%**	**50%**	**75%**	**10%**	**25%**	**50%**	**75%**	**10%**	**25%**	**50%**	**75%**	**10%**	**25%**	**50%**	**75%**
DGTPHA+RI	28934	30676	31268	31254	36578	39170	41263	41941	73254	81217	85225	85087	101146	113138	121141	122780	120473	135555	145207	146567
Time in *seconds*	0.37	0.38	0.38	0.4	0.41	0.41	0.43	0.42	0.47	0.49	0.51	0.5	0.59	0.59	0.59	0.6	0.87	0.88	0.89	0.88
DIO	52120	57394	59708	59988	84696	92269	97382	99053	183343	196015	201897	201641	262552	275157	283551	283618	317957	330970	335936	335021
Time in *seconds*	0.17	0.21	0.17	0.2	0.19	0.24	0.2	0.23	0.22	0.24	0.24	0.28	0.28	0.28	0.33	0.33	0.38	0.4	0.39	0.45
DGTPHA	39369	41157	42171	42818	47509	51282	53973	53973	89342	97509	102287	103045	115775	129682	138501	139892	130580	146554	158330	154755
Time in *seconds*	0.65	0.57	0.56	0.61	0.59	0.6	0.62	0.62	0.68	0.77	0.69	0.76	0.83	0.93	0.87	0.89	1.25	1.25	1.23	1.33
DGTPHA+NNRI	23760	24029	24095	23997	24316	24270	24320	24364	36688	36839	36724	36824	42997	42663	42786	42520	35619	35896	35778	35606
Time in *seconds*	0.37	0.4	0.41	0.48	0.4	0.43	0.45	0.51	0.51	0.51	0.52	0.61	0.6	0.64	0.64	0.67	0.88	0.94	0.95	0.96

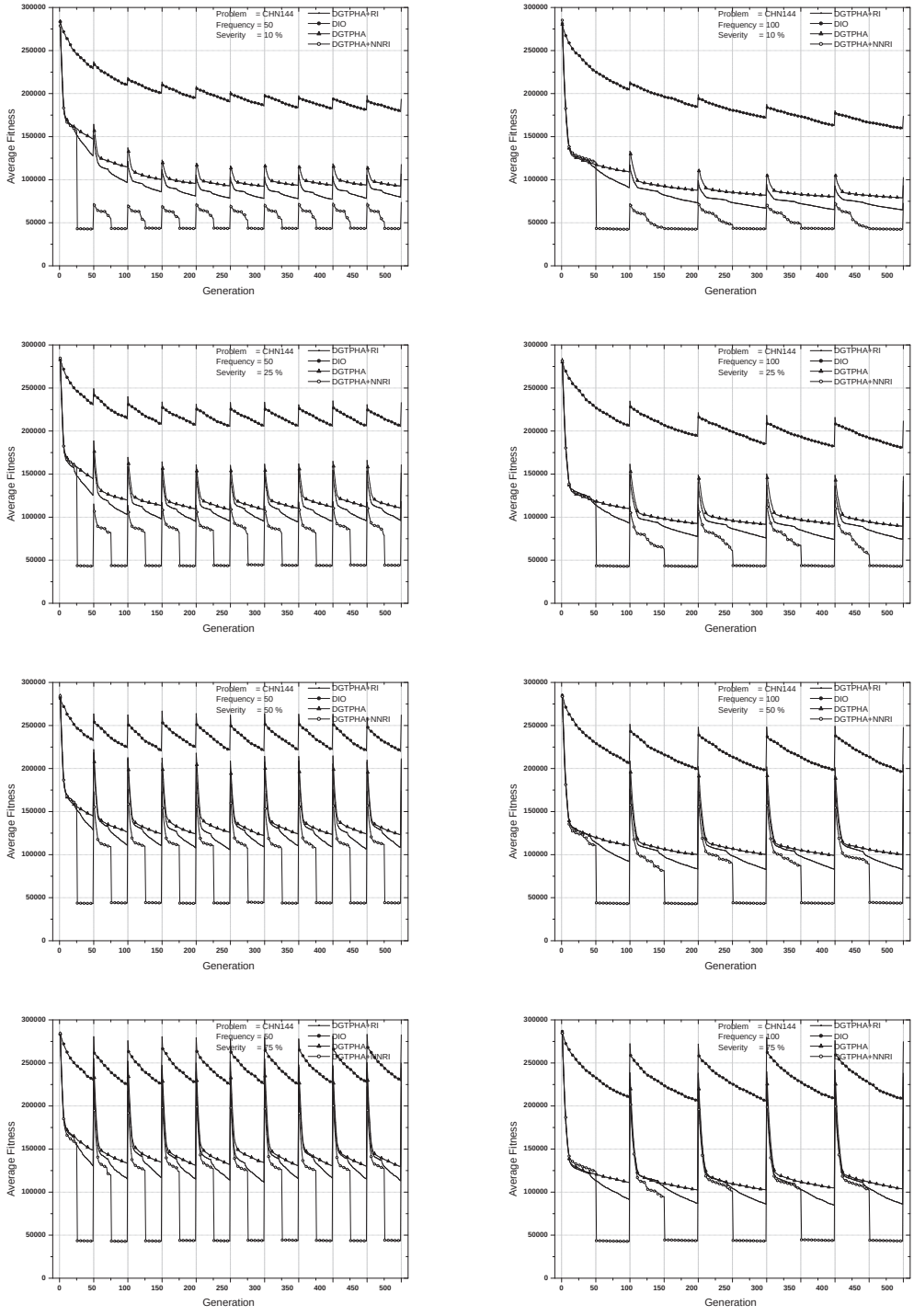


FIGURE 7.11: Experimental results of DGTPHA+RI (DGTPHA with random immigrants), DIO (Dynamic IO operator), DGTPHA, and DGTPHA+NNRI (DGTPHA with nearest neighbor based random immigrants) on the test bed based on changing the position of nodes (PN), where f was set to 50 and 100, and m was set to 10%, 25%, 50%, and 75% for the TSP instance *chn*144 with 144 nodes.

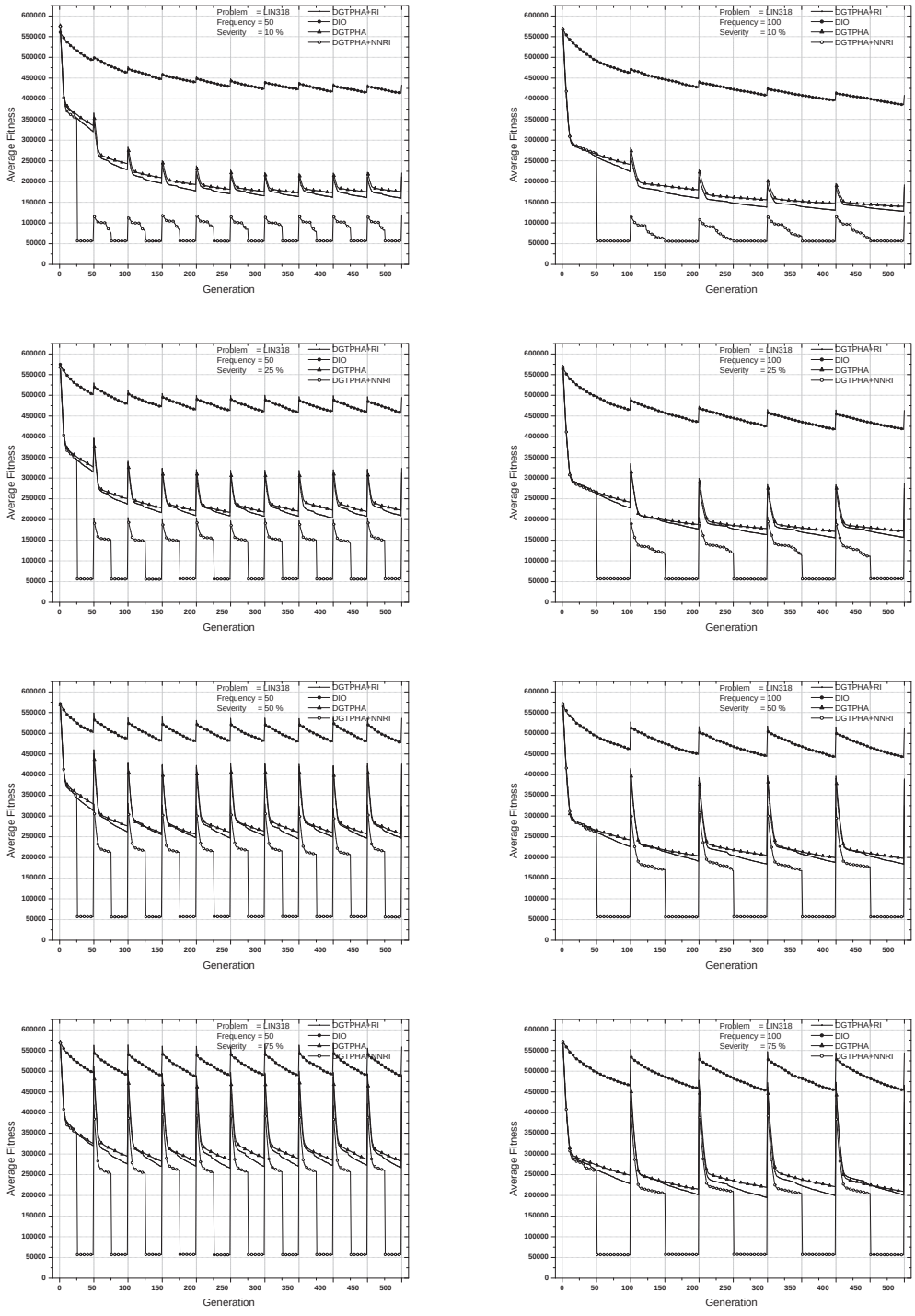


FIGURE 7.12: Experimental results of DGTPHA+RI (DGTPHA with random immigrants), DIO (Dynamic IO operator), DGTPHA, and DGTPHA+NNRI (DGTPHA with nearest neighbor based random immigrants) on the test bed based on changing the position of nodes (PN), where f was set to 50 and 100, and m was set to 10%, 25%, 50%, and 75% for the TSP instance $lin318$ with 318 nodes.

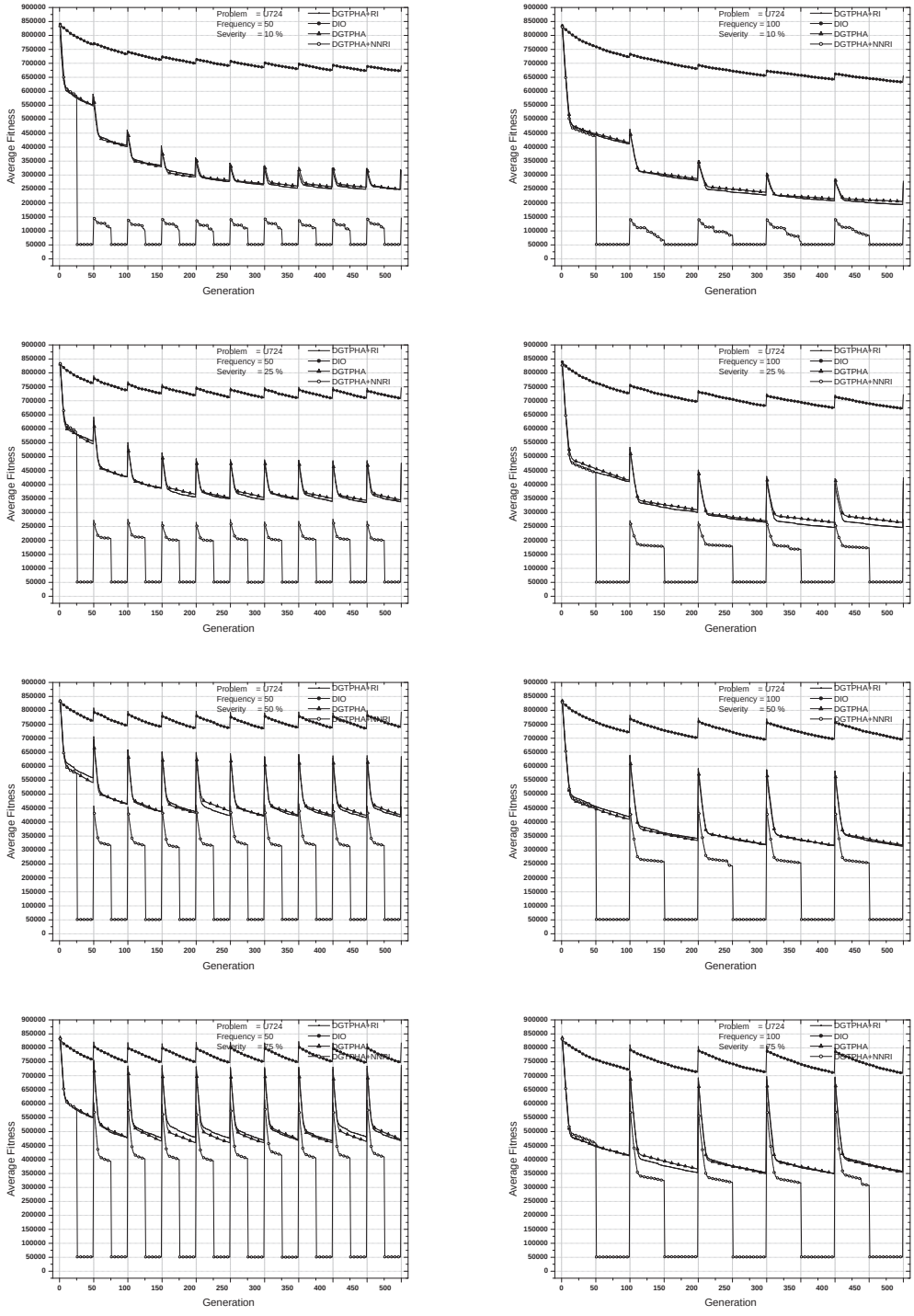


FIGURE 7.13: Experimental results of DGTPHA+RI (DGTPHA with random immigrants), DIO (Dynamic IO operator), DGTPHA, and DGTPHA+NNRI (DGTPHA with nearest neighbor based random immigrants) on the test bed based on changing the position of nodes (PN), where f was set to 50 and 100, and m was set to 10%, 25%, 50%, and 75% for the TSP instance $u724$ with 724 nodes.

TABLE 7.3: Experimental results on the test bed based on changing the positions of nodes, regarding the best average performance of DGTPHA (with RI and NNRI) and other MAs with the traditional Inver Over.

Test Bed Based on Changing Node Positions																				
Algorithm	**chn144**				**kroB200**				**lin318**				**pcb442**				**u724**			
***f*=50,*m* ⇒**	**10%**	**25%**	**50%**	**75%**	**10%**	**25%**	**50%**	**75%**	**10%**	**25%**	**50%**	**75%**	**10%**	**25%**	**50%**	**75%**	**10%**	**25%**	**50%**	**75%**
DGTPHA+RI	86014	99105	110242	115922	88433	103636	116723	124454	190165	221818	255952	274629	257138	307027	355886	386014	313120	378072	441451	480795
Time in *seconds*	0.49	0.50	0.51	0.52	0.60	0.62	0.64	0.64	0.89	0.92	0.89	0.93	1.29	1.29	1.32	1.30	2.58	2.57	2.59	2.60
DIO	193337	208982	222293	226142	210344	225432	234976	241001	435667	467172	481868	489329	607164	639595	656056	663875	697892	719505	739569	748222
Time in *seconds*	0.24	0.24	0.25	0.25	0.30	0.31	0.31	0.31	0.42	0.43	0.43	0.45	0.62	0.62	0.62	0.65	1.23	1.21	1.24	1.22
DGTPHA	101492	114835	125946	133245	98866.2	114685	126735	136941	203108	234246	267515	289754	271206	368346	394744	227148	314085	381407	442334	473425
Time in *seconds*	0.65	0.74	0.78	0.7	0.92	0.92	0.81	0.92	1.17	1.15	1.18	1.16	1.76	1.75	1.71	1.59	3.49	3.5	3.5	3.43
DGTPHA+NNRI	43068	43553	43595	43304	35844	35456	35533	35655	56179	55990	56425	56438	63598	63372	63759	63241	51141	50701	50818	50836
Time in *seconds*	0.52	0.52	0.52	0.54	0.63	0.63	0.66	0.67	0.92	0.96	0.97	0.98	1.35	1.39	1.39	1.41	2.73	2.78	2.81	2.81
***f*=100, *m* ⇒**	**10%**	**25%**	**50%**	**75%**	**10%**	**25%**	**50%**	**75%**	**10%**	**25%**	**50%**	**75%**	**10%**	**25%**	**50%**	**75%**	**10%**	**25%**	**50%**	**75%**
DGTPHA+RI	72090	78735	84493	86812	73016	81126	87771	91802	156028	175769	194399	204449	213990	241693	269626	287112	264389	293537	341485	365360
Time in *seconds*	0.46	0.47	0.49	0.48	0.54	0.55	0.55	0.55	0.79	0.78	0.79	0.79	1.12	1.1	1.11	1.12	2.08	2.09	2.14	2.14
DIO	176210	189276	198934	207256	196716	206031	215057	218878	415437	431385	447355	455927	579568	605101	619439	629698	665546	689910	700887	712655
Time in *seconds*	0.22	0.22	0.23	0.28	0.26	0.27	0.27	0.32	0.35	0.36	0.35	0.36	0.47	0.47	0.48	0.48	0.83	0.83	0.84	0.92
DGTPHA	87488	95019	101857	104837	85236	92332	100879	105981	172794	189964	209535	222500	227148	257010	283062	300922	272049	304656	338928	365957
Time in *seconds*	0.76	0.71	0.82	0.81	0.88	0.79	0.78	0.79	1.11	1.11	1.06	1.04	1.59	1.61	1.6	1.57	3.07	3.08	3.06	3.09
DGTPHA+NNRI	42526	42847	43111	43185	34847	35367	35616	35401	55776	56029	56065	56385	62973	63012	62990	63353	51052	50588	50767	50956
Time in *seconds*	0.44	0.47	0.49	0.51	0.5	0.56	0.61	0.68	0.73	0.81	0.84	0.93	1	1.19	1.2	1.27	2.02	2.17	2.26	2.25

First, DGTPHA with RI and NNRI has a more rapid and robust convergence capability than the other two approaches, which are the DGTPHA (this approach considers only base criteria) without RI and DIO. On all dynamic scenarios, *i.e.,* for gradual and drastic change, the proposed approach rapidly reached the near optimal value. This is because of the enforcement learning mechanism in the two phases of our guided MA. In Phase-1, distant nodes (cf. expensive edges, shown in Chapter 6 in Figure 6.1) are brought together and optimal sub-tours instead of non-optimal sub-tours are inserted. In Phase-2, random immigrants bring diversity along with preserving the parent and child population (containing better edges), and there is also the guidance provided by the gene pool which is generated with MST and elite population (*i.e.,* individuals with better fitness).

Second, the joint behaviour of two algorithms in the proposed approach yields better exploitation and exploration ability than the stand alone approaches.

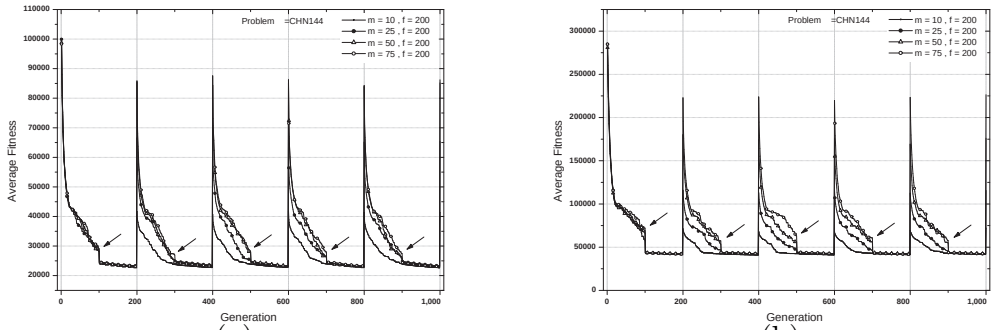


FIGURE 7.14: (a) Shows the learning effect of MS, which can accommodate the change when severity is 10, 25, 50, and 75. (b) Shows the learning effect of PN, which can accommodate the change when severity is 10, 25, 50 and 75.

The algorithm slightly shows a kind of learning behaviour as well, as can be observed from the figures with $f = 100$ and $m = 10$, *i.e.*, small changes. The algorithm collects good edges, so that as the algorithm proceeds and when the change happens, the Phase-1 almost copes with the effect of NNRI which is quite prominent in the first change (which is indicated by the arrow sign in Figure 7.14). The Phase-1 almost collects the nodes which ultimately balance the effect of NNRI. This behaviour shows

that the algorithm obtains a pool of better genes, which are, on the other hand, parts of the building block. On the other hand, this effect is not visible when the change is drastic.

In order to further elaborate the learning effect of the Phase-1, we have performed some analysis to show the learning behaviour, by changing the frequency f to 200 and the severity to 10, 25, 50, and 75, respectively. The experimental results are shown in Figure 7.14. From Figure 7.14, it can be seen that for the first static environment (i.e., from generation 0 to generation 200) for all m, the performance is the same before Phase 2 starts. The start of Phase 2 is shown by an arrow pointing to the adding of random individuals generated through the nearest neighbours algorithm. When the algorithm proceeds to the next change environment, that is, from 200 to 400, and so on, the gradual change for which m is 10 or 25 can accommodate the change by using the knowledge from the previous change, nearly cancelling the effect of NNRI which was absolutely visible in the first change. The performance increases further when m is 25. However, for $m = 75$, when the change is drastic and the problem changes almost completely, the first phase could hardly manage to accommodate the change. For the test bed with changing positions of nodes in Figure 7.14(b), the gradual change with $m = 25$ or $m = 50$ nearly cancels the effect of NNRI, which supports the concept of bringing the nodes near to each other in the Phase-1.

The t-test[1] results of statistically comparing investigated algorithms with 58 degrees of freedom at a 0.05 level of significance are shown in Table 7.4 and Table 7.5. In Table 7.4 and Table 7.5, the t-test result is shown as "$s+$", "$s-$", "$+$", "$-$", or "$\sim$" when the first algorithm is significantly better than, significantly worse than, insignificantly better than, insignificantly worse than, or statistically equivalent to the second algorithm, respectively. From Table 7.4 and Table 7.5, it can be seen that the performance of DGTPHA+NNRI is significantly better than the performance of

[1]For the t-test, we have used the GraphPad Prism software. Online available: (http://www.graphpad.com/prism/prism.htm)

the other three algorithms on all problem instances with different change frequencies f and change severities m. It can also be observed that the performance of DGTPHA is significantly better than the performance of DIO on all problem instances. This result indicates that a single heuristic is not enough for solving a DTSP. These results show that the integration of domain-specific knowledge and suitable LS and guided search techniques can greatly improve the performance of MAs for the DTSP.

TABLE 7.4: The t–test results of comparing algorithms of MS on problem instances with different values of f and m.

t-Test Results of Main and Spare Pool Experiments																				
Algorithm	**chn144**				**kroB200**				**lin318**				**pcb442**				**u724**			
f=50,m ⇒	10%	25%	50%	75%	10%	25%	50%	75%	10%	25%	50%	75%	10%	25%	50%	75%	10%	25%	50%	75%
DGTPHA+RI ⇔ DIO	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
DGTPHA+RI ⇔ DGTPHA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
DGTPHA+NNRI ⇔ DGTPHA+RI	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
DGTPHA+NNRI ⇔ DGTPHA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
DIO ⇔ DGTPHA+NNRI	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
DGTPHA⇔ DIO	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
f=100,m ⇒	10%	25%	50%	75%	10%	25%	50%	75%	10%	25%	50%	75%	10%	25%	50%	75%	10%	25%	50%	75%
DGTPHA+RI ⇔ DIO	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
DGTPHA+RI ⇔ DGTPHA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$+$
DGTPHA+NNRI ⇔ DGTPHA+RI	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
DGTPHA+NNRI ⇔ DGTPHA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
DIO ⇔ DGTPHA+NNRI	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
DGTPHA ⇔ DIO	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$

TABLE 7.5: The t–test results of comparing algorithms of PN on problem instances with different values of f and m.

t-Test Results of Position Based Experiments																				
Algorithm	**chn144**				**kroB200**				**lin318**				**pcb442**				**u724**			
f=50,m ⇒	10%	25%	50%	75%	10%	25%	50%	75%	10%	25%	50%	75%	10%	25%	50%	75%	10%	25%	50%	75%
DGTPHA+RI ⇔ DIO	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
DGTPHA+RI ⇔ DGTPHA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$\sim$	$\sim$	$\sim$	$+$
DGTPHA+NNRI ⇔ DGTPHA+RI	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
DGTPHA+NNRI ⇔ DGTPHA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
DIO ⇔ DGTPHA+NNRI	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
DGTPHA ⇔ DIO	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
f=100,m ⇒	10%	25%	50%	75%	10%	25%	50%	75%	10%	25%	50%	75%	10%	25%	50%	75%	10%	25%	50%	75%
DGTPHA+RI ⇔ DIO	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
DGTPHA+RI ⇔ DGTPHA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$+$	$s+$	$s+$	$s+$	$+$	$+$	$\sim$	$s+$	$s+$	$\sim$
DGTPHA+NNRI ⇔ DGTPHA+RI	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
DGTPHA+NNRI ⇔ DGTPHA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
DIO ⇔ DGTPHA+NNRI	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
DGTPHA+ ⇔ DIO	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$

From the above analysis, it can be observed that the winner is *DGTPHA+NNRI* and the runner up is *DGTPHA+RI*. From Figures 7.8 – 7.13, and Tables 7.2 and 7.3, it is quite clear that the suggested approach is far better than the traditional IO operator. It is obvious from Tables 7.2 and 7.3, the computational time of IO is much less than that of our proposed approaches. From the tables, we can get the observation that the Phase-1 algorithm is computationally expensive among all the approaches. However, from the plots, we can also observe that after some generations of running, the performance of Phase-1 is degraded and is unable to show any further progress towards the fitness gain. So, we have employed the Phase-2 which is based on the IO operator. From the results of the tables, the time and quality of our proposed DGTPHA+NNRI are well balanced together, which is quite clear from the execution time (measured in seconds) of each algorithm. It is a matter of fact that we have integrated several heuristic approaches to the existing framework, which should result in increase in the execution time. However, instead of increasing, the execution time is reduced a lot.

7.5 Chapter Summary

Different approaches have been applied to address dynamic travelling salesman problems (DTSPs). In this chapter, we have extended the framework of our two-phase hybrid memetic algorithm for the static travelling salesman problem (TSP) to solve the DTSP. Our approach is based on two phases: the first phase is based on traditional binary crossover and mutation operators, and the second phase is based on unary Inver Over (IO) operator. The first phase supports the second phase and makes the approach suitable not only for the static TSP but also for the DTSP. Our approach makes use of the knowledge of the current population for future generations. Simply, the next generation is dependent on the previous generation up to some extent or totally when the change is slight not drastic.

In the proposed approach, two techniques are used in the first phase, and three techniques are used in the second phase. In the first phase, the Nearest Neighbour technique is utilized to bring the nodes near to each other, and then the real essence of the evolutionary process is considered, which means that the algorithm can proceed in an open way to stabilize itself. The second approach applies local optimization on a small set of nodes instead of all the nodes in an individual in order to reduce the computational time.

In the second phase, a kind of domain-specific knowledge is used in the shape of gene pool generated by the minimum spanning tree. Another technique is the enforcement of the Inver Over operator to get the clue from the best individuals of the population, as it is quite natural that top individuals could contain good edges instead of bad ones. Finally, the traditional IO operator works in a fashion that when it gets an individual, it performs various inversions, which are not known unless the termination criteria of the local operator apply. We have changed the set-up of the traditional Inver Over by making it greedier, and it catches those inversions which go in favour of fitness gain. Although it puts a kind of extra load on the IO, but it results in better individuals at the end.

In order to test the performance of DGTPHA for the DTSP, experiments were performed to analyse the sensitivity of parameters and the effect of a few heuristic strategies for the performance of DGTPHA+NNRI based on a number of DTSP instances. The experimental results of DGTPHA+NNRI were also compared with several algorithms on the test DTSP instances. The experimental results also confirmed that the proposed approach is efficient for solving the DTSP. In general, with the hybridization of various operators, local search, and heuristic strategies, DGTPHA+NNRI is able to find good results for the tested DTSP instances.

The next chapter will conclude the work carried out in this thesis. Also, some future research directions are considered.

Chapter 8

Conclusions and Future Work

In this chapter, we conclude the thesis with a summary of the main technical contributions and major conclusions that can be made from the experimental studies carried out in the thesis. We will also discuss further research directions in the end, which should be undertaken in the future.

8.1 Technical Contributions

As mentioned in the beginning chapters of this thesis, there is no ending boundary for tackling the travelling salesman problem (TSP), which is an NP-hard problem. This problem has attracted significant research interests for the past several decades. Many researchers have had tremendous milestones, e.g., solving the TSP by keeping two issues in mind, which are the search space size (the exponential growth of the size of the search space with the problem size) and the computational time. There are so many solving techniques, which have been discussed in this thesis and in the literature. However, it is still an open problem for research and the wheel of investigation is moving faster.

Nowadays. it is a well-established fact that local search (LS) is useful within the framework of genetic algorithms (GAs). We have addressed the static TSP and dynamic TSP (DTSP) with the tool of GAs. In order to enhance the optimisation performance, a simple LS scheme based on sequences (sub-tours) is introduced into the GA to solve the TSP. The basic motivation of the LS scheme used in this thesis is quite simple and straightforward. In the proposed approaches, the major result emerging from the experiments is that LS can pass good traits on from one generation to another with the progress of the GA from the start to the end.

The proposed approach is composed of two phases. Each phase alone does not show good performance at the end. However, when the two phases are merged together as a hybrid algorithm, both approaches shoulder each other and the end product regarding the solution quality and time is optimal or near optimal. The algorithm in a sense allows the local and global components to influence each other at a finer granularity. The hypothesis here is that our LS scheme can find the building blocks that are very difficult for the recombination operator to obtain.

From the literature, we can conclude that in addition to crossover, mutation is also a big issue. According to Land in his dissertation [81], when Lamarckian evolution is considered, small mutation does not help a lot but large mutation helps a lot. In our approach, we slightly consider this issue. For example, when the algorithm gets stuck or the existing set of sequences or memory is unable to bring further changes, a kind of mutation is performed by reducing the size of sequences from a large set of nodes to an edge at the extreme in a sequence. This hypothesis worked to a great extent in the first phase of our approach.

For the second phase of our approaches, the fast Inver Over (IO) operator is employed. However, the original IO operator is blind in nature since it does not consider whether the clue obtained from the population is good or bad. Hence, we have introduced some techniques to reduce the blindness of IO. The first one, the parent and child populations of the last generation of the first phase are a kind of gene

pool of good edges. These valuable populations are kept and provided to the IO to further work on. Some random immigrants are also inserted, which enable the IO operator to explore the search space further. Additionally, the current approach is validated and examined by inserting immigrants constructed based on the nearest neighbour algorithm. Furthermore, IO is forced to get the clue from the elite population because good individuals in the elite population participate in adding and preserving good edges. In fact, this property is the key soul of solving the TSP in the best way. Along with this, a rotating technique is also introduced with the gene pool for solving the DTSP.

The major contributions achieved in this study in the domain of applying sub-tour approaches to the TSP and DTSP are summarised as follows:

1. **A Sequence Based Memetic Algorithm (SBMA) for the TSP**: In the proposed SBMA, a set of good sequences (sub-tours) are stored in the memory and further utilized to guide the algorithm. In our SBMA, a new local search scheme is introduced, which inserts a sequence, randomly selected from the memory, into the location of an individual, which results in the minimum increase in the total length of the individual.

2. **A Novel Technique of Hybridising Traditional Binary and Unary Operators**: A new memetic algorithm which is composed of binary crossover and inversion mutation are merged together with a unary operator which exhibits the characteristics of crossover and mutation. In addition, an embedded LS scheme is introduced. Here, the concept of embedded LS means that LS contributes in converting the partial individual to a complete individual by inserting a sub-tour in an optimal location. One of the traditional drawbacks that a GA suffers from is getting stuck in local optima. This is resolved by employing the IO operator. To keep the balance between *exploration* and *exploitation* and maintain the diversity, we also apply some techniques to adapt

the key parameters based on whether the best individual of the population improves or not. The proposed MA is capable of finding near optimal solutions for the test TSP instances.

3. **Embedding Several Heuristics with SBMA and IO**: The Nearest Neighbour approach is considered one of the best heuristics for constructing individuals. We utilized the NN-List of a node by bringing the nodes nearer to each other and prevent the MA to insert node, which is not in the neighbourhood of a node. Additionally, all the sequences before and after reducing the size were locally optimized by a traditional 2-opt improver. In addition, the second phase is strengthened by introducing the concept of an *elite* population. In this approach, IO is further guided by avoiding getting a random clue. The restricted inversion also plays a vital role. In this scheme, fruitful inversions are welcomed while those inversions that do not contribute in fitness gain are discarded. Moreover, preserving the characteristics of parent and child populations is also considered to be inherited in other individuals. Random immigrants are also beneficial to find good feasible solutions. It is worthwhile that all these considerations to our two phase, on the one hand, dramatically decrease the computational time but on the other hand, improved the overall performance by giving good-quality solutions.

4. **A Dynamic Guided Two Phase Hybrid Algorithm (DGTPHA) for the DTSP**: In this study, we have also tackled the DTSP as many real-world TSPs are time dependent and can be modelled as a DTSP. For solving the DTSP, the framework of the Guided Two Phase Hybrid Algorithm (GTPHA) for solving the static TSP is extended into the DGTPHA by combining the change operator. The DTSP requires the DGTPHA to quickly respond to the changes. For the DTSP, the time is very limited between the two changes. So, fast convergence and maintaining the diversity are the main concerns. The convergence speed in the second phase is further strengthened by introducing a rotating gene pool constructed based on the minimum spanning tree.

8.2 Conclusions

In order to justify the algorithms we developed in this thesis, we have carried out several sets of experiments systematically and analysed the experimental results regarding the performance of algorithms in comparison with other state-of-the-art approaches for the TSP taken from the literature. Here, we summarise the major conclusions based on the experimental results and relevant analyses carried out in this thesis as follows.

- The new LS technique (*i.e.*, the sequenced based LS, proposed in Chapter 4) enables the SBMA to escape from local optima and hence greatly enhances the performance of the SBMA for the static TSP. The initial results show that, by integrating sequences/sub-tours within the GA framework and adaptively changing the size of sequences, GAs can get better solutions for the TSP.

- From the experimental analysis in Chapter 5, the sequence based local search with different properties (*i.e.*, generating a sequence set and inserting it back to somewhere else in the individual with better fitness gain), and additionally when integrated with the IO operator, give significantly improved results. The sequence based strategy can be used to create good quality solutions based on previous good solutions and further guide the exploration of the search space as much as possible. The LS operator can enhance the quality of an individual by finding the local optimum around the individual. So, the cooperation between the sequence based approach and the IO operator can enhance the two phase hybrid approach (TPHA) to locate optimal or near-optimal solutions.

- We also noticed that the performance of TPHA can be enhanced with other heuristics as well, e.g., the nearest neighbour list and applying the 2-opt into the first phase. Moreover, in the second phase, further guiding the IO operator with the elite population (*i.e.*, getting clues from good individuals instead of

random ones) is very efficient to solve the TSP. Experimental results in Chapter 6 show that when the sequence based strategies and the elite population based IO operator are combined together, the performance is remarkably improved. We, furthermore, noticed that sequences constructed from the nearest neighbour list and 2-opt local search operator bring the nodes nearer to each other while the random sequence generation mechanism has the ability to explore a huge search space and collect useful knowledge, which can be further utilized by the second phase, i.e., the IO operator.

- We have shown that the above TPHA can be extended to handle the DTSP. In the proposed framework of DGTPHA, the integration of the nearest neighbour list, 2-Opt and adaptive parameter control in the first phase and the elite population with the rotating gene pool strategies in the second phase works well for the DTSP. In order to test the performance of the proposed approach for the DTSP, experiments were carried out based on two different DTSP test beds: one is based on the *main pool* and *spare pool*, and the other is based on *changing the positions of nodes*. From the study, it has been observed that the integrated heuristics or meta-heuristics are able to produce good quality solutions for the DTSP. We also observed the effect of the gene pool and immigrants generated with the nearest neighbour algorithm, which works well with all DTSP instances under different change frequencies and change severities.

Generally speaking, from the experimental results, we can conclude that the sequence based strategies and LS strategies proposed in this thesis greatly improve the performance of GAs for both static and dynamic TSPs.

8.3 Future Work

As Wolpert and Macreadys [148] stated "no search algorithm is superior on all problems". Due to this "No Free Lunch" theorem, one approach or one set of

parameters cannot be suitable for all problems. At the moment, the SBMA approach tackles two objectives. For example, the first phase does fast convergence but with the drawback of getting stuck in local optima and a bit computationally expensive. In the second phase, time could be controlled as it is fast in operation. For our study, we have followed a software engineering model that is called the spiral model. First, we have developed MAs by employing the basic GA operators: the order crossover was proposed in 1985, and the mutation operators are simple inversion and double bridge move operators. The experimental results show that the proposed approaches are able to produce some best results on small benchmark TSP instances. However, there are still some general research questions that arise from this research in each step of the spiral procedure. Here, we discuss them along with some suggestions on the future work.

1. We have observed in Chapter 4 that the SBMA can work better for solving small and medium-scale TSP instances. However, for large-scale instances, the solving speed is comparatively slow. On the behaviour of SBMA, one further improvement would be to make it more exploitable by making use of information extracted from the population, e.g., by creating an optimal set of sequences/sub-tours.

2. The observation that is made when we extended/merged the SBMA with IO showed that the IO phase is strongly dependent on the performance of the first phase to provide a good knowledge for the second phase. It is worthwhile to investigate the performance of some different heuristics which may build a better knowledge base for solving the TSP. We will also consider adaptive techniques to adjust key parameters for the SBMA during its searching progress.

3. In the approach mentioned in Chapter 6, we integrated many other heuristics, e.g., the nearest neighbor list, 2-opt for optimizing the sequences, adaptive parameter control, new shifting criteria from phase one to phase two, the elite population, RI and NNRI. All these heuristics are running in predefined

sequence. We do not consider any criteria regarding when to use which one during the course of execution of the algorithm. We feel the need that the current approach should be extended to a multi-phase MA instead of two-phase MA. It would be more desirable to introduce the concept of multi-meme and then fine-tune the parameters such that the algorithm should be intelligent enough to first analyse the landscape and then take the decision to select the most suitable heuristic among the other heuristics.

4. For tuning the current SBMA with IO for tackling the DTSP, it is necessary to enhance the response of the algorithm to tackle the dynamic changes because the dynamic characteristics of the problem may be gradual or drastic or may be of other kinds in nature.

In summary, the SBMA with IO and LS is productive for enhancing the traditional GA for solving the static and dynamic TSPs. The experimental results also confirmed that the proposed approach is competitive. In general, with the use of hybridization of various operators and local search strategies, NNSBMA can find optimal and near optimal results for the TSP instances. There are several other directions to pursue in the future as well. One future work would be further to extend it to large symmetric and asymmetric TSPs. We will also investigate how to further integrate advanced heuristics, meta-heuristics, and adaptive techniques to enhance the construction of better sequences/sub-tours and their insertion in proper locations of individuals. Furthermore, we will also consider making the sequence based strategies more generalized towards other crossover operators instead of the order crossover. It would be also interesting to extend the existing framework to other combinatorial optimization problems that are similar to or other variations of the TSP, *e.g.*, the sequential ordering, capacitated vehicle routing, and many other real-world problems, etc.

Bibliography

[1] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The Traveling Salesman Problems: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton University Press, Princeton, NJ, USA, 2007.

[2] D. L. Applegate, R. Bixby, V. Chvátal, and W. Cook. Finding Cuts in the TSP (A preliminary report), Technical Report 95-05, DIMACS, 1995.

[3] D. L. Applegate, R. Bixby, V. Chvátal, and W. Cook. On the Solution of Traveling Salesman Problems. *Documenta Mathematica*, Extra Volume ICM(3):645–656, 1998.

[4] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. Implementing the Dantzig-Fulkerson-Johnson algorithm for large traveling salesman problems, *Mathematical Programming* 97(1-2):91–153, 2003.

[5] R. Armañanzas, I. Inza, R. Santana, Y. Saeys, J. L. Flores, J. A. Lozano, Y. V. de Peer, R. Blanco, V. Robles, C. Bielza, and P. Larrañaga. A review of estimation of distribution algorithms in bioinformatics, *BioData Mining*, 1, 2008.

[6] S. Arshad, and S. Yang. A hybrid genetic algorithm and inver over approach for the traveling salesman problem. In: *Proceedings of the 2010 IEEE Congress on Evolutionary Computation*, pages 252–259, 2010.

[7] S. Arshad, S. Yang, and C. Li. A sequence based genetic algorithm with local search for the traveling salesman problem. In: *Proceedings of the 2009 UK Workshop on Computational Intelligence*, pages 98–105, 2009.

[8] H. Asoh, and H. Mühlenbein. On the mean convergence time of evolutionary algorithms without selection and mutation. In: *Proceedings of Parallel Problem Solving From Nature (PPSN)*, pages 88–97, 1994.

[9] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo. Algorithms for the on-line traveling salesman problem. *Algorithmica*, 29(4):560–581, 2001.

[10] T. Back, D. B. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation*. IOP Publishing Ltd., Bristol, UK, 1st edition, 1997.

[11] T. Bäck. Self-adaptation in genetic algorithms. In: *Proceedings of the 1st European Conference on Artificial Life*, pages 263–271, 1992.

[12] J. E. Baker. Adaptive selection methods for genetic algorithms. In: *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 101–111, 1985.

[13] J. E. Baker. Reducing bias and inefficiency in the selection algorithm. In: *Proceedings of the 2nd International Conference on Genetic Algorithms and Their Applications*, pages 14–21, 1987.

[14] R. Battiti, M. Brunato, and F. Mascia. Reactive Search and Intelligent Optimization, *Operations research/Computer Science Interfaces*, 45, Springer Verlag, 2008.

[15] R. Bland, and D. Shallcross. Large traveling salesman problems arising from experiments in X-ray crystallography: A preliminary report on computation. *Operations Research Letters*, 8(3):125–128, 1989.

[16] T. Blickle, and L. Thiele. A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, 4(4):361–394, 1996.

[17] J. Branke, and S. Hartmut. Designing evolutionary algorithms for dynamic optimization problems. In *Advances in Evolutionary Computing*, A. Ghosh and S. Tsutsui (Eds). Springer-Verlag New York, Inc., pages 239–262, 2003.

[18] P. A. Bosman, and D. Thierens. Linkage information processing in distribution estimation algorithms. In: *Proceedings of the Genetic and Evolutionary Computation Conference GECCO, Morgan Kaufmann Publishers, San Francisco*, pages 60–67, 1999.

[19] S. Carlson. Algorithm of the gods. *Scientific American*, 276:121–124, 1997.

[20] A. Colorni, M. Dorigo, and V. Maniezzo. Distributed Optimization by Ant Colonies. In F. J. Varela and P. Bourgine, editors, Towards a Practice of Autonomous Systems, In: *Proceedings of the First European Conference on Artificial Life*, pages 134-142. MIT Press, Cambridge, 1992.

[21] H. Crowder, and M. W. Padberg Solving Large Scale Symmetric Traveling Salesman Problems to Optimality, *Management Science*, 26:495–509, 1980.

[22] D. R. Cox, M. Burmeister, E. R. Price, S. Kim, and R. M. Myers. Radiation hybrid mapping: a somatic cell genetic method for constructing high-resolution maps of mammalian chromosomes. *Science*, 250(4978):245–50, 1990.

[23] G. A. Croes. A method for solving traveling salesman problemss. *Operations Research*, 6(6):791–812, 1958.

[24] G. B. Dantzig. Programming of interdependent activities. *Econometrica*, 17:193–211, 1949.

[25] G. B. Dantzig. Linear Programming and Extensions. Princeton Univ. Press, 1963.

[26] G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson. Solution of a Large-Scale Traveling Salesman Problem, *Operations Research*, 2(4):393–410, 1954.

[27] C. Darwin. *Origin of species*. New York, Boston H.M. Caldwell Co., 1900.

[28] R. Dawkins. *The Selfish Gene*. Oxford University Press, 1998.

[29] L. Davis. Applying adaptive algorithms to epistatic domains. In: *Proceedings of International Joint Conference on Artificial Intelligence*, pages 161–163, 1985.

[30] M. Dorigo, V. Maniezzo, and A. Colorni, Positive Feedback as a Search Strategy, Tech. Rep. 91-016, Politecnico di Milano, Milano, Italy, 1991.

[31] M. Dorigo, V. Maniezzo, and A. Colorni, The Ant System: Optimization by a Colony of Cooperating Agents, *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, 26(1):29–41, 1996.

[32] M. Dorigo, and G. D. Caro, The Ant Colony Optimization Meta-Heuristic, in New Ideas in Optimization, (D. Corne, M. Dorigo, and F. Glover, eds.), pages 11–32, McGraw Hill, 1999.

[33] M. Dorigo and T. Stützle. Ant Colony Optimization, MIT Press, Cambridge, MA, USA, 2004.

[34] W. L. Eastman. *Linear Programming With Pattern Constraints*. PhD thesis, Harvard, 1958.

[35] A. E. Eiben, Z. Michalewicz, M. Schoenauer, and J. E. Smith. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.

[36] A. E. Eiben, and J. E. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.

[37] T. A. El-Mihoub, A. A. Hopgood, L. Nolle, and A. Battersby. Hybrid genetic algorithms: A review. *Engineering Letters*, 13(2):124–137, 2006.

[38] L. J. Eshelman, and J. D. Schaffer. Preventing premature convergence in genetic algorithms by preventing incest. In: *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 115–122, 1991.

[39] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence Through Simulated Evolution*. Wiley, Chichester, WS, UK, 1966.

[40] D. B. Fogel. An evolutionary approach to the traveling salesman problem. *Biological Cybernatics*, 60(2):139–144, 1988.

[41] D. B. Fogel. Evolutionary Computation: The Fossil Record (1st ed.). Wiley-IEEE Press, 1998.

[42] R. M. French, and A. Messinger. Genes, Phenes and the Baldwin Effect: Learning and Evolution in a Simulated Population, In: *Proceedings of the 4th International Workshopon the Synthesis and Simulation of Living Systems Artificial Life IV*, (R. A. Brooks and P. Maes, eds.) MIT Press, pages 277–282, 1994.

[43] B. Gabrys, and D. Ruta. Genetic algorithms in classifier fusion, *Applied Soft Computing*, 6(4):337–347, 2006.

[44] L. M. Gambardella, and M. Dorigo. Solving symmetric and asymmetric traveling salesman problems by ant colonies. In: *Proceedings of the 1996 International Conference on Evolutionary Computation*, pages 622–627, 1996.

[45] R. S. Garfinkel. Minimizing wallpaper wast part 1 a class of traveling salesman problem. *Operations Research*, 25(5):741–751, 1977.

[46] M. R. Garey, and D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, 1979.

[47] P. C. Gilmore, and R. E. Gomory. Sequencing a One State-Variable Machine: A Solvable Case of the Traveling Salesman Problem. *Operations Research*, 12(5):655–679, 1964.

[48] F. Glover, E. Taillard, and D. de. Werra. A user's guide to tabu search. *Annals of Operations Research*, 41(1–4):3–28, 1993.

[49] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[50] D. E. Goldberg, and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In: *Proceedings of Foundations of Genetic Algorithms*, San Francisco, CA: Morgan Kaufman, pages 69–93, 1991.

[51] M. R. Garey, R. L. Graham, and D. S. Johnson. Some NP-complete geometric problems, In: *Proceedings of the 8th Annual ACM Symposium on Theory of Computing*, pages 10–22, 1976.

[52] J. J. Grefenstette. Incorporating problem specific knowledge into genetic algorithms. In: *Proceedings of Genetic algorithms and Simulated Annealing*, Morgan Kaufman Publishers, pages 42–60, 1987.

[53] J. J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16: 122–128, 1986.

[54] B. Grossman. Laser crystal techniques, 2005. On-line available at http://www.bathsheba.com/crystal/process).

[55] M. Grötschel, M. Jünger, and G. Reinelt. Optimal control of plotting and drilling machines: A case study. *Mathematical Methods of Operations Research*, 35(1):61–84, 1991.

[56] M. Grötschel. On the Symmetric Traveling Salesman Problem: Solution of a 120-city problem, *Mathematical Programming Studies*, 12:61–77, 1980.

[57] M. Grötschel, and O. Holland. Solution of Large-scale Symmetric Travelling Salesman Problems, *Mathematical Programming*, 51: 141–202, 1991.

[58] F. Gruau, and D. Whitley. Adding learning to the cellular development of neural networks: Evolution and the baldwin effect. *Evolutionary Computation*, 1(3):213–233, 1993.

[59] G. Gutin. Traveling Salesman Problems. Handbook of Graph Theory (J. Gross and J. Yellen, eds.), CRC Press, 2003.

[60] M. Guntsch, and M. Middendor. Pheromone modification strategies for ant algorithm applied to dynamic traveling salesman problem, in application of evolutionary computing. In: *Proceedings of the EvoWorkshops on Applications of Evolutionary Computing*, pages 213–220, 2001.

[61] M. Guntsch, J. Branke, M. Middendorf, and H. Schmeck. ACO strategies for dynamic traveling salesman problem. In: *Proceedings of the 2nd International Workshop on Ant Algorithms*, pages 59–62, 2000.

[62] M. Guntsch, M. Middendorf, and H. Schmeck. An ant colony optimization approach to dynamic travelling salesman problem. In: *Proceedings of the 2001 Genetic and Evolutionary Computation Conference (GECCO-2001)*, San Francisco, CA: Morgan Kaufmann Publishers, pages 860-867, 2001.

[63] W. Hart, N. Krasnogor, and J. Smith, editors. *Recent Advances in Memetic Algorithms*. Springer, Berlin, Heidelberg, New York, 2004.

[64] D. J. Hatfield, and J. Pierce. Production sequencing by combinatorial programming. Technical report, IBM Cambridge Scientific Center, Cambridge, Mass., 1966.

[65] K. Helsgaun. An effective implementation of the Lin-Kernighan travelling salesman problem. Technical report, Department of Computer Science, Roskilde University, 2000.

[66] Hgmis. DOE Human Genome Program Primer on Molecular Genetics. Genome, pages 1–44, 1992. Online available at: http://www.osti.gov/accomplishments/documents/individualPage/ACC0033/10.pdf.

[67] W. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. In: *Proceedings of the 9th annual International Conference of the Center for Non-linear Studies on Self-organizing, Collective, and Cooperative Phenomena in Natural and Artificial Computing Networks on Emergent computation*, pages 228–234, 1990.

[68] J. H.Holland. *Adaptation in Natural and Artificial Systems.* The University Michighan, 1975.

[69] I. Hong, A. B. Kahng, and B. Moon. Improved Large-Step Markov Chain Variants for the Symmetric TSP. *Journal of Heuristics*, 3(1):63–81, 1997.

[70] H. Hoos, and T. Stützle. *Stochastic Local Search: Foundations & Applications.* Elsevier / Morgan Kaufmann, 2004.

[71] K . A. D. Jong *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems.* PhD thesis, Ann Arbor, MI, USA, 1975.

[72] D. S. Johnson. Local optimization and the traveling salesman problem. In: *Proceedings of the 17th Colloquium on Automata, Languages, and programming, Lecture Notes in Computer Science*, pages 446–461, 1990.

[73] D. S. Johnson, and L. A. McGeoch. Experimental Analysis of Heuristics for the STSP, Local Search in Combinatorial Optimization, Wiley & Sons, 2001.

[74] L. Kang, Y. Xie, and S. Y. You. Nonnumerical Parallel Algorithms: Simulated Annealing Algorithm, Being:Science Press 1997.

[75] L. Kang, A. Zhou, B. McKay, Y. Li, and Z. Kang. Benchmarking algorithms for dynamic travelling salesman problems. In: *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pages 1286-1292, 2004.

[76] R. Karp. Reducibility among combinatorial problems. In R. Miller, and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[77] G. Kendall, E. Soubeiga, and P. Cowling. Choice function and random hyper heuristics. In: *Proceedings of the fourth Asia-Pacific Conference on Simulated Evolution and Learning*, pages 667–671, 2002.

[78] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[79] N. Krasnogor. Co-evolution of genes and memes in memetic algorithms ocean. Graduate Student Workshop, 1999.

[80] N. Krasnogor, and J. Smith. A tutorial for competent memetic algorithms: Model, taxonomy and design issues. *IEEE Transactions on Evolutionary Computation*, 9(5):474–488, 2005.

[81] M. W. S. Land. *Evolutionary Algorithms with Local Search for Combinatorial Optimization*. PhD thesis, University of Californai, San Diego, 1998.

[82] P. Larranaga, C. M. H. Kuijpers, R. Murga, I. Inza, and S. Dizdarevic. Genetic algorithms for the traveling salesman problem: A review of representations and operators. *Artificial Intelligence Review*, 13(2):129–170, 1999.

[83] P. Larranaga, and J. A. Lozano. Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation. Kluwer Academic Publishers; 2002.

[84] J. K. Lenstra, and R. Kan. Some simple applications of the traveling salesman problems. *Operational Research Quarterly*, 26(4):717–733, 1975.

[85] C. Li, M. Yang, and L. Kang. A new approach to solving dynamic traveling salesman problem. In: *Proceedings of the 6th International Conference on Simulated Evolution and Learning (SEAL)*, pages 236–243, 2006.

[86] S. Lin, and B. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21(2):498–516, 1973.

[87] J. D. Litke. An improved solution to the traveling salesman problem with thousands of nodes. *Communications of the ACM*, 27(12):1227–1236, 1984.

[88] J. D. C. Little, K. G. Murty, D. W. Sweeney, and C. Karel. An algorithm for the traveling salesman problems. *Operation Research*, 11(6):979–989, 1963.

[89] J. A. Lozano, P. Larranaga, I. Inza, and E. Bengoetxea. Towards a New Evolutionary Computation: Advances on Estimation of Distribution Algorithms. Springer-Verlag; 2006.

[90] S. W. Mahfoud. Boltzmann selection. In: *Proceedings of the 7th International Conference on Genetic Algorithms*, pages 1–4. Morgan Kaufmann, San Francisco, 1997.

[91] S. W. Mahfoud, and D. E. Goldberg. Parallel recombinative simulated annealing: a genetic algorithm. *Parallel Computing*, 21(1):1–28, January 1995.

[92] J. Martikainen, and S. J. Ovaska. Hierarchical two-population genetic algorithm. *International Journal of Computational Intelligence Research*, 2(4):367–380, 2006.

[93] P. Mascato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical report, Caltech Concurrent Computation Program, 1989.

[94] M. Mavrovouniotis, and S. Yang. A memetic ant colony optimization algorithm for the dynamic traveling salesman problems. *Soft Computing*, 15(7):1405–1425, 2011.

[95] P. Merz. *Memetic Algorithms for Combinatorial Optimization Problems: Fitness Landscapes and Effective Search Strategies*. PhD thesis, University of Gesamthochschule Siegen, 2001.

[96] P. Merz, and B. Freisleben. Memetic algorithms for the traveling salesman problem. *Complex Systems*, 13(4):297-345, 1993.

[97] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[98] Z. Michalewicz. *Genetic Algorithms + Data Structure = Evolutionary Programs*. Springer-Verlag Berlin Heidelberg New York, 1999.

[99] Z. Michalewicz, and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer, second, revised and extended edition, 2004.

[100] P. Moscato, and C. Cotta. A Gentle Introduction to Memetic Algorithms *Handbook of Metaheuristics*, Kluwer Academic Publishers, pages 105–144, 2001.

[101] H. Mühlenbein, and G. Paaß. From recombination of genes to the estimation of distributions. Binary parameters. In: *LNCS 1411: Parallel Problem Solving from Nature, PPSN IV*, pages 178–187, 1996.

[102] H. Mühlenbein, M. G. Scheleuter, and O. Kramer. Evolution algorithms in combinatorial optimization. *Parallel Computing*, 7(1):65–85, 1988.

[103] H. Mühlenbein, and D. Schlierkamp-Voosen. Analysis of selection, mutation and recombination in genetic algorithms. In: *Proceedings of Evolution and Bio computation*, pages 142-168, 1995.

[104] Y. Nagata. Fast EAX algorithm considering population diversity for traveling salesman problems. In: *Proceedings of EvoCOP*, pages 171–182, 2006.

[105] Y. Nagata, and S. Kobayashi. Edge assembly crossover: A high-power genetic algorithm for the traveling salesman problems. In: *Proceedings of ICGA*, pages 450–457, 1997.

[106] H. D. Nguyen, I. Yoshihara, K. Yamamori, and M. Yasunaga. Implementation of an effective hybrid GA for large-scale traveling salesman problems.

IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 37(1):92–99, 2007.

[107] C. Nilsson. Heuristics for the traveling salesman problem. Technical report, Linkoping University, 2003.

[108] Y. S. Ong, and A. J. Keane. Meta-lamarckian learning in memetic algorithms. *IEEE Transactions on Evolutionary Computation*, 8(2):99–110, 2004.

[109] E. Ozcan, and M. Erenturk. A brief review of memetic algorithms for solving euclidean 2D traveling salesman problem. Technical report, Department of Computer Engineering Yeditepe University Istanbul, 2004.

[110] M. Padberg, and G. Rinaldi. Optimization of a 532-city Symmetric Traveling Salesman Problem by Branch and Cut, *Operations Research Letters*, 6: 1–7, 1987.

[111] M. Padberg, and G. Rinaldi, A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems, *SIAM Review*, 33(1):60–100, 1991.

[112] J. Paredis. Co-evolutionary computation. *Artificial Life*, 2(4):355–375, 1995.

[113] R. J. Parsons, S. Forrest, and C. Burks. Genetic algorithms, operators, and DNA fragment assembly. *Machine Learning*, 21:11–33, 1995.

[114] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. The Addison-Wesley series in artificial intelligence. Addison-Wesley Pub (Sd), 1984.

[115] P. A. Pevzner, H. Tang, and M. S. Waterman. An eulerian path approach to DNA fragment assembly. *Proc Natl Acad Sci USA*, 98(17):9748–9753, 2001.

[116] D. Pisinger, S. Ropke Large Neighbourhood Search. *Handbook of Meta heuristics*, 146:1–22, 2010.

[117] R. D. Plante, T. J. Lowe, and R. Chandrasekaran. The product matrix traveling salesman problem: An application and solution heuristics. *Operations Research*, 35(5):772–783, 1987.

[118] R. C. Prim Shortest connection networks and some generalizations, *Bell Systems Technical Journal*, 36:1389–1401, 1957.

[119] P. Preux, and E.G. Talbi. Towards hybrid evolutionary algorithms *International Transactions in Operational Research*, 6(6):557–570, 1997.

[120] H. N. Psaraftis. Dynamic vehicle routing problems. In Vehicles Routing: Methods and Studies, Elsevier Science Publishers, 1988.

[121] S. S. Ray, S. Bandyopadhyay, and S. K. Pal. New operators of genetic algorithms for the traveling salesman problem. In: *Proceedings of the 2004 International Conference on Pattern Recognition (ICPR)*, pages 497–500, 2004.

[122] I. Rechenberg, and P. Bienert. Evolution strategies. Technical report, TU Berlin, 1969.

[123] G. Reinelt. *The traveling salesman problem, Computational Solutions for traveling salesman problem Applications*, *Lecture Notes in Computer Science*. Springer, 1994.

[124] G. Reinelt. TSPLIB95, University Heidelberg, 1995. Online avialable (http://www.iwr.uniheidelberg.de/groups/comopt/software/TSPlib95).

[125] M. Rocha, and J. Neves. Preventing premature convergence to local optima in genetic algorithms via random offspring generation. In: *Proceedings of the 12th International conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems: Multiple Approaches to Intelligent Systems*, pages 127–136, 1999.

[126] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis. An analysis of several heuristics for the traveling salesman problems. *SIAM Journal of Computation*, 6(3):563–581, 1977.

[127] H. M. Salkin, and K. Mathur. *Foundations of Integer Programming*. North-Holland, 1989.

[128] H. Schwefel. Numerische Optimierung von Computer-Modellen mittels der Evolutions-Strategie. *Interdisciplinary Systems Research, Basel: Birkhäuser Verlag*, 26, 1977.

[129] D. Shapiro. *Algorithms for the Solution of the Optimal Cost Traveling Salesman Problem*. PhD thesis, Washington University, St. Louis, 1966.

[130] S. N. Sivanandam, and S. N. Deepa. *Introduction to Genetic Algorithms (1st ed.)*. Springer Publishing Company, Incorporated, 2007.

[131] J. E. Smith. Co-evolving memetic algorithms: A review and progress report. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 37(1):6–17, 2007.

[132] J. Maynard-Smith. The Evolution of Sex. Cambridge University Press, Cambridge, UK, 1978.

[133] S. Sumathi, T. Hamsapriya, and P. Surekha. Evolutionary Intelligence: An Introduction to Theory and Applications with Matlab (1st ed.). Springer Publishing Company, 2008.

[134] G. Syswerda. Schedule optimization using genetic algorithms. In: *Handbook of Genetic Algorithm*, pages 332–349, 1991.

[135] T. Stützle, and H. H. Hoos. MAX-MIN Ant System, *Future Generation Computer Systems.*, 16(8):889–914, 2000.

[136] T. Stützle. ACOTSP, Version 1.0, available from http://www.aco-metaheuristic.org/aco-code, 2004.

[137] G. Syswerda. A study of reproduction in generational and steady state genetic algorithms. In: *Proceedings of FOGA*, pages 94–101, 1990.

[138] G. Tao, and Z. Michalewicz. Inver-over operator for the Travelling Salesman Problem. In: *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, pages 803–812, 1988.

[139] A. A. R. Townsend Genetic Algorithm A Tutorial [online]. Online Avilable: www-course.cs.york.ac.uk/evo/SupportingDocs/TutorialGAs.pdf, 2003.

[140] I. Troitski. Laser-induced damage creates interior images. Technical report, Optical Engineering Reports.

[141] H. K. Tsai, J. M. Yang, Y. F. Tsai, and C. Y. Kao. An evolutionary algorithm for large traveling salesman problem. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 34(4):1718–1729, 2004.

[142] C. Voudouris, and E. Tsang. Guided local search and its application to Travelling Salesman Problem. *European Journal of Operation Research*, 1998.

[143] J. D. Watson, and F. H. C. Crick Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid , *Nature*, 171(4356): 737–738, 1953.

[144] T. Weise. *Global optimization algorithms – Theory and Application.* Self-Published, second edition, 2009. Online available (http://www.it-weise.de/projects/book.pdf).

[145] D. Whitley, T. Starkweather, and D. Fuquay. Scheduling problems and traveling salesman problem: The genetic edge recombination operator. In: *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 133–140. Morgan Kaufamann Publishers, 1989.

[146] D. Whitley, V. S. Gordon, and K. Mathias. Lamarckian Evolution, The Baldwin Effect and Function Optimization. In: *Proceedings of Parallel Problem*

Solving From Nature PPSNIII, (Y. Davidor and H.-P. Schwefel, eds.), Springer, pages 6–15, 1994.

[147] D. Wolfgang, and D. Andreas. Einführung in Operations Research. Springer-Lehrbuch. Springer, 4. verb. Aufl., 1998.

[148] D. H. Wolpert, and W. G. Macready. No free lunch theorems for search. Technical report, Santa Fe Institute, Feb. 1995.

[149] X. F. Xie, and J. Liu. Multi-agent optimization system for solving the traveling salesman problem. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 39(2):489–502, 2009.

[150] H. Yang, L. Kang, and Y. Chen. A gene-pool based genetic algorithm for traveling salesman problem. *Wuhan University Journal of Natural Sciences*, 8:217–223, 2003.

[151] M. Yang, L. Kang, and J. Guan. Multi-algorithm co-evolution strategy for dynamic multi-objective traveling salesman problem. In: *Proceedings of 2008 IEEE World Congress on Computational Intelligence*, pages 466 –471, 2008.

[152] Q. Zhang, J. Sun, and E. Tsang, Evolutionary algorithm with the guided mutation for the maximum clique problem, *IEEE Transactions on Evolutionary Computation*, 9(2):192–200, 2005.

[153] A. Zhou, L. Kang, and Z. Yan. Solving dynamic traveling salesman problem with evolutionary approach in real time. In: *Proceedings of 2003 IEEE Congress on Evolutionary Computation*, pages 951–957, 2003.