A generic bee colony optimization framework for combinatorial optimization problems

Wong, Li Pei

2012

Wong, L. P. (2012). A generic bee colony optimization framework for combinatorial optimization problems. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/48091>

<https://doi.org/10.32657/10356/48091>

NANYANG
TECHNOLOGICAL
UNIVERSITY

**A GENERIC BEE COLONY OPTIMIZATION
FRAMEWORK FOR COMBINATORIAL OPTIMIZATION
PROBLEMS**

WONG LI PEI
SCHOOL OF COMPUTER ENGINEERING
2012

**A GENERIC BEE COLONY OPTIMIZATION
FRAMEWORK FOR COMBINATORIAL OPTIMIZATION
PROBLEMS**

WONG LI PEI

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for the degree of
Doctor of Philosophy

2012

Abstract

Combinatorial optimization is one of the areas that has been studied intensively in computer science and operations research. Solving a Combinatorial Optimization Problem (COP) is NP-hard [55] and it involves finding a set of feasible solutions with the least-cost in a discrete search space [90]. It becomes complicated when the COP being solved is with large scale data, constraints and variables. The significance of COPs can be found in various types of industrial applications. Examples include finding an optimized scan chains route in integrated chips testing and job scheduling in manufacturing industries.

This thesis presents a generic Bee Colony Optimization (BCO) framework for solving different COPs. The framework realizes computationally the collective intelligence shown in bee foraging behaviour. In a bee colony, bees are sent out of their beehive and travel across different locations to discover and to collect food. When a bee found a food source, it would perform waggle dance upon flying back to its hive. Its intension is to recruit more bees towards the newly discovered food source. Waggle dance operates as an important communication tool among bees. This collective intelligence shown in the bee foraging behaviour is modelled algorithmically as a generic framework to solve different COPs.

The generic BCO framework is designed using the object-oriented approach. A set of domain independent and abstract classes are defined. The domain independent classes implement the bee foraging behaviour and the abstract classes act as an interface to interact with domain specific classes. To solve a COP with the proposed framework, the implementation of domain specific tasks as defined in the abstract classes needs to be supplied to the framework. Besides the advantage of solving different COPs, the proposed BCO framework can be modified or enhanced flexibly and the effect of the enhancement will be applicable across all different COPs being solved.

Other than the waggle dance mechanism which encourages an intensive searching around a promising region, the proposed BCO framework is equipped with an initialization engine, a state transition rule, a local search and a pruning strategy. The initialization engine ensures that each bee is equipped with a feasible solution at the initial stage. The Fragmentation State Transition Rule aids a bee in constructing a feasible solution under the influence of arc fitness and heuristic distance. The local search method, of which its implementation is domain specific, helps in improving the resulting solutions generated by bees. The Dynamic Frequency-based Pruning Strategy allows a subset of resulting solutions to be locally optimized and hence minimizes the high computational overhead incurred by the local search procedure.

Three COPs are selected as the testbed for the proposed BCO framework. They are the Traveling Salesman Problem (TSP), the Quadratic Assignment Problem (QAP) and the Job Shop Scheduling Problem (JSSP). The results are compared against other bio-inspired algorithms such as Ant Colony Optimization, Particle Swarm Optimization and other variants of bee related algorithms. Out of 95 TSP benchmark instances, 53% of them are optimally solved and 93% of them record $< 1\%$ deviation from the known optimum. On average of all 95 instances, they are solved to 0.27% deviation from the known optimum. For QAP, a total of 134 benchmark instances are tested. 63% of them are optimally solved. On average of 134 instances, they are solved to 0.33% deviation from the known optimum. For JSSP, out of 82 benchmark instances, 37% of them are optimally solved. On average of 82 instances, they are solved to 3.02% deviation from the known optimum.

Acknowledgments

First and foremost, I would like to thank my supervisors, Dr. Malcolm Low Yoke Hean, Dr. Chong Chin Soon and Assistant Professor Dr. He Bingsheng, for guiding me through this domain of scheduling with meta-heuristic. Being open-minded and helpful, they have provided me tonnes of suggestions, ideas and criticism. They also have given me enough freedom in my study. Their effectiveness in providing a comfortable and conducive research environment and the facilities in the Parallel Distributed Computing Center (PDCC) have eased my pursuit of knowledge. Besides, I am grateful for their encouragement to me in submitting papers about this research to various conferences and journals.

Special thanks goes to the Universiti Sains Malaysia and the Ministry of Higher Education of Malaysia for the awarded scholarship, which provided me with financial security during the period of my PhD study. My thanks also goes to Professor Dr. Rosni Abdullah (Dean of School of Computer Sciences, Universiti Sains Malaysia) and Professor Dr. Ahamad Tajudin Khader (Deputy Dean of Graduate Studies & Research, School of Computer Sciences, Universiti Sains Malaysia) for the supports given to me.

My thanks also goes to PDCC lab technicians, Mdm. Irene Goh, Ms. Tay Siew Eng and Mr. Kenneth, for their courtesy and support in answering my technical inquiries on facilities in the PDCC. Without their help, it is difficult for me to run experiments and collect results for this research. I thank Mr. Sun Hequan and Mr. Alan Ch'ng Hee Siah for their generous helps and advice.

Not to forget my thanks to my parents (Wong Kim Yoo and Ong Moo Nooi), my beloved wife (Goh Ai Ris), my son (Wong Zi Zheng) and both of my sisters (Wong Lai Fong and Wong Lai Kun), for being supportive during the period of my study. Without their support and understanding, I might not be able to continue and focus on my study.

Contents

Abstract	i
Acknowledgments	iii
List of Figures	viii
List of Tables	x
List of Abbreviations	xiii
1 Introduction	1
1.1 Combinatorial Optimization Problems and Its Applications	1
1.2 Foraging Behaviour of Bees: An Interesting Natural Metaphor	3
1.3 Research Motivations and Objectives	4
1.4 An Overview of the Proposed Method on Combinatorial Optimization Problems	5
1.5 Research Contributions	7
1.6 Thesis Outline	10
2 Combinatorial Optimization Problems	13
2.1 Traveling Salesman Problem (TSP)	14
2.2 Quadratic Assignment Problem (QAP)	15
2.3 Job Shop Scheduling Problem (JSSP)	17
2.4 Benchmark Problem Instances	20
2.5 Summary	23
3 Related Work	25
3.1 Classification of Techniques for Combinatorial Optimization Problems . .	26
3.2 Review and Survey Literature on TSP, QAP and JSSP	27

3.3	Exact Algorithms	29
3.4	Approximate Algorithms	30
3.4.1	Constructive Approaches	30
3.4.2	Improvement Approaches	32
3.4.3	Nature Inspired Approaches	35
3.4.3.1	Ant Colony Optimization (ACO)	36
3.4.3.2	Particle Swarm Optimization (PSO)	43
3.4.3.3	Bee Colony Optimization (BCO)	47
3.5	Summary	54
4	A Generic Bee Colony Optimization Framework	57
4.1	Encoding Scheme	58
4.2	The Generic Framework	59
4.3	The Algorithm	67
4.4	Domain Specific Procedures	71
4.4.1	Initialization	71
4.4.2	Heuristic Distance Interpretation and Computation	72
4.4.3	Local Search Methods	75
4.4.3.1	Fixed-radius Near Neighbour (FRNN) 2-opt with Tabu List for TSP	76
4.4.3.2	2-opt with Tabu List for QAP	80
4.4.3.3	Two-enhancement Scheme with Neighbourhood N5 Per- turbation for JSSP	81
4.5	Summary	85
5	Bee Foraging Behaviour	86
5.1	Step-by-step State Transition Rule (SSTR)	87
5.1.1	Arc Fitness Factor	88
5.1.2	Heuristic Distance Factor	90
5.1.3	The Transition Rule	91
5.2	Fragmentation State Transition Rule (FSTR)	92
5.3	Reinforcement of Past Solutions	96

5.4	Waggle Dance	97
5.4.1	Waggle Dance Observation and Selection	98
5.4.2	Waggle Dance Performance	99
5.5	Summary	101
6	Frequency-based Pruning Strategy	103
6.1	The Building Block	104
6.2	Static Frequency-based Pruning Strategy (SFBPS)	105
6.3	Dynamic Frequency-based Pruning Strategy (DFBPS)	109
6.4	Summary	111
7	Findings and Results	112
7.1	Key Performance Indicators	112
7.2	BCO Parameters Configuration	113
7.3	Symbols, Notations, Experimental Platform and Setup	117
7.4	Performance of the BCO Algorithm on TSP	120
7.4.1	Parameters Tuning for TSP	121
7.4.2	Results for TSP	122
7.5	Performance of the BCO Algorithm on QAP	130
7.5.1	Parameters Tuning for QAP	131
7.5.2	Results for QAP	132
7.6	Performance of the BCO Algorithm on JSSP	140
7.6.1	Parameters Tuning for JSSP	140
7.6.2	Results for JSSP	141
7.7	Comparison Studies	147
7.8	Summary	152
8	Conclusions and Future Work	155
8.1	Future Work	159
A	Author's Related Publications	162
B	Parameters Tuning Results for TSP	164

C Parameters Tuning Results for QAP	166
D Parameters Tuning Results for JSSP	168
References	170

List of Figures

2.1	Allocation of G_F over G_L with a permutation of $\pi = (B, A, E, C, D)$. . .	16
2.2	A disjunctive graph for 2-job x 3-machine JSSP instance in Table 2.1. . .	19
2.3	A directed graph (feasible solution) for 2-job x 3-machine JSSP instance in Table 2.1.	19
2.4	Gantt chart for the directed graph in Fig. 2.3.	20
3.1	Classification of techniques which have been employed to solve the focus problem of this thesis.	27
4.1	Class diagram of the generic BCO framework for solving different COPs.	60
4.2	Local search procedure with Tabu List integration for the TSP, QAP and JSSP.	76
4.3	(a) Original closed tour, R . (b) R' after 2-opt transformation.	77
4.4	Searching for the second edge based on the FRNN method where E_{EF} is the first edge.	79
4.5	The compare and swap procedure on a π with cardinality of six, $ \pi = 6$.	80
4.6	Swapping two operations in π where $p = 2$ and $q = 5$	81
4.7	Inserting the $q = 5$ -th operation before the $p = 2$ -nd operation in π	81
5.1	Values assignment to the edges of (H, A), (H, C), (H, I) and (H, J). . . .	95
6.1	An adjacency list that shows the building blocks that are greater than $\eta = 5\%$ of the total row-based frequency. These blocks are marked as hot spots.	108
7.1	Experimental setup for testing the effect of the major components of the proposed BCO algorithm.	118

8.1	Linear function for duration calculation.	159
-----	---	-----

List of Tables

2.1	A 2-job $\times$ 3-machine JSSP. Each element is denoted by $x(y)$, where $x \equiv$ machine id and $y \equiv$ processing time.	18
2.2	Sources of the COP benchmark problems.	21
3.1	Review and survey literatures which are related to the domains of TSP, QAP and JSSP.	28
3.2	JSSP neighbourhood structures.	35
3.3	The Ant Colony Optimization (ACO) algorithm and its variants on TSP, QAP and JSSP.	38
3.4	The Particle Swarm Optimization (PSO) algorithm and its variants on TSP, QAP and JSSP.	44
3.5	Review and survey literatures which are related to the BCO algorithm and its variants.	51
3.6	The Bee Colony Optimization (BCO) algorithm and its variants on TSP, QAP and JSSP.	51
3.7	Macro characteristics of the ACO, the PSO and the BCO algorithms. . .	55
3.8	Differences at the micro level of the ACO, the PSO and the BCO algorithms.	56
4.1	A 2-job $\times$ 3-machine JSSP. Each element is denoted by $x(y)$, where $x \equiv$ machine id and $y \equiv$ processing time. It is a reproduction of Table 2.1 in Section 2.3.	75
5.1	Lookup table for P_{follow} adjustment.	98
7.1	The high (+1) and low (-1) levels of the BCO parameters for the three selected COPs. They are used in the parameter tuning experiments. . . .	114

7.2	$2^5 = 32$ different parameter settings for the proposed BCO algorithm. . .	115
7.3	Different BCO variants for the TSP, QAP and JSSP. The variants are to test the effect of different components in the proposed BCO algorithm. .	120
7.4	Recommended BCO parameter settings for the RAT575.	122
7.5	Effect of different variants of the BCO algorithm on 20 selected TSPLIB benchmark problems.	123
7.6	Improvement percentage in terms of average solution cost and μ_T among the different BCO variants on 20 selected TAPLIB benchmark problems.	124
7.7	Computation of improvement percentage in terms of average solution cost and μ_T	125
7.8	Computational results of the proposed BCO algorithm on the selected 95 TSPLIB benchmark problem instances.	126
7.9	Analysis on the results shown in Table 7.8.	130
7.10	Recommended BCO parameter settings for the SKO100A.	132
7.11	Effect of different variants of the BCO algorithm on 13 selected QAPLIB benchmark problems.	134
7.12	Improvement percentage in terms of average solution cost and μ_T among the different BCO variants on 13 selected QAPLIB benchmark problems.	134
7.13	Computational results of the proposed BCO algorithm on the selected 134 QAPLIB benchmark problem instances.	135
7.14	Analysis of the QAP results shown in Table 7.13. The analysis is based on different series of the QAPLIB benchmark problems.	138
7.15	Analysis of the QAP results shown in Table 7.13. The analysis is based on different <i>types</i> of the QAPLIB benchmark problems.	139
7.16	Recommended BCO parameter settings for the SWV01.	141
7.17	Effect of different variants of the BCO algorithm on 20 selected JSSP OR-Library benchmark problems.	142
7.18	Improvement percentage in terms of average solution cost and μ_T among the different BCO variants on 20 selected JSSP OR-Library benchmark problems.	143

7.19	Computational results of the proposed BCO algorithm on the 82 JSSP OR-Library benchmark problem instances.	144
7.20	Analysis of the JSSP results shown in Table 7.19. The analysis is based on different series of the JSSP OR-Library benchmark problems.	146
7.21	Performance comparison of the BCO algorithm versus various ACO, PSO and ILS algorithms. Average of δ_{min} and δ_{avg} for all the instances reported in each literature are shown.	148
7.22	Algorithm execution settings used by the benchmark approaches and the proposed BCO algorithm (BCO(A)) reported in Table 7.21.	148
7.23	Performance comparison of the BCO algorithm versus the BS algorithm.	151
7.24	Performance comparison of the BCO algorithm against the results reported in the DIMACS challenge.	153
7.25	A summary of the performance of the proposed BCO algorithm.	154
8.1	Performance summary of the proposed BCO algorithm.	158
B.1	Results of the BCO parameters tuning experiment for TSP. The RAT575 is used as the testbed.	164
B.2	P-value of the F-Test and T-Test together with the 95% CI for difference between μ_i and $\mu_{min} = 6824.5$ obtained from the RAT575 parameters tuning experiment.	165
C.1	Results of the BCO parameters tuning experiment for QAP. The SKO100A is used as the testbed.	166
C.2	P-value of the F-Test and T-Test together with the 95% CI for difference between μ_i and $\mu_{min} = 152381.0$ obtained from the SKO100A parameters tuning experiment.	167
D.1	Results of the BCO parameters tuning experiment for JSSP. The SWV01 is used as the testbed.	168
D.2	P-value of the F-Test and T-Test together with the 95% CI for difference between μ_i and $\mu_{min} = 1511.0$ obtained from the SWV01 parameters tuning experiment.	169

List of Abbreviations

ACO	Ant Colony Optimization.
ACOFT	Ant Colony Optimization with Fast Taboo.
ACOPSS	Ant Colony Optimization with Parameterized Search Space.
ACS	Ant Colony System.
APE	Adjacent Pairwise Exchange.
BAS	Backtracking Ant System.
BCBV	Bee Colony Optimization with Big Valley Landscape Exploitation.
BCNS	Bee Colony Optimization with N5 Neighbourhood Search.
BCO	Bee Colony Optimization.
BS	Bee System.
DFBPS	Dynamic Frequency-based Pruning Strategy.
C3DPSO	Particle Swarm Optimization with C3 Mutation Factor.
COP	Combinatorial Optimization Problem.
FBPS	Frequency-based Pruning Strategy.
FRNN 2-opt	Fixed-radius Near Neighbour 2-opt.
FSTR	Fragmentation State Transition Rule.
GA	Genetic Algorithm.
GACS	ACS with Generalized Insertion.
HDPSO	Hybrid Discrete Particle Swarm Optimization.
HPSO	Hybrid Particle Swarm Optimization.
HTSACO	Hybrid of Tabu Search and Ant Colony Optimization.
TSACS	Two-stage Ant Colony System.
JSSP	Job Shop Scheduling Problem.
MMAS	MAX-MIN Ant System.
MPSO	PSO with Simulated Annealing.
MWKR	Most Work Remaining.

NN Heuristic	Nearest Neighbourhood Heuristic.
PSO	Particle Swarm Optimization.
PSOFA	Particle Swarm Optimization with Forma Analysis.
PSO+LS	Particle Swarm Optimization with Local Search.
PHAS	Population-based Hybrid Ant System.
QAP	Quadratic Assignment Problem.
RAACS	Any Colony System with Randomized Algorithm.
SFBPS	Static Frequency-based Pruning Strategy.
SSTR	Step-by-step State Transition Rule.
TESN5	Two-enhancement Scheme with Neighbourhood N5 Perturbation.
TSP	Traveling Salesman Problem.

Chapter 1

Introduction

This thesis presents a method for solving Combinatorial Optimization Problems (COPs). The proposed method is inspired by the foraging behaviour of bees where waggle dances are used as a communication medium among bees. The proposed method is implemented as a generic framework for solving different COP instances.

In this chapter, an introduction to the entire research is provided. It starts with a brief discussion on COP and its applications. It is then followed by an introduction to the bee foraging behaviour found in a typical bee colony. Subsequently, the research motivations, scope, as well as the research contributions are presented. Finally, a thesis outline is included at the end of the chapter.

1.1 Combinatorial Optimization Problems and Its Applications

Solving a Combinatorial Optimization Problem (COP) involves finding a set of feasible solutions to a mathematical problem in a discrete search space such that only solutions with the minimum or maximum cost are identified [90]. Examples of common COPs are Traveling Salesman Problem (TSP), Quadratic Assignment Problem (QAP) and Job Shop Scheduling Problem (JSSP).

Combinatorial optimization is one of the areas that has been studied intensively in computer science and operations research. In the domain of computational mathematics, COPs are employed as testbed for a new proposed computational algorithm. The success of solving benchmark COPs indicates that the algorithm can be used to solve real world problems. Although a COP is easy to describe, the difficulty of solving it becomes more and more obvious with increasing problem dimension. To further explain this point, consider an example of solving a TSP instance with 13509 cities in USA (USA13509) using the Concorde TSP Solver¹. As mentioned in [7], a total of 554.8 CPU days is needed to solve the USA13509 problem to optimum by using a cluster of 2.66GHz Intel Xeon processors.

Over the years, COPs have been studied and investigated from various perspectives. Some of these works focus on how to achieve a high level of computational accuracy on a set of benchmark problems [49, 52] and others focus on how to increase the speed to obtain an optimum solution [23, 119]. The research direction has now been shifted such that a wide range of large-scale COP instances are attempted. For this, many methods have been proposed. The methods can be either classified as an exact or an approximate algorithm. Examples of the exact algorithms are Branch-and-Bound [23, 70], Branch-and-Cut [7, 48] etc. Examples of the approximate algorithms are Ant Colony Optimization [52, 136], Particle Swarm Optimization [84, 131], Insertion heuristic [56, 57] etc. These algorithms will be further discussed in Chapter 3.

In this thesis, three COPs are selected to be the testing ground for the proposed algorithm. They are TSP, QAP and JSSP. A brief introduction is given to each of them as follows. Solving a TSP requires the finding of a tour with the shortest distance such that a salesman starts at a city, visits each city exactly once and returns to the start

¹The Concorde TSP Solver combines most of the theories and algorithms explained in [7]. To date, it is one of the best solvers that is able to solve a very large TSP instances to optimum. Further information about it can be found in Section 3.3.

city eventually. Solving a QAP on the other hand requires the finding of the least-cost² allocation plan of a set of facilities to a set of locations. Solving a JSSP requires the finding of a schedule of operations with the minimum makespan, such that the schedule complies with the job precedence and machine capacity constraints. Further information about TSP, QAP and JSSP can be found in Chapter 2.

The significance of these three selected problems is shown in many industrial applications. Examples include finding an optimized scan chains route in integrated chips testing, parcels collection and delivery in logistics companies, genome sequencing, job scheduling in manufacturing companies, typewriter keyboard design and facility-location planning in retail businesses. Finding an optimum solution for such COPs with large scale data, constraints and variables is NP-hard [55]. The wide range of real-world applications for COPs shows that COPs are important problems to solve and carry out research on.

1.2 Foraging Behaviour of Bees: An Interesting Natural Metaphor

Bees are highly organized social insects. The survival of the entire colony depends on every individual bee carrying out its tasks in a cooperative manner. Bees practise a systematic task segregation among themselves to ensure a continued existence of its colony. They perform various tasks such as foraging, reproduction, taking care of young, patrolling, housekeeping and constructing hive. Of these, foraging is one of the most important activity as bees have to ensure an uninterrupted supply of food to the colony.

When a bee found a new food source, upon flying back to its hive, the bee would perform an informative waggle dance to its hive mates. Its intention is to attract more

²The cost can be construction cost, transportation cost or other cost measurement that is related to the nature of the problem being solved.

bees towards the newly discovered food source. Via the waggle dance, important information about the newly discovered food source (e.g. direction and distance) is conveyed to other bees. The waggle dance serves as an important communication medium among bees. Further information about waggle dance can be found in Section 3.4.3.3 where literatures with discussion from biology perspective are cited. The unique bee foraging behaviour is interesting and it has been modelled as a computational algorithm to solve complex problems such as COPs in this thesis.

1.3 Research Motivations and Objectives

One of the motivations of this research is to study how a COP is solved using a bee-inspired algorithm. We would like to evaluate the performance of the algorithm in terms of computational accuracy and speed. The bee foraging behaviour as described in Section 1.2 has a close resemblance with the nature of solving a COP [96]. In nature, a bee would travel from one location to another location in order to find a new food source. When a new food source is found, the bee would perform waggle dance to attract more of its hive mates towards the newly discovered food source. Similarly, when a COP is being solved, a solution is build incrementally until a feasible solution is acquired. When a feasible solution is obtained at a specific region within the search space, the searching around that region would be intensified so that a better feasible solution could be acquired.

Given the close match between the foraging behaviour of bees and the nature of solving a COP, we are motivated to investigate and computationally realize the bee foraging behaviour. Various approaches and strategies are adapted to model the foraging behaviour and this enriches the domain of Bee Colony Optimization. Besides waggle dance modeling, this thesis proposes a fragmented solution construction mechanism and a dynamic pruning strategy which are described in Chapters 5 and 6 respectively.

We also aim to design and develop a generic Bee Colony Optimization framework that supports multiple instances of COPs. Any enhancement/modification on the framework would be applicable to all different COP instances being solved by the framework. Three COPs are identified as testbed for such framework, i.e. TSP, QAP and JSSP.

Solving a COP with large dimension can take a long time. However, in most cases, obtaining a reasonably good solution within a short period of time is more important than acquiring an optimum solution with much longer time. Decision makers could be satisfied by these sub-optimum solutions when a quick decision needs to be made. The proposed BCO algorithm in this thesis is a meta-heuristic which has the mentioned advantage. By mimicking the bee foraging behaviour found in nature, the proposed algorithm requires less domain specific knowledge and it is able to generate sub-optimum solutions quickly.

Last but not least, this research is also motivated by the challenge to address some unsolved benchmark instances. For instance, in the domains of QAP and JSSP, 31% and 43% of the QAPLIB and Taillard instances are not solved to optimum. The details of these unsolved benchmark problem instances are described in Section 2.4.

1.4 An Overview of the Proposed Method on Combinatorial Optimization Problems

This section provides an overview of the proposed Bee Colony Optimization algorithm that will be applied to solve the Combinatorial Optimization Problems. Also, this overview can be viewed as the boundary or scope of this research.

As mentioned in Section 1.3, we aim to design and develop a generic Bee Colony Optimization framework for solving different COP instances. The underlying mechanism of the generic BCO framework is the bee foraging behaviour in a typical bee colony where bees travel to different locations to discover new food sources. When new food sources are discovered, waggle dances are used as a communication medium among bees

to exchange information about the newly discovered food sources. To develop the generic BCO framework, an encoding scheme to represent a solution for a COP is defined. The encoding scheme is named as the “permutation without repetition”. Details about the encoding scheme and the generic BCO framework design can be found in Chapter 4.

In the proposed generic BCO framework, a bee is able to construct a feasible solution incrementally with the aid of a state transition rule. Before a bee leaves its hive to forage, it observes waggle dances performed by its mates. If it decides to follow a dance, it will be equipped with a preferred path which serves as a guide during the foraging process. The state transition rule aids a bee to explore and exploit the search space by repeatedly connecting a fragment of nodes at one time under the influence of the preferred path and heuristic distance. Besides the ability to construct a feasible solution with the aid of the state transition rule, the bee is also able to reinforce some past solutions acquired from its past foraging activities.

After a bee has constructed a feasible solution, upon its return back to its hive, the bee will start dancing to share information about its finding with other hive mates. The waggle dances by bees are implemented as an accumulation of a set of feasible solutions. Each feasible solution is accumulated and stored for a certain duration. If a bee’s profitability³ is better than average profitability of the entire colony, the waggle dance performed by that bee will stay longer in the accumulation list. Otherwise, the dance duration is shorter. In terms of observing and following a dance, these are implemented via a definition of a probability value, namely: the probability to follow a waggle dance. It is dynamically adjusted with the aid of a lookup table. Details on the incremental path construction, state transition rule, reinforcement of past solutions and waggle dance performance can be found in Chapter 5.

³The bee’s profitability refers to the quality of the solution generated by a bee. The higher the profitability, the better the solution quality. For instance, if a TSP is being solved, the profitability is defined as the reciprocal of the tour length. Details on this can be found in Chapter 5.4.

The feasible solution constructed by a bee is locally optimized by a neighbourhood search operator. The neighbourhood search operator is domain specific and depends on the COP being solved (refer to Section 4.4.3). In the proposed BCO algorithm, not all feasible solution constructed by bees will undergo the local search operation. A pruning method which is named as the Dynamic Frequency-based Pruning Strategy (DFBPS) is proposed for this purpose (refer to Chapter 6). The DFBPS is able to reduce the high overhead of a local search operator and maintain the computational accuracy of the proposed BCO algorithm.

Like other population-based algorithms such as Genetic Algorithm and Ant Colony Optimization, the search process of the proposed BCO algorithm would very likely to be trapped in a local optimum. To help the search process to escape from such local optimum, a memory resetting policy is defined such that every bees will lose part of their past foraging experience. Details on this resetting policy can be found in Section 5.4.2.

The proposed BCO algorithm shows characteristics such as autonomy, distributed functioning and self organization in its problem solving process. It is similar to what is shown in insect systems such as bees or ants where these simple organisms are able to perform highly complex tasks by dynamically interacting with each other.

The proposed BCO framework will be tested on a set of benchmark problems (i.e. TSP, QAP and JSSP). The performance of the proposed BCO framework will be compared against other existing nature inspired algorithms. Such comparison studies are presented in Section 7.7.

1.5 Research Contributions

This research is to design and develop a BCO algorithm with its core problem solving process inspired by the bee foraging behaviour. In a bee colony, the two major activities in foraging are bee movement and bee recruitment via waggle dance performance. These

two activities are computationally realized to be the major components of the proposed BCO algorithm.

To develop the bee movement mechanism, a step-by-step state transition rule that aids a bee to construct a solution is proposed. The transition rule works under the influence of arc fitness and heuristic distance. It allows a bee to travel from one point to another point until it finds a food source. The point is selected via a guided stochastic approach rather than via a greedy approach. In this case, the greedy approach refers to always picking the nearest point.

Due to the nature of this state transition rule which constructs a solution on a step-by-step basis, it requires long computational time when it is applied to a problem with large dimension and hence it is not scalable for this type of problems. To overcome this limitation, a state transition rule which selects a fragment of points rather than one single point is proposed. This mechanism is named as Fragmentation State Transition Rule. The proposal to replace the Step-by-step State Transition Rule with the Fragmentation State Transition Rule leads to a lower computational overhead while maintaining the advantage to pick a fragment of points via a guided stochastic process.

On the other hand, to develop the mechanism of bee recruitment via waggle dance performance, a linear duration function together with a lookup table are proposed. The linear duration function ensures that a bee that finds a better food source will dance longer. The lookup table is used to determine if a bee needs to observe and follow the dance performed by its mates before leaving its hive to start foraging.

The development of the bee movement through a state transition rule and the bee waggle dance performance through a linear duration function and a lookup table contributes to the state of the art of the bee colony related algorithms. It provides another alternative to model the bee foraging behaviour which uses waggle dances to recruit more bees towards the newly discovered food source. Details on these mechanisms are described in Chapter 5.

Besides the mechanisms of the bee movement and waggle dance, a pruning strategy is also integrated as part of the BCO algorithm. The proposed pruning strategy is another contribution of this research. The fundamental idea behind this pruning strategy is as follows. It decides if a solution generated by a bee needs to undergo a local optimization procedure by considering the frequency of the smallest building block of the solution. In another word, the pruning strategy prohibits a subset of solutions from undergoing the local optimization procedure. This would decrease the computational overhead incurred by the local search procedure as not every generated solution would undergo the local optimization procedure. The pruning strategy was initially developed as a static approach, namely: the Static Frequency-based Pruning Strategy. The Static Frequency-based Pruning Strategy requires a tuning on two parameters defined in the approach. The static tuning of two parameters motivated us to proposed another pruning strategy which is named as the Dynamic Frequency-based Pruning Strategy. In this strategy, the two parameters are dynamically tuned and hence the static configuration is avoided. This makes the Dynamic Frequency-based Pruning Strategy a preferable one to be used in the BCO framework. Details on these pruning strategies are described in Chapter 6.

As mentioned in Section 1.1, many real world applications exhibit characteristics of COP. The success of solving a COP with the proposed bee-inspired algorithm would demonstrate the feasibility of solving these real world applications with similar technique. Thus, the development of the proposed algorithm serves as an explanatory example for such real world applications. To achieve this, the proposed BCO algorithm is developed as a generic framework such that it is able to solve different COPs. To demonstrate such flexibility to solve different COPs, three showcases are presented. They are the Traveling Salesman Problem, Quadratic Assignment Problem and Job Shop Scheduling Problem. In the proposed generic BCO framework, a set of domain independent classes and abstract classes are defined. The advantages of implementing the BCO algorithm

as a generic framework are two-fold. First, by inheriting and implementing abstract classes, a different COP can be ported into the generic BCO framework. The porting of different COPs includes the implementation of different local optimization procedures and initialization techniques. Second, any enhancement on the domain independent classes will be applicable to all the problem instances being solved. Details on the generic BCO framework are further discussed in Chapter 4.

1.6 Thesis Outline

This thesis consists of eight chapters. It begins with this introduction chapter (Chapter 1) which provides an overview of the entire research. It also highlights the research motivations and significance of this research.

Chapter 2 describes the three COP instances in detail, namely: the TSP, QAP and JSSP. These three selected problems will be the testbed for the proposed BCO algorithm. The selected problems are described from various perspectives. This includes problem definitions and formulations, graph representation, real world applications and various sources of benchmark problems.

Chapter 3 reviews some related works which have been proposed to solve TSP, QAP and JSSP. Each method discussed in this chapter is either grouped as an exact or approximate approach. A figure (Fig. 3.1) that illustrates the major groups of techniques that have been employed to solve the TSP, QAP and JSSP is included in Section 3.1. Besides bee colony related algorithms, other approaches such as the Branch-and-Bound, Branch-and-Cut, construction approaches (e.g. Shifting Bottleneck procedure, Nearest Neighbourhood heuristic), improvement approaches (e.g. Adjacent Pairwise Exchange, LIN-KER heuristic), Ant Colony Optimization and Particle Swarm Optimization are described. The reported results from these reviewed publications will be used in our comparison study in Chapter 7.

Chapter 4 presents a generic BCO framework for solving different COP instances. The proposed generic framework is designed using the object-oriented approach and a class diagram is enclosed in Section 4.2. The generic BCO framework consists of a set of domain independent classes and abstract classes. When a specific COP is ported into the framework via inheriting and implementing the abstract classes, the COP needs to be associated with a set of domain specific classes. The proposed generic BCO framework is flexible in terms of testing new COP instances as well as framework enhancement. Any enhancement of the domain independent classes will be applicable to all the problem instances being solved. Three domain specific procedures, namely: the initialization, heuristic distance interpretation and computation, and local search methods are also explained. Three domain specific local search approaches are highlighted in this chapter. They are the Fixed-radius Neighbour 2-opt (FRNN 2-opt) for TSP, a 2-opt approach for QAP and the Two-enhancement Scheme with N5 Neighbourhood for JSSP.

Chapter 5 describes how the bee foraging behaviour in a real bee colony is computationally realized as an effective tool to solve different types of COPs. Two state transition rules are described, namely: the Step-by-step State Transition Rule (SSTR) and the Fragmentation State Transition Rule (FSTR). These two rules aid the bees in the BCO algorithm to construct a set of feasible solutions. Besides, the bees are allowed to reinforce past solutions which are acquired from the past. This chapter also describes how the waggle dance model is computationally realized. The major motivation of having waggle dance in the proposed BCO algorithm is to intensify the search around a promising area in the search space so that better solutions can be acquired. When a bee dances, the dance will last for a certain duration which is linearly proportional to the bee's profitability.

Chapter 6 discusses a pruning strategy which is integrated as part of the domain independent classes in the proposed generic BCO framework. This pruning strategy

is named as the Frequency-based Pruning Strategy. Its objective is to prohibit a set of solutions from being locally optimized by a local search method, due to the high computation overhead incurred by the local search method. This strategy is analyzed from two different perspectives: static pruning versus dynamic pruning. The dynamic pruning strategy is preferred as it eliminates the parameters tuning procedure required for the static pruning strategy.

Chapter 7 presents the findings and results of the experiments for solving the three selected problems with the proposed BCO algorithm. It also highlights the results obtained from the enhancement on the BCO domain independent classes. From the reported results, it is noted that the enhancement is effective for all the three focus problems. A comparison study is included in this chapter where the performance of the BCO algorithm is compared against some benchmark algorithms described in Chapter 3.

Finally, Chapter 8 concludes the thesis. It also outlines some possible future studies of this research.

Chapter 2

Combinatorial Optimization Problems

Combinatorial Optimization Problems (COPs) appear in various types of industrial applications. Examples include finding an optimized scan chains route in integrated chips testing, parcels collection and delivery in logistics companies, job scheduling in manufacturing companies and facility-location planning in retail businesses. COPs still remains as an interesting and challenging problem in computer science and operations research. The problem becomes much more complicated when the dimension of the COP being solved is large.

Three instances of COPs (i.e. TSP, QAP and JSSP) will be described in this thesis. These three problems will become our test scenarios in examining the performance of the proposed BCO algorithm.

This chapter will first describe the three instances of COPs. The definitions and formulations of these three COP instances together with their possible applications will be explained in Sections 2.1, 2.2 and 2.3. Section 2.4 describes various sources of benchmark problems that will be used throughout the testing of the proposed BCO algorithm. Finally, a summary is presented in Section 2.5.

2.1 Traveling Salesman Problem (TSP)

In TSP, suppose a salesman is given a set of n cities associated with a traveling distance matrix from any city to any other cities, the salesman is required to make a Hamiltonian tour with minimum distance/cost. The Hamiltonian tour is also referred to as a round trip tour whereby the salesman must visit each city once and only once, and eventually return to the starting city [7, 86].

TSP can be denoted by graph notations as follows. Let $G = (V, E)$ be a weighted complete graph where V is a set of n cities ($V = \{v_1, \dots, v_n\}$) and E is a set of arcs or edges ($E = \{(r, s) : r, s \in V\}$). E is associated with a distance matrix, $D = \{d_{r,s}\}$ where $d_{r,s}$ denotes the distance between city r and city s . For all the combinations, if $d_{r,s} = d_{s,r}$, the problem is a symmetric TSP (STSP). Otherwise, it becomes an asymmetric TSP (ATSP). If Π denotes all possible permutations of set V , then solving the TSP is to determine a permutation $\pi \in \Pi$ that minimizes the total round trip distance as shown in Eq. 2.1, where $\pi(i) \in V$ indicates the i -th element in π .

$$C_{TSP}(\pi \in \Pi) = \sum_{i=1}^{n-1} d_{\pi(i), \pi(i+1)} + d_{\pi(n), \pi(1)} \quad (\text{Eq. 2.1})$$

The significance of TSP can be found in many areas. One of them is the application of TSP in logistics and transportation domain. This domain deals with movement of people, goods and equipments. In the school bus routing problem, the determination of a bus schedule for picking up school children is a TSP [51, 94]. Postal or parcel delivery is another application that exhibits TSP characteristics [45, 46].

In the area of medicine and bio-informatics, TSP plays an important role in genome sequencing. Before uncovering the genome sequence, it is important to have an accurate placement of markers that serve as landmarks for the genome maps. In this context, TSP provides an instrument for constructing sequences from experimental data on the closeness of pairs of markers [32].

In the area of semiconductor manufacturing such as integrated circuits production, post testing is a crucial step to ensure that products are in good quality. One of the common testing procedures is scan chains. A scan chain allows test data to be loaded into the scan points via the input end, and evaluates the results from the output end. It can be modeled as a TSP [66] as each scan point is treated as a city and the challenge is to find an ordering of scan points that reduces the time required in the testing phases.

Some literatures that describe TSP and its real world applications have been provided here. Further information on the history of TSP and its variations can be found in [7, 67, 72, 118].

2.2 Quadratic Assignment Problem (QAP)

In QAP, if a planner is given a set of n facilities associated with a distance matrix from any location to any other locations and a flow matrix from any facility to any other facilities, the planner needs to come out with an assignment plan of n facilities to n locations with minimum cost [100].

Similar to TSP, QAP can be expressed by a graph formulation. Let's consider two undirected weighted complete graphs, $G_F = (V_F, E_F)$ and $G_L = (V_L, E_L)$, which denotes the facility flows and location distances respectively. V_F is a set of n facilities and V_L is a set of n locations. E_F and E_L are sets of edges. E_F is associated with a $n \times n$ matrix $F = \{f_{r,s}\}$ that represents flow between facilities r and s (i.e. flow of products, peoples, information etc). E_L is associated with a $n \times n$ matrix $D = \{d_{i,j}\}$ that represents distance between locations i and j . With this configuration, solving the QAP is to obtain an optimal allocation of the vertices of G_F over G_L (as shown in Fig. 2.1). The aim is to determine a permutation $\pi \in \Pi$ (Π is all the permutations of set V_F) that minimizes the following:

$$C_{QAP}(\pi \in \Pi) = \sum_{i=1}^n \sum_{j=1}^n f_{i,j} \times d_{\pi(i),\pi(j)} \quad (\text{Eq. 2.2})$$

Figure 2.1: Allocation of G_F over G_L with a permutation of $\pi = (B, A, E, C, D)$.

where $\pi(i)$ indicates the location of node i in π .

Since QAP was introduced by Koopmans and Beckmann [87], it has become one of the most studied problems in combinatorial optimization. Among the applications that demonstrate the QAP characteristics are backboard wiring problem, plants-to-locations problem, typewriter keyboard design etc.

In the backboard wiring problem [133], the objective is to place components on the computer backboard such that the length of interconnecting wiring is minimized. This will reduce manufacturing cost and improve computational speed.

As for the plants-to-locations problem, decision makers would want to have an allocation plan such that manufacturing plants are built at various locations. The plan should minimize the inter-transportation cost among locations and at the same time it should fulfill customer demand in each location. Another problem which is similar to this is the hospital layout design as proposed in [47]. In the hospital layout design problem, Elshafei intended to place hospital departments so that the total distance traveled by patients is minimized.

In the typewriter keyboard design problem [34], the aim is to position the keys of keyboard such that the time required to write a text is minimal. Similar to the formulation in Eq. 2.2, two matrices are defined in this problem, namely: a frequency matrix

$F = \{f_{r,s}\}$, and a time matrix $D = \{d_{i,j}\}$. The elements in the frequency matrix denote the appearance frequency of the ordered pair of keys r and s . As for the time matrix, the elements represent the time required to press the key at location j after pressing the key at location i .

Some literatures that describe QAP have been provided here. Further information on QAP and survey of techniques that have been applied to solve it can be found in [18, 19, 24, 100].

2.3 Job Shop Scheduling Problem (JSSP)

As presented in [2], JSSP is defined by a set J of n jobs, $J = \{1, 2, \dots, n\}$ and a set M of m machines, $M = \{1, 2, \dots, m\}$. Each job J_i consists of a chain of m operations $\{O_{i1}, O_{i2}, \dots, O_{im}\}$. Each operations of J_i is to be processed on m machines in a pre-determined order (precedence constraints). O_{ij} denotes the j -th operation of job J_i which operates on a machine m_x for an uninterrupted processing time of τ_{ijx} . At a particular time, each machine can handle at most one operation (capacity constraints). No operation pre-emption is allowed along the jobs execution. With this, when a JSSP is being solved, the objective is to obtain a schedule that allocates operations of each job $J_i \in J$ on a set of machines subject to the precedence and capacity constraints. The generated schedule should show an optimization of a performance metric. The commonly used metrics that are related to JSSP include machine (resource) utilization, cycle time, throughput rate and inventory level. The optimization of machine utilization is important as it leads to a better throughput rate and lower product cost. One of the alternatives to measure the machine utilization is the makespan of a schedule and this performance metric will be used in this thesis. The makespan is the duration in which all operations of all jobs are accomplished.

Table 2.1: A 2-job $\times$ 3-machine JSSP. Each element is denoted by $x(y)$, where $x \equiv$ machine id and $y \equiv$ processing time.

	Operations		
	1	2	3
Job 1	2(3)	1(7)	3(6)
Job 2	1(8)	2(4)	3(5)

JSSP can be modeled by using a disjunctive graph $G = (O, E_{Con} \cup E_{Dis})$. O is the set of all operations of n jobs plus two fabricated operations “0” and “*”. The two fabricated operations “0” and “*” denote the “first” and the “last” operations respectively, $O = \{0, O_{11}, \dots, O_{1m}, O_{n1}, \dots, O_{nm}, *\}$. Hence, the cardinality of O is $|O| = (n \times m) + 2$. E_{Con} is a set of directed conjunctive arcs which denote the precedence constraints of each job. E_{Dis} is a set of bi-directional disjunctive arcs which denote the capacity constraints of each machine. This set of bi-directional disjunctive arcs tie all the operations that need to be handled by a machine.

Let's consider a two-job $\times$ three-machine problem ($n \times m \equiv 2 \times 3$ JSSP) as shown in Table 2.1. Each row represents a pre-defined machine precedence order for each job with the processing time in parentheses. As for the capacity constraint, operations that are processed by machine 1 is $m_1 = \{O_{12}, O_{21}\}$. Similarly, $m_2 = \{O_{11}, O_{22}\}$ and $m_3 = \{O_{13}, O_{23}\}$.

The problem in Table 2.1 is illustrated by a disjunctive graph shown in Fig. 2.2. In the disjunctive graph model, a solution π is obtained by converting the bi-directional disjunctive arcs to become directed arcs such that no cycle exists in the graph. This arcs conversion is equivalent to determining the starting times for each operation, $t_{ijx} \geq 0$. The aim is to minimize the makespan while fulfilling the precedence and capacity constraints as follows:

$$C_{JSSP}(\pi \in \Pi) = \max(t_{ijx} + \tau_{ijx}), \forall O_{ij} \in O, m_x \in M \quad (\text{Eq. 2.3})$$

Figure 2.2: A disjunctive graph for 2-job x 3-machine JSSP instance in Table 2.1.

Figure 2.3: A directed graph (feasible solution) for 2-job x 3-machine JSSP instance in Table 2.1.

where Π denotes the set of feasible solutions. This will eventually turn the disjunctive graph in Fig. 2.2 to a directed graph in Fig. 2.3.

When a feasible solution for a JSSP is obtained, critical path identification is carried out. The critical path identification will help in identifying a set of operations which cannot be delayed such that all jobs are completed on time. The sum of all the critical operations processing time will be the makespan of the solution. Fig. 2.4 shows the

Figure 2.4: Gantt chart for the directed graph in Fig. 2.3.

critical path of the schedule represented in Fig. 2.3 which consists of four operations (shaded boxes). It is composed of the following: $0 \rightarrow O_{21} \rightarrow O_{22} \rightarrow O_{23} \rightarrow O_{13} \rightarrow *$. The sum of all the operation processing time is $0 + 8 + 4 + 5 + 6 + 0 = 23$.

The critical path can be decomposed into a set of q blocks, $B = \{b_1, b_2, \dots, b_q\}$ as defined in [63, 108]. Each block contains an order set of operations which are processed on the same machine $b_i = \{\phi_{i1}, \phi_{i2}, \dots, \phi_{i|b_i|}\}$ where $\phi_{ij} \in O$ is the j -th operation of block i . Two consecutive blocks in B must contain operations which are processed on different machines. The following example illustrates the block decomposition of the critical path found in Fig. 2.4. By performing the block decomposition on the critical path, three blocks are identified where $q = 3$. Hence, $B = \{b_1, b_2, b_3\}$. The content of each block is as follows: $b_1 = \{O_{21}\}$, $b_2 = \{O_{22}\}$ and $b_3 = \{O_{23}, O_{13}\}$. The described block decomposition will be the central idea in the implementation of the neighbourhood operator which will be discussed in Section 4.4.3.3.

2.4 Benchmark Problem Instances

When a new algorithm is proposed, it is important to test the proposed algorithm on a set of benchmark problems. By doing so, the performance of the proposed algorithm can be investigated in a more objective and precise manner. A good source of benchmark problems should fulfill criteria as follows:

- (i) It is a common benchmark in a research community. In other words, the benchmark problems are utilized by most of the researchers in the community.
- (ii) It can be easily obtained, e.g. dataset of the benchmark problems is downloadable from a website and the information about its latest progress is frequently updated.

Table 2.2 shows the sources of benchmark problems that fulfill the criteria as mentioned above. Datasets from the TSPLIB, QAPLIB and OR-Library are employed to test the BCO algorithm in this thesis.

Table 2.2: Sources of the COP benchmark problems.

Domains	Sources	Descriptions
TSP	TSPLIB ¹	It is maintained by Professor Reinelt from the University Heidelberg. The TSP problem instances range in size from 14 cities up to 85900 cities. The source covers a variety of industrial applications, geographical distances and academic related problems. All 112 problem instances are solved to optimality in the year of 2007. However, this set of problem instances are still popular in the TSP research community.
TSP	GATECH ² [7]	It is maintained by Professor Cook from the School of Industrial and Systems Engineering of Georgia Tech (Georgia Institute of Technology, United States). The source covers a variety of problem instances such as the World TSPs, National TSPs and very-large-scale integration (VLSI) TSP collections.

Continued on next page...

¹<http://comopt.ifl.uni-heidelberg.de/software/TSPLIB95/index.html>

²<http://www.tsp.gatech.edu/index.html>

Table 2.2 – continued from previous page

Domains	Sources	Descriptions
QAP	QAPLIB ³ [20, 21]	The QAPLIB was initiated by Burkard et al. and it is now maintained by Peter Hahn from the School of Engineering and Applied Science, University of Pennsylvania. The source contains problem instances with the dimension ranges from 12 to 256. Out of the total of 134 instances, 31% of them are not solved to optimality as of Dec 2010. The problem instances are categorized into 15 groups and each of them corresponds to a problem which is QAP related. For example, the STE series are the instances that model the backboard wiring problem.
JSSP	OR-Library ⁴ [10, 11]	OR-Library is a collection of benchmark data sets for various Operations Research problems. It is maintained by Professor Beasley from the Department of Mathematical Sciences, Brunel University in United Kingdom. The 82 JSSP dataset are our interest as they are commonly cited in the literatures related to JSSP research. The 82 instances are categorized into six major groups as follows: 3 instances by Fisher and Thompson (FT or MT) [50], 40 instances by Lawrence (LA) [88], 5 instances by Adam et al. (ABZ) [2], 10 instances by Applegate and Cook (ORB) [6], 20 instances by Storer et al. (SWV) [134] and 4 instances by Yamada and Nakano (YN) [160]. The dimension of these 82 instances ranges from 6-job $\times$ 6-machine to 50-job $\times$ 10-machine.

Continued on next page...

³<http://www.seas.upenn.edu/qaplib/>⁴<http://people.brunel.ac.uk/mastjjb/jeb/info.html>

Table 2.2 – continued from previous page

Domains	Sources	Descriptions
JSSP	Taillard ⁵ [139]	The Taillard’s dataset is a collection of 80 JSSP with the dimension ranges from 15-job $\times$ 15-machine to 100-job $\times$ 20-machine. The library is maintained by Professor Taillard from the University of Applied Sciences of Western Switzerland. Out of the total instances, 43% of them are not solved to optimality as of Dec 2010. The problems with the label of ta11 to ta50 are difficult problems whereas the rest are relatively easy problems which have been solved to optimum. Besides JSSP, the website also hosts other benchmark instances for flow shop and open shop scheduling.

2.5 Summary

In this chapter, three COPs (i.e. TSP, QAP and JSSP) have been described in detail. Their definitions and problem formulations have been discussed. Solving a TSP is about finding a Hamiltonian tour with the minimum distance. For QAP, it is to generate an allocation plan that assigns a set of facilities to a set of locations with the minimum cost. Solving a JSSP requires the finding of a schedule of operations with the minimum makespan, such that the schedule complies with the job precedence and machine capacity constraints. TSP, QAP and JSSP share a common characteristic where they all involve the finding of solution with minimum cost and hence they are minimization problems. The difficulty of solving these three selected problems increases with the problem dimension. The problem becomes complicated when the problem dimension is large.

TSP, QAP and JSSP are selected to be the testbed for the proposed BCO algorithm in this thesis. Various sources of benchmark problems for the three selected COPs are

⁵<http://mistic.heig-vd.ch/taillard/>

presented. Out of the five sources of benchmark problems, datasets from the TSPLIB, QAPLIB and OR-Library are employed to test the proposed BCO algorithm.

In the subsequent chapter, some other works that are related to this research will be described.

Chapter 3

Related Work

Due to the NP-hard nature of the Combinatorial Optimization Problems (COPs), the COPs is still a popular problem among researchers in various disciplines such as computer science and operations research. The problem has been investigated from various perspectives. To solve a COP, generic approaches or problem specific methods can be applied. Besides, hybrid approaches which combine generic and problem specific strategies are proposed. This chapter aims to provide an overview on the methods that have been used in solving the COPs in particularly the TSP, QAP and JSSP. The reported results from the literatures will be used in our comparison study which is presented in Chapter 7.

This chapter is organized as follows. Section 3.1 first discusses the classification of different techniques developed to solve COPs. It is followed by Section 3.2 which describes a list of review and survey literatures. Sections 3.3 and 3.4 provide an overview of the exact and approximate methods that have been used to solve the focus problems. Finally, this chapter ends with a summary.

3.1 Classification of Techniques for Combinatorial Optimization Problems

This section depicts a classification of techniques that have been applied to solve COPs. This classification will serve as the basis of our discussion in the subsequent sections throughout this chapter.

Fig. 3.1 illustrates the major groups of techniques that have been used to solve the TSP, QAP and JSSP. A technique can be classified as an exact algorithm or an approximate algorithm. If a technique is classified as an exact algorithm, it utilizes mathematical model in its problem solving process. If the technique is an approximate algorithm, it employs heuristics and iterative improvements in its problem solving process. The classification is further organized such that the approximate group has three sub-clusters, namely: the constructive approaches, the local search approaches and the nature inspired approaches. The focus of the literature review will be more on the nature inspired approaches that are based on natural process metaphors such as foraging and swarming behaviours. Thus, our discussion will concentrate on the Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO) and Bee Colony Optimization (BCO).

One of the major advantages of the exact algorithms is that it guarantees global convergence. However, this guarantee is limited to small problems. For bigger problems, the exact algorithms might take a long time for the algorithm to reach optimality. On the other hand, the approximate algorithms are generally speedy, but they do not promise optimal solutions. The approximate algorithms are able to find sub-optimal solutions within polynomial computing time and reasonable resources. This feature is attractive to decision makers who want to obtain satisfactory solutions in a short period of time.

We introduce a citation notation which is used in this chapter. This notation is aimed to help readers differentiate the literatures cited in this chapter, whether they are related to the domains of TSP, QAP or JSSP. Literatures in this chapter will be cited as $[n]^a$

Figure 3.1: Classification of techniques which have been employed to solve the focus problem of this thesis.

where $a \in \{T, Q, J\}$. If a literature is cited as $[n]^T$, this indicates the mentioned literature is related to TSP. Likewise, $[n]^Q$ and $[n]^J$ denote that the cited references are related to QAP and JSSP respectively. If there is no subscript letter, it denotes that the literature content relates to the general class of problems.

3.2 Review and Survey Literature on TSP, QAP and JSSP

Table 3.1 lists the review and survey literatures over the last twenty years which are related to the focus problems of the thesis. They are sorted according to the domains and then according to the year of publication.

A wide range of discussions can be found in these literatures. They include the problem definitions and formulations, benchmark problems and their historical achievement, algorithms and approaches that have been tested on the focus problems and future trends

Table 3.1: Review and survey literatures which are related to the domains of TSP, QAP and JSSP.

Domains	Author(s)	Year	Source	Title
TSP, QAP & JSSP	Jourdan et al. [79]	2009	European Journal of Operational Research.	Hybridizing exact methods and metaheuristics: A taxonomy.
TSP, QAP & JSSP	Angel [5]	2006	Efficient Approximation and Online Algorithms, Recent Progress on Classical Combinatorial Optimization Problems and New Applications.	A survey of approximation results for local search algorithms.
TSP	Gutin [67] ^T	2007	Handbook of Graph Theory.	Traveling salesman problems.
TSP	Pasquier et al. [111] ^T	2007	Proceedings of Sixth Mexican International Conference on Artificial Intelligence - Special Session, 2007 (MICAI 2007).	A comparative study of three metaheuristics applied to the traveling salesman problem.
TSP	Applegate et al. [7] ^T	2006	The Traveling Salesman Problem: A Computational Study.	The traveling salesman problem: A computational study.
QAP	Loiola et al. [100] ^Q	2007	European Journal of Operational Research.	A survey for the quadratic assignment problem.
QAP	Singh & Sharma [132] ^Q	2006	The International Journal of Advanced Manufacturing Technology.	A review of different approaches to the facility layout problems.
JSSP	Potts & Strusevich [117] ^J	2009	Journal of the Operational Research Society.	Fifty years of scheduling: A survey of milestones.
JSSP	Gupta & Sivakumar [65] ^J	2006	International Journal of Advanced Manufacturing Technology.	Job shop scheduling techniques in semiconductor manufacturing.

or open problems which require further investigation.

3.3 Exact Algorithms

The Branch-and-Bound approach is perhaps one of the most recognized methods that is used to solve the focus problems of this thesis. The approach is to enumerate all candidate solutions systematically and at the same time, prune all non-rewarding solutions. The Branch-and-Bound implementation generally consists of three strategies as follows: branching, bounding and node selection. The branch with a node which its lower bound is larger than the current best lower bound will not be further considered and hence it will be fathomed. The following literatures employ the Branch-and-Bound technique as its problem solving process: Little et al. (1963) [99]^T, Volgenant & Jonker (1982) [150]^T, Radharamanan & Choi (1986) [122]^T, Hahn et al. (1998) [70]^Q and Brucker et al. (1994) [17]^J. Recently, the Branch-and-Bound approach is hybridized with parallel implementation so that bigger instances are attempted. This parallel Branch-and-Bound technique can be found in Tschoke et al. (1995) [145]^T, Clausen & Perregaard (1997) [28]^Q, Clausen & Perregaard (1998) [29]^J and Carrabs et al. (2007) [23]^T.

There is an alternative approach to the Branch-and-Bound in solving an integer program¹, which is named as the Cutting Plane method. The core idea behind the Cutting Plane method is to gradually introduce cuts until the problem being solved obtains integer values as its feasible solutions. By adding cuts to the formulation, the original problem is modified and hence more constraints are introduced to the formulation. Whenever a cut is introduced, a region in the solution space is pruned from further consideration. This gradually produces a smaller solution space and eventually turns the fractional solution to become an integer solution. However, due to the slow convergence and high computation overhead over iterations, the cutting plane method is appropriate only for small problems.

¹The TSP, QAP and JSSP can be modeled as an integer program.

To overcome the limitation of the Cutting Plane method and to expedite the convergence towards optimum solution, a hybrid of the Branch-and-Bound and Cutting Plane method was proposed, namely: the Branch-and-Cut. The Branch-and-Cut approach works as follows. When fathoming cannot be performed in any Branch-and-Bound execution after solving the Linear Programming relaxation, cuts are introduced to the formulation as additional constraints. With this, a portion of the solution space is pruned. If such cuts cannot be found, branching is performed. These steps are continued until a feasible integer solution is obtained. The Branch-and-Cut method converges faster than the standalone Branch-and-Bound or Cutting Plane method. In the domain of TSP, the most successful application of the Branch-and-Cut approach is the Concorde TSP Solver² [7]^T. The Concorde TSP Solver is currently the state-of-the-art of the exact algorithm implementation in solving TSP. In the domains of QAP and JSSP, related details can be found in [48]^Q and [129]^J.

3.4 Approximate Algorithms

This section describes the approximate algorithms that have been used to solve the focus problems of the thesis. This set of algorithms will be discussed from three different perspectives, according to the nature of the algorithms. Constructive approaches is first discussed, followed by improvement methods and nature inspired methods.

3.4.1 Constructive Approaches

A constructive approach is a technique that builds a solution from scratch. In the case of constructing a permutation of nodes as one of the feasible solutions, a constructive approach starts with adding a node or a chunk of nodes at the initial stage. At each

²<http://www.tsp.gatech.edu/concorde/index.html>

iteration of the algorithm, the node(s) is/are added by means of problem dependent heuristics. The construction process terminates when a feasible solution is acquired.

In the domain of TSP, the Nearest Neighbourhood (NN) heuristic is one of the constructive approaches. The NN heuristic starts with a random city. To move ahead, it then picks a city which is not yet part of the solution and is the nearest city from the last city added. If there is a tie in distances, random selection is applied. These processes are repeated until a feasible solution is obtained. The NN heuristic is simple to implement. However, the size of the TSP being solved dictates the performance of the NN heuristic. If x denotes the dimension of a TSP being solved, the NN heuristic produces a tour length that is a factor $\Omega(\log_2 x)$ above the length of the optimal tour³ [74, 127, 141]^T.

The insertion heuristic is also considered as a constructive approach for TSP. The basic idea of an insertion heuristic is to add a vertex which is not yet part of the solution to the current tour until a complete solution is acquired. An insertion heuristic may start with a single vertex or a subset of all vertices. The initial sub-tour can be the convex hull of the TSP. There are a variety of insertion heuristics. They include the Closest Point Insertion, the Farthest Point Insertion, the Cheapest Insertion and etc. Generally, the performance ratio of the Insertion Heuristic is a factor of $\Omega(\log_2 x)$ where x is the TSP dimension⁴. However, the performance ratio of the Closest Point Insertion and the Cheapest Insertion heuristics can be reduced to a constant of 2, which is to say that the yielded tour length of the mentioned insertion heuristics cannot be more than twice the optimum tour length, regardless of the TSP dimension [127]^T. The application of the Insertion heuristics on the TSP can be found in the following literatures [56, 57, 106]^T.

In the domain of QAP, as the nature of the problem involves two entities, which are facilities and locations, the heuristic used in the solution construction is more complex. In

³The performance of the NN heuristic is based on the following assumptions: i. The lengths among all cities are non-negative. ii. The TSP being solved is a symmetrical TSP instance. iii. The triangle inequality is complied with.

⁴Assumptions for the NN heuristic apply here.

1962, Gilmore proposed a permutation production method which involves the allocation of facilities on a set of locations [60]^Q. Since then, various constructive heuristics were proposed including the heuristics formulated by Gutin & Yeo (2002) [68]^Q and Yu & Sarker (2003) [161]^Q.

In the domain of JSSP, various priority dispatching rules were proposed in the early days. The idea of these dispatching rules is to prioritize all the operations which have not been allocated as part of the schedule, and the operation with the highest priority will be picked one by one until a complete feasible solution is acquired. The priority heuristics range from first-in first-out, last-in first-out, longest processing time, shortest processing time, least work remaining, most work remaining to work in the next queue. The following literatures provide an overview on these dispatching rules: [15, 30, 71, 110]^J. In order to obtain better results, these dispatching rules can be linearly or arbitrarily combined [88, 110]^J.

Another constructive algorithm in the domain of JSSP is the Shifting Bottleneck Procedure (SBP) proposed by Adam et al. [2]^J. The SBP consists of different phases as follows: identify sub-problem, select bottleneck, examine sub-problem solution and re-optimize schedule [37]^J. The basic idea of the SBP is to split the actual problem into multiple one machine problems and solve each sub-problem at one time by using the Carrier's approach [22]^J. Each of the sub-problem solution is examined in order to identify the bottleneck machine. Re-optimization is performed whenever the bottleneck machine is scheduled. Some innovative works that are related to the SBP are as follows: Balas & Vazacopoulos (1998) [8]^J, Wenqi & Aihua (2004) [153]^J.

3.4.2 Improvement Approaches

This section describes some useful improvement heuristics for TSP, QAP and JSSP. The improvement heuristics are also named as local search algorithms. An improvement

heuristic starts with a feasible solution and let's assume this feasible solution is labelled as R . The heuristic will try to improve R by looking for other solutions around its neighbourhood. If a neighbourhood solution (R') is found and its cost is better than R , R will be replaced by R' . The process is continued until no further improvement is obtained. At this stage, R is said to be locally optimum. Algorithm 1 illustrates the basic steps of an improvement heuristic.

Algorithm 1 General steps of an improvement heuristic.

```

procedure Improvement_Heuristic
  while stop criteria are not fulfilled do
    Generate a feasible solution,  $R$ 
    while  $\neg R.isLocallyOptimum()$  do
       $R' \leftarrow R.improve()$ 
      if  $R'.cost < R.cost$  then
         $R \leftarrow R'$ 
      end if
    end while
  end while
end procedure Improvement_Heuristic

```

One of the improvement heuristics is the Adjacent Pairwise Exchange (APE) method. The basic idea of the APE method is to exchange two adjacent relevant entities in a feasible solution. Let's consider a feasible solution⁵ 1-5-3-4-2 for a problem with dimension $x = 5$, the APE method exchanges the following adjacent pairs: $\{[1, 5], [5, 3], [3, 4], [4, 2]\}$. The APE method can be implemented such that the operation is halted whenever the exchange generates a better solution compared to the original solution. Otherwise, a thorough examination on all the $(x - 1)$ pairs is performed in order to allocate the best neighbourhood solution.

The 2-opt approach is another improvement heuristic which was originally proposed by Croes [33]^T to solve the TSP. Croes's idea was to eliminate crossings found in a tour

⁵The solution of 1-5-3-4-2 is feasible for all the three selected problems in the thesis where each number in the solution represents a city (TSP), a facility (QAP) and an operation (JSSP).

and to re-order the cities involved in the crossings so that a better solution was expected. This idea is then extended such that an elimination of two edges is performed in order to acquire two different paths. These two resulting paths will be re-connected in the other possible way to get a neighbourhood solution. This elimination-and-reconnection procedure is reiterated for all the possible pairs of edges. If a feasible solution of a 5-city TSP is as follows: 1-5-3-4-2-1, the following $\frac{x(x-1)}{2} = \frac{5(5-1)}{2} = 10$ pairs of edges will be considered for exchange: $\{[1-5, 5-3], [1-5, 3-4], [1-5, 4-2], [1-5, 2-1], [5-3, 3-4], [5-3, 4-2], [5-3, 2-1], [3-4, 4-2], [3-4, 2-1], [4-2, 2-1]\}$. Thus, the time complexity of this exhaustive 2-opt approach is $\Omega(x^2)$ [78]^T. The 2-opt approach can also be employed in the domain of QAP as explained in [135]^Q where its core operation is to exchange two facilities r and s at locations L_r and L_s iteratively. This method is named as Iterated Local Search (ILS). The performance of the ILS approach on the benchmark instances is promising as shown in [101, 123]^Q.

In 1973, Lin & Kernighan [97]^T proposed an effective improvement heuristic for symmetric TSP, namely: the LIN-KER heuristic. The idea is similar to the 2-opt approach. Nevertheless, instead of exchanging two edges at one time as what is done by the 2-opt approach, the LIN-KER heuristic attempts to allocate sequentially two sets of edges $X = \{x_1, \dots, x_k\}$, $Y = \{y_1, \dots, y_k\}$ such that the edges in X are deleted and replaced by the edges in Y . This approach is also named as variable k -opt which involves selecting the most suitable k edges in its elimination-and-reconnection process. Although the complexity of the LIN-KER heuristic is approximately $\Omega(x^{2.2})$, making it slower than the exhaustive 2-opt with complexity of $\Omega(x^2)$, it is still one of the best performing improvement heuristics for the symmetric TSP [7]^T.

In the domain of JSSP, the improvement heuristics are all based on a certain neighbourhood structure. The neighbourhood structure is exploited so that a new feasible solution is acquired. Blazewicz et al. (1996) [16]^J provided a detailed description on the

Table 3.2: JSSP neighbourhood structures.

Neighbourhood	Initiator(s)	Year
N1	van Laarhoven et al. [148] ^J	1992
N2	Matsuo et al. [105] ^J	1988
N3 & N4	Dell’Amico & Trubian [35] ^T	1993
N5	Nowicki & Smutnicki [108] ^J	1996
N6	Balas & Vazacopoulos [8] ^J	1998

commonly used neighbourhood structures and labeled them as N1, N2, N3, N4, N5 and N6 as shown in Table 3.2. The N5 neighbourhood will be our focus here as it is employed as the local search approach for the generic BCO framework to solve the JSSP (refer to Section 4.4.3.3). The N5 neighbourhood is defined as the interchange of two successive operations in a block structure that belongs to a critical path. The interchanges are done on the first two or the last two operations in any intermediate blocks. However, only the last two operations in the first block and the first two operations in the last block involve in the interchange. The N5 neighbourhood has the following properties: the interchange of two adjacent critical operations cannot produce an infeasible solution; the interchange of any non-critical operations cannot reduce the makespan and may produce an infeasible solution. Nowicki & Smutnicki [108, 109]^J proposed the N5 neighbourhood and used it as part of the Tabu Search (TS) method. This technique remains as one of the most efficient approaches to solve the JSSP.

3.4.3 Nature Inspired Approaches

The heuristics discussed in Sections 3.4.1 and 3.4.2 are mainly specific and custom-designed for a given problem. This section describes another COPs solving paradigm, namely: the metaheuristic. A metaheuristic is a general purpose algorithm which is able to solve a given COP. Some of the metaheuristics are based on natural metaphors and some are based on theoretical and experimental considerations [100]. The metaheuristics that are based on natural metaphors include Simulated Annealing (SA), Ant Colony

Optimization (ACO), Particle Swarm Optimization (PSO), Bee Colony Optimization (BCO), Genetic Algorithms (GA) and etc. As for the metaheuristics which are based on theoretical and experimental considerations, they include Tabu Search (TS), Greedy Randomized Adaptive Search Procedure (GRASP), Iterated Local Search (ILS), Memetic Algorithm (MA) and etc. More recently, these metaheuristics are hybridized with other approaches. One of the hybridization choices is as follows. A metaheuristic is initialized with a constructive method and is locally optimized by a local search technique. Constructive methods and local search techniques have been discussed in Sections 3.4.1 and 3.4.2 respectively. Although the hybridization does not guarantee optimum solutions, its fast performance to obtain relatively good feasible solutions makes this method an attractive alternative among practitioners.

In the subsequent section, three nature inspired approaches will be described, namely: the ACO, PSO and BCO.

3.4.3.1 Ant Colony Optimization (ACO)

This section describes the ACO algorithm. Ants are highly organized social insects. The survival of the entire ant colony depends on every individual unit. Each ant plays an important role in order to guarantee the continued existence of its colony.

Many of the ants' behaviours are unique and self-organized. One of them is the food foraging practice. The ability of ants to find the shortest path from their colony to a food source has motivated ethologists to study ants' behaviours in great detail. It is recognized that ants deposit chemical substance known as pheromone along the way from their colony to food source (and from the food source back to the colony). This pheromone trails are useful to ants and they are used as a communication medium among ants [38, 39].

Consider an example on how ants utilize the pheromone trails to communicate with other ants. Supposed two unexplored paths are established, linking an ant colony and

a food source. The paths are arranged such that one path is longer than the other one. Ants will then be following these paths to forage for food. At the initial stages, ants will choose one of the paths randomly (as the paths are unexplored and hence ants have no pheromone indication). As a result, the number of ants in both paths appear to be similar. Over a period of time, because ants take shorter time to make a round trip between the colony and the food source, a higher concentration of pheromone will be accumulated along the shorter path. This eventually causes the shorter path to be favoured as ants tend to follow the trail with a higher concentration of pheromone. Via the pheromone secretion mechanism, an ant is *notified* by other ants about the choices of which path should be taken. An indirect communication has taken place among ants via the deposit of pheromone. Further explanation on this mechanism can be found in [40, 42].

The unique foraging behaviours of ants have been used to solve COPs including TSP. Among the earliest attempts on TSP were [52, 136, 137]^T (as shown in Table 3.3).

A Backtracking Ant System (BAS) was introduced by Al-Shihabi [4]^T where the backtracking mechanism is built on top of the Max-Min Ant System (MMAS) [136, 137]^T. Two types of ants are introduced: engineering and scout ants. The role of engineering ants is to follow routes that are built previously and guide other ants towards the destination via pheromone secretion. In contrast, the role of scout ants is to follow partially the route shown by the engineering ants and discover the rest of the route by themselves. If the scout ants found a better route than the current route, the algorithm will backtrack to a larger sub-region that encloses the new route. The main idea of backtracking is to make sure that ants move from one sub-region to another sub-region rather than moving between solution points.

Le Louarn et al. proposed an ACS with Generalized Insertion heuristic (GACS) for TSP [89]^T. The Generalized Insertion heuristic is an iterative insertion heuristic

proposed by Gendreau et al. [56]^T. In this heuristic, a new node u is inserted between two nodes i and j that are not necessarily consecutive in the current tour. However, node u must be adjacent to nodes i and j after the insertion. The insertion heuristic has a limitation where it will always look for the nearest node in its insertion operations. To avoid this, a state transition rule was proposed by the authors with the aim to increase diversification. This allows the heuristic to have more nodes to choose from during its insertion operations. The difference between the GACS and an ordinary ACS is in their path construction mechanism where the GACS utilizes an insertion heuristic in its path construction and the ACS employs a state transition rule.

Table 3.3: The Ant Colony Optimization (ACO) algorithm and its variants on TSP, QAP and JSSP.

Approaches	Abbrev.	Year	Initiator(s)	Description
Ant Colony System	ACS [52] ^T	1996	Gambardella & Dorigo	It is built on top of Ant System (AS) and AS with Q-Learning(Ant-Q). Ants build path with state transition rule (heuristic distance & pheromone trails). Various local updating rules are reported.
MAX-MIN Ant System	MMAS [136, 137] ^T	1997	Stützle & Hoos	It modifies the AS model. The pheromone level on trails are limited within [MAX, MIN] level to offer more control on the trails strength.
Backtracking Ant System	BAS [4] ^T	2004	Al-Shihabi	Backtracking mechanism is built on top of MMAS. Scout and engineer ants are established. Scout ants perform backtracking to jump from a region to another region in the search space.
ACS with Generalized Insertion	GACS [89] ^T	2004	Le Louarn et al.	The 2-opt and the Generalized Insertion heuristic are integrated. It alters the Generalized Insertion heuristic by adding a state transition rule to increase the diversity.

Continued on next page...

Table 3.3 – continued from previous page

Approaches	Abbrev.	Year	Initiator(s)	Description
Two-stage ACS	TSACS [119] ^T	2007	Puris et al.	The aim is to expedite ACO operations in its path construction mechanism. The search process is divided into two stages where the first stage outcomes are used by ants in the second stage. The results show that the computational time is reduced by the two-stage approach. The 2-opt is integrated in the proposed method.
ACS with Randomized Algorithm	RAACS [120] ^T	2007	Qi	Similar to [119] ^T , it is built with the aim to reduce the computational time. A randomized algorithm is used to specify a partial neighbourhood in the state transition module.
ACO with Genetic Algorithm	ACO+GA [164] ^T	2008	Zhao et al.	The ACO algorithm is employed to produce a pool of solutions that constitute the initial population of the GA. The GA will then optimize the solutions. The GA crossover operator is altered. It becomes a greedy operator such that it considers three factors in its offspring generation.
Hybrid ACO with a local search	HAS-QAP [53] ^Q	1999	Gambardella et al.	The ACO algorithm is coupled with a local search where the local search is driven by pheromone trail information. The HAS-QAP performs best for the real world, irregular and structured problems.
Hybrid ACO with the Simulated Annealing	AntSimulated [36] ^Q	2006	Demirel & Toksari	The ACO algorithm is hybridized with the Simulated Annealing. A different exploitation mechanism is employed such that it considers the best three solutions found so far in generating new neighbourhood solutions. The performance is better than the HAS-QAP algorithm.

Continued on next page...

Table 3.3 – continued from previous page

Approaches	Abbrev.	Year	Initiator(s)	Description
Population-based Hybrid Ant System	PHAS [124] ^Q	2009	Ramkumar et al.	A population of ants (which is set at twice of the problem dimension) are initialized with random solutions. The initial solution attached to each ant is locally optimized by the 2-opt with the first improving jump strategy. The results show that by applying the defined local search on the initial solutions, the PHAS algorithm is able to obtain an average of 0.42% from known optimum for a total of 109 QAPLIB benchmark instances.
ACO with Fast Taboo	ACOFT [73] ^J	2008	Huang & Liao	The Fast Taboo algorithm [108] ^J is incorporated in the ACO framework. The algorithm applies a shifting bottleneck liked pheromone trail mechanism and an insertion algorithm as the constructive algorithm. The solutions are then locally optimized by the Fast Taboo algorithm.
Hybrid of Tabu Search and ACO	HTSACO [49] ^J	2009	Eswaramurthy & Tamilarasi	The HTSACO applies a dynamic tabu length and uses the ACO algorithm to pick neighbourhood solution in the Tabu Search operations. These two strategies ensure that the system would not be trapped in local optima and prevent cycle in its solution construction operation.
ACO with parameterized search space	ACOPSS [130] ^J	2010	Seo & Kim	The ACO algorithm is combined with the algorithm proposed by Storer et al. [134] ^J . The ACOPSS algorithm aims to search for optimum solutions in a smaller search space which contains only active schedules.

Another hybridization technique for TSP, namely: the ACO with embedded Genetic Algorithm (ACO+GA) [164]^T, was proposed by Zhao et al. Within the proposed algorithm, the ACO algorithm is used to construct solutions that comprised initial population for the GA. Next, the GA plays its role to simulate the consultation mechanism among ants. In other words, a “*discussion*” (simulated by the evolutionary operations) is cre-

ated to exchange information among ants with the intention to optimize the solutions. A greedy heuristic crossover operator that utilizes adjacency information, distances and pheromone trails between cities was proposed to generate offspring.

The literatures discussed so far mainly focuses on how to acquire better solutions (approaching optimality) for TSP. The issue of algorithm complexity (computational speed) was not emphasized in most of the approaches. Two ACO related approaches that are able to expedite the computation [119, 120]^T are introduced here. The first approach is the Two-stage Ant Colony System (TSACS) proposed by Puris et al. [119]^T. From the experimental results, the authors found that most of the computational time of a typical ACS was occupied by the state transition rule. To reduce the time occupied by the state transition rule, Puris et al. divided the search process into two different stages. The outcomes from the first stage will be used by ants in the second stage. In the first stage, only a subset of the cities will be considered by a certain number of ants. In the second stage, the remaining ants will then continue to search for a complete path. Their results show that the computational time is reduced by 40-60% when compared to an ordinary ACS. Similar idea was utilized by Qi [120]^T and the approach was named as the Ant Colony System with Randomized Algorithm (RAACS). With the Randomized Algorithm, the state transition rule considers only a certain percentage of cities rather than the entire neighbourhood.

Table 3.3 shows a list of examples of the application of the ACO algorithm on QAP. Gambardella et al. proposed the HAS-QAP in solving the QAP [53]^Q. It is a method that combines the ACO algorithm with a local search where the local search is directed by the pheromone trail information. The HAS-QAP was tested on the QAPLIB benchmark problem instances which are categorized into four different categories as follows [123, 135]: (i) random and unstructured instances; (ii) real-life instances; (iii) grid-based distance matrix instances; and (iv) real-life like instances . The results show that its performance

is good when the given benchmark QAP is of categories (ii), (iii) and (iv). It is less competitive for the benchmark problems in category (i).

Demirel & Toksari proposed the AntSimulated algorithm to solve the QAP [36]^Q. In their proposed method, the ACO algorithm is employed together with the Simulated Annealing as the local search technique. Besides the ordinary pheromone evaporation mechanism in the ACO algorithm, the AntSimulated algorithm is modified such that the evaporation mechanism considers the best three solutions acquired so far in its pheromone trails update. This new function is able to allow the AntSimulated algorithm to exploit the neighbourhood so that better solutions can be found. The results show that the AntSimulated algorithm performs better than the HAS-QAP [53]^Q.

Ramkumar et al. proposed a population-based hybrid ant system (PHAS) for QAP. The major finding of this work is as follows. When a population of ants is initialized with solutions resulted from applying the 2-opt local search with first improving jump strategy, the PHAS algorithm solves 73% of 110 selected QAP benchmark instances to optimum. The reported results show that the size of the population should be set at two times of the problem dimension, that is, if a given QAP has x dimension, $2x$ ants will be created.

In the domain of JSSP, the literatures [49, 73, 130]^J listed in Table 3.3 are described here. Huang & Liao proposed the ACO with Fast Taboo (ACOF_T) and applied it on benchmark instances from the OR-Library (82 datasets) as well as the Taillard datasets. A specific pheromone trail definition inspired from the Shifting Bottleneck and an insertion greedy heuristic are incorporated in the ACOF_T algorithm. These two approaches generate a feasible solution and the solution will be locally optimized by the Fast Taboo algorithm. The ACOF_T has advanced the best upper bound of one open benchmark instance from Taillard datasets, namely: the TA41.

Eswaramurthy & Tamilarasi proposed a hybrid variant of the Tabu Search and ACO algorithm (HTSACO) for JSSP [49]^J. It is similar to the work proposed by Huang &

Liao [73]^J. The major difference between the two algorithms is as follows. The ACOFT algorithm generates solutions with the ACO algorithm and locally optimizes them with the Fast Taboo algorithm whereas the HTSACO algorithm uses the ACO algorithm to select neighbourhood solution in the Tabu Search process. In other words, the core mechanism of the HTSACO is the Tabu Search and the ACO algorithm is used to assist in picking neighbourhood solution. The HTSACO algorithm also applies a dynamic tabu length strategy in its solution construction. It was tested on benchmark instances from the OR-Library and Taillard datasets. The results show that it is comparable to the tabu search method guided by shifting bottleneck [112].

Lastly, the work proposed by Seo & Kim is highlighted here. They proposed an algorithm (ACOPSS) that utilizes the ACO algorithm combined with an algorithm that is able to reduce the search space for a set of active schedules [134]^J. Their aim was to let the ACO algorithm perform the search in a smaller search space which contains only active schedules. The ACOPSS algorithm was tested on 20 benchmark problems (LA01-LA20). The results show an average of 0.93% from known optimum.

3.4.3.2 Particle Swarm Optimization (PSO)

PSO is a computational technique inspired from behaviour of fish schools, bird flocks, and swarming theory. Fish schools come in many different shapes and sizes such as stationary swarms, predator avoiding vacuoles and flash expansions. So do the bird flocks. Examples include the V-shaped patterns of migrating geese and seagulls gathering over a harbour. In [31], Couzin explained that regardless of the variety of shapes and motions in these collective behaviours and patterns, they are generated by two rules that are followed by each individual in a population as follows: (i) keeping a minimum distance from nearby neighbours; and (ii) adapting the same direction and aligning with nearby neighbours. These rules (also known as self-propelled particle models) have been applied

in many computer-based simulations, as an attempt to have a better understanding on the interactions between group members in a swarm.

Among the earliest PSO framework was the one proposed by Kennedy [84, 85], where the PSO was used to optimize a set of continuous nonlinear functions. The fundamental of the suggested framework is positioned on the simulation of the movement effects of social creatures (e.g. bird flocks). Each particle will perform searching in the entire search space, guided by its previous best position (*pbest*) and the best position of the swarm (*gbest*). The searching is assisted by two operators which are defined via a position equation and a velocity equation.

The PSO algorithm has been widely applied in optimizing many continuous problems. Besides PSO's applicability in solving continuous problems, it can be modified and adapted to tackle discrete problems such as the TSP, QAP and JSSP. Table 3.4 lists the works within these domains which adapt the PSO concept to solve COPs.

Table 3.4: The Particle Swarm Optimization (PSO) algorithm and its variants on TSP, QAP and JSSP.

Approaches	Abbrev.	Year	Initiator(s)	Description
PSO with local search	PSO+LS [61] ^T	2006	Goldbarg et al.	Velocity equation is replaced by three local search techniques. Lin-Ker heuristic is integrated.
Hybrid Discrete PSO	HDPSO [95] ^T	2006	Li et al.	A binary matrix and integration of tour construction mechanism are proposed to replace the encoding scheme and velocity equation. 2-opt & 3-opt are integrated.
PSO with Mutation Factor	C3DPSO [165] ^T	2007	Zhong et al.	Generate permutations on a set of edges rather than on numbers. A mutation operator, namely: C3, is utilized to keep a balance trade-off between global search versus local search.
PSO with forma analysis	PSOFA [62] ^Q	2009	Gong & Tuson	The PSO operator is generalized using a forma analysis approach. The derived operator can be applied to a problem which the solutions are defined in a discrete space.

Continued on next page...

Table 3.4 – continued from previous page

Approaches	Abbrev.	Year	Initiator(s)	Description
Hybrid PSO	HPSO [131] ^J	2006	Sha & Hsu	A preference list-based representation, a swap operator, and a tabu list are employed to model the particle position, movement and velocity respectively. The Tabu Search algorithm is applied to further improve the solution quality.
PSO with Simulated Annealing	MPSO [98] ^J	2010	Lin et al.	A random-key encoding scheme and a multi-type individual enhancement scheme are incorporated into the PSO algorithm (MPSO). The random-key encoding scheme is used to perform the continuous-to-discrete mapping. The multi-type individual enhancement scheme is based on the Simulated Annealing algorithm, which acts as a local optimizer.

In solving a TSP, among the attempts are as follows. Goldberg et al. proposed a PSO model with a local search (PSO+LS) [61]^T where modifications were made on the Kennedy’s PSO framework in which the movement option for each particle in a swarm is determined by a velocity equation:

- (i) To follow in its own way.
- (ii) To go back to its best previous position (*pbest*).
- (iii) To go towards the global best solution (*gbest*).

In the PSO+LS [61]^T model, the modifications are as follows. The first option is replaced by two local search techniques: simple inversion and Lin-Ker heuristic. The second and third options are replaced by a path re-linking operator. Although local search techniques are efficient in locating good quality solutions, it incurs a high overhead. To achieve a better trade-off, the simple inversion and the Lin-Ker heuristic have a higher occurrence rate than the path re-linking operator. Only at the later stages of the model execution are the path re-linking operator promoted so that the search is intensified in good regions.

A Hybrid Discrete Particle Swarm Optimization with local search (HDPSO) was proposed in [95]^T. Unlike the classical PSO algorithm, instead of real number matrix, a binary matrix is employed as its encoding scheme for particles. The velocity equation is enriched by a tour construction mechanism which is similar to the ACO algorithm. Local search techniques such as the 2-opt and 3-opt are integrated in the model. To avoid expensive computational cost incurred by the 2-opt and 3-opt, a gradually increment strategy was proposed. The strategy shares a certain level of similarity with the strategy described in [61]^T. The aim of the strategy is to ensure that more and more particles perform local search over the generations such that the search is amplified in promising regions.

Another endeavor to apply the PSO algorithm on TSP, namely: the Discrete PSO with C3 parameter (C3DPSO) was proposed by Zhong et al. [165]^T. In this method, the particles are prohibited from generating permutations of numbers. Instead, particles generate permutations of a set of edges. A mutation parameter (C3) is utilized in their model to keep a balance trade-off between exploration and exploitation (global search versus local search). The role of C3 is to contribute some new edges into the solutions generated by PSO operators, according to a pre-defined likelihood. It is a parameter that encourages exploration.

Gong & Tuson proposed a PSO approach to solve the QAP [62]^Q as listed in Table 3.4. The PSO operator has been generalized in a formal manner to discrete domain using forma analysis. The aim is to make sure that the PSO features which is suitable for a continuous space can be generalized to solve a problem in a discrete space (in this case, the QAP). The derived operator using the forma analysis was compared against the Genetic Algorithms. The reported results show that they are both comparable.

As in the domain of JSSP, Table 3.4 indexes two related literatures which use the PSO as the core process. Sha & Hsu proposed a hybrid PSO model (HPSO) to solve

JSSP [131]^J. Since JSSP is a discrete problem, they modified the particle position representation, particle movement and particle velocity as follows. A preference list-based representation, a swap operator, and a tabu list are employed to model the particle position, movement and velocity respectively. The computational results show that it is better than other traditional metaheuristics such as the Shifting Bottleneck, Tabu Search and Genetic Algorithms.

Lin et al. integrated a random-key encoding scheme and a multi-type individual enhancement scheme into the PSO algorithm (MPSO) to solve the JSSP [98]^J. The random-key encoding scheme is used to map a position in a continuous space to a position in a discrete space. The multi-type individual enhancement scheme is based on the Simulated Annealing algorithm which is used to perform local search to allocate better neighbourhood solutions.

3.4.3.3 Bee Colony Optimization (BCO)

In this section, the BCO algorithm and its variants will be discussed. Firstly, an introduction of the behavioural characteristics of bee colony related algorithms will be given. Some domain examples that have been attempted by using the BCO variants are presented. This is then followed by a discussion of the application of the BCO algorithm on the focus problems of this thesis.

Like ants, bees are self-organized social insects. They live in groups and each member in the colony has its own role, e.g. queen bee - to reproduce, worker bees - to forage etc. Some unique behaviours practised by bees have been adapted algorithmically to solve complex problems. Among them are *foraging behaviours*, *reproduction behaviours* and *queen bee concept* [9]. The subsequent discussions in this section will be based on these three behavioural characteristics.

The foraging behaviour in a bee colony remains mysterious for many years until von Frisch translated the language embedded in bee waggle dances [151]. Waggle dance

operates as a communication tool among bees. Suppose a bee found a rich food source. Upon its return to the hive, it starts to dance in a figure-eight pattern. This figure-eight dance consists of a straight waggle run followed by a turn to the right back to the starting point, and then another straight waggle run followed by a turn to the left and back to the starting point again. The bee usually repeats these for a few times. Remarkably, via this informative dance, the bee has actually informed its hive mates about the direction and distance of the food source. The direction is expressed via the angle of dance relative to the sun position whereas the distance is expressed via the length of the straight waggle run. This coding and decoding process will eventually bring more bees towards the new food discovery.

In [9], it was described that the success of foraging behaviours very much depends on the different types of bees in a bee colony such as *unemployed foragers* (scout and recruited bees), *employed foragers* and *experienced foragers*. They all play different roles to ensure an undisrupted food supply to the colony. Foragers will either explore new food sources or attract more bees towards a good food source. Bees also perform reinforcement on certain paths to food sources. This is done via a frequent visit to a food source without sharing the food source information with their hive mates. Further details about waggle dance can be found in [14, 43].

Lemmens et al. developed a bee colony foraging simulation model [92], namely: BEEHAVE, to investigate and compare the usefulness of the bee foraging model against the ACO algorithm. Their results show that the bee inspired algorithm is significantly more effective and more scalable than the ant inspired algorithm in allocating food resources. However, in terms of adaptability, the ant inspired algorithm outperforms the bee inspired algorithm. To make it more adaptive, Lemmens et al. proposed another model, which includes the use of inhibition pheromones [91]. By integrating inhibition pheromones into the model, [91] unites the best features of both ant inspired and bee inspired algorithms.

Pham et al. proposed a population-based search algorithm called the Bees Algorithm [113]. This algorithm is also based on the food foraging behavior of honey bees. Various engineering problems have been attempted with the proposed Bees Algorithms. They include environmental or economic power dispatch [115], optimization on printed circuit board assembly [116] and fuzzy logic controller [114].

Various bee related models that employ *foraging behaviours* have been attempted in different domains ranging from dynamic server allocation for Internet hosting center [107], numerical function optimization [81, 82], telecommunication network routing [152] to stochastic vehicle routing problem [104, 142, 144].

Another behaviour of bees that has been adapted algorithmically as a problem solving tool is the reproduction process of a queen bee. A typical bee colony consists of queen(s), drones, worker bees and broods. The queen bee acts as the mother of all bees in a hive. Its major responsibility is to mate with drones and to reproduce bees. The egg which turns into a queen has no difference compared to other eggs which turn into worker bees. The queen is developed from larvae selected by worker bees and exclusively fed with lots of royal jelly. With extra nutrient, the larvae develop differently and become a queen bee after a period of time.

When a virgin queen is ready to reproduce, she would fly out from her hive to an area with lots of drones. The area will be a place for her to mate with drones. After mating with a drone, she stores the sperms in her spermatheca (an organ in the reproductive tracts of female insects which is used to store sperms from male). The mating process usually continues for a few days, depending on the weather condition. In other words, a queen would make a few mating flights out of her hive and would mate with multiple drones until she is fully mated. The accumulated sperms in the queen's spermatheca form a genetic pool of the colony. The sperms will be utilized whenever the queen lays eggs.

This unique behaviour had been adapted by Abbass [1] to solve the 3-SAT problem. The approach is named as Marriage Bee Optimization (MBO). Similar to the natural metaphor of the reproduction process, the mating flight of a queen bee is illustrated as a process to find solutions in a solution space. Firstly, the queen is equipped with a solution. This solution can be viewed as queen's genotype. The queen will mate with a set of drones and different sperms (different partial solutions) will be collected in her spermatheca. When she reaches her hive, she lays eggs and fertilizes them with the accumulated sperms. The "*fertilization of eggs*" is implemented as a series of crossover and mutation processes. Broods are created by these reproduction processes and they are improved by a certain number of heuristics encoded in the program. If a brood appears to be fitter than the queen, the brood replaces the queen and the remaining broods would be demolished. After that, another mating flight takes place until a set of pre-defined criteria is fulfilled.

Besides the 3-SAT problem, the framework inspired by the queen bee *reproduction behaviour* has been attempted in many other area such as water resources management [69], continuous optimization [3] and data mining [12].

Utilization and modification of *queen bee mating patterns* in solving complex problems is noted in some literatures. The main objective of these works is to advance the capability of the GA. In [138], a queen bee mating pattern replaces the roulette wheel selection in the GA. In [83], several crossover patterns were proposed. When a crossover occurs, one of the parents is considered as a queen bee and the other one is considered as a drone. The idea of utilizing the queen bee mating patterns has also been tested in the hex game playing program [149] and the power dispatch problem [121].

Table 3.5 lists review and survey literatures which are related to the BCO algorithm and its variants to help readers to have a better understanding on the BCO algorithm.

Table 3.5: Review and survey literatures which are related to the BCO algorithm and its variants.

Author(s)	Year	Title
Teodorović [142]	2008	Swarm intelligence systems for transportation engineering: Principles and applications.
Karaboga & Akay [80]	2009	A survey: Algorithms simulating bee swarm intelligence.
Teodorović [143]	2009	Bee Colony Optimization (BCO).

The discussions on the literatures that employ the BCO algorithm and its variants in solving the three selected problems (refer to Table 3.6) are presented here. The algorithms listed in the table are mainly based on the bee foraging behaviours.

Table 3.6: The Bee Colony Optimization (BCO) algorithm and its variants on TSP, QAP and JSSP.

Approaches	Abbrev.	Year	Initiator(s)	Description
Bee System	BS [102, 103] ^T	2003	Lučić & Teodorović	Inspired by the bee foraging behaviour. Interaction among bees is implemented via memory and waggle dance. 2-opt & 3-opt are integrated. PR1002: 3.19/3.82 %
BCO	BCO [154] ^T	2008	Wong et al.	Inspired by the bee foraging behaviour. Bees use a state transition equation to construct complete routes. Integration of preferred path concept in the transition equation. Waggle dance duration is implemented via a linear function. Bees decide to follow a dance based on a lookup table. LIN318: 1.05/1.54 %
BCO with 2-opt	BCO+2-opt [155] ^T	2008	Wong et al.	Extension of [154] ^T . The 2-opt technique is incorporated as the local optimizer. LIN318: 0.00/0.09 %
BCO with N5 neighbourhood search	BCNS [26, 27] ^J	2007	Chong et al.	The N5 neighbourhood search is integrated into the BCO algorithm. A set of JSSP solutions are created by using dispatching rules at the initial stage. These solutions are further optimized by the BCO algorithm and the neighbourhood search. LIN318: 0.00/0.09 %

Continued on next page...

Table 3.6 – continued from previous page

Approaches	Abbrev.	Year	Initiator(s)	Description
BCO with big valley landscape exploitation	BCBV [157] ^J	2010	Wong et al.	Besides the effect of the BCO algorithm, the searching is intensified in the area where big valley is encountered. LIN318: 0.00/0.09 %

Lučić & Teodorović proposed a Bee System (BS) to solve TSP [102, 103]^T. The BS algorithm is inspired by the bee foraging behaviour where the waggle dance is the main characteristic to realize computationally. In their model, bees are used to construct partial tours in every iteration. The position of the hive will be reallocated randomly to a new city after every iteration. The interaction among bees in their model is implemented in two ways: memory and waggle dance. A bee is given an ability to memorize the number of visits by its other hive mates on certain links in the past. A bee is also able to advertise its partial route to other bees. When a bee finishes building a partial path, it flies back to its hive and participate in the following decision making process:

- (i) To recruit its nest mates by dancing.
- (ii) To continue foraging at the food source without recruiting new nest mates.
- (iii) To abandon the food source.

The bee would opt for one of these options with different probabilities. The second and the third options have a low probability compared to the first option. They depend on the length of a constructed partial path. If the tour length of the path is short, it is very unlikely that a bee will choose the second or third alternative. On the other hand, if a bee does not use the same partial tour, it will decide to follow a dance performed by its hive mates according to a probability function. This function factors in the total length of the partial route and the number of bees that are promoting this route. To enhance the performance of BS, 2-opt and 3-opt are incorporated before the process

of relocating the hive. The authors tested BS on ten TSPLIB instances (Section 2.4). With the successful adaptation on TSP, they extended the proposed approach to solve stochastic vehicle routing problem [104, 144, 142].

Wong et al. proposed a BCO [154, 155]^T algorithm for TSP which is based on the bee foraging behaviour. Similar to the PSO and ACO algorithms, the BCO algorithm is a population-based algorithm. The fundamental design of the BCO algorithm is as follows. It consists of several steps which will be executed iteratively until a set of criteria is met. These steps are defined such that they computationally realize the bee foraging behaviours and the waggle dance performance. Via these iterative steps, the BCO algorithm is able to generate, modify and evaluate solutions for TSP. To further optimize each and every resulting solutions, an exhaustive 2-opt is incorporated. The algorithm was tested on 20 selected TSPLIB benchmark problems and it solved 19 of them to optimum.

Although both approaches, the BCO and BS algorithms developed by Wong et al. and Lučić & Teodorović respectively, are inspired by the bee foraging behaviour, some distinctions are noted as follows:

- (i) Bees in the BCO algorithm do not remember the number of bees that have visited a particular edge between two cities.
- (ii) Bees in the BCO algorithm advertise the entire feasible path rather than a partial tour in its waggle dance modeling.
- (iii) Bees in the BCO algorithm are influenced by both arc fitness and distance between cities in its state transition rule.
- (iv) The duration of waggle dance shown by bees in the BCO algorithm is determined by a linear function, which factors in the bee's profitability and the profitability of the entire colony.
- (v) The beehive in the BCO algorithm is assumed to have equal distance to all cities.

Chong et al. proposed a BCO algorithm specifically designed for JSSP [26]^J and incorporated a local search technique to further improve the performance of the BCO algorithm (BCNS) [27]^J. The local search technique is based on the N5 neighbourhood structure which has been described in Section 3.4.2 and Table 3.2. In terms of computational accuracy and execution speed, the BCO algorithm outperforms the ACO algorithm, Tabu Search and Shifting Bottleneck algorithms.

Wong et al. proposed a BCO algorithm with Big Valley landscape exploitation (BCBV) [157]^J to solve the Taillard's JSSP datasets 2.4. In the BCBV algorithm, the dances are treated as clusters of solutions to the JSSP. These clusters of solutions are distributed as a Big Valley landscape structure. Via a dance accumulation strategy as well as an effective search in multiple clusters in the entire landscape, the BCBV algorithm is able to generate relatively good solutions for the JSSP. Experimental results comparing the BCBV algorithm with the Shifting Bottleneck heuristic, the Tabu Search algorithm and the Parameter-free Genetic algorithm show that the BCBV algorithm is comparable to these approaches.

3.5 Summary

In this chapter, we have provided an overview on the methods that have been used to solve the COPs in particularly the TSP, QAP and JSSP. The methods that have been described are mainly from the category of nature inspired approaches, namely: the ACO, PSO and BCO algorithms. These three algorithms are summarized by highlighting their macro and micro characteristics in Tables 3.7 and 3.8 respectively.

The ACO, PSO and BCO algorithms shows a high similarity at the macro level (refer to Table 3.7). They are population-based algorithms which consist of a set of iterative steps. During the algorithm execution, the functional structures are self-organized. They

Table 3.7: Macro characteristics of the ACO, the PSO and the BCO algorithms.

Attributes	ACO	PSO	BCO
Biological analogy	Foraging behaviours and pheromone secretion on trails	Swarming or movement	Foraging behaviours, waggle dance, reproduction behaviours and queen bee concept
Functional structures	Ants	Particles (e.g. fishes or birds)	Bees
Population based?	Yes	Yes	Yes
Iterative improvement?	Yes	Yes	Yes
Self organization?	Yes	Yes	Yes
Autocatalytic?	Yes	Yes	Yes
Parameters involvement?	Yes	Yes	Yes
Possible to integrate with other domain specific heuristics? (e.g. 2-opt)	Yes	Yes	Yes

are sensitive to a set of pre-defined parameters and they can be integrated with other domain specific heuristics.

However, these algorithms are different from each other at the micro level in terms of characteristics such as selection schemes and search operators. Table 3.8 summarizes the differences. Selection scheme is found in the ACO and BCO algorithms. In the ACO algorithm, an ant is selected to performed global update. In the BCO algorithm, only selected bees are able to perform waggle dance. In terms of search operator, the ACO algorithm uses a construction rule that is based on heuristic distance and pheromone trails. The PSO algorithm uses two mathematical equations (velocity and position equations) in directing the search. For BCO algorithm, it uses a construction rule that is based on heuristic distance and a preferred path obtained from the waggle dance observation.

Table 3.8: Differences at the micro level of the ACO, the PSO and the BCO algorithms.

Attributes	ACO	PSO	BCO
Selection schemes	Selection of ants to update the trail globally based on the quality of the route taken.	None.	Selection of bees to perform waggle dance based on the profitability.
Search operators	A construction rule that is based on heuristic distance and quantity of pheromone on trails.	An update scheme that is based on a velocity and a position equation.	A construction rule that is based on heuristic distance and a preferred path obtained from the waggle dance observation.
Exploitation	Influence of pheromone trails and heuristic distance.	The use of <i>gBest</i> and <i>pBest</i> in directing a search.	Influence of edges rating and heuristic distance.
Exploration	Pheromone evaporation.	<i>pBest</i> reset on certain particles.	Stochastic choice of a bee to follow a path shown in the waggle dance.

In the subsequent chapters (Chapter 4 to Chapter 6), a BCO algorithm with a local search procedure and a pruning strategy will be described. The BCO algorithm is developed as a generic framework with a set of domain independent classes and abstract classes. The framework is designed for solving different types of COPs where the solution can be represented as a permutation of numbers without repetition. To demonstrate the flexibility of the generic framework to solve different COPs, three COPs has been identified as the showcases, namely: the TSP, QAP and JSSP.

Chapter 4

A Generic Bee Colony Optimization Framework

This chapter describes the generic BCO framework for solving different types of COPs. A set of domain independent and abstract classes are defined in the generic BCO framework. The abstract classes serve as the interface between the domain independent classes and the domain specific classes. The core concept of the BCO framework is based on the foraging behaviour of bees. Besides mimicking the natural metaphor such as the recruitment of bees towards a newly discovered food source via waggle dances, the framework is enhanced with local search and initialization methods as well as a pruning strategy. The enhancement is to ensure that the framework strikes a balance between speed performance and computational accuracy.

This chapter is organized as follows. It first explains the encoding scheme being employed in the framework (refer to Section 4.1). This is followed by a discussion on the various aspects of the generic BCO framework (refer to Section 4.2) with the aid of a class diagram. The BCO algorithm is explained in Section 4.3 and finally three domain specific procedures, namely: the initialization, heuristic distance interpretation and computation, and local search methods are explained in Section 4.4.

4.1 Encoding Scheme

Permutation without repetition of $Z^+ = \{1, 2, \dots, +\infty\}$ is employed as the encoding scheme in the BCO framework. Each of the generated solution is encoded as a string of positive numbers where each number occurs exactly once. Take a permutation $\pi = \{5, 3, 6, 2, 4, 1\}$ as an example. Each number represents a single node where the node represents a city in TSP, a facility in QAP or an operation of a job in JSSP.

In TSP, π denotes that the first city to be visited is city 5, followed by city 3 and so forth. The start city, which is city 5, must be re-visited in order to complete the tour.

In QAP, π denotes that facility 5 is assigned to location 1, facility 3 is assigned to location 2 and so forth.

In JSSP, if $\pi = \{5, 3, 6, 2, 4, 1\}$ is one of the feasible solutions for the example shown in Table 2.1 (refer to Section 2.3), the permutation π will first be converted to a string such that each number corresponds to a job according to a conversion rule $\frac{\pi(i)-1}{m} + 1$, where $\pi(i)$ indicates the i -th element in π and m denotes the number of machines (i.e. $m = 3$). For instance, $\pi(1) = 5$ corresponds to job 2 (i.e. $\frac{5-1}{3} + 1 = 2$). After the conversion, the permutation π becomes $\{2, 1, 2, 1, 2, 1\}$. This permutation indicates that the first operation to be put into the schedule is the first operation of job 2 (to be processed on m_1 for 8 units of time). Next, the first operation of job 1 will be put into the schedule (to be processed on m_2 for 3 units of time). Generally, the schedule sequence complies to the rule as follows. By traversing the converted permutation π starting from left to right, the k -th occurrence of job $\pi(i)$ denotes the k -th operation in the precedence sequence of job $\pi(i)$.

To compute the cost of π for TSP, Eq. 2.1 is applied. Similarly, Eq. 2.2 and Eq. 2.3 are applied in order to compute the cost of π for QAP and JSSP respectively. Information about these equations can be obtained in Chapter 2.

4.2 The Generic Framework

Fig. 4.1 shows the class diagram of the generic BCO framework for solving different COPs. The framework consists of a set of domain independent classes (right pane of the diagram) where the `BeeColonyOptimization` class appears as the base class and a set of abstract classes (middle pane of the diagram).

The `BeeColonyOptimization` class is composed of five domain independent classes, namely: the `Bee`, `DynamicFBPS`, `EliteSolutions`, `ProfitLookupTable` and `Results`. Apart from the composition classes, the `BeeColonyOptimization` class utilizes the classes of `LocalOptimizer`, `Domain`, `Initialization`, `Parameters` and `WaggleDance` in solving a particular COP.

The classes of `LocalOptimizer`, `Domain` and `Initialization` are defined as abstract classes. The `Domain` class is composed of the `LocalOptimizer` and `Initialization` classes. All the three abstract classes contain the definitions of common attributes and methods which are utilized by the `BeeColonyOptimization` class. In the `Domain` class, three abstract methods are defined, namely: the `readInput()`, `getDistance()` and `calcSolutionCost()` (refer to Listing 4.1). For each of the `Initialization` class and `LocalOptimizer` class, one abstract method is defined. They are the `initialize()` and `localSearch()` methods respectively (refer to Listings 4.2 and 4.3). The major functionality of these abstract classes is to interact with the domain specific classes which are shown in the left pane of the class diagram.

The descriptions of each class defined in the proposed generic BCO framework are as follows:

- (i) `DynamicFBPS` (Domain independent class) - This class performs the Frequency-based Pruning Strategy in the proposed BCO framework. It allows a subset of solutions generated by bees to undergo the local search procedure. Its advantage

Figure 4.1: Class diagram of the generic BCO framework for solving different COPs.

```
public abstract class Domain{
    LocalOptimizer lo;
    Initialization init;

    public abstract boolean readInput(String fName);
    public abstract double getDistance(int n1, int n2);
    public abstract double calcSolutionCost(Vector sol);
}
```

Listing 4.1: Domain class.

```
public abstract class Initialization{
    Domain dom;

    Initialization(Domain d){
        dom = d;
    }

    public abstract void initialize(Vector sol);
}
```

Listing 4.2: Initialization class.

```
public abstract class LocalOptimizer{
    Domain dom;

    LocalOptimizer(Domain d){
        dom = d;
    }

    public abstract void localSearch(Vector sol);
}
```

Listing 4.3: Local optimizer class.

is to reduce the high computational overhead of the local search procedure. It performs the pruning via the methods as follows. Further details can be found in Section 6.3.

- **prune()** - This method takes in a solution π generated by a bee and compares the solution with a frequency matrix H . It prunes the solution from undergoing the local search procedure if the solution has a certain percentage of dissimilarity when compared to the frequency matrix.
- **updateMatrix()** - This method tracks the frequency of the smallest building blocks of a solution π and stores the information in a frequency matrix H .

(ii) **Bee** (Domain independent class) - The **Bee** class is composed of the **EncodeList** class and **PastSolutions** class. It's main function is to generate a solution π via a set of methods as follows:

- **resetBestCost()** - This method implements the bee's memory adjustment policy. It is invoked when no bee dances for φ consecutive cycles (e.g. $\varphi = 100$). The aim is to encourage more bees to dance by lowering the personal best cost obtained in previous cycle. Details of this method are described in Section 5.4.2.
- **forageByFollowPastSol()** - This method implements the reinforcement of past solutions acquired in previous cycles. A bee would select and reinforce a solution from a pool of past solutions based on a selection strategy. This reinforcement strategy will be described in Section 5.3.
- **forageByFSTR()** - This method aids a bee to generate a solution π by complying with a state transition rule, which is named as the Fragmentation State Transition Rule. The rule is a function of two factors: arc fitness and heuristic distance. At every step during the phase of constructing a solution, a fragment of nodes is chosen and added until a complete solution is acquired. Details about the transition rule can be found in Section 5.2.
- **observeAndSelectDance()** - This method allows a bee to observe and to follow a dance before starting the foraging process. The probability to follow a dance is determined with the aid of a lookup table, which is named as **ProfitLookupTable** class in the generic BCO framework. A bee is more likely to follow a dance if its cost is worse than the average cost of the colony. Details of this method is described in Section 5.4.1.

- `performWaggleDance()` - This method implements the waggle dance performance by bees. Only bees with a better cost compared to its own previous personal best cost are allowed to dance. The dance would last for a certain duration. The duration is determined according to a linear function where a bee with a better solution would dance longer. Details of this method is described in Section 5.4.2.
- (iii) `BeeColonyOptimization` (Domain independent class) - This is the base class of the proposed BCO framework. It is implemented such that the bee foraging behaviour in nature is computationally realized. This class will be further described algorithmically in Section 4.3. In the proposed algorithm, besides the bee foraging behaviour, it is integrated with other mechanisms such as initialization and local search with a pruning strategy.
- (iv) `Domain` (Abstract class) - This class serves as an important interface that links a problem with the proposed framework. Three abstract methods are defined as follows:
- `readInput()` - This method is domain specific and its main function is to read in an input file and initialize the heuristic distance matrix `HDM` defined in the `Domain` class.
 - `getDistance()` - This method returns the heuristic distance between two nodes. More details can be found in Section 4.4.
 - `calcSolutionCost()` - This method computes the cost of the solution generated by a bee. For the three selected problems of TSP, QAP and JSSP, the `calcSolutionCost()` is implemented according to Eq. 2.1, Eq. 2.2 and Eq. 2.3 respectively. These equations are defined in Chapter 2.

- (v) **EliteSolutions** (Domain independent class) - The **EliteSolutions** class is composed of **WaggleDance** class. The function of this class is to maintain the waggle dances performed by bees as a vector list, *ES*. The accumulation and replacement of waggle dance entries in the *ES* are controlled by the following methods and further details are presented in Chapter 5.4.
- **addEliteDance()** - This method implements the elitism strategy where the best solution found so far will be kept in the *ES* throughout the execution of the algorithm. Details of this method is discussed in Section 4.3.
 - **discardWaggleDance()** - This method discards the expiry entries in the *ES*.
 - **isEmpty()** - This method is to check if the *ES* is empty. If it is empty for φ consecutive cycles (e.g. $\varphi = 100$), the bee's memory adjustment policy will be executed.
- (vi) **EncodeList** (Domain independent class) - This class implements the encoding scheme described in Section 4.1. A solution π is represented as a permutation of number without repetition.
- (vii) **FSTR** (Domain independent class) - This class implements the Fragmentation State Transition Rule which will be described in Section 5.2. The rule allows a bee to add a fragment of nodes in its solution construction.
- **runFSTR()** - This method selects a waggle dance or a past solution and uses it as a guidance in finding a new solution under the influence of arc fitness and heuristic distance. The heuristic distance between two nodes is computed via the abstract method **getDistance()** defined in the **Domain** class.
- (viii) **Initialization** (Abstract class) - This class is one of the composition classes of the **Domain** class. Its function is to initialize the bees in the proposed framework

with a feasible solution during the initial phase of the algorithm execution. This is done via the abstract method `initialize()`. More details can be found in Section 4.4.

(ix) **LocalOptimizer** (Abstract class) - This class is one of the composition classes of the **Domain** class. Its function is to allow the local search procedure to be performed on the solution generated by a bee. This is done via the abstract method of `localSearch()`. More details will be presented in Section 4.4

(x) **Parameters** (Domain independent class) - This class contains a set of BCO parameters which need to be configured prior to the algorithm execution. The details of the parameter tuning can be found in Section 7.2.

- `setParameters()` - This method takes in a set of user-defined parameter values and assigns them to the corresponding BCO parameters.

(xi) **PastSolutions** (Domain independent class) - The function of this class is to maintain the past solutions of each bee and implement it as a vector list *PS*. The *PS* is a bounded list with size of ϖ , which is a BCO parameter in the **Parameters** class. The retrieval and the replacement of a entry in the *PS* is controlled by the following methods:

- `getPastSol()` - This method selects a past solution from the *PS* via a multiple selection strategy which will be discussed in Section 5.3.
- `updatePastSol()` - This method discards an entry from the *PS* and replaces it with a new entry via a replacement strategy which will be discussed in Section 5.3.

(xii) **ProfitLookupTable** (Domain independent class) - This class aids a bee to decide if it needs to follow a waggle dance or not via the method as follows:

- **isFollowing()** - This method provides a mechanism which encourages a bee with a higher profitability compared to the average profitability of the colony to follow a dance performed by its mate. If it decides to follow a dance, *true* is returned or otherwise. The details of this method can be found in Section 5.4.1.
- (xiii) **Results** (Domain independent class) - This class keeps track of the results acquired from the execution of the proposed algorithm via the method as follows:
- **updateResults()** - This method updates the best score found so far and the time stamp to acquire such best score. The solution π that corresponds to the best score is captured.
- (xiv) **WaggleDance** (Domain independent class) - The instances of this class will be the elements of the *ES* defined in the **EliteSolutions** class.

One of the advantages of this framework design is its flexibility when dealing with different types of COPs. In order to solve a COP (e.g. a TSP), the codes that handle domain specific tasks related to TSP will need to be supplied to the framework. The domain specific tasks for TSP are as follows:

- (i) reading in TSP data/input files. This task corresponds to the **readInput()** method defined in the **Domain** class.
- (ii) returning the heuristic distance between two cities. This task corresponds to the **getDistance()** method defined in the **Domain** class.
- (iii) calculating the tour length of a solution path. This task corresponds to the **calcSolutionCost()** method defined in the **Domain** class.
- (iv) initializing a set of feasible solutions with an initialization technique (e.g. the Nearest Neighbourhood heuristic, random initialization). This task corresponds to the **initialize()** method defined in the **Initialization** class.

- (v) optimizing a solution with a local search approach (e.g. the Fixed-radius Near Neighbour (FRNN) 2-opt, the 2-opt). This task corresponds to the `localSearch()` method defined in the `LocalOptimizer` class.

Similarly, if QAP or JSSP is being solved, the domain specific implementation of all the five abstract methods defined in the `LocalOptimizer`, `Domain` and `Initialization` will need to be provided to the framework. The supply of the implementation for domain specific classes is done via a mix-and-match strategy.

Another advantage of such design is its flexibility when modifying/enhancing the BCO framework. As the `BeeColonyOptimization` base class and all other classes are domain independent, any enhancement on the framework will be applicable to all COPs being solved. For example, if scout bees were to be added to the BCO framework in order to increase randomness of the entire system, the effect will be applicable across all COPs being solved (e.g. TSP, QAP or JSSP).

4.3 The Algorithm

The outline of the BCO algorithm is shown in Algorithm 2. It is the implementation of the `BeeColonyOptimization` class in Fig. 4.1. During the initial phase of the algorithm execution, the creation of the instances of the classes of `DynamicFBPS`, `ProfitLookupTable`, `EliteSolutions`, `WaggleDance`, `Parameters` and `Results` is performed. Next, the BCO algorithm reads in an input file by executing the `readInput()` method in the `Domain` class. The heuristic distance matrix is then assigned according to the input file. The BCO parameters are then assigned with a user-defined value via the method of `setParameters()` in the `Parameters` class. The BCO parameters will be explained further in Chapter 7.2.

A group of bees bee , where $|bee| = N_{Bee}$, is created after the BCO parameters assignment. These bees are initialized with a solution (a permutation of numbers without repe-

tition as described in Section 4.1) via an initialization procedure. The implementation of such initialization procedure is domain specific and it corresponds to the `initialize()` method of the `Initialization` class in Algorithm 2 and Fig. 4.1. More details on the initialization procedure will be described in Section 4.4.

Once the bees are initiated, the foraging process starts (see Chapter 5). Bees are allowed to explore and exploit the search space to construct a feasible solution for a problem being solved (e.g. TSP, QAP or JSSP) either using the Fragmentation State Transition Rule (corresponds to `forageByFSTR()` in Algorithm 2 and Fig. 4.1) or the reinforcement of past solutions (corresponds to `forageByFollowPastSol()` in Algorithm 2 and Fig. 4.1). These two mechanisms share an equal likelihood to be performed on a bee and they are described in detail in Sections 5.2 and 5.3 respectively.

If a bee has to perform its solution construction with the Fragmentation State Transition Rule, it will first need to decide if it needs to follow a dance. This is implemented via the `observeAndSelectDance()` method in Algorithm 2 and Fig. 4.1. When a bee follows a selected dance wd , the dance will serve as the guidance in the Fragmentation State Transition Rule. Otherwise, a past solution will be used in the Fragmentation State Transition Rule.

After a bee completes a solution π , the Dynamic Frequency-based Pruning Strategy (DFBPS) will check if π needs a transformation by a local search method by comparing π with a frequency matrix defined in the `DynamicFBPS` class. The checking and the local search procedures are performed via the methods of `prune()` and `localSearch()` in the `DynamicFBPS` class and `LocalOptimizer` class. The pruning strategy and the local search techniques are further explained in Sections 4.4.3 and 6.3. The frequency matrix is then updated according to the content of the resulting solution π via the `updateMatrix()`. Also, the solution π is added as one of the past solutions for the bee via the method of `updatePastSol()`.

Algorithm 2 The BCO algorithm with local search for COPs.

procedure BCO

Require: *dom* (instance of the abstract class Domain)

Require: *init* (instance of the abstract class Initialization)

Require: *lo* (instance of the abstract class LocalOptimizer)

DynamicFBPS *dfbps* {creation of an instance of the DynamicFBPS class.}

ProfitLookupTable *plt* {creation of an instance of the ProfitLookupTable class.}

EliteSolutions *elite* {creation of an instance of the EliteSolutions class.}

WaggleDance *wd* {creation of an instance of the WaggleDance class.}

Parameters *param* {creation of an instance of the Parameters class.}

Results *res* {creation of an instance of the Results class.}

$C_{Best} \leftarrow \infty$ { C_{Best} refers to the cost of global best solution.}

dom.readInput()

param.setParameters()

Bee *bee* {creation of N_{Bee} instances of the Bee class.}

bee $\leftarrow$ *init.initialize()*

while stop criteria are not fulfilled **do**

elite.discardWaggleDance()

if *elite.isEmpty()* for φ cycles **then**

for $\forall b_i \in bee$ **do**

b_i.resetBestScore(b_i.bestCost) {refer to Section 5.4.2.}

end for

end if

for $\forall b_i \in bee$ **do**

b_i.EncodeList. π $\leftarrow \{\}$

if *Math.random()* < 0.50 **then**

if *plt.isFollowing()* = true **then**

wd $\leftarrow$ *b_i.observeAndSelectDance()* {refer to Section 5.4.1.}

b_i.EncodeList. π $\leftarrow$ *b_i.forageByFSTR(wd)* {refer to Section 5.2.}

else

b_i.EncodeList. π $\leftarrow$ *b_i.forageByFSTR(b_i.PastSolutions.getPastSol())*

end if

else

b_i.EncodeList. π $\leftarrow$ *b_i.forageByFollowPastSol()*

end if

if *dfbps.prune(b_i.EncodeList. π)* $\neq$ true {refer to Chapter 6.} **then**

$\pi \leftarrow lo.localSearch(b_i.EncodeList.\pi)$ {refer to Section 4.4.3.}

end if

dfbps.updateMatrix()

b_i.PastSolutions.PS $\leftarrow$ *b_i.PastSolutions.updatePastSol(b_i.EncodeList. π)*

b_i.cost $\leftarrow$ *dom.calcSolutionCost(b_i.EncodeList. π)*

if *b_i.cost* $<$ *b_i.bestCost* **then**

elite.ES $\leftarrow$ *b_i.performWaggleDance(b_i.EncodeList. π)* {refer to Section 5.4.2.}

end if

if *b_i.cost* $<$ C_{Best} **then**

elite.addEliteDance(b_i.EncodeList. π)

$C_{Best} \leftarrow b_i.cost$

res.updateResults()

end if

end for

end while

end procedure BCO

The bee will start performing waggle dance (`performWaggleDance()`) if the cost of the resulting solution ($b_i.cost$) is better than its own personal best cost ($b_i.bestCost$). Also, if the cost of the resulting solution is better than the global best cost (C_{Best}), the global best cost will be replaced. When this occurs, the elitism process is invoked where the resulting solution is added to the waggle dance collection list, ES . The principle of the elitism is to let an outperforming bee to continue to dance so that more bees are recruited towards the profitable food source. The bee will only stop dancing and recruiting when a new bee manages to find a better food source. In the BCO algorithm, this is done as follows. Once a dance is added to ES , the current global best solution will always be kept in the list until it is replaced by a new one. This elitism strategy corresponds to the `addEliteDance()` method in Algorithm 2 and Fig. 4.1.

When all the bees have successfully found a solution, it is considered as one complete bee cycle. After a number of iterations, the BCO algorithm might encounter a situation where no bee is performing waggle dance (trapped in local optima). This is detected when ES contains only the most elite solution. This situation should be avoided as the waggle dance plays an important role in guiding other bees to find a better solution and hence optimality could be achieved. To overcome this problem, a bee's memory reset policy is defined via the method of `resetBestScore()`. The policy is explained in Section 5.4.2 and it is similar to the negative feedback mechanism in homeostasis, which is an example of hormones regulation in human body [64]. The aim of having this self-regulated mechanism in the BCO algorithm is to maintain a balance and stable situation such that waggle dances are performed constantly.

All the steps as shown in Algorithm 2 will be repeated until a set of stopping criteria is fulfilled. Examples of the stopping criteria can be a maximum number of iterations (i.e. BC_{Max}) or a maximum amount of time for the algorithm execution.

4.4 Domain Specific Procedures

This section describes the domain specific procedures defined in the generic BCO framework. Three domain specific functions are highlighted, namely: the initialization (Section 4.4.1), computation of heuristic distance (Section 4.4.2) and local search (Section 4.4.3). These three functions correspond to the abstract methods `initialize()`, `getDistance()` and `localSearch()` shown in Algorithm 2 and Fig. 4.1.

4.4.1 Initialization

During the initial stage of the BCO algorithm execution, every bee is initialized with a feasible solution. The initialization is a domain specific task and it depends on the COP being solved. An initialization strategy for a domain might not be suitable to another domain. They can be different in terms of implementation and the working mechanism. In this section, the initialization strategies for the three selected COPs are described.

For the TSP and QAP, the Nearest Neighbourhood heuristic (NN) [74], the state transition heuristic [156] and a random permutation method are employed as the initialization procedures. Bees in the BCO algorithm choose one of these three initialization strategies fairly. The working mechanism of the NN heuristic is as follows. A bee starts with a random node. In order for a bee to move ahead, it picks the nearest node which is not yet part of the solution from the last node added. If there is a tie in selecting a node, a random selection is applied to break the tie. These steps are repeated until a feasible solution is obtained. On the other hand, the state transition heuristic aids a bee to construct a feasible solution under the influence of heuristic distance. A bee is more likely to select a nearer node to move ahead. However, it might select other nodes with a certain probability. If a bee opts for the random permutation method as its initialization strategy, then the feasible solution is constructed node by node, according to a random pattern.

For the JSSP, a set of dispatching rules is used in generating initial solutions. They range from first-in first-out (FIFO), last-in first-out(LIFO), longest processing time (LPF), least work remaining (LWKR), most work remaining (MWKR), work in next queue (WINQ), shortest processing time with most work remaining (SPF+MWKR) and shortest processing time with work in next queue (SPF+WINQ). These dispatching rules are applied in a round-robin manner for all bees in *bee*. Implementation details for the mentioned dispatching rules can be obtained in [159].

4.4.2 Heuristic Distance Interpretation and Computation

Heuristic distance is a terminology that refers to the cost between two nodes. The heuristic distance is one of the criteria that aids a bee in deciding which node to visit next in its foraging process. The heuristic distance interpretation and computation are different from one domain to another. This section explains the heuristic distance interpretation and computation for the three selected problems (i.e. TSP, QAP and JSSP). The procedure of computing the heuristic distance between two nodes corresponds to the abstract method `getDistance()` in the `Domain` class.

In the TSP, if a six-city symmetric problem is being solved and it is associated with a distance/cost matrix D as follows:

$$D = \begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 0 & 8 & 1 & 4 & 3 & 5 \\ 2 & 8 & 0 & 7 & 5 & 9 & 3 \\ 3 & 1 & 7 & 0 & 2 & 4 & 8 \\ 4 & 4 & 5 & 2 & 0 & 9 & 5 \\ 5 & 3 & 9 & 4 & 9 & 0 & 4 \\ 6 & 5 & 3 & 8 & 5 & 4 & 0 \end{array} \quad (\text{Eq. 4.1})$$

the heuristic distance therefore refers to the distance/cost between two cities. Let's consider a bee stops at city 3 and it has to select a city from cities 1, 2, 4, 5 and 6 as the

next city to visit. Thus, the heuristic distance from city 3 to cities 1, 2, 4, 5 or 6 is 1, 7, 2, 4 or 8 respectively.

The entries in D are computed based on a distance computation method. As mentioned in Section 2.4, the benchmark problem instances obtained from the TSPLIB are employed to test the proposed BCO algorithm. Three different distance computation methods are considered, namely: the 2D Euclidean distance, pseudo-Euclidean distance and geographical distance. The 2D Euclidean distance and pseudo-Euclidean distance are computed based on floating point coordinates. The geographical distance is computed based on coordinates denoted in geographical latitude and longitude of the points on the Earth. The spherical law of cosines which considers the Earth radius is used to compute the geographical distance. Details of each distance computation method are explained in the TSPLIB. Also, in the TSPLIB, some problem instances come with a set of pre-defined distances represented in a matrix. No further computation is needed for these problem instances as every matrix entry denotes the distance between two cities.

In the QAP, as the nature of the problem consists of two matrices, namely: the flow matrix F and the distance matrix D (refer to Section 2.2), a consolidation technique is applied to F and D such that only one matrix $Y = \{y_{i,j}\}$ is produced [41]. Each element in Y is interpreted as a consolidated heuristic value of assigning facility i to location j . Let's consider a symmetric QAP with dimension of six ($n = 6$). This problem is assumed to have a distance matrix D (same as the previous TSP example shown in Eq. 4.1) and a flow matrix F as follows:

$$F = \begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 0 & 38 & 21 & 43 & 32 & 25 \\ 2 & 38 & 0 & 17 & 25 & 29 & 13 \\ 3 & 21 & 17 & 0 & 21 & 14 & 38 \\ 4 & 43 & 25 & 21 & 0 & 29 & 55 \\ 5 & 32 & 29 & 14 & 29 & 0 & 64 \\ 6 & 25 & 13 & 38 & 55 & 64 & 0 \end{array} \quad (\text{Eq. 4.2})$$

Firstly, the sum of each row of the two matrices is computed. This produces $F(\Sigma) = [159, 122, 111, 173, 168, 195]$ and $D(\Sigma) = [21, 32, 22, 25, 29, 25]$. The consolidated matrix Y is produced such that it complies with the equation as follows:

$$y_{i,j} = f(\Sigma)_i \cdot d(\Sigma)_j, \quad 0 < (\forall i, j) \leq n \quad (\text{Eq. 4.3})$$

This leads to Y that contains elements as follows.

	1	2	3	4	5	6
1	3339	5088	3498	3975	4611	3975
2	2562	3904	2684	3050	3538	3050
3	2331	3552	2442	2775	3219	2775
4	3633	5536	3806	4325	5017	4325
5	3528	5376	3696	4200	4872	4200
6	4095	6240	4290	4875	5655	4875

If a bee has allocated facility 3 in location 1 and it has a choice to assign facilities 1, 2, 4, 5 or 6 in location 2, the heuristic values of such allocation are 5088, 3904, 5536, 5376 and 6240 respectively.

In the JSSP, the interpretation of the heuristic distance depends on the dispatching rule being employed in the solution construction. The Most Work Remaining (MWKR) heuristic is described here [71]. If a bee employs the MWKR heuristic in its path construction, the operation belongs to the job with the most remaining work is favoured. Let's consider an example where a bee is trying to solve the two-job $\times$ three-machine JSSP. The problem is described in Section 2.3 and Table 2.1 and it is re-illustrated in Table 4.1.

The bee is assumed to have selected node 3 as its solution and nodes 1, 2, 4, 5 and 6 are yet to be part of the solution. With the conversion method discussed in Section 4.1, the partial solution is interpreted as the first operation of job 1 has been scheduled. This implies that job 1 has two remaining operations which have not been scheduled. They are

Table 4.1: A 2-job $\times$ 3-machine JSSP. Each element is denoted by $x(y)$, where $x \equiv$ machine id and $y \equiv$ processing time. It is a reproduction of Table 2.1 in Section 2.3.

	Operations		
	1	2	3
Job 1	2(3)	1(7)	3(6)
Job 2	1(8)	2(4)	3(5)

the second and the third operations with 7 and 6 units of processing time respectively. Job 2 has three remaining operations which have not been scheduled. They are the first, the second and the third operations with 8, 4 and 5 units of processing time respectively. The heuristic value to schedule a node j as part of the solution is defined as the total number of remaining operations of job $\frac{j-1}{m} + 1$. Thus, to schedule nodes 1, 2, 4, 5 or 6 as the second node in the partial solution, the heuristic value for each option is 2, 2, 3, 3 or 3. With this, a bee would favour node 4, 5 or 6 to be scheduled next to node 3 due to the MWKR heuristic.

The MWKR heuristic can be modified such that heuristic value denotes the total processing time of remaining operations of job $\frac{j-1}{m} + 1$. If this is the case, the heuristic distance values to schedule nodes 1, 2, 4, 5 or 6 would be as follows: 31, 32, 34, 35 or 36. The node with a higher total processing time of remaining operations of a job would be favoured.

In solving JSSP, other heuristics (e.g. the Shortest Processing Time, the Least Work Remaining etc) can be integrated as part of the bee's path construction mechanism provided the heuristic gives a valid interpretation of measuring the distance between two nodes.

4.4.3 Local Search Methods

In the proposed BCO algorithm, a solution generated by a bee is locally optimized by a local search method. The implementation of the local search methods depends on the

Figure 4.2: Local search procedure with Tabu List integration for the TSP, QAP and JSSP.

domain being solved and hence it is part of the domain specific classes depicted in Fig. 4.1. Details that are presented in this section corresponds to the `localSearch()` function in Algorithm 2 and Listing 4.3.

The organization of this section is as follows. Firstly, the Fixed-radius Near Neighbour (FRNN) 2-opt is described (Section 4.4.3.1). It is followed by the descriptions of the 2-opt (Section 4.4.3.2) and the Two-enhancement Scheme with Neighbourhood N5 Perturbation (TESN5) (Section 4.4.3.3). Each of these methods is the local search procedure for TSP, QAP and JSSP respectively and they all are incorporated with a Tabu List as shown in Fig. 4.2.

4.4.3.1 Fixed-radius Near Neighbour (FRNN) 2-opt with Tabu List for TSP

Before looking at the FRNN 2-opt, let's consider a typical 2-opt operation which is applied on TSP. The basic idea of the 2-opt heuristic is as follows. It first eliminates two arcs in order to obtain two different paths. These two paths are then reconnected in the other possible way. If a feasible solution π is with a permutation of $\{A, B, C, D, E, F, A\}$ ¹ and its total tour length is 8 units as shown in Fig. 4.3(a), this closed tour is then transformed by firstly eliminating two arcs (B, C) and (E, F) and thus producing two separate paths "F, A, B" and "C, D, E". These two paths are then reconnected by two replacement edges (C, F) and (B, E) to produce another closed tour, $\pi = \{A, B, E, D, C, F, A\}$, as shown in Fig. 4.3(b). Note that there is only one possible way to reconnect

¹Note that a complete solution for TSP contains the start city as the last element.

Figure 4.3: (a) Original closed tour, R . (b) R' after 2-opt transformation.

these two paths in order to preserve the length of the other four arcs. If the reconnection is done via edges (B, F) and (C, E), it would break the tour into two disjoint cycles. This is not desirable as it produces an infeasible solution for the problem being solved.

By performing the two-arc transformation, the length of the closed tour is reduced from 8 to 6 units. The 2-opt transformation can be generalized to cater for more arcs in its eliminate-and-reconnect process. k -opt is the general terminology for this approach [25].

An exhaustive 2-opt reiterates the basic step of considering all $N(N - 1)/2$ pairs of arcs in a tour, and executing a swap if it gives a better tour length. Hence, the time complexity of a 2-opt swap is $O(N^2)$ [78]. In most cases, the algorithm might need to go through many round of swapping before reaching 2-optimal. Due to the high overhead, the integration of this exhaustive 2-opt in the BCO algorithm would lead to long computational time especially when solving a TSP with a larger dimension. To expedite the 2-opt operations, the 2-opt algorithm is implemented differently as proposed by Bentley [13]. Modifying the exhaustive 2-opt to become the FRNN 2-opt helps to avoid an exhaustive search during the swap process. The results that show the improvement of replacing the exhaustive 2-opt with the FRNN 2-opt were presented in [156].

Similar to the exhaustive 2-opt, the basic operation of the FRNN 2-opt is continuously picking two edges for a swap. If the swap reduces the tour length, the swap will be made persistent, that is to say a successful swap has occurred. Whenever a swap takes place, it is observed that at least one node has an edge that decreases the tour length. Otherwise, the swap would not be successful as both edges increase the tour length. Due to this observation, the exploration of the second edge could be restricted to a circle centered at a node of the first edge with a radius equal to the edge length. This restriction mechanism allows the FRNN 2-opt to have a better computation performance than the exhaustive 2-opt as it only considers nodes within the radius vicinity in forming the candidates of the second edge.

Consider a tour which is drawn on a two dimensional surface as shown in Fig. 4.4. The tour is with a permutation of $\pi = \{ E, F, I, H, C, G, A, B, D, E \}$ and it will be optimized by the FRNN 2-opt. While the edge (E, F) serves as the first edge, a search of the second edge based on the FRNN method centered at E is performed with a radius of d_{EF} . All the nodes within the vicinity will be considered to form the second edge together with its single appropriate neighbour. In Fig. 4.4, node B is found within the vicinity. Edges (B, A) and (B, D) will then be picked as potential candidates for the second edge in the 2-opt swap.

If the exhaustive 2-opt local search is applied on the same permutation of $\pi = \{ E, F, I, H, C, G, A, B, D, E \}$ and (E, F) is chosen as the first edge, the potential candidates for the second edge are (I, H), (H, C), (C, G), (G, A), (A, B) and (B, D). In the worst case scenario, the exhaustive 2-opt has to perform six comparisons before a successful swap. However, only two comparisons are needed in FRNN 2-opt. It is equivalent to a 67% reduction in terms of the number of comparisons.

The FRNN 2-opt in the proposed BCO algorithm is integrated with a Tabu List (referred as *TL* thereafter). The main objective of *TL* is to make sure that the local

Figure 4.4: Searching for the second edge based on the FRNN method where E_{EF} is the first edge.

search moves which have been performed before are not reiterated for a period of time. The working mechanism of TL is as follows. At every particular time, a maximum of 20 local search moves can be stored in TL . These local search moves will be stored in TL for a certain period of time. The staying duration is defined as the dimension of a problem being solved divided by four. If a TSP with a dimension of 400 is being solved, each move will stay in TL for $\frac{400}{4} = 100$ iterations. This duration concept is adapted from the work by Tsubakitani and Evans [146]. Once a move is expired, it is discarded to make room for a new move. The addition of moves is done such that no duplication is permitted in TL . A move in TL has the structure $[a, b, c, d, e]$. a , b , c , and d denote the four required cities in an edges exchange. e designates the staying duration of that move in TL .

When a local search move is to be performed, the local search algorithm will first traverse through TL to check if this move has been performed before and hence identical

$$\begin{aligned}
 &\underline{6} \underline{5} 4 3 2 1 \rightarrow \underline{5} \underline{6} 4 3 2 1; \quad 6 \underline{5} \underline{4} 3 2 1 \rightarrow 6 \underline{4} \underline{5} 3 2 1; \quad 6 5 \underline{4} 3 \underline{2} 1 \rightarrow 6 5 \underline{2} 3 \underline{4} 1; \\
 &\underline{6} 5 \underline{4} 3 2 1 \rightarrow \underline{4} 5 \underline{6} 3 2 1; \quad 6 \underline{5} 4 \underline{3} 2 1 \rightarrow 6 \underline{3} 4 \underline{5} 2 1; \quad 6 5 \underline{4} 3 2 \underline{1} \rightarrow 6 5 \underline{1} 3 2 \underline{4}; \\
 &\underline{6} 5 4 \underline{3} 2 1 \rightarrow \underline{3} 5 4 \underline{6} 2 1; \quad 6 \underline{5} 4 3 \underline{2} 1 \rightarrow 6 \underline{2} 4 3 \underline{5} 1; \quad 6 5 4 \underline{3} \underline{2} 1 \rightarrow 6 5 4 \underline{2} \underline{3} 1; \\
 &\underline{6} 5 4 3 \underline{2} 1 \rightarrow \underline{2} 5 4 3 \underline{6} 1; \quad 6 \underline{5} 4 3 2 \underline{1} \rightarrow 6 \underline{1} 4 3 2 \underline{5}; \quad 6 5 4 \underline{3} 2 \underline{1} \rightarrow 6 5 4 \underline{1} 2 \underline{3}; \\
 &\underline{6} 5 4 3 2 \underline{1} \rightarrow \underline{1} 5 4 3 2 \underline{6}; \quad 6 5 \underline{4} \underline{3} 2 1 \rightarrow 6 5 \underline{3} \underline{4} 2 1; \quad 6 5 4 3 \underline{2} \underline{1} \rightarrow 6 5 4 3 \underline{1} \underline{2}.
 \end{aligned}$$

Figure 4.5: The compare and swap procedure on a π with cardinality of six, $|\pi| = 6$.

move is avoided. A strict checking mechanism is employed whereby a move candidate is compared symmetrically against all the moves in TL .

4.4.3.2 2-opt with Tabu List for QAP

The local search method for QAP is the 2-opt. The 2-opt is implemented such that a compare and swap strategy is applied as follows [123]. Assume that a generated solution π has the cardinality of six ($|\pi| = 6$). This indicates that π is an assignment of six facilities $\{1, 2, 3, 4, 5, 6\}$ on six different locations. Let $\pi(i)$ denotes the i -th element in π . The compare and swap procedure will be performed on the following $\frac{|\pi| \times (|\pi| - 1)}{2}$ pairs: $[\pi(1), \pi(2)]$, $[\pi(1), \pi(3)]$, $[\pi(1), \pi(4)]$, $[\pi(1), \pi(5)]$, $[\pi(1), \pi(6)]$, $[\pi(2), \pi(3)]$, $[\pi(2), \pi(4)]$, $[\pi(2), \pi(5)]$, $[\pi(2), \pi(6)]$, $[\pi(3), \pi(4)]$, $[\pi(3), \pi(5)]$, $[\pi(3), \pi(6)]$, $[\pi(4), \pi(5)]$, $[\pi(4), \pi(6)]$ and $[\pi(5), \pi(6)]$. Fig. 4.5 illustrates the details on the compare and swap procedure.

In the BCO algorithm, the 2-opt local search method is implemented as follows. A swap persists if it leads to a lower cost or otherwise. Once a swap is made persistent, the local search on the current solution π is halted. If all the possible swaps are examined and they bring no improvement, the local search will stop and the BCO algorithm resumes its next step (refer to Algorithm 2).

Another alternative is to implement the 2-opt for QAP such that the entire neighbourhood structure is examined and the swap that brings the maximum decrement in

$$1 \underline{2} 3 4 \underline{5} 6 \xrightarrow{\text{SWAP}} 1 \underline{5} 3 4 \underline{2} 6$$

Figure 4.6: Swapping two operations in π where $p = 2$ and $q = 5$.

$$1 \underline{2} 3 4 \underline{5} 6 \xrightarrow{\text{INSERT}} 1 \underline{5} \underline{2} 3 4 6$$

Figure 4.7: Inserting the $q = 5$ -th operation before the $p = 2$ -nd operation in π .

cost is identified. This swap would be made persistent if it is found to bring the maximum decrement in cost. The overhead for this option is high and hence this puts the option at a disadvantage.

The 2-opt for QAP is also integrated with a TL as mentioned in Section 4.4.3.1. The working mechanism is the same as the TL in the FRNN 2-opt for TSP. The TL has the maximum size of 20 and each element will stay for a period of time, according to the dimension of the problem being solved. A move in TL has the structure $[a, b, c, d, e]$. a and b denote the facilities to be swapped and c and d denote the locus of that two facilities in π . e designates the staying duration of that move in TL .

4.4.3.3 Two-enhancement Scheme with Neighbourhood N5 Perturbation for JSSP

For JSSP, a Two-enhancement Scheme with Neighbourhood N5 Perturbation (TESN5) is proposed to locally optimize a solution π generated by a bee [98, 108]. The TESP5 algorithm consists of two phases. The first phase is to locally optimize a solution with a swapping procedure or an insertion procedure [98]. The swapping procedure swaps two operations which are the p -th and q -th elements ($p \neq q$) in π . The insertion procedure removes the q -th operation and inserts it before p -th operation ($p \neq q$) in π . The procedures of the swap and insertion are shown in Fig. 4.6 and 4.7 respectively.

The swapping and insertion operations are executed iteratively as part of the Simulated Annealing² (SA) algorithm. We name the embedded swapping and insertion operations in the SA algorithm as the Two-enhancement Scheme. The Two-enhancement Scheme is developed such that a worse solution is occasionally accepted according to a defined probability. This mechanism would increase the probability for the system to jump out from a local optimum and getting a better solution. The Two-enhancement Scheme is explained in Algorithm 3 (shown by lines 2-17). The Two-enhancement Scheme starts with an initial temperature T by taking in a solution π and perturbs it with the swap or insertion procedure illustrated in Fig. 4.6 and 4.7. The perturbation would produce a new solution which is denoted as π' . The difference of makespan (Δ) between π and π' is computed. If π' has a worse makespan compared to π (implied by $\Delta \geq 0$), then the π' is accepted as a new solution with a probability of $\min(1, \exp^{-\frac{\Delta}{T}})$. Otherwise, π' would be accepted $\pi \leftarrow \pi'$. The Two-enhancement Scheme starts from a high temperature and T is slowly decreased according to a cooling schedule (defined by the cooling rate CR in Algorithm 3). The Two-enhancement Scheme will continue reiterating its operations as long as T is higher than a pre-defined final temperature which is configured at a value near to zero (denoted as $T > T_f$ in Algorithm 3). In our experiment with JSSP, the following settings are used: initial $T = 20$, $CR = 0.9$ and $T_f = 0.1$.

The solution π generated from the Two-enhancement Scheme is then further improved according to the neighbourhood structure as proposed by Nowicki and Smutnicki [108]. This neighbourhood structure ensures that a new set of neighbour solutions is obtained by applying a minor perturbation on a given solution and it is named as the neighbourhood N5 [163]. The perturbation is referred as a move l_j . The central idea in the implementation of the neighbourhood N5 is based on the block decomposition structure explained in Section 2.3.

²An algorithm which is developed based on the heating and cooling behaviour of physical particles.

In order to produce a set of moves on a solution π , the N5 structure shown in Eq. 4.4 is identified on a critical path of π , which is decomposed into a set of blocks, $B = \{b_1, b_2, \dots, b_q\}$ where $q = |B|$, $b_i = \{\phi_{i1}, \phi_{i2}, \dots, \phi_{i|b_i|}\}$ and $\phi_{ij} \in O$ is the j -th operation of block i . The $N5_i(b_i)$ in Eq. 4.4 is further defined by Eq. 4.5, Eq. 4.6 and Eq. 4.7.

$$N5(\pi) = \bigcup_{i=1}^q N5_i(b_i), \forall b_i \in B, q = |B| \quad (\text{Eq. 4.4})$$

$$N5_1(b_1) = \left\{ \begin{array}{ll} \{(\phi_{1(x-1)}, \phi_{1x})\} & , \text{ if } (x = |b_1|) > 1; \\ \{\} & , \text{ otherwise.} \end{array} \right\} \text{ for } q \geq 1; i = 1. \quad (\text{Eq. 4.5})$$

$$N5_i(b_i) = \left\{ \begin{array}{ll} \{(\phi_{i1}, \phi_{i2}), (\phi_{i(x-1)}, \phi_{ix})\} & , \text{ if } (x = |b_i|) > 3; \\ \{(\phi_{i1}, \phi_{i2})\} & , \text{ if } (x = |b_i|) = 2; \\ \{\} & , \text{ otherwise.} \end{array} \right\} \text{ for } q \geq 3; \forall i = 2, \dots, q-1. \quad (\text{Eq. 4.6})$$

$$N5_q(b_q) = \left\{ \begin{array}{ll} \{(\phi_{q1}, \phi_{q2})\} & , \text{ if } |b_q| > 1; \\ \{\} & , \text{ otherwise.} \end{array} \right\} \text{ for } q \geq 2; i = q. \quad (\text{Eq. 4.7})$$

Eq. 4.5 states that swapping is allowed on the last two operations in the first block. Eq. 4.6 states that swapping is permitted on the first two (and the last two) operations in every intermediate block $\{b_2, b_3, \dots, b_{q-1}\}$. Eq. 4.7 states that swapping is allowed on the first two operations in the last block. Once the neighbourhood search completes its operation, $N5(\pi)$ contains all the moves which are grouped together via a union operator. If any of these moves is applied, a new feasible neighbour solution is obtained.

Once the entire neighbourhood $N5(\pi)$ is obtained, it is filtered such that only moves that bring an improvement in terms of makespan are of interest. These moves are denoted as a set of improving moves I . Of these moves in I , the move that is not in TL^3 ,

³The working mechanism is the same as the TL in the FRNN 2-opt for TSP in Section 4.4.3.1.

$l_j \in (I - TL)$ and brings the maximum improvement to the makespan is performed. The newly opted move is then inserted into TL . These steps are shown by lines 19-30 in Algorithm 3.

Algorithm 3 The *localSearch* algorithm with a move selection strategy for JSSP.

procedure localSearch(π)

Require: π (require a solution generated by the FSTR).

Require: T (a starting temperature), T_f (a final temperature) and CR (a cooling rate).

Ensure: π (return a solution that has been locally optimized).

```

1:
2: while  $T > T_f$  do
3:   if  $\text{Math.random}() < 0.50$  then
4:      $\pi' \leftarrow \pi.\text{swap}(p, q)$  {both  $p$  and  $q$  denote two random elements in  $\pi$ .}
5:   else
6:      $\pi' \leftarrow \pi.\text{insert}(p, q)$  {both  $p$  and  $q$  denote two random elements in  $\pi$ .}
7:   end if
8:    $\Delta \leftarrow C_{JSSP}(\pi') - C_{JSSP}(\pi)$ 
9:   if  $\Delta \geq 0$  then
10:    if  $\text{Math.random}() < \min(1, \exp^{-\frac{\Delta}{T}})$  then
11:       $\pi \leftarrow \pi'$ 
12:    end if
13:  else
14:     $\pi \leftarrow \pi'$ 
15:  end if
16:   $T \leftarrow CR \times T$ 
17: end while
18:
19: for  $\forall \text{move } l_j \in N5(\pi)$  do
20:    $\pi' \leftarrow \pi.\text{performMove}(l_j)$ 
21:   if  $C_{JSSP}(\pi') < C_{JSSP}(\pi)$  then
22:      $I \leftarrow l_j$ 
23:   end if
24: end for
25:
26: if  $I - TL \neq \{\}$  then
27:   Find a move  $l_j \in (I - TL)$  that leads to the maximum advancement
28:    $\pi \leftarrow \pi.\text{performMove}(l_j)$ 
29:    $TL \leftarrow l_j$ 
30: end if
31: return( $\pi$ )
end procedure localSearch( $\pi$ )

```

4.5 Summary

In this chapter, a generic BCO framework has been described. Various aspects of the framework has been introduced such as the generic encoding scheme, the design of the framework which is based on the object-oriented approach and the outline of the BCO algorithm. The generic encoding scheme that is used in the BCO framework is a permutation without repetition, $Z^+ = \{1, 2, \dots, +\infty\}$. The BCO framework is designed such that it consists of a set of domain independent classes and a set of abstract classes. The framework interacts with domain specific classes via the abstract classes. The domain specific classes are unique and they are outside of the BCO framework. The implementation of these classes is specific to the problem being solved.

The generic BCO framework is able to solve other COPs as long as the domain specific classes that handle domain specific tasks are provided (according to Listings 4.1, 4.2 and 4.3). Also, when an enhancement is added to the BCO framework, the effect is applicable to all COPs being solved. The flexibility to deal with different COPs and to implement modifications/enhancements are among the two major advantages of the proposed BCO framework.

In the subsequent chapter, a model that shows how bees in the BCO algorithm construct a feasible solution will be described. Also, the waggle dance performance by bees is described in the subsequent chapter.

Chapter 5

Bee Foraging Behaviour

In a bee colony, bees need to keep on foraging in order to ensure an undisrupted food supply. With sufficient food supply, the survival of the entire colony is guaranteed. Thus, foraging is one of the important activities in a bee colony. Naturally, bees travel from one spot to another spot in their foraging process. For example, in a honey bee colony, bees would travel from one flower patch to another flower patch in order to discover new food source and to collect nectar back to the hive. When a bee has found a food source, it starts to perform a waggle dance to other hive mates upon returning to its hive. The aim is to recruit more bees towards the newly discovered food source.

This chapter describes the mechanism of artificial bees in constructing a feasible solution for a problem being solved as well as how the waggle dance model is computationally realized.

In the BCO algorithm, artificial bees have the ability to travel from one node to another node until a complete solution is obtained. This process is similar to what has been performed by bees in nature where bees travel from one spot to another spot to discover new food source. The artificial bees in the BCO algorithm are aided by a mechanism which helps them to decide on which node to select as the next node to travel to. Two mechanisms will be described, namely: the Step-by-step State Transition Rule (SSTR) and the Fragmentation State Transition Rule (FSTR). Details of the SSTR and

the FSTR will be explained in Sections 5.1 and 5.2 respectively. Also, the bees in the proposed algorithm are able to reinforce past solutions generated in the previous cycles. The reinforcement strategy is described in Section 5.3.

The discussion of the waggle dance model is in Sections 5.4.1 and 5.4.2. Section 5.4.1 describes the dance observation and selection strategies and Section 5.4.2 presents the mechanism of the waggle dance performance by bees.

5.1 Step-by-step State Transition Rule (SSTR)

Before a bee starts the path construction in its foraging process, it randomly observes dances performed by the other bees. Once the bee has selected a dance to follow, it will be equipped with an ordered set of moves. This set of moves, named as a *preferred path* and denoted as θ , will then serve as a guidance in its foraging process. θ contains a complete set of stopping points that has been previously explored by its mate. It is one of the possible permutations Π as described in Chapter 2, $\theta \in \Pi$.

During the foraging process, a bee has to move from one node to another node in order to collect food. To aid a bee in its decision making on which node to visit next, a transition rule is proposed. This rule will be described in Section 5.1.3. The transition rule consists of two factors: arc fitness and heuristic distance. The arc fitness is computed as follows. If a bee stops at node x , all the possible arcs that link to the node x will be assigned a fitness value. A higher fitness value is assigned to the arc which is part of the preferred path θ so that the bee tends to choose the next node to be visited based on its mate's past experience. On the other hand, under the heuristic distance influence, a bee tends to choose the next node to be visited which is near to its current node.

Heuristic distance is a general terminology which refers to different cost measurements between two nodes and it depends on the problem being solved (i.e. TSP, QAP and JSSP). Section 5.1.2 will describe this in detail.

To describe more on this path construction mechanism, two examples will be presented in the subsequent Sections 5.1.1 and 5.1.2. The first example describes path finding under the influence of arc fitness whereas the second example describes the influence of heuristic distance. In both examples, a bee is required to collect food from a set of six nodes, denoted by $\{1, 2, 3, 4, 5, 6\}$. In addition, the bee is assumed to start its foraging process from its hive and randomly pick a node as the first node to stop, that is node 3.

5.1.1 Arc Fitness Factor

This section depicts the example of how a bee is influenced by the arc fitness factor in its path construction. This bee has observed a dance performed by another bee before leaving its hive. Thus, the bee is equipped with a preferred path θ and it contains the following entries in the particular order: $\theta = \{2, 1, 4, 3, 5, 6\}$.

To illustrate that the bee is influenced by the arc fitness factor only, an assumption is defined as follows. All the nodes are equidistance among each other. With this, the bee will be biased towards collecting food according to the sequence in θ .

When a bee is at node i after t transitions, two sets of nodes are derived, namely: a set of ***not-yet-selected nodes*** $A_{i,t}$ and a set of ***preferred next node*** $G_{i,t}$. $A_{i,t}$ is defined as a set of nodes that have not been selected yet and they can be reached from node i at transition t . $G_{i,t}$ is a set that contains one node which the bee prefers to move to from node i at transition t as recommended by θ . Only the forward adjacent node of node i in θ is considered. Let $\theta(y)$ denotes the y -th element in θ . If the current node is $\theta(y)$ after t transitions, then $G_{\theta(y),t} = \{\theta(y+1)\}$.

When $t = 1$, the bee stops at node 3 after leaving its hive due to the random selection assumption made in Section 5.1. With a reference to $\theta = \{2, 1, 4, 3, 5, 6\}$, two sets of nodes are identified: $A_{3,1} = \{1, 2, 4, 5, 6\}$ and $G_{3,1} = \{5\}$. All the arcs that connect

node 3 and the nodes in $A_{3,1}$ will be assigned with a fitness value as follow: λ is assigned to arc (3, 5) and $\frac{1-\lambda}{4}$ is assigned to arcs (3, 1), (3, 2), (3, 4) and (3, 6).

The fitness values at transition t from node i can be illustrated by an arc fitness matrix, $\Phi_{i,t}$ with dimensions of $1 \times |A_{i,t}|$. Each entry of $\Phi_{i,t}$ is denoted by $\rho_{ij,t}$, which indicates the fitness value assigned to the arc from node i to node j where $j \in A_{i,t}$. The following matrix represents arc fitness values from node 3 to all the nodes in $A_{i,t}$ ($\forall j \in A_{i,t}$) when $t = 1$:

$$\begin{aligned}\Phi_{3,1} &= [\rho_{31,1} \quad \rho_{32,1} \quad \rho_{34,1} \quad \rho_{35,1} \quad \rho_{36,1}] \\ &= [\frac{1-\lambda}{4} \quad \frac{1-\lambda}{4} \quad \frac{1-\lambda}{4} \quad \lambda \quad \frac{1-\lambda}{4}]\end{aligned}$$

λ represents the probability of trailing a node in θ . Note that the arc fitness matrix is associated with the following property as shown in Eq. 5.1.

$$\sum_{j \in A_{i,t}} \rho_{ij,t} = 1 \quad (\text{Eq. 5.1})$$

The idea of this mechanism is to ensure that the node proposed by the preferred path θ has a higher probability (λ in this case) while the rest of the arcs share the probability of $1 - \lambda$.

Let's assume node 5 is picked when $t = 1$. t now becomes 2 and this leads to the following: $A_{5,2} = \{1, 2, 4, 6\}$ and $G_{5,2} = \{6\}$. Likewise, the arc fitness matrix $\Phi_{5,2}$ is shown as follows:

$$\begin{aligned}\Phi_{5,2} &= [\rho_{51,2} \quad \rho_{52,2} \quad \rho_{54,2} \quad \rho_{56,2}] \\ &= [\frac{1-\lambda}{3} \quad \frac{1-\lambda}{3} \quad \frac{1-\lambda}{3} \quad \lambda]\end{aligned}$$

If node 4 is picked when $t = 1$. When proceeding to $t = 2$, the content of $A_{4,2}$ is $\{1, 2, 5, 6\}$ and $G_{4,2}$ is $\{3\}$. Note that node 3 as listed in $G_{4,2}$ is no longer a feasible node as it has been picked when $t = 0$. Hence, all the arcs that connect node 4 and the nodes in $A_{4,2}$ will be assigned a probability of $\frac{1}{4}$ as follows:

$$\begin{aligned}\Phi_{4,2} &= [\rho_{41,2} \quad \rho_{42,2} \quad \rho_{45,2} \quad \rho_{46,2}] \\ &= [\frac{1}{4} \quad \frac{1}{4} \quad \frac{1}{4} \quad \frac{1}{4}]\end{aligned}$$

This indicates that all the four arcs are now sharing the same likelihood to be picked by the bee as its next node to be visited.

Besides the uniform preference of the remaining arcs where all the remaining arcs share the same selection likelihood, it is also possible to integrate the mechanism with an ordered preference of the remaining arcs. The remaining arcs can be ordered/biased according to a priority metric (e.g. heuristic distance, visit frequency). In the proposed BCO algorithm, the path construction mechanism is influenced by another factor, which is the heuristic distance factor. This factor is independently implemented and it is not part of the arc fitness factor. The path construction mechanism under the heuristic distance influence will be described in the subsequent section (Section 5.1.2).

5.1.2 Heuristic Distance Factor

After examining the example that describes the path construction mechanism by a bee under the arc fitness influence in Section 5.1.1, let's look at another example where a bee construct its path under the heuristic distance influence.

In the example explained in Section 5.1.1 the arc fitness factor acts exclusively on a bee. In this example, the heuristic distance factor acts exclusively on a bee. The bee has been assumed to stop at node 3 when $t = 1$. The bee is going to pick and add one node at a time until a complete path is obtained. At any stopping node i after t transitions, the bee is influenced by the heuristic distance d_{ij} where $\forall j \in A_{i,t}$. The bee is assumed to have just started its first foraging expedition and there is no dance to be observed at the hive. Thus, the bee has no past experience or prior information about the foraging journey and it is not equipped with a preferred path θ . This also indicates that at any stopping node i , the set of preferred next node will be empty, $G_{i,t} = \{\}$.

When $t = 1$, the content of $A_{3,1}$ is $\{1, 2, 4, 5, 6\}$ due to the assumption that the bee stops at node 3 after a random selection. The bee is now to decide which node to

pick from $A_{3,1}$ under the heuristic distance influence. Let's assume the heuristic distance values from the stopping node i to all other nodes $j \in A_{i,t}$ are represented by a matrix $\Omega_{i,t}$ which the matrix is of size $1 \times |A_{i,t}|$. Each entry of $\Omega_{i,t}$ is denoted by $\frac{1}{d_{ij}}$ as follows:

$$\begin{aligned}\Omega_{3,1} &= \begin{bmatrix} \frac{1}{d_{31}} & \frac{1}{d_{32}} & \frac{1}{d_{34}} & \frac{1}{d_{35}} & \frac{1}{d_{36}} \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{1} & \frac{1}{7} & \frac{1}{2} & \frac{1}{4} & \frac{1}{8} \end{bmatrix}\end{aligned}$$

Out of these nodes, node 1 is favoured by the bee at the next transition $t = 2$. The idea of this method is to allow the arc with a smaller heuristic distance d_{ij} to be selected (implied by a bigger value after the reciprocal for d_{ij} is taken).

The way of determining the entries in $\Omega_{i,t}$ (heuristic distance) is different from one domain to another domain. Further information on how to compute and interpret the heuristic distance in different domains can be found in Section 4.4.2.

5.1.3 The Transition Rule

Two examples have been described in Sections 5.1.1 and 5.1.2. The examples show the working mechanism of a bee on how to construct a solution by adding one node at a time under the influence of arc fitness and heuristic distance factors. These two factors are integrated to form a state transition rule, namely: the Step-by-step State Transition Rule (SSTR). The SSTR is described in this section.

During the foraging process in the SSTR, a bee is aided by a rule in deciding which node to be picked as the next node. The chosen node will be appended as part of π until a complete solution is obtained. The state transition probability $P_{ij,t}$ determines the likelihood to move from node i to node j after t transitions. It is a function of two factors: a heuristic distance that measures the closeness of nodes i and j and arc fitness that presents on the connecting edge of nodes i and j . Formally, it is defined as in Eq. 5.2 as follows:

$$P_{ij,t} = \frac{[\rho_{ij,t}]^\alpha \cdot \left[\frac{1}{d_{ij}}\right]^\beta}{\sum_{j \in A_{i,t}} \left([\rho_{ij,t}]^\alpha \cdot \left[\frac{1}{d_{ij}}\right]^\beta\right)} \quad (\text{Eq. 5.2})$$

where $\rho_{ij,t}$ is the arc fitness that presents on the connecting edge of nodes i and j after t transitions. d_{ij} is the heuristic distance between nodes i and j . Note that $P_{ij,t}$ is inversely proportional to the nodes distance. The lower the heuristic value, the higher is the likelihood of that node to be selected. $A_{i,t}$ is a set of not-yet-selected nodes that can be reached from node i at transition t (refer to Section 5.1.1). α is a binary variable ($\alpha \in \{0, 1\}$) that turns on or off the arc fitness influence in the algorithm. β ($\beta \in \mathbb{Z}^+$) controls the significance level of the heuristic distance factor. Throughout the thesis, α is configured as 1. This denotes that the arc fitness influence is always activated. If α is set to 0, it indicates that the heuristic distance factor has an exclusive effect to the path construction mechanism.

The arc fitness $\rho_{ij,t}$, that is described in Section 5.1.1, is defined in Eq. 5.3. Note that $|A_{i,t} \cap G_{i,t}|$ is 1 if there is a common instance in both $A_{i,t}$ and $G_{i,t}$. Otherwise, $|A_{i,t} \cap G_{i,t}| = 0$. $A_{i,t} - G_{i,t}$ denotes the difference between sets $A_{i,t}$ and $G_{i,t}$. It contains all elements of $A_{i,t}$ that are not present in $G_{i,t}$.

$$\rho_{ij,t} = \left\{ \begin{array}{ll} \lambda & , j \in G_{i,t}, |A_{i,t}| > 1 \\ \frac{1-\lambda|A_{i,t} \cap G_{i,t}|}{|A_{i,t} - G_{i,t}|} & , j \notin G_{i,t}, |A_{i,t}| > 1 \\ 1 & , |A_{i,t}| = 1 \end{array} \right\} \forall j \in A_{i,t}, 0 \leq \lambda \leq 1 \quad (\text{Eq. 5.3})$$

The first two conditions ensure that the arc suggested by the preferred path θ (if there is one) is assigned with a value of λ whereas the rest of the arcs are assigned with an equal likelihood value.

The third condition caters for a situation when there is only one node left in $A_{i,t}$. If this is the case, $\rho_{ij,t}$ is set to 1 to indicate that the node is the only choice. This can be observed at the last transition.

5.2 Fragmentation State Transition Rule (FSTR)

In Section 5.1, the SSTR has been described. The main characteristic of the SSTR is to enable bees to add one node at one time until a complete solution is obtained. This

mechanism is a time consuming process and it is not scalable for problems with a larger dimension.

A modified version of the state transition rule will be described in this section, namely: the Fragmentation State Transition Rule (FSTR). The major characteristic of the FSTR is as follows. Rather than adding one node at a time, a bee will repeatedly add a few nodes (fragments of nodes) at a time during the phase of solution construction. Since the encoding scheme used in the algorithm is a string of numbers without repetition (see Section 4.1), a bee will add a fragment of numbers at a time until a complete solution is generated. By doing so, the BCO algorithm would be more scalable and it is able to solve problems with a larger dimension. The modification of the SSTR to become FSTR also serves as an example that illustrates the flexibility to alter the generic BCO framework. Once the FSTR is plugged into the framework as one of the domain independent classes, the effect of the alteration will be applicable to all the problems in different domains (e.g. TSP, QAP and JSSP).

The description of the FSTR presented in this section corresponds to the function of `forageByFSTR()` in Algorithm 2 and Fig. 4.1.

The FSTR is inspired by the Nearest Fragment (NF) operator demonstrated together with the Genetic Algorithm in solving TSP and microarray gene ordering [125]. The NF operator suggests that a chromosome is randomly sliced into multiple fragments. The first element and the last element of each fragment will serve as connection points. In the reconnection process of these fragments via the connection points, the Nearest Neighbourhood heuristic is applied.

The following example explains the working mechanism of the FSTR. Before a bee leaves its hive for the foraging process, it will be equipped with a set of moves θ that is either obtained from the dance observation or from the path that it explored in the previous iterations. Information about the preferred path θ can be obtained in Section

5.1. In this example, we assume that the bee randomly observes a dance performed by one of its hive mates and it is a string of alphabets (instead of numbers), i.e. $\theta = \{A, B, C, D, E, F, G, H, I, J\}$. To start the FSTR, the content of θ will be cloned and assigned to ϵ , $\epsilon \leftarrow \theta$. At this moment, the content of ϵ and θ is identical. ϵ then undergoes a slicing operation at a given position so that multiple fragments of nodes are produced. The slicing strategies varies from variable-length or fixed-length fragmentation. The example in this section assumes that ϵ will undergo a variable-length fragmentation. If ϵ is randomly sliced at the positions three and eight, three different fragments are generated, $\epsilon = \{A, B, C \mid D, E, F, G, H \mid I, J\}$. On the other hand, if the fixed-length fragmentation takes place where each fragment must have exactly y (e.g. $y = 3$) elements, the resulting fragments will be $\epsilon = \{A, B, C \mid D, E, F \mid G, H, I \mid J\}$. Now, every fragment contains three elements except the last fragment where it contains only a single element. In general, for the fixed-length fragmentation strategy, the last fragment contains $(|\epsilon| \text{ modulo } y)$ element(s). If the last fragment contains y element(s), the number of fragments obtained will be $|\epsilon|/y$. Otherwise, there will be $(|\epsilon|/y) + 1$ fragment(s).

At the beginning, solution $\pi = \{\}$. Recall that $\epsilon = \{A, B, C \mid D, E, F, G, H \mid I, J\}$. The bee will randomly choose any of these three fragments from ϵ as part of π . Assuming that the second fragment is selected, then $\pi = \{D, E, F, G, H\}$ and now $\epsilon = \{A, B, C \mid I, J\}$. The bee is now assumed to stop at node H. It will decide which fragment to connect next to node H in the subsequent transition. In order to connect one of the remaining two fragments in ϵ , next to node H, nodes A, C, I and J will act as connection points. Node A is the first node and node C is the last node of the fragment “A, B, C” in ϵ . Similarly, nodes I and J are the first node and the last node of the fragment “I, J” in ϵ .

To continue the path construction, the state transition rules as defined in Eq. 5.2 and Eq. 5.3 are used to aid the bee in selecting the next node to connect. At transition $t = 1$

Figure 5.1: Values assignment to the edges of (H, A), (H, C), (H, I) and (H, J).

where the bee stops at node H, $A_{H,1} = \{A, C, I, J\}$. At the same time, $G_{H,1} = \{I\}$ ¹. A value of $\rho_{ij,1}^\alpha \times \frac{1}{d_{ij}^\beta}$ will be assigned to the edges of (H, A), (H, C), (H, I) and (H, J) as shown in Fig. 5.1 where i is node H and $\forall j \in A_{H,1}$. Each of the $\rho_{HA,1}$, $\rho_{HC,1}$ and $\rho_{HJ,1}$ will be rewarded $\frac{1-\lambda}{3}$ and $\rho_{HI,1}$ will be rewarded λ . If $G_{H,1} = \{\}$, all the mentioned edges will be sharing a value of $\frac{1-\lambda}{|A_{H,1}|} = \frac{1-\lambda}{4}$. This mechanism ensures that the edge recommended by the preferred path θ is awarded with λ (a higher value) and hence it is favoured by bees. The rest of the non-recommended edges will be sharing an average value of $1 - \lambda$ (a lower value).

Eq. 5.2 will then be used to decide the probability value to travel from node H to each of the node in $A_{H,1}$. Assuming that node C is chosen, the fragment of “A, B, C” will be linked next to node H in a reverse order. The resulting tour after such linkage is $\pi = \{D, E, F, G, H, C, B, A\}$. If node A is chosen, the reverse connection is not necessary and thus producing $\pi = \{D, E, F, G, H, A, B, C\}$. As nodes A, B and C are now part of the resulting solution π , they are marked as selected nodes. ϵ is updated to $\epsilon = \{I, J\}$. The process continues until all the nodes in ϵ have been added to π .

The given example illustrates the executions of 6 branching considerations, 3 slicing operations and 2 re-connections. If a bee has to construct a path which consists of ten cities (A, B, C, D, E, F, G, H, I, J) with the SSSTR, a total of 45 branching considerations

¹Recall that $\theta = \{A, B, C, D, E, F, G, H, I, J\}$. Hence, $\theta(8) = H$. This leads to $G_{H,1} = \{\theta(8+1)\} = \{I\}$ as only the forward adjacent node is considered.

and 9 connections have to be performed. In terms of branching considerations, a saving of $\frac{45-6}{45}\% = 86.67\%$ is noticed. In terms of connections need to be made, a saving of $\frac{9-2}{9}\% = 77.78\%$ is noticed. Although the FSTR incurs overhead when performing the fragmentation mechanism, this could be compensated with the savings from the reduction of branching considerations and re-connections. Also, the computation efficiency of the FSTR can be improved by developing a parallel version of the algorithm. Results obtained by experiments which show a reduction in computational cost, due to the replacement of the SSTR with the FSTR, can be found in 7.6, 7.12 and 7.18.

Connecting a fragment of nodes rather than connecting a node at a time is the main idea of the FSTR. The advantages of the FSTR are two-fold. Firstly, the FSTR reduces the neighbourhood size in the solution building phase. Thus, it leads to a lower computational cost. Secondly, instead of always picking the nearest node in its fragments reconnection process, which is a greedy approach, other nodes will still have a chance to be selected probabilistically under the influence of Eq. 5.2 and Eq. 5.3.

5.3 Reinforcement of Past Solutions

This section aims to describe how the artificial bees in the proposed BCO algorithm reinforce past solutions which were generated in the previous cycles. In Algorithm 2, it is noted that a bee shares an equal likelihood to either construct a solution with the aid of a state transition rule or via reinforcing a past solution acquired in the previous cycles. The latter construction method is denoted as `forageByFollowPastSol()` in the `Bee` class.

In the proposed BCO algorithm, an artificial bee has the ability to remember a set of past solutions, PS . The size of PS is limited to a maximum of ϖ elements at one particular time. If $|PS| < \varpi$ elements, a solution is added to PS without going through the replacement scheme. Otherwise, an existing solution $ps_i \in PS$ is discarded to make

room for the new solution. In other words, if a solution has been added in as part of PS , it will remain in PS until it is replaced by a new solution.

Let's assume a new solution π is ready to be added in as part of the PS and the existing solution with the maximum or minimum cost is denoted as ps_{max} or ps_{min} respectively. Both $ps_{max}, ps_{min} \in PS$. Also, $PS(x)$ denotes the x -th element in PS . The replacement scheme consists of five different policies as follows: (i) replace ps_{max} with π ; (ii) replace ps_{max} with π if $C(\pi) < C(ps_{max})$; (iii) replace $PS(1)$ with π ; (iv) replace $PS(\varpi)$ with π ; and (v) replace a random element $ps_i \in PS$ with π .

When a bee decides to reinforce a past solution, it selects one solution from PS with the following acquisition policies: (i) select ps_{max} ; (ii) select ps_{min} ; (iii) select $PS(1)$; (iv) select $PS(\varpi)$; and (v) select a random element $ps_i \in PS$.

Each of the replacement and acquisition policy shares an equal likelihood to occur. The advantage of having multiple replacement and acquisition policies is that a bee is able to reinforce a solution which is selected from a pool of past solutions. This avoids a situation where the past solution from the immediate previous cycle is always replaced or reinforced.

5.4 Waggle Dance

This section describes how the waggle dance model is computationally realized in the proposed BCO algorithm. The model description corresponds to the functions in Algorithm 2 and Fig. 4.1 as follows:

- `isFollowing()`,
- `observeAndSelectDance()`,
- `performWaggleDance()`,
- `addEliteDance()` and
- `discardWaggleDance()`.

Table 5.1: Lookup table for P_{follow} adjustment.

Profitability Scores	P_{follow}
$Pf_i < 0.95Pf_{colony}$	0.80
$0.95Pf_{colony} \leq Pf_i < 0.975Pf_{colony}$	0.20
$0.975Pf_{colony} \leq Pf_i < 0.99Pf_{colony}$	0.02
$0.99Pf_{colony} \leq Pf_i$	0.00

5.4.1 Waggle Dance Observation and Selection

Before a bee leaves its hive, it determines if it needs to observe and follow a dance shown by previous dancers (bees) with a probability of P_{follow} . P_{follow} is adjusted dynamically according to the profitability score of the bee and the colony, based on the lookup table in Table 5.1. In the extreme case where P_{follow} is zero, the bee will keep to its own path. This table is adapted from [107] and the entries are configured empirically based on a benchmark problem instances in the TSPLIB (i.e. EIL101). We assume that this set of values is applicable to all benchmark instances across different problem domains.

Pf_i denotes the profitability score of bee i as defined in Eq. 5.4. Pf_i is defined as a reciprocal value of the solution cost so that it is inversely proportional to the solution cost. The solution cost is different from one problem to another problem as defined in Eq. 2.1, Eq. 2.2 or Eq. 2.3 (refer to Chapter 2). Pf_{colony} denotes the bee colony's average profitability as in Eq. 5.5 and it is updated after each bee completes the construction of a solution.

$$Pf_i = \frac{1}{C(\pi)}, \quad C(\pi) = \text{solution cost.} \quad (\text{Eq. 5.4})$$

$$Pf_{colony} = \frac{1}{N_{Bee}} \sum_{i=1}^{N_{Bee}} Pf_i \quad (\text{Eq. 5.5})$$

A bee is more likely to randomly observe and follow a waggle dance if its profitability rating is low when compared to the average profitability of the colony. Although a bee

tends to be influenced either by its own or other bees' experience, it may still occasionally wander off from its preferred path as indicated by the transition rule as explained in Section 5.1.3.

The values of parameters listed in Table 5.1 are configured such that the solution space can be effectively explored or exploited. The proposed BCO algorithm can act as a more explorative algorithm if P_{follow} is increased. On the other hand, the BCO algorithm will be more exploitative algorithm if P_{follow} is decreased. How to obtain a balance between exploration and exploitation via the tuning of parameters listed in Table 5.1 is one of the studies which needs further investigation.

After a bee has decided to observe and follow a dance, the next step is to select a dance from a pool of dances shown by its hive mates. In the BCO algorithm, a random selection is applied whereby a bee will randomly pick one dance from the pool of dances to follow. Other selection strategies can be implemented such as the Most Recently Added (MRA), the Round Robin (RR) and the Roulette Wheel (RW). In the MRA, the dance which is shown recently is of the bee's interest. In the RR, every dance shown by its hive mates is selected one after another in succession. In the RW, a dance with higher profitability has a higher chance to be picked.

5.4.2 Waggle Dance Performance

After a local search procedure, a dance will be performed by the bee to other hive mates according to the policy as follows. Only bees that own a better solution compared to its own previous best solution are allowed to dance. Thus, bees in the BCO algorithm are equipped with the ability to remember their previous solution with the “*bestCost*” (as shown in Algorithm 2) obtained during the algorithm execution.

If a bee dances, the waggle dance will last for a certain duration. The dance duration of bee i , D_i is computed by a linear function (see Eq. 5.6). This equation shows that if a

bee has a higher Pf_i , it will dance longer. K is a user-defined scaling factor that controls the magnitude of the dance duration. The experiments to configure the value for K can be found in Section 7.2.

$$D_i = K \cdot \frac{Pf_i}{Pf_{colony}} \quad (\text{Eq. 5.6})$$

When a dance expires, it will be discarded. In terms of implementation, waggle dances are accumulated in a vector list, ES . The list will be refreshed and updated after every bee cycle such that expiry entries are discarded. After many iterations, this might create a situation where no bee is performing waggle dance, due to the failure of bees to get better solutions. This situation should be avoided as waggle dance is an important communication tool among bees. Thus, a memory reset policy is used to avoid such situation (corresponds to the `resetBestCost()` function in Algorithm 2). The aim of having this self-regulated mechanism in the BCO algorithm is to maintain a balance and stable situation such that waggle dances are performed constantly. The policy suggests that when no bee dances for φ consecutive cycles (e.g. $\varphi = 100$), the *bestCost* of each bee in the population will be reset.

The situation where no further waggle dance is observed indicates that the algorithm execution is trapped in a local optimum. Similar situation is observed in the Genetic Algorithm execution. When the Genetic Algorithm is executed over time, the search can sometimes be trapped in a local optima and hence a random search is integrated. This is to avoid the execution from being trapped in a local optimum. The random search in the Genetic Algorithm and the proposed memory reset policy in the BCO algorithm share a common point where they prevent the search from being trapped in a local optimum. However, in terms of the working mechanism, they are different.

5.5 Summary

In this chapter, two state transition rules (i.e. the SSTR and the FSTR) are described. They are part of the foraging process where bees construct a set of feasible solutions for a problem being solved. The main feature of the SSTR is that it allows a node to be added as part of π at a time. Decision is made under the influence of arc fitness and heuristic distance. The added node is stochastically picked rather than deterministically via a greedy approach. However, the SSTR may not be scalable for problems with a larger dimension due to its time consuming construction process whereby the construction is performed via a step-by-step pattern.

To overcome this, the FSTR is proposed. The main feature of the FSTR is to add a fragment of nodes at one time. This leads to a lower computational cost while maintaining the advantage of picking a fragment of nodes via a stochastic process. The fragments reconnection process is repeated until a complete solution is obtained.

The proposal to replace the SSTR with the FSTR serves as a showcase that illustrates the flexibility of altering and enhancing the generic BCO framework. Once the FSTR is added into the framework as one of the domain independent classes, the effect of the FSTR is applicable to all problems from different domains (e.g. TSP, QAP and JSSP). Results that shows the improvement of this replacement are presented in Chapter 7.

The reinforcement of past solutions and the waggle dance model in the BCO algorithm has also been described. Bees are equipped with a memory to remember a set of past solutions so that they can be reinforced in its future foraging process. The waggle dance model implemented in the proposed algorithm is similar to a bee colony in nature. The aim is to ensure that more bees are recruited towards the newly discovered food source. By doing so, more bees will intensively explore the promising area and hence produce a better solution.

The waggle dance model is implemented as follows. A bee would observe dances before leaving its hive to start the foraging process. Once the bee has decided to follow a dance, it would pick a dance from a pool of dances to follow. If the bee successfully found a better solution when compared to its past experience, it would start to dance upon returning to its hive. The dance would last for a longer duration if the solution is much better than the average profitability of the entire colony.

In the subsequent section, a pruning strategy is proposed as part of the BCO framework. The objective of the pruning strategy is to selectively allow a subset of bees to undergo a local search procedure. As not all bees are allowed to be locally optimized by a local search procedure, this would reduce the high computational overhead of a local search procedure.

Chapter 6

Frequency-based Pruning Strategy

In Chapter 4.4.3, it is mentioned that the BCO algorithm is integrated with a local search method and this method is implemented as a domain specific class. The local search acts on a feasible solution π with the aim to identify a better neighbourhood solution. However, if each and every solution generated by the artificial bees undergoes the local search optimization, this will lead to high computational cost. To reduce the computational overhead, a pruning strategy is proposed so that only a subset of solutions are allowed to undergo the local optimization.

The organization of this chapter is as follows. Section 6.1 provides some background information of the pruning strategy especially the usage of building blocks in the model. Section 6.2 describes the mechanism of the frequency-based pruning strategy and shows how its two parameters are statically configured. This strategy is named as Static Frequency-based Pruning Strategy (SFBPS). To overcome the limitation of the SFBPS where a static parameter configuration is involved, a dynamic parameter configuration is proposed in Section 6.3, namely: the Dynamic Frequency-based Pruning Strategy (DFBPS). Finally, this chapter ends with a summary.

6.1 The Building Block

Before looking at a specific example of the pruning strategy, some background information of the strategy is described here. A solution π consists of a string of numbers. If π is dissected into multiple blocks, each of them consists of a substring of numbers. Hence, we can say that π is made up of a set of building blocks. The proposed pruning strategy prohibits a solution π from undergoing a local optimization by comparing π with the common building blocks which are identified throughout the foraging process.

The building block concept is prevalent in many disciplines. One of them is the Genetic Algorithms (GA) in computer science. In GA, some commonly used encoding schemes are as follows: binary, gray code, fixed or floating numbers etc. Although the listed encoding schemes are different from each other, they are similar in terms of the usage of a block of numbers to encode a chromosome. These chromosomes are then further transformed by genetic operators such as crossover and mutation, according to the concept of evolution: survival of the fittest.

Gero et al. conducted a study on the building blocks in GA chromosomes [58, 59]. A population of chromosomes is segregated into two groups, one above a threshold (better fitness), and the other one below the threshold (lower fitness). They proposed a mechanism to identify prevalent building blocks in the better fitness group and absent building blocks in the lower fitness group. The identified building blocks are then combined as a single gene and they are prevented from further modifications by crossover and mutation. The proposed approach was tested on a space layout planning problem and results showed that if these blocks are frequently reused in the evolutionary process, a significant reduction of computational time can be achieved.

Wu and Lindsay [158] investigated the effects of location-independent building blocks in GA. Urquhart et al. [147] proposed a building block mutation for GA to solve the

street based routing problem. Both techniques ensure a frequent reusability of prevalent building blocks in the evolutionary process of GA.

Gao et al. proposed an Ant Colony Optimization algorithm with Association Rule discovery for TSP [54]. This method is able to identify a set of important sequential patterns (building blocks) found in the permutations of TSP solutions with a certain minimum support. These identified blocks will have a higher odd to be utilized in its optimization operations. A 30-city TSP with known optimal was tested and the best result showed a deviation of 0.95% from the optimal.

All the works mentioned in this section describe the usefulness of building blocks. By identifying and utilizing them frequently, improvement to the TSP solutions was achieved.

6.2 Static Frequency-based Pruning Strategy (SFBPS)

The proposed pruning strategy presented in this section is the Static Frequency-based Pruning Strategy (SFBPS). This strategy is based on the accumulated frequency of the smallest building blocks (two-element blocks) of every solution π generated by bees during the foraging process.

Consider the following example that explains the working mechanism of the SFBPS. Firstly, a $|\pi| \times |\pi|$ matrix is created with each entry of the matrix assigned to zero. If a solution π is with the cardinality of six, $|\pi| = 6$, a matrix H with the size of 6×6 is

created as follows¹:

$$H = \begin{array}{c|cccccc} & A & B & C & D & E & F \\ \hline A & 0 & 0 & 0 & 0 & 0 & 0 \\ B & 0 & 0 & 0 & 0 & 0 & 0 \\ C & 0 & 0 & 0 & 0 & 0 & 0 \\ D & 0 & 0 & 0 & 0 & 0 & 0 \\ E & 0 & 0 & 0 & 0 & 0 & 0 \\ F & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

The purpose of creating H is to record the accumulated frequency of the smallest building blocks of every solutions generated by bees. In this case, the smallest building block contains two elements. If a solution $\pi = \{A, B, C, D, E, F\}$ is considered, H will be updated according to the block configurations A-B, B-C, C-D, D-E and E-F as follows:

$$H = \begin{array}{c|cccccc} & A & B & C & D & E & F \\ \hline A & 0 & 1 & 0 & 0 & 0 & 0 \\ B & 0 & 0 & 1 & 0 & 0 & 0 \\ C & 0 & 0 & 0 & 1 & 0 & 0 \\ D & 0 & 0 & 0 & 0 & 1 & 0 \\ E & 0 & 0 & 0 & 0 & 0 & 1 \\ F & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

If another solution π with a permutation of $\pi = \{A, B, E, D, C, F\}$ is added, H becomes:

$$H = \begin{array}{c|cccccc} & A & B & C & D & E & F \\ \hline A & 0 & 2 & 0 & 0 & 0 & 0 \\ B & 0 & 0 & 1 & 0 & 1 & 0 \\ C & 0 & 0 & 0 & 1 & 0 & 1 \\ D & 0 & 0 & 1 & 0 & 1 & 0 \\ E & 0 & 0 & 0 & 1 & 0 & 1 \\ F & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

The SFBPS considers only the smallest building blocks in a solution, but it can be extended to consider building blocks in different sizes e.g. three-element blocks and so on.

¹The example uses permutations of six alphabets $\{A, B, C, D, E, F\}$ rather than permutations of numbers for π .

The following example illustrates the use of building blocks in different sizes (i.e. three-element and four-element blocks) with respect to a particular solution $\pi = \{A, B, C, D, E, F, G\}$. When three-element blocks are considered, 10 blocks A-BC, AB-C, BC-D, B-CD, C-DE, CD-E, D-EF, DE-F, E-FG and EF-G are identified. When four-element blocks are considered, 12 blocks A-BCD, AB-CD, ABC-D, B-CDE, BC-DE, BCD-E, C-DEF, CD-EF, CDE-F, D-EFG, DE-FG and DEF-G are identified. If these two approaches are compared to the approach which considers two-element blocks², the computational cost in terms of frequency update is increased by 67% (three-element blocks) and 100% (four-element blocks). Although an increment in terms of computational cost is noticed with the increment of building block size, this might lead to a better result as some blocks has to be tied together to give an optimum solution. Also, the corresponding storage cost will increase with the increase of block size. This topic will be one of our future studies where a trade-off investigation between accuracy and computational cost (e.g. frequency update cost and storage structure) would be conducted. A sensitivity analysis would be carried out as well.

Based on the two-element blocks frequency update approach, H might end up with the following entries after several cycles of update:

	A	B	C	D	E	F
A	0	88	929	22	113	23
B	88	0	754	355	105	4
$H =$ C	929	754	0	11	826	2
D	22	355	11	0	176	933
E	113	105	826	176	0	56
F	23	4	2	933	56	0

By examining H , building blocks with high frequency of access are identified. Taking the first row of H as an example, where it records the frequency of the building blocks

²Six blocks are identified: A-B, B-C, C-D, D-E, E-F and F-G.

Figure 6.1: An adjacency list that shows the building blocks that are greater than $\eta = 5\%$ of the total row-based frequency. These blocks are marked as hot spots.

with A as the first element $\{AB, AC, AD, AE, AF\}$. It has a total frequency of 1175 (summation of 88, 929, 22, 113 and 23). The building block of AA is disregarded as the system does not allow any recursive visit. Of these five building blocks, AB contributes 7.49% of the total occurrences ($\frac{88}{1175} \times 100\%$). AC, AD, AE and AF contribute 79.06%, 1.87%, 9.62% and 1.96% respectively. Similar calculation is carried out on the rest of the entries in H , and the following matrix is obtained:

	A	B	C	D	E	F
A	0.00	7.49	79.06	1.87	9.62	1.96
B	6.74	0.00	57.73	27.18	8.04	0.31
C	36.84	29.90	0.00	0.44	32.75	0.08
D	1.47	23.71	0.73	0.00	11.76	62.32
E	8.86	8.23	64.73	13.79	0.00	4.39
F	2.26	0.39	0.20	91.65	5.50	0.00

Based on $H_{\%} = \{h_{i,j}\}$, matrix elements that are greater than or equal to the hot spots inclusion threshold ($h_{i,j} \geq \eta\%$) will be marked as hot spots. For instance, if $\eta = 5\%$, AB, AC and AE are the hot spots with A as its first element (as shown in Fig. 6.1).

The identified hot spots will then be employed in the decision making process. The pruning criterion is set as follows: if the dissimilarity of a solution π , $\Delta\pi$ is greater than or equal to a pre-defined value ($\Delta\pi \geq \kappa\%$), when compared to the identified hot

spots (refer to Fig. 6.1), the solution will then be prohibited from undergoing a local optimization. Otherwise, it will be further optimized by a local search method. If a permutation of $\pi = \{A, D, C, F, B, E\}$ is compared against Fig. 6.1, the building blocks of AD, DC, CF and FB are not hot spots. Only BE is hot spot. If κ is configured at 25%, this permutation will be pruned from a local optimization as 80% of the building blocks appear to be dissimilar. If another permutation of $\pi = \{A, D, F, E, C, B\}$ is considered, it will be accepted to perform local optimization as only one dissimilar building blocks is observed ($20\% < \kappa$ where $\kappa = 25\%$). κ is a user-defined pruning threshold.

η and κ are user-defined threshold values. In our previous work [156], η and κ are tuned empirically through a set of experiments on the TSP domain. The tuning experiment was based on the LIN318 from the TSPLIB³. Further details of the η and κ configuration experiments can be found in [156]. The results from the tuning experiments for η and κ were statically applied throughout the entire algorithm execution. Also, similar setting of η and κ was applied on the rest of the TSP benchmark problems with different dimensions. This static tuning approach has a limitation whereby it assumes the η and κ are problem independent. This might not be the case if different values of η and κ are needed for different problems with different dimensions. To avoid this, the DFBPS is proposed and it is discussed in the subsequent section (refer to Section 6.3).

6.3 Dynamic Frequency-based Pruning Strategy (DFBPS)

As mentioned in Section 6.2, the configuration of η and κ is done statically. In this section, an dynamic configuration of these two parameters is proposed. By incorporating a problem and domain independent tuning approach into the BCO framework, tedious work of setting η and κ when solving a new domain (e.g. QAP or JSSP) would be

³Details on the TSPLIB can be found in Section 2.4.

avoided. The Dynamic Frequency-based Pruning Strategy (DFBPS) discussed in this section corresponds to the `DynamicFBPS` class in Fig. 4.1 (refer to Section 4.2) and the `prune()` function in Algorithm 2 (refer to Section 4.3).

The dynamic computation of η and κ works as follows. Assume that each of the solution π generated by bees is of size x , which mean $|\pi| = x$. Thus, the size of $H_{\%} = \{h_{i,j}\}$ will be $x \times x$. The mean and standard deviation of all non-diagonal entries (entries with $i \neq j$) in the $H_{\%}$ are computed according to Eq. 6.1 and Eq. 6.2. As $H_{\%}$ is a square matrix, the mean and standard deviation computations will be based on $x^2 - x$ entries.

$$\mu_{H_{\%}} = \frac{\sum_{i=1}^x \sum_{j=1}^x h_{i,j}}{x(x-1)}, \forall i \neq j \quad (\text{Eq. 6.1})$$

$$\sigma_{H_{\%}} = \sqrt{\frac{\sum_{i=1}^x (h_{i,j} - \mu_{H_{\%}})^2}{x(x-1)}}, \forall i \neq j \quad (\text{Eq. 6.2})$$

The hot spots inclusion threshold, η is configured at $\eta = \mu_{H_{\%}} + \sigma_{H_{\%}}$. This value is obtained after conducting a series of tuning experiments (e.g. $\eta = \mu_{H_{\%}} \pm 1\sigma_{H_{\%}}$, $\eta = \mu_{H_{\%}} \pm 2\sigma_{H_{\%}}$ etc). This implies that any entry in the matrix with $h_{i,j} \geq \mu_{H_{\%}} + \sigma_{H_{\%}}$ will be considered as a hot spot. Intuitively, these hot spots are frequent building blocks in most of the solutions generated by the bees.

If the $H_{\%}$ in Section 6.2 is referred, the computations of $\mu_{H_{\%}}$ and $\sigma_{H_{\%}}$ are as follows:

$$\begin{aligned} \mu_{H_{\%}} &= \frac{\sum_{i=1}^x \sum_{j=1}^x h_{i,j}}{x(x-1)}, \forall i \neq j \\ &= \frac{600.00\%}{6(6-1)} \\ &= 20.00\% \\ \sigma_{H_{\%}} &= \sqrt{\frac{\sum_{i=1}^x (h_{i,j} - \mu_{H_{\%}})^2}{x(x-1)}}, \forall i \neq j \\ &= \sqrt{\frac{19479.95}{6(6-1)}}\% \\ &= 25.48\% \end{aligned}$$

$\mu_{H\%}$ and $\sigma_{H\%}$ hold a value of 20.00% and 25.48% respectively. Hence, $\eta = 20.00\% + 25.48\% = 45.48\%$. Entries of $H\%$ which are greater than or equal to η will be marked as hot spots. Building blocks of AC, BC, DF, EC and FD are marked as hot spots.

On the other hand, κ is a moving average of $\Delta\pi$ generated by the last Ψ bees (i.e. $\Psi = 100$) at a particular moment during the algorithm execution. It is defined as in Eq. 6.3:

$$\kappa = \frac{\sum_{i=1}^{\Psi} \Delta\pi}{\Psi} \quad (\text{Eq. 6.3})$$

6.4 Summary

In this chapter, a pruning strategy which is based on the frequency of the smallest building block of every solution π is proposed as part of the BCO algorithm. Firstly, the SFBPS is described. The SFBPS allows only a subset of solutions to be locally optimized by a local search method in order to reduce the high overhead of the local optimization. A limitation of the SFBPS is the static configuration of two parameters found in the model, which are η and κ . η is the hot spots inclusion threshold whereas κ is the solution dissimilarity threshold. These two threshold values are used to decide which solution to be pruned from a local optimization.

To overcome the limitation, the DFBPS is proposed where the η and κ are dynamically tuned. By doing so, the DFBPS is able to act on problems from different domains. Hence, it makes the pruning strategy more robust.

The replacement of the SFBPS with the DFBPS serves as another showcase that illustrates the flexibility to alter the generic BCO framework. The effect of the replacement is applicable to all COPs discussed in this thesis, namely: the TSP, QAP and JSSP.

In the subsequent chapter, we will carry out a set of experiments based on the topics discussed in Chapters 4 to 6 and discuss the experimental results.

Chapter 7

Findings and Results

This chapter presents the findings and results of the proposed BCO algorithm on the three selected COPs (i.e. TSP, QAP and JSSP). The organization of this chapter is as follows. Firstly, we describe the key performance indicators used in measuring the performance of the proposed algorithm (refer to Section 7.1). It is followed by a discussion on the BCO parameters configuration in Section 7.2. In Section 7.3, we describe the symbols and notations used in presenting the results of the proposed BCO algorithm on the three selected problems. Also, Section 7.3 describes the experimental platform and setup for this research. The results of the BCO algorithm on TSP, QAP and JSSP are presented in Sections 7.4, 7.5 and 7.6 respectively. This is followed by a discussion on the comparison studies (refer to Section 7.7). Finally, a summary is included to end this chapter (refer to Section 7.8).

7.1 Key Performance Indicators

In order to measure the performance of the proposed BCO algorithm, two key performance indicators are defined as follows: percentage deviation from known optimum/upper bound δ (measured in %) and computational time to obtain the best score (measured in seconds).

If the optimal score or the upper bound of an instance is denoted by C^* and the score of a resulting solution generated by the BCO algorithm is denoted by $C(\pi)$, then δ is computed as follows:

$$\delta = \frac{C(\pi) - C^*}{C^*} \times 100\% \quad (\text{Eq. 7.1})$$

7.2 BCO Parameters Configuration

The proposed BCO algorithm is equipped with five user-defined parameters, namely: β , λ , K , N_{Bee} and ϖ . Here is a brief description for each of them. β is a parameter ($\beta \in \mathbb{Z}^+$) in Eq. 5.2 that controls the significance level of heuristic distance factor (refer to Section 5.1.3). The higher the value of β , the more significant the influence of heuristic distance in the state transition rule. λ is a value that is assigned to the arc that connects two nodes as suggested by the preferred path (refer to Eq. 5.3 in Section 5.1.3). It is a positive bounded parameter with values of $0 \leq \lambda \leq 1$. K is a scaling parameter that controls the magnitude of the dance duration (refer to Eq. 5.6 in Section 5.4.2). K is a bounded parameter which is a positive integers $K \in \mathbb{Z}^+$. N_{Bee} denotes the number of bees being used in the proposed BCO algorithm (refer to Section 4.3). The range of N_{Bee} is between 1 and n (both inclusive), where n represents the dimension of the problem being solved. Finally, ϖ denotes the capacity of past memory associated with each bee in the proposed BCO algorithm (refer to Section 5.3). The bigger the value of ϖ , the greater the memory of a bee in keeping its past solutions. ϖ is a parameter which takes a positive integer greater than 1 ($\varpi \in \mathbb{Z}^+$).

To tune the parameters in the proposed BCO algorithm, a 2^5 factorial experiment with ten replicates was conducted. The factorial design is a generic parameter tuning approach which can be used across multiple problem domains [36, 126, 128]. When the proposed BCO algorithm is applied to a new problem domain, a representative/famous benchmark problem in that domain can be employed in the parameters tuning experiments using

Table 7.1: The high (+1) and low (-1) levels of the BCO parameters for the three selected COPs. They are used in the parameter tuning experiments.

Factors	TSP		QAP		JSSP	
	(-1)	(+1)	(-1)	(+1)	(-1)	(+1)
β	1	10	1	10	1	10
λ	0.1	0.9	0.1	0.9	0.1	0.9
K	10	100	10	100	10	100
N_{Bee}	10	50	10	50	10	25
ϖ	10	25	10	25	10	25

the factorial design. After obtaining parameter setting results, the results can then be applied to other problem instances with similar characteristics within the same domain.

In this thesis, for every selected problem (i.e. TSP, QAP and JSSP), each factor is configured at either high (+1) or low (-1) level as shown in Table 7.1. This creates a total of 32 different parameter settings as illustrated in Table 7.2. The high (+1) and low (-1) values listed in Table 7.1 are obtained after they are tuned empirically. In such empirical tuning experiments, factors such as computational cost, accuracy performance and expert's past experience are considered. Generally, the higher the values of these five parameters, the higher the computational cost. Some existing works have demonstrated the use of an optimization algorithm (e.g. GA, data mining approach) to optimally tune parameters in metaheuristics [44, 93, 162]. We will apply some of these techniques in our future work to investigate how an optimization algorithm can be applied to optimally determine the upper and lower bounds for the parameters listed in Table 7.1.

For each configuration i in Table 7.2, ten replicates of algorithm executions were conducted. Each replicate produces a solution score. The average score of ten replicates is denoted as μ_i and it is computed as follows where x_{ji} denotes the score for j -th replicate of configuration i :

$$\mu_i = \frac{\sum_{j=1}^{10} x_{ji}}{10}; i = 1, \dots, 32; j = 1, \dots, 10$$

Table 7.2: $2^5 = 32$ different parameter settings for the proposed BCO algorithm.

Config.	β	λ	K	N_{Bee}	ϖ	Config.	β	λ	K	N_{Bee}	ϖ
$i = 01$	-1	-1	-1	-1	-1	$i = 17$	-1	-1	-1	-1	+1
$i = 02$	+1	-1	-1	-1	-1	$i = 18$	+1	-1	-1	-1	+1
$i = 03$	-1	+1	-1	-1	-1	$i = 19$	-1	+1	-1	-1	+1
$i = 04$	+1	+1	-1	-1	-1	$i = 20$	+1	+1	-1	-1	+1
$i = 05$	-1	-1	+1	-1	-1	$i = 21$	-1	-1	+1	-1	+1
$i = 06$	+1	-1	+1	-1	-1	$i = 22$	+1	-1	+1	-1	+1
$i = 07$	-1	+1	+1	-1	-1	$i = 23$	-1	+1	+1	-1	+1
$i = 08$	+1	+1	+1	-1	-1	$i = 24$	+1	+1	+1	-1	+1
$i = 09$	-1	-1	-1	+1	-1	$i = 25$	-1	-1	-1	+1	+1
$i = 10$	+1	-1	-1	+1	-1	$i = 26$	+1	-1	-1	+1	+1
$i = 11$	-1	+1	-1	+1	-1	$i = 27$	-1	+1	-1	+1	+1
$i = 12$	+1	+1	-1	+1	-1	$i = 28$	+1	+1	-1	+1	+1
$i = 13$	-1	-1	+1	+1	-1	$i = 29$	-1	-1	+1	+1	+1
$i = 14$	+1	-1	+1	+1	-1	$i = 30$	+1	-1	+1	+1	+1
$i = 15$	-1	+1	+1	+1	-1	$i = 31$	-1	+1	+1	+1	+1
$i = 16$	+1	+1	+1	+1	-1	$i = 32$	+1	+1	+1	+1	+1

This creates a set of 32 average values and let's name this set as Y . The lowest average $\mu_{min} \in Y$ was identified and this would be the reference point in the comparison of μ_{min} against the rest of averages in $Y = \{\mu_1, \mu_2, \dots, \mu_{32}\}$ by using the 2-Sample T-Test. In another word, for every entry of $\mu_i \in Y$, the following hypothesis testing was conducted with 95% confidence interval:

$$H_0 : \mu_i = \mu_{min} ; H_\alpha : \mu_i \neq \mu_{min}$$

The aim of the hypothesis testing is to identify a set of average values in Y which is statistically equal to μ_{min} .

After conducting the 2-Sample T-Test on every entry of $\mu_i \in Y$, the instances with a conclusion of “fail to reject the null hypothesis H_0 ” were identified. The identification

was performed with 5% statistical significance level. This means that any instance with P-value less than 0.05 is statistically significant and hence causing the rejection of H_0 . If z entries in Y were found to be statistically equal to μ_{min} , the parameter configuration of these z entries would be grouped together for a bit-by-bit mode comparison. The bit-by-bit mode comparison will be explained via an example as follows. Let's consider $z = 4$ entries in Y were found to be statistically equal to μ_{min} , namely: μ_{30} , μ_{27} , μ_{13} and μ_{25} . Each of this entry is with a parameter configuration as shown as follows where β , λ , K , N_{Bee} and ϖ are the five factors being examined:

	β	λ	K	N_{Bee}	ϖ
μ_{30}	+1	-1	+1	+1	+1
μ_{27}	-1	+1	-1	+1	+1
μ_{13}	-1	-1	+1	+1	-1
μ_{25}	-1	-1	-1	+1	+1
<i>Mode :</i>	-1	-1	+1/-1	+1	+1

Based on these four entries, the mode of every factor (with the highest frequency) is identified. For instance, the mode of factor β is -1 (frequency of -1 is 3 and frequency of +1 is 1), the mode of factor λ is -1, the mode of factor K is either +1/-1 (bimodal as the frequencies of -1 and +1 are 2), the mode of factor N_{Bee} is +1 and the mode of factor ϖ is +1. After the bit-by-bit mode comparison, the result indicates that we should configure factors β and λ at low level (due to the modes for β and λ are -1), factor K at either low or high level (due to the bimodal factor) and factors N_{Bee} and ϖ at high level (due to the modes for N_{Bee} and ϖ are +1). If these low and high levels were to be converted to numerical values, the values correspond to the high and low levels indicated in Table 7.1 could be applied.

In this illustrative example, which is not tied to any selected problem domain, β and λ are configured at low level. This indicates that a bee in the proposed algorithm would be able to generate a relatively better solution if the influence of heuristic distance and

preferred path is minimal. When these two parameters are configured at low level, the effect of β and λ is reduced. By lowering down the effect of β and λ , bees are able to explore other areas in the solution space as they may wander off from its preferred path or the path with heuristic distance influence. The high level of K indicates that a bee would dance longer and thus the likelihood to attract more bees to visit the surrounding area of the newly discovered food source is increased. In this particular illustrative example, both low and high level of factor K (bimodal) play equally important role to generate relatively better solutions. Also, the high level of N_{Bee} indicates that better performance is observed if more bees are employed to search for the best solution in the solution space. The high level of ϖ indicates that a bee is able to memorize more past solutions and thus it has more past solutions to refer to in the past solutions reinforcement strategy.

Interpretation and justification of the parameter settings for the illustrative example explained in this section can be used to analyze the experimental results of parameter settings for TSP, QAP and JSSP, where the results are presented in Sections 7.4.1, 7.5.1 and 7.6.1.

7.3 Symbols, Notations, Experimental Platform and Setup

This section describes the symbols and notations used in presenting the results of the proposed BCO algorithm on the three selected problems. Also, it describes the experimental platform and setup for this research.

When a benchmark problem with an optimum score or an upper bound C^* is solved by the proposed BCO algorithm, a total of x replications will be conducted. The score produced by each replication and the computational time to obtain such score are recorded. This leads to the creation of $C = \{c_1, c_2, \dots, c_x\}$ and $T = \{t_1, t_2, \dots, t_x\}$ where C is a set of scores and T is a set of computational time corresponding to the time to obtain each

Figure 7.1: Experimental setup for testing the effect of the major components of the proposed BCO algorithm.

score in C . For set C , the minimum and maximum entries as well as the average and standard deviation of x scores are identified. They are denoted as C_{min} , C_{max} , μ_C and σ_C respectively. Eq. 7.1 is then applied to C_{min} , C_{max} , μ_C so that the deviation percentage from C^* are computed and they are denoted as δ_{min} , δ_{max} and δ_{avg} respectively. For set T , the average and standard deviation of computational time are computed and they are denoted as μ_T and σ_T . μ_T is the average time to obtain the best score in each replication.

The experiments reported in this chapter were conducted on a Windows OS cluster environment with 2GHz Intel processors unless it is stated otherwise. At any particular time, one processor was restricted to run one test only.

In order to examine the effect of the major components of the proposed BCO algorithm, four different BCO variants are developed. These four BCO variants are denoted as Λ_1 , Λ_2 , Λ_3 and Λ_4 as illustrated in Fig. 7.1. All these four BCO variants are integrated with a local search mechanism whereby its implementation is not part of the proposed framework and the effect is contributed via the `LocalOptimizer` abstract class (refer to Fig. 4.1). The aim of the experiments illustrated in Fig. 7.1 is to compare the effectiveness and efficiency of the major components within the proposed framework (i.e.

path construction mechanisms and pruning strategies) but not the components outside the framework (i.e. local search mechanisms). Therefore, the experiments which examine the BCO variants with or without local search integration are not included here.

The implementation of a local search mechanism is domain dependent. An effective and efficient local search mechanism for a problem domain might not be applicable or having the same effect when it is applied to another problem domain due to the different neighbourhood structure. For instance, the Lin-Ker heuristic [97], which is very effective on the TSP, cannot be applied to the JSSP. Also, within a problem domain (e.g. TSP, QAP or JSSP), some local search algorithms are superior to the others [5]. All these factors contribute to the decision whereby the experiments which examine the local search effect are not included. Therefore, for each problem domain, a local search method is integrated for a fair comparison.

In Fig. 7.1, variants Λ_1 and Λ_2 form the first phase of the experiment, with the aim to examine different path construction mechanisms in the proposed BCO algorithm. Variant Λ_1 is a BCO algorithm with the SSTR as its path construction mechanism. Variant Λ_2 is a BCO algorithm with the FSTR as its path construction mechanism. Both variants Λ_1 and Λ_2 employ a local search (LS) procedure as the local optimization engine and every bee is allowed to perform the local search operation (i.e. no pruning strategy). Variants Λ_1 and Λ_2 are denoted as BCO_SSTR+LS and BCO_FSTR+LS respectively.

Variants Λ_3 and Λ_4 form the second phase of the experiment, with the aim to examine different pruning strategies in the proposed BCO algorithm. Both variants Λ_3 and Λ_4 employ the FSTR and a local search (LS) procedure as the path construction mechanism and the local optimization engine respectively. The difference between these two variants is that variant Λ_3 uses the SFBPS and variant Λ_4 uses the DFBPS as the pruning strategy. The pruning strategy allows only a subset of resulting solutions to undergo the local search operation. Variants Λ_3 and Λ_4 are denoted as BCO_FSTR+LS+SFBPS and BCO_FSTR+LS+DFBPS respectively as shown in Fig. 7.1.

Table 7.3: Different BCO variants for the TSP, QAP and JSSP. The variants are to test the effect of different components in the proposed BCO algorithm.

Domains	BCO Variants
TSP	Λ_1 : BCO_SSTR+FRNN_2-opt_TL
	Λ_2 : BCO_FSTR+FRNN_2-opt_TL
	Λ_3 : BCO_FSTR+FRNN_2-opt_TL+SFBPS
	Λ_4 : BCO_FSTR+FRNN_2-opt_TL+DFBPS
QAP	Λ_1 : BCO_SSTR+2-opt_TL
	Λ_2 : BCO_FSTR+2-opt_TL
	Λ_3 : BCO_FSTR+2-opt_TL+SFBPS
	Λ_4 : BCO_FSTR+2-opt_TL+DFBPS
JSSP	Λ_1 : BCO_SSTR+TESN5_TL
	Λ_2 : BCO_FSTR+TESN5_TL
	Λ_3 : BCO_FSTR+TESN5_TL+SFBPS
	Λ_4 : BCO_FSTR+TESN5_TL+DFBPS

Since the local search procedure is specific to a problem domain being solved (refer to Fig. 4.2 in Section 4.4.3), variants Λ_1 , Λ_2 , Λ_3 and Λ_4 for the TSP, QAP and JSSP are integrated with different local search procedures and they are denoted by the entries shown in Table 7.3. The results which shows the effect of the major components within the proposed framework are presented in Sections 7.4.2, 7.5.2 and 7.6.2.

7.4 Performance of the BCO Algorithm on TSP

This section presents the results and findings of the proposed BCO algorithm on a set of TSPLIB benchmark problems. A total of 95 problem instances with dimensions between [14, 2392] are used to test the proposed BCO algorithm. These problem instances are selected as they appear in many TSP related literatures including the literatures reviewed in Chapter 3.

The chosen instances from the TSPLIB follow a naming notation which starts with

alphabetical characters (denote the origin of the problem) and followed by a numerical figure. The numerical figure denotes the dimension of the problem. For example, ATT48 is a 48-city problem from the ATT series; LIN318 is a 318-city problem from the LIN series and so forth. Table 7.8 lists all the 95 problem instances that have been tested using the proposed BCO algorithm.

Besides the dimension, the 95 instances taken from the TSPLIB are different from each other in terms of data file presentation and format. The data file presentation and format are dictated by the distance computation method. Four approaches described in Section 4.4.2 are developed, namely: the 2D Euclidean distance, pseudo-Euclidean distance, geographical distance and matrix with pre-defined distances. Further details on the TSPLIB can be obtained in Section 2.4.

7.4.1 Parameters Tuning for TSP

In the parameter tuning experiment for TSP, the RAT575 problem from the TSPLIB¹ is used as the sample. Table 7.1 in Section 7.2 illustrates the five BCO parameters with their high (+1) and low (-1) levels for this experiment. The BCO algorithm is executed for 3000 iterations, that is $BC_{Max} = 3000$.

Table B.1 in Appendix B presents the experiment results for each configuration. The table is sorted according to the average tour length of each configuration. From Table B.1, $\mu_{min} = \mu_{29} = 6824.5$ and it is obtained via the configuration $i = 29$. This μ_{min} will be used in the hypothesis testing so that the parameter setting for the proposed BCO algorithm is determined.

The hypothesis testing results for the RAT575 problem are shown in Table B.2 (refer to Appendix B). The table illustrates the P-value of F-Test and T-Test together with the 95% confidence interval (CI) for difference between μ_i and μ_{min} . The F-Test was

¹Details about the TSPLIB are discussed in Section 2.4.

Table 7.4: Recommended BCO parameter settings for the RAT575.

Config.	β	λ	K	N_{Bee}	ϖ
$i = 29$	-1	-1	+1	+1	+1
$i = 25$	-1	-1	-1	+1	+1
$i = 16$	+1	+1	+1	+1	-1
$i = 14$	+1	-1	+1	+1	-1
Mode	-1/+1	-1	+1	+1	-1/+1

conducted to check if the standard deviation of configuration i is statistically equal to the standard deviation of the configuration with the lowest average.

From Table B.2, the T-Test shows that μ_{25} , μ_{16} and μ_{14} (they are in bold and italic font which their P-value is > 0.05) are statistically equal to $\mu_{min} = \mu_{29}$. After performing a bit-by-bit comparison as explained in Section 7.2, the recommended parameter setting for the RAT575 is shown in Table 7.4. We assume that the setting of these five parameters shown in Table 7.4 is applicable to all problems in the TSP domain. This parameter setting, which is equivalent to the configuration $i = 29$ in Table 7.2, will be applied throughout the TSP experiments. The TSP experimental results will be described in Section 7.4.2.

7.4.2 Results for TSP

Before the results of the entire 95 TSPLIB problem instances are presented, the results that show the effect of the major components of the proposed BCO algorithm are described. Four variants of the BCO algorithm for the TSP as shown in Fig. 7.1 and Table 7.3 are created. A total of 20 selected problems with dimensions of [101, 532] were employed to test the BCO algorithm.

The recommended BCO parameter setting for TSP as described in Section 7.4.1 was applied: $\beta = 1$, $\lambda = 0.1$, $K = 100$, $N_{Bee} = 50$ and $\varpi = 25$. The BCO algorithm was

Table 7.5: Effect of different variants of the BCO algorithm on 20 selected TSPLIB benchmark problems.

Inst.	Opt.	Λ_1			Λ_2			Λ_3			Λ_4		
		δ_{min}	δ_{avg}	μ_T (s)	δ_{min}	δ_{avg}	μ_T (s)	δ_{min}	δ_{avg}	μ_T (s)	δ_{min}	δ_{avg}	μ_T (s)
EIL101	629	0.795	1.240	558.30	0.000	0.000	74.95	0.000	0.032	42.67	0.000	0.000	17.73
PR136	96772	0.616	0.753	524.86	0.023	0.054	181.71	0.013	0.040	107.27	0.013	0.018	127.05
KROA150	26524	0.415	0.639	487.83	0.000	0.001	60.51	0.000	0.002	28.99	0.000	0.000	88.42
KROB150	26130	0.111	0.474	470.52	0.000	0.002	95.79	0.000	0.000	28.57	0.000	0.000	20.03
RAT195	2323	2.497	2.910	919.98	0.129	0.232	181.94	0.000	0.224	183.28	0.000	0.198	119.66
D198	15780	0.792	0.866	846.29	0.006	0.032	371.53	0.006	0.023	183.89	0.019	0.072	295.62
KROA200	29368	1.124	1.269	519.28	0.000	0.000	85.25	0.000	0.000	41.98	0.000	0.000	43.98
KROB200	29437	1.124	1.393	782.28	0.000	0.001	261.12	0.000	0.001	158.53	0.000	0.002	160.37
TSP225	3916	2.477	2.835	761.73	0.000	0.000	247.28	0.000	0.015	148.57	0.000	0.000	128.06
GIL262	2378	2.607	3.045	1196.99	0.000	0.034	278.57	0.000	0.034	302.11	0.000	0.000	168.77
A280	2579	2.404	2.831	1282.79	0.000	0.000	85.08	0.000	0.000	102.85	0.000	0.000	105.89
PR299	48191	1.882	2.050	2291.56	0.000	0.005	408.81	0.000	0.013	511.20	0.000	0.029	416.28
LIN318	42029	2.306	2.471	1462.30	0.048	0.126	670.23	0.050	0.129	323.69	0.128	0.159	207.32
RD400	15281	4.005	4.201	3732.13	0.137	0.267	2263.71	0.118	0.242	1445.83	0.190	0.229	1467.13
FL417	11861	1.029	1.106	8386.31	0.076	0.115	1112.70	0.067	0.103	2479.25	0.076	0.130	1171.07
GR431	171414	3.274	3.447	6615.24	0.205	0.424	2681.42	0.166	0.371	2370.28	0.410	0.582	1877.06
PR439	107217	1.761	2.134	5991.03	0.055	0.063	1792.32	0.031	0.071	1963.08	0.031	0.041	1373.00
PCB442	50778	2.842	3.087	1761.30	0.402	0.523	1420.24	0.351	0.489	1564.70	0.313	0.423	866.14
D493	35002	2.948	3.064	12233.29	0.386	0.497	3045.36	0.434	0.493	3207.28	0.157	0.354	1733.04
ATT532	27686	3.634	3.878	10396.57	0.184	0.276	2949.08	0.275	0.349	2088.35	0.267	0.351	2053.40
Average:		1.932	2.185	3061.03	0.083	0.132	913.38	0.076	0.132	864.12	0.080	0.129	622.00

executed for 10000 BC_{Max} for five replications on each benchmark instance. Table 7.5 presents the results for this experiment. In terms of the average of δ_{avg} for each BCO variants (Λ_1 , Λ_2 , Λ_3 and Λ_4), they are 2.185%, 0.132%, 0.132%, 0.129%. This indicates that with the integration of the FSTR and the DFBPS, the results are improved. In terms of the number of instances that have been solved to optimum, Λ_4 solves seven of them to optimum, followed by Λ_2 (four instances) and Λ_3 (three instances). Table 7.5 also shows the average time to obtain the best score for each benchmark problem. It is denoted as μ_T . The average μ_T for Λ_1 , Λ_2 , Λ_3 and Λ_4 are 3061.029s (≈ 51 minutes), 913.380s (≈ 15 minutes), 864.117s (≈ 14 minutes) and 622.000s (≈ 10 minutes) respectively.

Table 7.6 shows the improvement percentage in terms of average solution cost and μ_T for each benchmark instances. A negative percentage implies a reduction in average solution cost or μ_T . A positive percentage implies an increase in average solution cost or μ_T . In general, negative improvement percentages for both average solution cost and μ_T

Table 7.6: Improvement percentage in terms of average solution cost and μ_T among the different BCO variants on 20 selected TAPLIB benchmark problems.

Inst.	Λ_1 VS Λ_2 (%)		Λ_1 VS Λ_3 (%)		Λ_1 VS Λ_4 (%)		Λ_2 VS Λ_3 (%)		Λ_2 VS Λ_4 (%)		Λ_3 VS Λ_4 (%)	
	Cost	μ_T	Cost	μ_T	Cost	μ_T	Cost	μ_T	Cost	μ_T	Cost	μ_T
EIL101	-1.225	-86.58	-1.193	-92.36	-1.225	-96.82	0.032	-43.06	0.000	-76.34	-0.032	-58.45
PR136	-0.694	-65.38	-0.707	-79.56	-0.729	-75.79	-0.014	-40.97	-0.036	-30.08	-0.022	18.44
KROA150	-0.635	-87.60	-0.633	-94.06	-0.635	-81.87	0.002	-52.09	-0.001	46.14	-0.002	205.05
KROB150	-0.470	-79.64	-0.472	-93.93	-0.472	-95.74	-0.002	-70.18	-0.002	-79.09	0.000	-29.88
RAT195	-2.602	-80.22	-2.610	-80.08	-2.635	-86.99	-0.009	0.74	-0.034	-34.23	-0.026	-34.71
D198	-0.827	-56.10	-0.836	-78.27	-0.787	-65.07	-0.009	-50.51	0.041	-20.43	0.049	60.76
KROA200	-1.253	-83.58	-1.253	-91.92	-1.253	-91.53	0.000	-50.76	0.000	-48.40	0.000	4.79
KROB200	-1.373	-66.62	-1.372	-79.73	-1.372	-79.50	0.001	-39.29	0.001	-38.59	0.001	1.16
TSP225	-2.756	-67.54	-2.741	-80.50	-2.756	-83.19	0.015	-39.92	0.000	-48.21	-0.015	-13.81
GIL262	-2.922	-76.73	-2.922	-74.76	-2.955	-85.90	0.000	8.45	-0.034	-39.42	-0.034	-44.13
A280	-2.753	-93.37	-2.753	-91.98	-2.753	-91.75	0.000	20.88	0.000	24.45	0.000	2.96
PR299	-2.004	-82.16	-1.996	-77.69	-1.980	-81.83	0.009	25.05	0.024	1.83	0.016	-18.57
LIN318	-2.289	-54.17	-2.285	-77.86	-2.256	-85.82	0.004	-51.71	0.033	-69.07	0.029	-35.95
RD400	-3.776	-39.35	-3.800	-61.26	-3.812	-60.69	-0.025	-36.13	-0.038	-35.19	-0.013	1.47
FL417	-0.981	-86.73	-0.992	-70.44	-0.966	-86.04	-0.012	122.81	0.015	5.25	0.027	-52.77
GR431	-2.922	-59.47	-2.974	-64.17	-2.769	-71.63	-0.053	-11.60	0.158	-30.00	0.211	-20.81
PR439	-2.028	-70.08	-2.020	-67.23	-2.050	-77.08	0.008	9.53	-0.022	-23.40	-0.030	-30.06
PCB442	-2.487	-19.36	-2.521	-11.16	-2.584	-50.82	-0.034	10.17	-0.099	-39.01	-0.065	-44.64
D493	-2.492	-75.11	-2.495	-73.78	-2.630	-85.83	-0.003	5.32	-0.142	-43.09	-0.138	-45.97
ATT532	-3.467	-71.63	-3.397	-79.91	-3.395	-80.25	0.073	-29.19	0.075	-30.37	0.002	-1.67
Average:	-1.998	-70.07	-1.999	-76.03	-2.001	-80.71	-0.001	-15.62	-0.003	-30.36	-0.002	-6.84

are favorable as they show that the solution quality is improved with faster computational time. The results in Table 7.6 are based on the following: Λ_1 is compared against Λ_2 , Λ_3 and Λ_4 ; Λ_2 is compared against Λ_3 and Λ_4 ; and finally Λ_3 is compared against Λ_4 . The computations of the percentage improvement of average solution cost and μ_T for each comparison are shown in Table 7.7.

Considering the EIL101 in Table 7.6, μ_T for the EIL101 is reduced 86.58% from 558.30s (Λ_1) to 74.95s (Λ_2). This significant reduction is obtained via the replacement of the SSSTR with the FSTR. When the BCO variant Λ_2 is integrated with the DFBPS, which is denoted as Λ_4 , μ_T is further reduced to 17.73s, which is equivalent to a 96.82% reduction.

Considering all 20 problems together, Tables 7.5 and 7.6 show the results as follows. When Λ_1 is compared against Λ_2 , Λ_3 or Λ_4 , the average improvement percentages in terms of solution cost are -1.998%, -1.999% and -2.001% while the average reduction in terms

Table 7.7: Computation of improvement percentage in terms of average solution cost and μ_T .

Comparison	Average Solution Cost	μ_T
Λ_1 VS Λ_2	$\frac{\delta_{avg}(\Lambda_2) - \delta_{avg}(\Lambda_1)}{100 + \delta_{avg}(\Lambda_1)}$	$\frac{\mu_T(\Lambda_2) - \mu_T(\Lambda_1)}{\mu_T(\Lambda_1)}$
Λ_1 VS Λ_3	$\frac{\delta_{avg}(\Lambda_3) - \delta_{avg}(\Lambda_1)}{100 + \delta_{avg}(\Lambda_1)}$	$\frac{\mu_T(\Lambda_3) - \mu_T(\Lambda_1)}{\mu_T(\Lambda_1)}$
Λ_1 VS Λ_4	$\frac{\delta_{avg}(\Lambda_4) - \delta_{avg}(\Lambda_1)}{100 + \delta_{avg}(\Lambda_1)}$	$\frac{\mu_T(\Lambda_4) - \mu_T(\Lambda_1)}{\mu_T(\Lambda_1)}$
Λ_2 VS Λ_3	$\frac{\delta_{avg}(\Lambda_3) - \delta_{avg}(\Lambda_2)}{100 + \delta_{avg}(\Lambda_2)}$	$\frac{\mu_T(\Lambda_3) - \mu_T(\Lambda_2)}{\mu_T(\Lambda_2)}$
Λ_2 VS Λ_4	$\frac{\delta_{avg}(\Lambda_4) - \delta_{avg}(\Lambda_2)}{100 + \delta_{avg}(\Lambda_2)}$	$\frac{\mu_T(\Lambda_4) - \mu_T(\Lambda_2)}{\mu_T(\Lambda_2)}$
Λ_3 VS Λ_4	$\frac{\delta_{avg}(\Lambda_4) - \delta_{avg}(\Lambda_3)}{100 + \delta_{avg}(\Lambda_3)}$	$\frac{\mu_T(\Lambda_4) - \mu_T(\Lambda_3)}{\mu_T(\Lambda_3)}$

of μ_T are -70.07%, -76.03% and -80.71%. When Λ_2 is compared against Λ_3 or Λ_4 , the average improvement percentages in terms of solution cost are -0.001% and -0.003% while the average improvement percentages in terms of μ_T are -15.62% and -30.36%. When Λ_3 is compared against Λ_4 , the average improvement percentages in terms of solution cost and μ_T are -0.002% and -6.84%. These results imply that the solution quality is improved (for Λ_1 VS Λ_2 , Λ_3 or Λ_4) or sustained (for Λ_2 VS Λ_3 or Λ_4 , and Λ_3 VS Λ_4) while a reduction of μ_T is observed (for all cases). Justifications of the results shown in Tables 7.5 and 7.6 are as follows:

- Observation I - When the SSTR in BCO variant Λ_1 is replaced with the FSTR in the BCO variant Λ_2 , the results show improvement in terms of average solution cost and reduction in terms of computational time (μ_T). In the SSTR, a bee incrementally constructs a solution by adding one node at each pass. In the FSTR, a fragment of nodes are added at each pass until a complete feasible solution is acquired and hence brings reduction in terms of computational time (refer Section 5.2 for the FSTR mechanism). Also, the break-and-connect operations between fragments in the FSTR encourage exploration within the solution space. This causes the solution space to be explored more effectively and thus it improves solution quality.

- Observation II - When the BCO variant Λ_2 is incorporated with a pruning strategy, which forms the BCO variants Λ_3 (with the SFBPS) and Λ_4 (with the DFBPS), a reduction in terms of μ_T is observed. In the BCO variant Λ_2 , each and every bee is allowed to undergo the local search procedure and this creates a high overhead. In the BCO variants Λ_3 and Λ_4 , only a subset of bees is allowed to undergo the local search procedure and hence computational time is reduced. The results also show that the SFBPS or DFBPS integration does not affect the solution quality.
- Observation III - When the SFBPS in BCO variant Λ_3 is replaced with the DFBPS in the BCO variant Λ_4 , a reduction in terms of μ_T is observed while the solution quality is maintained. This is because the pruning in the DFBPS can be performed more efficiently due to its dynamic tuning approach whereby the pruning decision adaptively changes over a period of time according to the accumulated frequency of the solution building blocks made by a population of bees. This increases the pruning accuracy. In the SFBPS, once the parameters are statically fixed, the pruning decision will remain unchanged throughout the entire algorithm execution.

Table 7.8 shows the results of the proposed BCO algorithm on the 95 TSPLIB benchmark problem instances with dimensions between [14, 2392]. The variant that produces such results is Λ_4 with the following settings: $\beta = 1$, $\lambda = 0.1$, $K = 100$, $N_{Bee} = 50$ and $\varpi = 25$ and $BC_{Max} = 10000$. Five replications of experiment are conducted for each benchmark instance.

Table 7.8: Computational results of the proposed BCO algorithm on the selected 95 TSPLIB benchmark problem instances.

Instances	Optimum	μ_C	σ_C	δ_{avg} (%)	δ_{min} (%)	δ_{max} (%)	μ_T (s)	σ_T (s)
BURMA14	3323	3323.0	0.000	0.000	0.000	0.000	< 0.01	0.01
ULYSSES16	6859	6859.0	0.000	0.000	0.000	0.000	0.02	0.01
GR17	2085	2085.0	0.000	0.000	0.000	0.000	0.02	0.01
GR21	2707	2707.0	0.000	0.000	0.000	0.000	0.02	0.02
ULYSSES22	7013	7013.0	0.000	0.000	0.000	0.000	0.03	0.03

Continued on next page...

Table 7.8 – continued from previous page

Instances	Optimum	μ_C	σ_C	δ_{avg} (%)	δ_{min} (%)	δ_{max} (%)	μ_T (s)	σ_T (s)
GR24	1272	1272.0	0.000	0.000	0.000	0.000	0.05	0.03
FRI26	937	937.0	0.000	0.000	0.000	0.000	0.04	0.03
BAYG29	1610	1610.0	0.000	0.000	0.000	0.000	0.13	0.06
BAYS29	2020	2020.0	0.000	0.000	0.000	0.000	0.06	0.03
DANTZIG42	699	699.0	0.000	0.000	0.000	0.000	0.29	0.12
SWISS42	1273	1273.0	0.000	0.000	0.000	0.000	0.07	0.05
ATT48	10628	10628.0	0.000	0.000	0.000	0.000	1.44	0.37
GR48	5046	5046.0	0.000	0.000	0.000	0.000	0.58	0.44
HK48	11461	11461.0	0.000	0.000	0.000	0.000	0.26	0.18
EIL51	426	426.0	0.000	0.000	0.000	0.000	2.05	1.62
BERLIN52	7542	7542.0	0.000	0.000	0.000	0.000	0.19	0.09
BRAZIL58	25395	25395.0	0.000	0.000	0.000	0.000	0.56	0.42
ST70	675	675.0	0.000	0.000	0.000	0.000	1.00	0.26
EIL76	538	538.0	0.000	0.000	0.000	0.000	3.93	3.91
PR76	108159	108159.0	0.000	0.000	0.000	0.000	1.90	1.12
GR96	55209	55209.0	0.000	0.000	0.000	0.000	14.77	14.82
RAT99	1211	1211.0	0.000	0.000	0.000	0.000	4.04	1.42
KROA100	21282	21282.0	0.000	0.000	0.000	0.000	1.76	0.67
KROB100	22141	22141.0	0.000	0.000	0.000	0.000	3.00	0.61
KROC100	20749	20749.0	0.000	0.000	0.000	0.000	1.93	0.59
KROD100	21294	21294.0	0.000	0.000	0.000	0.000	11.51	9.82
KROE100	22068	22068.0	0.000	0.000	0.000	0.000	18.22	12.59
RD100	7910	7910.0	0.000	0.000	0.000	0.000	5.52	5.65
EIL101	629	629.0	0.000	0.000	0.000	0.000	17.73	16.20
LIN105	14379	14379.0	0.000	0.000	0.000	0.000	1.18	0.66
PR107	44303	44303.0	0.000	0.000	0.000	0.000	0.78	0.39
GR120	6942	6947.4	4.930	0.078	0.000	0.130	86.97	59.44
PR124	59030	59030.0	0.000	0.000	0.000	0.000	1.99	1.21
BIER127	118282	118282.0	0.000	0.000	0.000	0.000	11.95	6.61
CH130	6110	6110.0	0.000	0.000	0.000	0.000	22.18	24.67
PR136	96772	96789.6	4.506	0.018	0.013	0.023	127.05	64.17
GR137	69853	69853.0	0.000	0.000	0.000	0.000	12.57	5.15
PR144	58537	58537.0	0.000	0.000	0.000	0.000	14.22	5.83
CH150	6528	6528.0	0.000	0.000	0.000	0.000	40.94	22.13
KROA150	26524	26524.0	0.000	0.000	0.000	0.000	88.42	69.22
KROB150	26130	26130.0	0.000	0.000	0.000	0.000	20.03	8.62
PR152	73682	73682.0	0.000	0.000	0.000	0.000	4.58	1.67
U159	42080	42080.0	0.000	0.000	0.000	0.000	4.43	0.41
SI175	21407	21407.0	0.000	0.000	0.000	0.000	43.73	38.11
RAT195	2323	2327.6	2.702	0.198	0.000	0.301	119.66	25.98
D198	15780	15791.4	5.128	0.072	0.019	0.095	295.62	112.13
KROA200	29368	29368.0	0.000	0.000	0.000	0.000	43.98	30.91
KROB200	29437	29437.6	0.548	0.002	0.000	0.003	160.37	120.46
GR202	40160	40170.8	14.789	0.027	0.000	0.067	157.22	35.20
TS225	126643	126643.0	0.000	0.000	0.000	0.000	25.05	6.94
TSP225	3916	3916.0	0.000	0.000	0.000	0.000	128.06	91.52
PR226	80369	80369.0	0.000	0.000	0.000	0.000	160.43	145.10
GR229	134602	134616.0	0.000	0.010	0.010	0.010	148.54	109.54

Continued on next page...

Table 7.8 – continued from previous page

Instances	Optimum	μ_C	σ_C	δ_{avg} (%)	δ_{min} (%)	δ_{max} (%)	μ_T (s)	σ_T (s)
GIL262	2378	2378.0	0.000	0.000	0.000	0.000	168.77	127.87
PR264	49135	49135.0	0.000	0.000	0.000	0.000	55.02	20.13
A280	2579	2579.0	0.000	0.000	0.000	0.000	105.89	68.55
PR299	48191	48205.0	17.015	0.029	0.000	0.068	416.28	269.26
LIN318	42029	42095.8	15.595	0.159	0.128	0.224	207.32	133.20
RD400	15281	15316.0	6.364	0.229	0.190	0.281	1467.13	422.55
FL417	11861	11876.4	3.578	0.130	0.076	0.143	1171.07	900.27
GR431	171414	172412.4	270.485	0.582	0.410	0.736	1877.06	617.44
PR439	107217	107260.6	19.074	0.041	0.031	0.072	1373.00	853.89
PCB442	50778	50993.0	71.746	0.423	0.313	0.664	866.14	396.73
D493	35002	35126.0	45.563	0.354	0.157	0.506	1733.04	1150.76
ATT532	27686	27783.2	24.994	0.351	0.267	0.477	2053.40	1310.54
ALI535	202339	202546.6	237.387	0.103	0.029	0.311	1867.01	1056.80
SI535	48450	48466.6	8.142	0.034	0.019	0.058	938.15	340.19
PA561	2763	2789.2	3.271	0.948	0.832	1.122	1154.70	853.16
U574	36905	37162.4	55.406	0.697	0.466	0.883	2511.78	1224.17
RAT575	6773	6809.4	16.471	0.537	0.221	0.797	2238.61	1326.33
P654	34643	34659.6	4.669	0.048	0.029	0.064	3161.03	1342.15
D657	48912	49129.8	26.386	0.445	0.386	0.511	3306.19	1187.82
GR666	294358	295986.2	204.705	0.553	0.456	0.633	4484.27	556.09
U724	41910	42170.8	69.712	0.622	0.453	0.859	3436.89	942.87
RAT783	8806	8884.8	11.323	0.895	0.738	1.056	5217.05	1321.12
PR1002	259045	261255.4	385.584	0.853	0.607	1.004	8511.25	2842.29
SI1032	92650	92650.0	0.000	0.000	0.000	0.000	818.54	774.30
U1060	224094	225790.2	268.412	0.757	0.620	0.885	8466.07	3116.04
VM1084	239297	240481.8	395.565	0.495	0.247	0.697	8489.21	2949.15
PCB1173	56892	57417.6	238.123	0.924	0.320	1.452	12804.69	144.73
D1291	50801	51028.0	95.958	0.447	0.287	0.758	12959.64	3676.52
RL1304	252948	254420.0	771.984	0.582	0.082	0.891	12620.80	6460.17
RL1323	270199	271538.4	505.192	0.496	0.323	0.709	9396.76	4753.49
NRW1379	56638	57406.8	127.133	1.357	1.081	1.633	12408.56	2833.32
FL1400	20127	20196.6	24.966	0.346	0.268	0.566	16530.97	7266.16
U1432	152970	156066.0	419.159	2.024	1.647	2.339	9866.34	3451.63
FL1577	22249	22368.2	22.961	0.536	0.378	0.656	10332.37	2750.91
D1655	62128	62787.8	95.035	1.062	0.837	1.209	22426.53	6117.06
VM1748	336556	339697.2	418.647	0.933	0.742	1.086	24750.99	5827.55
U1817	57201	57976.6	110.835	1.356	1.119	1.629	19154.02	6840.11
RL1889	316536	319296.8	423.069	0.872	0.694	1.014	24409.27	7532.62
D2103	80450	81069.8	80.596	0.770	0.677	0.925	24600.67	8143.72
U2152	64253	65214.2	215.056	1.496	0.971	1.770	24377.15	4151.39
U2319	234256	238396.4	172.784	1.767	1.685	1.886	24965.49	8860.91
PR2392	378032	381980.0	1912.284	1.044	0.519	1.668	44667.15	9338.16

The δ_{avg} column in Table 7.8 shows that the proposed BCO algorithm (variant Λ_4 , BCO_FSTR+FRNN_2-opt_TL+DFBPS) is able to obtain $\leq 1\%$ deviation from known

optimum for 88 out of 95 instances. Instances of NRW1379, U1432, D1655, U1817, U2152, U2319 and PR2392 show a deviation of $> 1\%$. The largest deviation from known optimum for the entire dataset is 2.024%, which is obtained when solving the U1432. Furthermore, the proposed BCO algorithm solves 50 instances to optimum, which is equivalent to 52.63% of the 95 instances. The average of δ_{avg} , δ_{min} and δ_{max} in Table 7.8 are 0.271%, 0.193% and 0.346% respectively.

The results shown in Table 7.8 are further organized as follows. The 95 instances are grouped into four different classes. Class I consists of 64 instances with the dimension of $[1, 500]$. Class II consists of 11 instances with the dimension $[501, 1000]$. Class III consists of 11 instances with the dimension of $[1001, 1500]$. Class IV consists of 9 instances with the dimension of $[1501, 2500]$. Class V represents the entire benchmark dataset. Table 7.9 shows the average of δ_{avg} , δ_{min} , δ_{max} and μ_T for each class. On average, the proposed BCO algorithm is able to obtain 0.037% deviation from known optimum for instances in Class I. For instances in Classes II, III and IV, the BCO algorithm obtains 0.391%, 0.753% and 1.093% respectively. In terms of the average computational time, Class I requires 175.81s to obtain the best score. This also means that the BCO algorithm is able to obtain the best score for each instance in Class I within 3.0 minutes. For Classes II, III and IV, the average time to acquire the best score are 2168.46s (equivalent to 36.1 minutes), 10261.17s (equivalent to 2.9 hours) and 24409.29s (equivalent to 6.8 hours) respectively. Considering all 95 problem instances (Class V), the proposed BCO algorithm requires 3938.71s (equivalent to 65.6 minutes) to solve all the problem instances to an average of 0.271% from known optimum.

In Table 7.9, it is noticed that the averages of δ_{avg} , δ_{min} , δ_{max} and μ_T increase across Classes I to IV. This is due to the increment of problem dimension across the mentioned classes. The solution space increases when the problem dimension becomes larger and larger. Therefore, bees in the proposed BCO algorithms will need more computational

Table 7.9: Analysis on the results shown in Table 7.8.

Class	Dimension	Total Inst.	Avg. δ_{avg}	Avg. δ_{min}	Avg. δ_{max}	Avg. μ_T
I	[1, 500]	64	0.037%	0.021%	0.052%	175.81s
II	[501, 1000]	11	0.391%	0.283%	0.513%	2168.46s
III	[1001, 1500]	11	0.753%	0.498%	0.994%	10261.17s
IV	[1501, 2500]	9	1.093%	0.847%	1.316%	24409.29s
V	[1, 2500]	95	0.271%	0.193%	0.346%	3938.71s

time to explore and exploit the solution space in order to allocate solutions with minimum tour length. With the aid from the components within the generic BCO framework (e.g. the FSTR, waggle dance performance and the DFBPS) and domain specific local search method, the BCO algorithm is able to find near-optimum solutions for many benchmark problem instances as shown by the TSP results reported in this section.

Comparison studies which assess the performance of the proposed BCO algorithm compared to various approaches reviewed in Section 3.4 can be found in Section 7.7.

7.5 Performance of the BCO Algorithm on QAP

This section presents the results and findings of the proposed BCO algorithm on a set of QAPLIB benchmark problems. A total of 134 problem instances with dimensions between [12, 256] are used to test the proposed BCO algorithm. The instances from the QAPLIB are grouped into 15 different series. They are BUR, CHR, ELS, HAD, KRA, LIPA, NUG, ROU, SCR, SKO, STE, TAI, THO and WIL.

The QAPLIB instances can also be grouped into four different classes, according to the method used in generating a problem instance. These four classes are as follows:

- Class I - unstructured, randomly generated instances (TAI**A and ROU**);
- Class II - grid-based distance matrix instances (NUG**, THO*** and SKO***);

- Class III - real-world instances (BUR***, CHR***, KRA***, ELS** and STE***);
and
- Class IV - real-world-like instances (TAI**B).

The type of a QAP instance has a significant influence on the performance of proposed methods. Further information on how to generate the QAPLIB benchmark problem instances according to the mentioned classes can be found in [124, 135].

7.5.1 Parameters Tuning for QAP

In the parameter tuning experiment for QAP, the SKO100A problem from the QAPLIB is used as the sample. Table 7.1 in Section 7.2 illustrates the five BCO parameters with their high (+1) and low (-1) levels for this experiment. The BCO algorithm is executed for 3000 iterations, that is $BC_{Max} = 3000$.

Table C.1 (refer to Appendix C) presents the parameter tuning experiment results. The table is sorted according to the average assignment cost of each configuration. From Table C.1, $\mu_{min} = \mu_{13} = 152381.0$ and it is obtained via the configuration $i = 13$. This μ_{min} will be used in the hypothesis testing so that the parameter setting for the proposed BCO algorithm is determined.

The hypothesis testing results for SKO100A are shown in Table C.2 (refer to Appendix C). The T-Test shows that μ_{10} , μ_{09} , μ_{14} , μ_{25} and μ_{15} (they are in bold and italic font) are statistically equal to $\mu_{min} = \mu_{13}$. After performing a bit-by-bit comparison as explained in Section 7.2, the recommended parameter setting for the SKO100A is shown in Table 7.10. We assume that the setting of these five parameters shown in Table 7.10 is applicable to all problems in the QAP domain. Hence, this parameter setting, which is equivalent to the configuration $i = 13$ in Table 7.2, will be applied throughout the QAP experiments. The QAP experimental results will be described in Section 7.5.2.

Table 7.10: Recommended BCO parameter settings for the SKO100A.

Config.	β	λ	K	N_{Bee}	ϖ
$i = 13$	-1	-1	+1	+1	-1
$i = 10$	+1	-1	-1	+1	-1
$i = 09$	-1	-1	-1	+1	-1
$i = 14$	+1	-1	+1	+1	-1
$i = 25$	-1	-1	-1	+1	+1
$i = 15$	-1	+1	+1	+1	-1
Mode	-1	-1	-1/+1	+1	-1

7.5.2 Results for QAP

In Section 4.2, it is mentioned that if an enhancement is made to the generic BCO framework, the effect of the enhancement will be applicable to all COPs. The results in Section 7.4.2 shows that the TSP obtains the benefit from the enhancements made to the generic BCO framework. To demonstrate the enhancement effect is also applicable to QAP, 13 selected QAPLIB problem instances from the SKO series are tested with different BCO variants. The problem instances from the SKO series have not been solved to optimum and hence upper bounds are used as reference. The BCO variants for the QAP (Λ_1 , Λ_2 , Λ_3 and Λ_4) as shown in Fig. 7.1 and Table 7.3 are employed in such experiment.

The recommended BCO parameter setting for QAP as illustrated in Table 7.10 was applied: $\beta = 1$, $\lambda = 0.1$, $K = 100$, $N_{Bee} = 50$ and $\varpi = 10$. The BCO algorithm was executed for 10000 BC_{Max} for five replications on each benchmark instance. Table 7.11 presents the results for this experiment. In terms of the average of δ_{avg} for each BCO variants (Λ_1 , Λ_2 , Λ_3 and Λ_4), they are 0.638%, 0.077%, 0.111%, 0.127%. Table 7.11 also shows the average time to obtain the best score for each benchmark problem. It is denoted as μ_T . The average μ_T for Λ_1 , Λ_2 , Λ_3 and Λ_4 are 5393.68s (≈ 90 minutes), 5417.91s

(≈ 90 minutes), 3783.79s (≈ 63 minutes) and 2754.25s (≈ 46 minutes) respectively. The improvement percentage in terms of μ_T for each benchmark instances is shown in Table 7.12

Considering all 13 problems together, Tables 7.11 and 7.12 show the results as follows. When Λ_1 is compared against Λ_2 , the average improvement percentage in terms of solution cost is -0.558% while a slight increment in μ_T (+7.30%) is observed. When Λ_1 is compared against Λ_3 or Λ_4 , the average improvement percentages in terms of solution cost are -0.523% and -0.507% while the average reduction in terms of μ_T are -25.60% and -51.05%. When Λ_2 is compared against Λ_3 or Λ_4 , the average improvement percentages in terms of solution cost are +0.035% and +0.051% while the average improvement percentages in terms of μ_T are -28.18% and -49.68%. When Λ_3 is compared against Λ_4 , the average improvement percentages in terms of solution cost and μ_T are +0.016% and -17.18%. These results imply that the solution quality is improved (for Λ_1 VS Λ_2 , Λ_3 or Λ_4) or sustained (for Λ_2 VS Λ_3 or Λ_4 , and Λ_3 VS Λ_4) while a reduction of μ_T is observed (for all cases except Λ_1 VS Λ_2). Similar trends are observed as what have been reported for the TSP results. Therefore, only differences are highlighted here. For the comparison between the BCO variants Λ_1 and Λ_2 (the SSTR is replaced with the FSTR), a slight increment of 7.30% in terms of μ_T is observed while the δ_{avg} drops from 0.638% to 0.077%. In the BCO variant Λ_1 , bees are trapped in a local optimum whereby they are only able to find solutions with $\delta_{avg} = 0.638\%$. When the replacement is performed, bees are able to explore the solution space more effectively due to the break-and-connect operations between fragments in the FSTR. Thus, this improves the solution quality. The reduction in terms of δ_{avg} compensates the slight increment in terms of μ_T . The discussions of the effects of integrating the BCO variant with a pruning strategy (Λ_2 VS Λ_3 or Λ_4) and replacing the SFBPS with the DFBPS (Λ_3 VS Λ_4) in QAP can be found Section 7.4.2.

Table 7.13 shows the results of the proposed BCO algorithm on the 134 QAPLIB benchmark problem instances with dimensions between [12, 256]. If a QAPLIB problem

Table 7.11: Effect of different variants of the BCO algorithm on 13 selected QAPLIB benchmark problems.

Inst.	U.B.	Λ_1			Λ_2			Λ_3			Λ_4		
		δ_{min}	δ_{avg}	μ_T (s)	δ_{min}	δ_{avg}	μ_T (s)	δ_{min}	δ_{avg}	μ_T (s)	δ_{min}	δ_{avg}	μ_T (s)
SKO42	15812	0.240	0.463	526.62	0.000	0.000	343.72	0.000	0.000	288.51	0.000	0.000	308.74
SKO49	23386	0.316	0.573	1392.68	0.034	0.068	1199.98	0.000	0.027	706.01	0.068	0.074	231.05
SKO56	34458	0.534	0.576	1592.62	0.023	0.034	1847.56	0.017	0.029	1289.89	0.023	0.081	675.04
SKO64	48498	0.520	0.616	2841.59	0.000	0.027	3810.37	0.000	0.003	1826.36	0.004	0.058	838.59
SKO72	66256	0.525	0.639	2144.38	0.036	0.068	2864.62	0.085	0.136	2756.94	0.100	0.167	1031.16
SKO81	90998	0.503	0.605	4595.49	0.046	0.105	6686.93	0.048	0.086	3909.25	0.088	0.134	2577.60
SKO90	115534	0.649	0.707	5593.04	0.073	0.111	7811.74	0.158	0.186	4621.95	0.090	0.215	2390.10
SKO100A	152002	0.621	0.671	6597.89	0.113	0.137	7271.47	0.078	0.117	6987.35	0.108	0.161	4673.10
SKO100B	153890	0.560	0.642	11547.57	0.043	0.082	8910.48	0.066	0.133	5700.68	0.071	0.123	3801.62
SKO100C	147862	0.582	0.618	9013.63	0.024	0.066	6486.97	0.049	0.132	2016.01	0.087	0.133	7134.83
SKO100D	149576	0.603	0.704	5778.99	0.027	0.092	9371.43	0.166	0.218	5822.69	0.067	0.167	4324.82
SKO100E	149150	0.654	0.751	8432.89	0.043	0.076	8143.30	0.043	0.094	5156.45	0.039	0.138	3606.16
SKO100F	149036	0.676	0.731	10060.44	0.094	0.130	5684.31	0.209	0.286	8107.21	0.072	0.205	4212.44
Average:		0.537	0.638	5393.68	0.043	0.077	5417.91	0.071	0.111	3783.79	0.063	0.127	2754.25

Table 7.12: Improvement percentage in terms of average solution cost and μ_T among the different BCO variants on 13 selected QAPLIB benchmark problems.

Inst.	Λ_1 VS Λ_2 (%)		Λ_1 VS Λ_3 (%)		Λ_1 VS Λ_4 (%)		Λ_2 VS Λ_3 (%)		Λ_2 VS Λ_4 (%)		Λ_3 VS Λ_4 (%)	
	Cost	μ_T	Cost	μ_T	Cost	μ_T	Cost	μ_T	Cost	μ_T	Cost	μ_T
SKO42	-0.461	-34.73	-0.461	-45.22	-0.461	-41.37	0.000	-16.06	0.000	-10.18	0.000	7.01
SKO49	-0.502	-13.84	-0.543	-49.31	-0.497	-83.41	-0.041	-41.16	0.005	-80.75	0.046	-67.27
SKO56	-0.539	16.01	-0.544	-19.01	-0.492	-57.61	-0.005	-30.18	0.048	-63.46	0.052	-47.67
SKO64	-0.585	34.09	-0.609	-35.73	-0.555	-70.49	-0.024	-52.07	0.031	-77.99	0.054	-54.08
SKO72	-0.567	33.59	-0.499	28.57	-0.469	-51.91	0.068	-3.76	0.098	-64.00	0.030	-62.60
SKO81	-0.497	45.51	-0.516	-14.93	-0.468	-43.91	-0.019	-41.54	0.029	-61.45	0.048	-34.06
SKO90	-0.592	39.67	-0.517	-17.36	-0.489	-57.27	0.075	-40.83	0.104	-69.40	0.029	-48.29
SKO100A	-0.530	10.21	-0.550	5.90	-0.507	-29.17	-0.020	-3.91	0.024	-35.73	0.044	-33.12
SKO100B	-0.557	-22.84	-0.506	-50.63	-0.516	-67.08	0.051	-36.02	0.041	-57.34	-0.010	-33.31
SKO100C	-0.548	-28.03	-0.483	-77.63	-0.482	-20.84	0.066	-68.92	0.067	9.99	0.001	253.91
SKO100D	-0.607	62.16	-0.483	0.76	-0.533	-25.16	0.126	-37.87	0.075	-53.85	-0.050	-25.72
SKO100E	-0.670	-3.43	-0.652	-38.85	-0.608	-57.24	0.018	-36.68	0.062	-55.72	0.044	-30.07
SKO100F	-0.597	-43.50	-0.441	-19.41	-0.521	-58.13	0.157	42.62	0.076	-25.89	-0.081	-48.04
Average	-0.558	7.30	-0.523	-25.60	-0.507	-51.05	0.035	-28.18	0.051	-49.68	0.016	-17.18

instance has been solved to optimum, the name of that particular instance is marked with “*”. The variant that produces such results is Λ_4 with the following settings: $\beta = 1$, $\lambda = 0.1$, $K = 100$, $N_{Bee} = 50$ and $\varpi = 10$ and $BC_{Max} = 10000$. Five replications of experiment are conducted for each benchmark instance.

The δ_{avg} column in Table 7.13 shows that the proposed BCO algorithm (variant Λ_4 , BCO_FSTR+2-opt_TL+DFBPS) is able to obtain $\leq 1\%$ deviation from known optimum

or upper bound for 122 out of 134 instances. This is equivalent to 91.04% of the 134 QAPLIB benchmark instances being solved to $\leq 1\%$ deviation from known optimum or upper bound. In addition, the proposed BCO algorithm solves 84 instances to optimum, which is equivalent to 62.69% of the 134 instances. The average of δ_{avg} , δ_{min} and δ_{max} in Table 7.13 are 0.33%, 0.15% and 0.59% respectively.

Table 7.13: Computational results of the proposed BCO algorithm on the selected 134 QAPLIB benchmark problem instances.

Instances	Opt./U.B.	μ_C	σ_C	δ_{avg} (%)	δ_{min} (%)	δ_{max} (%)	μ_T (s)	σ_T (s)
*BUR26A	5426670	5426670.0	0.00	0.00	0.00	0.00	4.25	3.50
*BUR26B	3817852	3817852.0	0.00	0.00	0.00	0.00	5.76	8.38
*BUR26C	5426795	5426795.0	0.00	0.00	0.00	0.00	12.77	12.50
*BUR26D	3821225	3821225.0	0.00	0.00	0.00	0.00	10.51	6.38
*BUR26E	5386879	5386879.0	0.00	0.00	0.00	0.00	7.49	6.73
*BUR26F	3782044	3782044.0	0.00	0.00	0.00	0.00	4.48	3.77
*BUR26G	10117172	10117172.0	0.00	0.00	0.00	0.00	25.52	24.62
*BUR26H	7098658	7098658.0	0.00	0.00	0.00	0.00	9.31	7.91
*CHR12A	9552	9552.0	0.00	0.00	0.00	0.00	0.20	0.11
*CHR12B	9742	9742.0	0.00	0.00	0.00	0.00	0.17	0.16
*CHR12C	11156	11162.0	13.42	0.05	0.00	0.27	4.78	4.24
*CHR15A	9896	9912.0	21.91	0.16	0.00	0.40	11.81	7.79
*CHR15B	7990	7990.0	0.00	0.00	0.00	0.00	1.67	0.88
*CHR15C	9504	9504.0	0.00	0.00	0.00	0.00	2.07	1.30
*CHR18A	11098	11098.0	0.00	0.00	0.00	0.00	8.91	4.80
*CHR18B	1534	1534.0	0.00	0.00	0.00	0.00	0.85	0.47
*CHR20A	2192	2266.4	47.21	3.39	1.46	6.39	19.66	15.21
*CHR20B	2298	2385.6	26.05	3.81	2.35	4.96	20.40	20.29
*CHR20C	14142	14275.6	298.74	0.94	0.00	4.72	15.95	8.26
*CHR22A	6156	6198.8	50.35	0.70	0.00	2.08	21.28	19.08
*CHR22B	6194	6274.8	36.60	1.30	0.58	1.94	24.00	11.32
*CHR25A	3796	3963.6	119.13	4.42	2.05	9.80	40.71	30.09
*ELS19	17212548	17212548.0	0.00	0.00	0.00	0.00	1.34	0.63
*ESC16A	68	68.0	0.00	0.00	0.00	0.00	0.03	0.01
*ESC16B	292	292.0	0.00	0.00	0.00	0.00	< 0.01	0.01
*ESC16C	160	160.0	0.00	0.00	0.00	0.00	0.02	0.01
*ESC16D	16	16.0	0.00	0.00	0.00	0.00	0.01	0.01
*ESC16E	28	28.0	0.00	0.00	0.00	0.00	0.02	0.01
*ESC16F	0	0.0	0.00	0.00	0.00	0.00	< 0.01	0.00
*ESC16G	26	26.0	0.00	0.00	0.00	0.00	0.01	0.01
*ESC16H	996	996.0	0.00	0.00	0.00	0.00	0.02	0.01
*ESC16I	14	14.0	0.00	0.00	0.00	0.00	0.01	0.02
*ESC16J	8	8.0	0.00	0.00	0.00	0.00	< 0.01	0.00
ESC32A	130	130.4	0.89	0.31	0.00	1.54	50.65	52.46
ESC32B	168	168.0	0.00	0.00	0.00	0.00	2.94	2.89
ESC32C	642	642.0	0.00	0.00	0.00	0.00	0.07	0.03

Continued on next page...

Table 7.13 – continued from previous page

Instances	Opt./U.B.	μ_C	σ_C	δ_{avg} (%)	δ_{min} (%)	δ_{max} (%)	μ_T (s)	σ_T (s)
ESC32D	200	200.0	0.00	0.00	0.00	0.00	0.19	0.13
*ESC32E	2	2.0	0.00	0.00	0.00	0.00	< 0.01	0.01
*ESC32G	6	6.0	0.00	0.00	0.00	0.00	0.02	0.01
ESC32H	438	438.0	0.00	0.00	0.00	0.00	0.72	0.41
ESC64A	116	116.0	0.00	0.00	0.00	0.00	0.06	0.03
ESC128	64	64.0	0.00	0.00	0.00	0.00	5.47	4.11
*HAD12	1652	1652.0	0.00	0.00	0.00	0.00	0.10	0.05
*HAD14	2724	2724.0	0.00	0.00	0.00	0.00	0.07	0.03
*HAD16	3720	3720.0	0.00	0.00	0.00	0.00	0.08	0.02
*HAD18	5358	5358.0	0.00	0.00	0.00	0.00	0.52	0.21
*HAD20	6922	6922.0	0.00	0.00	0.00	0.00	5.66	4.84
*KRA30A	88900	88900.0	0.00	0.00	0.00	0.00	24.91	16.99
*KRA30B	91420	91420.0	0.00	0.00	0.00	0.00	90.42	29.65
*KRA32	88700	88700.0	0.00	0.00	0.00	0.00	16.37	25.01
*LIPA20A	3683	3683.0	0.00	0.00	0.00	0.00	3.35	0.68
*LIPA20B	27076	27076.0	0.00	0.00	0.00	0.00	0.19	0.08
*LIPA30A	13178	13178.0	0.00	0.00	0.00	0.00	8.53	1.16
*LIPA30B	151426	151426.0	0.00	0.00	0.00	0.00	0.70	0.41
*LIPA40A	31538	31538.0	0.00	0.00	0.00	0.00	71.67	22.19
*LIPA40B	476581	476581.0	0.00	0.00	0.00	0.00	3.16	0.75
*LIPA50A	62093	62431.0	284.97	0.54	0.00	0.95	475.18	174.16
*LIPA50B	1210244	1210244.0	0.00	0.00	0.00	0.00	13.62	7.29
*LIPA60A	107218	108100.8	34.30	0.82	0.79	0.87	453.01	364.56
*LIPA60B	2520135	2520135.0	0.00	0.00	0.00	0.00	285.98	283.53
*LIPA70A	169755	171032.2	40.98	0.75	0.72	0.78	2363.95	740.34
*LIPA70B	4603200	4603200.0	0.00	0.00	0.00	0.00	171.61	109.58
*LIPA80A	253195	254881.6	56.34	0.67	0.64	0.70	1820.41	1361.23
*LIPA80B	7763962	7763962.0	0.00	0.00	0.00	0.00	411.31	504.99
*LIPA90A	360630	362902.8	50.01	0.63	0.61	0.64	3825.38	1916.19
*LIPA90B	12490441	13508545.6	1394102.92	8.15	0.00	20.42	1286.57	1226.82
*NUG12	578	578.0	0.00	0.00	0.00	0.00	0.06	0.02
*NUG14	1014	1014.0	0.00	0.00	0.00	0.00	0.98	0.67
*NUG15	1150	1150.0	0.00	0.00	0.00	0.00	0.26	0.35
*NUG16A	1610	1610.0	0.00	0.00	0.00	0.00	3.76	2.53
*NUG16B	1240	1240.0	0.00	0.00	0.00	0.00	0.16	0.09
*NUG17	1732	1732.0	0.00	0.00	0.00	0.00	3.87	3.55
*NUG18	1930	1930.0	0.00	0.00	0.00	0.00	1.85	2.56
*NUG20	2570	2570.0	0.00	0.00	0.00	0.00	1.48	1.17
*NUG21	2438	2438.0	0.00	0.00	0.00	0.00	1.14	1.08
*NUG22	3596	3596.0	0.00	0.00	0.00	0.00	0.69	0.82
*NUG24	3488	3488.0	0.00	0.00	0.00	0.00	1.11	0.69
*NUG25	3744	3744.0	0.00	0.00	0.00	0.00	11.63	13.80
*NUG27	5234	5234.0	0.00	0.00	0.00	0.00	34.84	22.48
*NUG28	5166	5172.4	6.07	0.12	0.00	0.23	51.32	35.97
*NUG30	6124	6124.0	0.00	0.00	0.00	0.00	52.64	32.75
*ROU12	235528	235528.0	0.00	0.00	0.00	0.00	0.65	0.46
*ROU15	354210	354210.0	0.00	0.00	0.00	0.00	0.88	0.29
*ROU20	725522	726034.0	642.24	0.07	0.00	0.18	22.70	17.04
*SCR12	31410	31410.0	0.00	0.00	0.00	0.00	0.05	0.01

Continued on next page...

Table 7.13 – continued from previous page

Instances	Opt./U.B.	μ_C	σ_C	δ_{avg} (%)	δ_{min} (%)	δ_{max} (%)	μ_T (s)	σ_T (s)
*SCR15	51140	51140.0	0.00	0.00	0.00	0.00	0.76	0.61
*SCR20	110030	110030.0	0.00	0.00	0.00	0.00	7.03	2.85
SKO42	15812	15812.0	0.00	0.00	0.00	0.00	308.74	335.70
SKO49	23386	23403.2	1.79	0.07	0.07	0.09	231.05	185.18
SKO56	34458	34486.0	24.12	0.08	0.02	0.20	675.04	283.67
SKO64	48498	48526.0	24.08	0.06	0.00	0.12	838.59	351.77
SKO72	66256	66366.4	47.00	0.17	0.10	0.25	1031.16	309.90
SKO81	90998	91120.0	38.65	0.13	0.09	0.20	2577.60	787.60
SKO90	115534	115782.0	133.23	0.21	0.09	0.40	2390.10	1449.82
SKO100A	152002	152246.4	85.62	0.16	0.11	0.24	4673.10	1937.53
SKO100B	153890	154078.8	66.63	0.12	0.07	0.19	3801.62	2101.55
SKO100C	147862	148058.8	61.07	0.13	0.09	0.19	7134.83	2390.28
SKO100D	149576	149826.4	136.57	0.17	0.07	0.27	4324.82	1715.71
SKO100E	149150	149356.4	159.39	0.14	0.04	0.31	3606.16	2130.02
SKO100F	149036	149342.0	181.02	0.21	0.07	0.39	4212.44	839.81
*STE36A	9526	9528.0	4.47	0.02	0.00	0.10	103.20	57.29
*STE36B	15852	15852.0	0.00	0.00	0.00	0.00	48.21	42.83
*STE36C	8239110	8239110.0	0.00	0.00	0.00	0.00	77.44	43.57
*TAI12A	224416	224416.0	0.00	0.00	0.00	0.00	0.35	0.24
*TAI12B	39464925	39464925.0	0.00	0.00	0.00	0.00	0.90	0.81
*TAI15A	388214	388214.0	0.00	0.00	0.00	0.00	3.15	1.43
*TAI15B	51765268	51765268.0	0.00	0.00	0.00	0.00	2.07	1.72
*TAI17A	491812	491812.0	0.00	0.00	0.00	0.00	3.48	5.21
*TAI20A	703482	705335.2	2052.29	0.26	0.00	0.71	20.37	9.09
*TAI20B	122455319	122455319.0	0.00	0.00	0.00	0.00	2.17	1.38
*TAI25A	1167256	1177721.6	6374.37	0.90	0.00	1.37	37.43	25.89
*TAI25B	344355646	344355646.0	0.00	0.00	0.00	0.00	4.08	3.52
TAI30A	1818146	1837216.0	7566.07	1.05	0.48	1.59	62.62	37.76
TAI30B	637117113	637121280.6	9319.04	0.00	0.00	0.00	75.59	48.43
TAI35A	2422002	2455950.4	9120.69	1.40	0.86	1.89	108.07	55.18
TAI35B	283315445	283315445.0	0.00	0.00	0.00	0.00	53.80	19.61
TAI40A	3139370	3187202.0	10291.32	1.52	1.00	1.83	138.30	111.02
TAI40B	637250948	637250948.0	0.00	0.00	0.00	0.00	82.17	101.22
TAI50A	4938796	5042024.4	12020.38	2.09	1.72	2.37	187.64	161.97
TAI50B	458821517	458961705.8	122381.16	0.03	0.00	0.06	299.38	181.53
TAI60A	7205962	7358711.6	7264.08	2.12	1.96	2.22	576.71	256.14
TAI60B	608215054	608323031.8	112415.10	0.02	0.00	0.05	593.54	406.16
TAI64C	1855928	1855928.0	0.00	0.00	0.00	0.00	0.50	0.65
TAI80A	13511780	13767906.8	35210.86	1.90	1.46	2.15	1852.76	459.94
TAI80B	818415043	819759662.4	1314306.57	0.16	0.06	0.45	1199.08	840.11
TAI100A	21052466	21457081.6	29777.88	1.92	1.68	2.05	2464.53	1493.21
TAI100B	1185996137	1191194761.8	2075185.02	0.44	0.25	0.70	3288.10	1837.43
TAI150B	498896643	502178911.4	618211.56	0.66	0.52	0.83	16106.84	7099.39
TAI256C	44759294	44816702.8	11022.64	0.13	0.10	0.16	56395.31	44041.50
*THO30	149936	149936.0	0.00	0.00	0.00	0.00	64.88	53.39
THO40	240516	240577.6	63.83	0.03	0.00	0.06	197.24	113.16
THO150	8133398	8153398.4	5846.74	0.25	0.15	0.32	11262.74	5980.24
WIL50	48816	48835.6	12.20	0.04	0.00	0.06	478.92	435.63
WIL100	273038	273326.8	69.51	0.11	0.07	0.14	3930.68	2010.44

Table 7.14: Analysis of the QAP results shown in Table 7.13. The analysis is based on different series of the QAPLIB benchmark problems.

Dimension	Total Instances	Average of				Standard Deviation of			
		$\delta_{min}(\%)$	$\delta_{avg}(\%)$	$\delta_{max}(\%)$	$\mu_T(s)$	$\delta_{min}(\%)$	$\delta_{avg}(\%)$	$\delta_{max}(\%)$	$\mu_T(s)$
BUR [26, 26]	8	0.00	0.00	0.00	10.01	0.00	0.00	0.00	6.94
CHR [12, 25]	14	0.46	1.06	2.18	12.32	0.84	1.59	3.11	11.99
ELS [19, 19]	1	0.00	0.00	0.00	1.34	0.00	0.00	0.00	0.00
ESC [16, 128]	19	0.00	0.02	0.08	3.17	0.00	0.07	0.35	11.58
HAD [12, 20]	5	0.00	0.00	0.00	1.29	0.00	0.00	0.00	2.45
KRA [30, 32]	3	0.00	0.00	0.00	43.90	0.00	0.00	0.00	40.51
LIPA [20, 90]	16	0.17	0.72	1.52	699.66	0.31	2.01	5.05	1097.73
NUG [12, 30]	15	0.00	0.01	0.02	11.05	0.00	0.03	0.06	18.82
ROU [12, 20]	3	0.00	0.02	0.06	8.08	0.00	0.04	0.10	12.67
SCR [12, 20]	3	0.00	0.00	0.00	2.61	0.00	0.00	0.00	3.84
SKO [42, 100]	13	0.06	0.13	0.22	2754.25	0.04	0.06	0.11	2096.82
STE [36, 36]	3	0.00	0.01	0.03	76.28	0.00	0.01	0.06	27.51
TAI [12, 256]	26	0.39	0.56	0.71	3213.80	0.64	0.77	0.88	11302.98
THO [30, 150]	3	0.05	0.09	0.13	3841.62	0.08	0.14	0.17	6427.22
WIL [50, 100]	2	0.03	0.07	0.10	2204.80	0.05	0.05	0.06	2440.77

The results shown in Table 7.13 are further analyzed according to a classification technique where the 134 QAPLIB problems are categorized into 15 different series. The analysis results are reported in Table 7.14. Taking the BUR series as an example, the BCO variant Λ_4 solves all the problem instances in that series to optimum within 10.01s. Besides the BUR series, the ELS, HAD, KRA and SCR series are solved to optimum within 3.17s, 1.29s, 43.90s, and 2.61s respectively. For the CHR series, it is noticed that the average of δ_{avg} and μ_T is 1.06% and 12.32s respectively. We anticipate that if more time is allocated to the CHR series, the results would be improved further.

Table 7.15 shows another analysis results of the QAPLIB benchmark problem instances. The analysis is done based on different types of problems which have been discussed in Section 7.5. If the column of Avg. δ_{avg} is referred, it is noticed that the proposed BCO algorithm is able to solve the benchmark instances in all classes to $\leq 1\%$ deviation from the know optimum or known upper bound. In Classes I and II, the average δ_{avg} are 0.88% and 0.07% respectively. In Classes III and IV, the average δ_{avg} are 0.51%

Table 7.15: Analysis of the QAP results shown in Table 7.13. The analysis is based on different *types* of the QAPLIB benchmark problems.

Types	Dimension	Total Inst.	Avg. δ_{avg}	Avg. δ_{min}	Avg. δ_{max}	Avg. μ_T
I	[12, 100]	15	0.88%	0.61%	1.09%	365.31s
II	[12, 150]	31	0.07%	0.03%	0.11%	1532.13s
III	[12, 36]	29	0.51%	0.22%	1.06%	21.19s
IV	[12, 150]	12	0.11%	0.07%	0.17%	1808.98s

Legends:

Class I - unstructured, randomly generated instances (TAI**A and ROU**).

Class II - grid-based distance matrix instances (NUG**, THO*** and SKO***).

Class III - real-world instances (BUR***, CHR***, KRA***, ELS** and STE***).

Class IV - real-world-like instances (TAI**B).

and 0.11% respectively. The slightly higher average δ_{avg} for unstructured and randomly generated benchmark instances in Class I is due to the distribution of relatively good solutions in the entire feasible solution space. In [140], via the entropy measure of local optima, it is explained that different local optima are very slightly correlated in Class I instances. This indicates that relatively good solutions of these problems are spread in the whole feasible solution space. The proposed BCO algorithm is able to find structure of relatively good solutions. If all the good solutions are distributed close to each other like the real-world and real-world-like instances in Classes III and IV, the proposed BCO algorithm performs well. On the contrary, if all the good solutions are distributed all over a large solution space, the proposed BCO algorithm is able to find near-optimum solutions but not able to get to the global best solution.

Comparison studies which assess the performance of the proposed BCO algorithm compared to various approaches reviewed in Section 3.4 can be found in Section 7.7.

7.6 Performance of the BCO Algorithm on JSSP

This section presents the results and findings of the proposed BCO algorithm on a set of JSSP benchmark problems obtained from the OR-Library. A total of 82 problem instances are employed to test the proposed BCO algorithm. The dimensions of these 82 instances range from 6-job $\times$ 6-machine to 50-job $\times$ 10-machine. The 82 instances are categorized into six major groups as follows:

- 5 instances by Adam et al. (ABZ) [2],
- 10 instances by Applegate and Cook (ORB) [6],
- 3 instances by Fisher and Thompson (FT or MT) [50],
- 40 instances by Lawrence (LA) [88],
- 20 instances by Storer et al. (SWV) [134], and
- 4 instances by Yamada and Nakano (YN) [160].

7.6.1 Parameters Tuning for JSSP

In the parameter tuning experiment for JSSP, the SWV01 problem from the JSSP OR-Library is used as the sample. Table 7.1 in Section 7.2 illustrates the five BCO parameters with their high (+1) and low (-1) levels for this experiment. The BCO algorithm is executed for 3000 iterations, that is $BC_{Max} = 3000$.

Table D.1 in Appendix D presents the experiment results for each configuration. The table is sorted according to the average makespan of each configuration. From Table D.1, $\mu_{min} = \mu_{31} = 1511.0$ and it is obtained via the configuration $i = 31$. This μ_{min} will be used in the hypothesis testing so that the parameter setting for the proposed BCO algorithm is determined.

The hypothesis testing results for the SWV01 problem are shown in Table D.2 (refer to Appendix D). The table illustrates the P-value of F-Test and T-Test together with the

Table 7.16: Recommended BCO parameter settings for the SWV01.

Config.	β	λ	K	N_{Bee}	ϖ
$i = 31$	-1	+1	+1	+1	+1
$i = 30$	+1	-1	+1	+1	+1
$i = 13$	-1	-1	+1	+1	-1
$i = 32$	+1	+1	+1	+1	+1
$i = 16$	+1	+1	+1	+1	-1
Mode	+1	+1	+1	+1	+1

95% confidence interval (CI) for difference between μ_i and μ_{min} . From Table D.2, the T-Test shows that μ_{30} , μ_{13} , μ_{32} and μ_{16} (they are in bold and italic font which their P-value is > 0.05) are statistically equal to $\mu_{min} = \mu_{31}$. After performing a bit-by-bit comparison as explained in Section 7.2, the recommended parameter setting for the SWV01 is shown in Table 7.16. We assume that the setting of these five parameters shown in Table 7.16 is applicable to all problem instances in the JSSP domain. This parameter setting, which is equivalent to the configuration $i = 32$ in Table 7.2, will be applied throughout the JSSP experiments. The results for JSSP experiments will be described in Section 7.6.2.

7.6.2 Results for JSSP

To measure the effect of the major components of the proposed BCO algorithm in the JSSP, 20 selected benchmark problem instances from the JSSP OR-Library are tested with four BCO variants (Λ_1 , Λ_2 , Λ_3 and Λ_4) as shown in Fig. 7.1 and Table 7.3.

In this set of experiments, the recommended BCO parameter setting as illustrated in Table 7.16 was applied: $\beta = 10$, $\lambda = 0.9$, $K = 100$, $N_{Bee} = 25$ and $\varpi = 25$. The BCO algorithm was executed for 10000 BC_{Max} for five replications on each benchmark instance.

Table 7.17 presents the results for this experiment. In terms of the average of δ_{avg} for each BCO variants (Λ_1 , Λ_2 , Λ_3 and Λ_4), they are 1.612%, 1.494%, 1.526%, 1.573%.

Table 7.17: Effect of different variants of the BCO algorithm on 20 selected JSSP OR-Library benchmark problems.

Inst.	Opt.	Λ_1			Λ_2			Λ_3			Λ_4		
		δ_{min}	δ_{avg}	μ_T (s)	δ_{min}	δ_{avg}	μ_T (s)	δ_{min}	δ_{avg}	μ_T (s)	δ_{min}	δ_{avg}	μ_T (s)
LA21	1046	0.860	2.314	2847.08	0.860	1.625	1791.31	1.338	1.759	659.90	1.530	2.103	667.94
LA22	927	1.079	1.381	2464.51	0.863	0.992	1463.26	0.863	1.122	901.48	0.539	1.122	862.72
LA23	1032	0.000	0.000	233.52	0.000	0.000	62.37	0.000	0.000	51.19	0.000	0.000	76.17
LA24	935	1.390	1.882	2026.67	1.604	2.139	1115.26	0.963	1.797	1255.70	1.711	2.246	1224.24
LA25	977	1.126	1.556	3392.76	0.921	1.228	1689.39	0.819	1.576	1001.75	1.126	1.515	915.19
LA26	1218	0.000	0.115	4002.23	0.000	0.000	2380.27	0.000	0.066	792.48	0.000	0.213	1151.80
LA27	1235	2.753	3.109	3577.01	2.429	2.996	2289.58	2.672	2.834	3040.87	2.753	3.304	1567.93
LA28	1216	2.056	2.516	2593.41	1.069	1.924	1575.46	1.480	2.072	2095.83	0.576	1.398	1659.00
LA29	1152	4.774	5.608	5016.73	4.514	5.469	1727.34	5.122	5.313	2226.85	5.122	5.469	1605.16
LA30	1355	0.000	0.000	2848.08	0.000	0.000	794.02	0.000	0.103	1511.56	0.000	0.000	873.65
LA31	1784	0.000	0.000	254.28	0.000	0.000	59.46	0.000	0.000	104.37	0.000	0.000	56.59
LA32	1850	0.000	0.000	96.54	0.000	0.000	121.87	0.000	0.000	125.27	0.000	0.000	111.10
LA33	1719	0.000	0.000	140.83	0.000	0.000	77.52	0.000	0.000	88.84	0.000	0.000	62.72
LA34	1721	0.000	0.000	5992.41	0.000	0.000	1395.67	0.000	0.000	1228.32	0.000	0.000	1112.69
LA35	1888	0.000	0.000	829.44	0.000	0.000	153.29	0.000	0.000	152.91	0.000	0.000	156.26
LA36	1268	1.814	2.177	5768.46	1.814	2.224	2498.61	1.893	2.129	1183.42	1.814	2.366	1731.63
LA37	1397	1.503	2.133	4613.86	1.790	2.162	2648.58	1.074	2.004	1611.30	1.360	1.875	2211.82
LA38	1196	3.094	4.548	5323.85	3.428	4.615	2251.42	3.512	4.916	1737.48	3.595	4.632	1757.96
LA39	1233	1.460	2.693	8023.93	1.865	2.336	2540.63	1.460	2.303	2576.61	1.703	2.709	1095.61
LA40	1222	1.555	2.209	2868.53	0.982	2.177	2866.45	1.964	2.520	1752.89	1.964	2.504	730.49
Average:		1.173	1.612	3145.71	1.107	1.494	1475.09	1.158	1.526	1204.95	1.190	1.573	981.53

Table 7.17 also shows the average time to obtain the best score for each benchmark problem. It is denoted as μ_T . The average μ_T for Λ_1 , Λ_2 , Λ_3 and Λ_4 are 3145.71s (≈ 52 minutes), 1475.09s (≈ 25 minutes), 1204.95s (≈ 20 minutes) and 981.53s (≈ 16 minutes) respectively. Tables 7.17 also shows that the maximum δ_{avg} is 1.612% and the minimum δ_{avg} is 1.494%. The difference between the maximum and the minimum δ_{avg} is 0.118%.

The improvement percentage in terms of average solution cost and μ_T for each benchmark instances is shown in Table 7.18. When Λ_1 is compared against Λ_2 , Λ_3 or Λ_4 , the average improvement percentages in terms of solution cost are -0.024%, +0.010% and +0.032% while the average reduction in terms of μ_T are -47.90%, -54.59% and -63.06%. When Λ_2 is compared against Λ_3 or Λ_4 , the average improvement percentages in terms of solution cost are +0.035% and +0.056% while the average improvement percentages in terms of μ_T are -7.14% and -25.16%. When Λ_3 is compared against Λ_4 , the average improvement percentages in terms of solution cost and μ_T are +0.021% and -12.47%. These

Table 7.18: Improvement percentage in terms of average solution cost and μ_T among the different BCO variants on 20 selected JSSP OR-Library benchmark problems.

Inst.	Λ_1 VS Λ_2 (%)		Λ_1 VS Λ_3 (%)		Λ_1 VS Λ_4 (%)		Λ_2 VS Λ_3 (%)		Λ_2 VS Λ_4 (%)		Λ_3 VS Λ_4 (%)	
	Cost	μ_T	Cost	μ_T	Cost	μ_T	Cost	μ_T	Cost	μ_T	Cost	μ_T
LA21	-0.673	-37.08	-0.542	-76.82	-0.206	-76.54	0.132	-63.16	0.470	-62.71	0.338	1.22
LA22	-0.383	-40.63	-0.255	-63.42	-0.255	-64.99	0.128	-38.39	0.128	-41.04	0.000	-4.30
LA23	0.000	-73.29	0.000	-78.08	0.000	-67.38	0.000	-17.92	0.000	22.12	0.000	48.78
LA24	0.252	-44.97	-0.084	-38.04	0.357	-39.59	-0.335	12.59	0.105	9.77	0.441	-2.50
LA25	-0.323	-50.21	0.020	-70.47	-0.040	-73.03	0.344	-40.70	0.283	-45.83	-0.060	-8.64
LA26	-0.115	-40.53	-0.049	-80.20	0.098	-71.22	0.066	-66.71	0.213	-51.61	0.148	45.34
LA27	-0.110	-35.99	-0.267	-14.99	0.188	-56.17	-0.157	32.81	0.299	-31.52	0.457	-48.44
LA28	-0.578	-39.25	-0.433	-19.19	-1.091	-36.03	0.145	33.03	-0.516	5.30	-0.661	-20.84
LA29	-0.132	-65.57	-0.279	-55.61	-0.132	-68.00	-0.148	28.92	0.000	-7.07	0.148	-27.92
LA30	0.000	-72.12	0.103	-46.93	0.000	-69.32	0.103	90.37	0.000	10.03	-0.103	-42.20
LA31	0.000	-76.62	0.000	-58.95	0.000	-77.75	0.000	75.53	0.000	-4.83	0.000	-45.78
LA32	0.000	26.24	0.000	29.76	0.000	15.09	0.000	2.79	0.000	-8.83	0.000	-11.31
LA33	0.000	-44.95	0.000	-36.92	0.000	-55.46	0.000	14.59	0.000	-19.09	0.000	-29.40
LA34	0.000	-76.71	0.000	-79.50	0.000	-81.43	0.000	-11.99	0.000	-20.28	0.000	-9.41
LA35	0.000	-81.52	0.000	-81.57	0.000	-81.16	0.000	-0.25	0.000	1.94	0.000	2.20
LA36	0.046	-56.69	-0.046	-79.48	0.185	-69.98	-0.093	-52.64	0.139	-30.70	0.232	46.32
LA37	0.028	-42.60	-0.126	-65.08	-0.252	-52.06	-0.154	-39.16	-0.280	-16.49	-0.126	37.27
LA38	0.064	-57.71	0.352	-67.36	0.080	-66.98	0.288	-22.83	0.016	-21.92	-0.271	1.18
LA39	-0.347	-68.34	-0.379	-67.89	0.016	-86.35	-0.032	1.42	0.365	-56.88	0.396	-57.48
LA40	-0.032	-0.07	0.304	-38.89	0.288	-74.53	0.336	-38.85	0.320	-74.52	-0.016	-58.33
Average:	-0.024	-47.90	0.010	-54.59	0.032	-63.06	0.035	-7.14	0.056	-25.16	0.021	-12.47

results imply that the solution quality is sustained while a reduction of μ_T is observed (for all cases). Similar trends as for the TSP results are observed where the enhancements made to the proposed BCO framework (e.g. replacement of the SSSTR with the FSTR, the integration of the BCO variant with a pruning strategy and the replacement of the SFBPS with the DFBPS) sustain the solution quality while a reduction in terms of μ_T is observed. Analysis of these results can be found in Section 7.4.2.

Table 7.19 shows the results of the proposed BCO algorithm on the 82 JSSP OR-Library benchmark problem instances with dimensions range from 6-job $\times$ 6-machine to 50-job $\times$ 10-machine. The instances that have been solved to optimum are marked with *. The variant that produces such results is Λ_4 with the following settings: $\beta = 10$, $\lambda = 0.9$, $K = 100$, $N_{Bee} = 25$ and $\varpi = 25$ and $BC_{Max} = 10000$. Five replications of experiment are conducted for each benchmark instance.

Table 7.19: Computational results of the proposed BCO algorithm on the 82 JSSP OR-Library benchmark problem instances.

Instances	$n \times m$	Optimum	μ_C	σ_C	δ_{avg} (%)	δ_{min} (%)	δ_{max} (%)	μ_T (s)	σ_T (s)
*ABZ05	10×10	1234	1237.6	0.89	0.29	0.16	0.32	359.18	416.08
*ABZ06	10×10	943	944.6	2.19	0.17	0.00	0.42	526.40	284.80
*ABZ07	20×15	656	697.6	1.67	6.34	5.95	6.55	2028.78	1753.50
ABZ08	20×15	665	714.6	2.07	7.46	7.22	7.97	2622.34	1348.81
ABZ09	20×15	679	735.2	5.02	8.28	7.66	9.43	2253.46	1485.48
*FTP06	6×6	55	55.0	0.00	0.00	0.00	0.00	0.43	0.15
*FTP10	10×10	930	949.4	8.26	2.09	0.75	2.80	529.56	392.17
*FTP20	20×5	1165	1173.0	0.00	0.69	0.69	0.69	553.50	381.59
*LA01	10×5	666	666.0	0.00	0.00	0.00	0.00	0.63	0.30
*LA02	10×5	655	655.0	0.00	0.00	0.00	0.00	19.09	5.96
*LA03	10×5	597	597.0	0.00	0.00	0.00	0.00	17.48	14.80
*LA04	10×5	590	590.0	0.00	0.00	0.00	0.00	24.28	22.73
*LA05	10×5	593	593.0	0.00	0.00	0.00	0.00	0.13	0.08
*LA06	10×5	926	926.0	0.00	0.00	0.00	0.00	0.50	0.27
*LA07	10×5	890	890.0	0.00	0.00	0.00	0.00	1.73	0.87
*LA08	10×5	863	863.0	0.00	0.00	0.00	0.00	1.98	0.67
*LA09	10×5	951	951.0	0.00	0.00	0.00	0.00	0.64	0.31
*LA10	10×5	958	958.0	0.00	0.00	0.00	0.00	0.48	0.43
*LA11	20×5	1222	1222.0	0.00	0.00	0.00	0.00	1.18	0.57
*LA12	20×5	1039	1039.0	0.00	0.00	0.00	0.00	0.77	0.30
*LA13	20×5	1150	1150.0	0.00	0.00	0.00	0.00	0.88	0.66
*LA14	20×5	1292	1292.0	0.00	0.00	0.00	0.00	0.34	0.43
*LA15	20×5	1207	1207.0	0.00	0.00	0.00	0.00	15.71	3.28
*LA16	10×10	945	945.2	0.45	0.02	0.00	0.11	301.64	238.49
*LA17	10×10	784	784.0	0.00	0.00	0.00	0.00	68.77	50.14
*LA18	10×10	848	848.0	0.00	0.00	0.00	0.00	286.03	329.82
*LA19	10×10	842	845.8	3.63	0.45	0.00	0.95	504.88	137.89
*LA20	10×10	902	906.0	2.24	0.44	0.00	0.55	198.55	346.23
LA21	15×10	1046	1068.0	5.34	2.10	1.53	2.68	667.94	558.29
LA22	15×10	927	937.4	4.16	1.12	0.54	1.62	862.72	316.92
LA23	15×10	1032	1032.0	0.00	0.00	0.00	0.00	76.17	44.28
*LA24	15×10	935	956.0	2.92	2.25	1.71	2.46	1224.24	521.50
*LA25	15×10	977	991.8	4.32	1.51	1.13	2.05	915.19	681.05
LA26	20×10	1218	1220.6	4.77	0.21	0.00	0.90	1151.80	685.63
*LA27	20×10	1235	1275.8	7.73	3.30	2.75	4.21	1567.93	384.26
LA28	20×10	1216	1233.0	7.18	1.40	0.58	2.06	1659.00	973.53
LA29	20×10	1152	1215.0	3.39	5.47	5.12	5.82	1605.16	821.65
LA30	20×10	1355	1355.0	0.00	0.00	0.00	0.00	873.65	356.89
LA31	30×10	1784	1784.0	0.00	0.00	0.00	0.00	56.59	26.54
LA32	30×10	1850	1850.0	0.00	0.00	0.00	0.00	111.10	76.09
LA33	30×10	1719	1719.0	0.00	0.00	0.00	0.00	62.72	29.91
LA34	30×10	1721	1721.0	0.00	0.00	0.00	0.00	1112.69	760.40
LA35	30×10	1888	1888.0	0.00	0.00	0.00	0.00	156.26	119.71
*LA36	15×15	1268	1298.0	5.20	2.37	1.81	2.76	1731.63	915.71
*LA37	15×15	1397	1423.2	7.05	1.88	1.36	2.43	2211.82	869.92
*LA38	15×15	1196	1251.4	9.24	4.63	3.60	5.60	1757.96	752.90

Continued on next page...

Table 7.19 – continued from previous page

Instances	$n \times m$	Optimum	μ_C	σ_C	δ_{avg} (%)	δ_{min} (%)	δ_{max} (%)	μ_T (s)	σ_T (s)
*LA39	15×15	1233	1266.4	7.23	2.71	1.70	3.16	1095.61	689.72
*LA40	15×15	1222	1252.6	5.81	2.50	1.96	3.27	730.49	442.82
*ORB01	10×10	1059	1064.0	6.93	0.47	0.00	1.32	269.94	371.04
*ORB02	10×10	888	889.0	0.00	0.11	0.11	0.11	370.61	304.87
*ORB03	10×10	1005	1024.0	12.21	1.89	0.00	3.38	567.63	494.99
*ORB04	10×10	1005	1015.6	4.98	1.05	0.60	1.79	581.97	325.01
*ORB05	10×10	887	889.0	0.00	0.23	0.23	0.23	407.46	131.23
ORB06	10×10	1010	1019.6	2.41	0.95	0.69	1.29	514.50	227.51
ORB07	10×10	397	397.8	1.79	0.20	0.00	1.01	707.40	235.03
ORB08	10×10	899	908.4	0.89	1.05	1.00	1.22	352.45	278.49
ORB09	10×10	934	937.0	2.74	0.32	0.00	0.54	433.12	250.33
ORB10	10×10	944	946.2	2.86	0.23	0.00	0.74	535.82	265.94
SWV01	20×10	1407	1511.8	11.82	7.45	6.33	8.39	1691.20	889.69
SWV02	20×10	1475	1566.2	7.56	6.18	5.63	6.64	2002.91	1202.97
SWV03	20×10	1398	1525.6	14.66	9.13	8.23	10.80	1920.06	967.63
SWV04	20×10	1470	1592.0	26.14	8.30	5.31	10.07	1638.60	694.52
SWV05	20×10	1424	1567.6	20.70	10.08	8.01	11.87	2262.47	468.29
SWV06	20×15	1678	1838.4	6.91	9.56	9.12	10.13	2848.77	1975.41
SWV07	20×15	1600	1740.2	11.26	8.76	7.69	9.63	2784.40	1532.94
SWV08	20×15	1756	1962.8	5.22	11.78	11.39	12.07	3401.91	1956.95
SWV09	20×15	1661	1848.8	10.47	11.31	10.72	12.10	4298.45	1457.31
SWV10	20×15	1754	1928.8	15.80	9.97	8.44	10.72	3485.78	1689.29
*SWV11	50×10	2983	3400.6	38.32	14.00	12.57	15.92	6730.82	960.86
SWV12	50×10	3003	3389.6	6.66	12.87	12.52	13.09	2972.04	2016.99
*SWV13	50×10	3104	3508.2	11.19	13.02	12.73	13.60	3395.18	2681.42
SWV14	50×10	2968	3285.0	11.85	10.68	10.31	11.19	3965.70	2807.73
SWV15	50×10	2904	3237.8	13.41	11.49	10.71	11.88	5643.20	1327.18
SWV16	50×10	2924	2924.0	0.00	0.00	0.00	0.00	11.31	11.06
SWV17	50×10	2794	2794.0	0.00	0.00	0.00	0.00	100.83	84.77
SWV18	50×10	2852	2852.0	0.00	0.00	0.00	0.00	17.78	11.53
SWV19	50×10	2843	2843.0	0.00	0.00	0.00	0.00	383.11	67.93
SWV20	50×10	2823	2823.0	0.00	0.00	0.00	0.00	17.14	15.24
YN01	20×20	885	940.6	7.30	6.28	4.86	6.89	2428.50	1370.40
YN02	20×20	909	966.8	8.67	6.36	5.17	7.37	4147.01	2313.43
YN03	20×20	892	956.2	3.11	7.20	6.61	7.51	5171.78	2615.46
YN04	20×20	968	1055.4	9.24	9.03	7.75	10.23	2359.63	2260.99

The δ_{avg} column in Table 7.19 shows that the proposed BCO algorithm (variant Λ_4 , BCO_FSTR+TESN5_TL+DFBPS) is able to obtain $\leq 1\%$ deviation from known optimum for 44 out of 82 instances. This indicates that 53.66% of the 82 problem instances are solved to $\leq 1\%$ from optimum. The largest deviation from known optimum for the entire dataset is 14.00%, which is obtained when solving the SWV11. Furthermore,

Table 7.20: Analysis of the JSSP results shown in Table 7.19. The analysis is based on different series of the JSSP OR-Library benchmark problems.

Series	Total Instances	Average of				Standard Deviation of			
		$\delta_{min}(\%)$	$\delta_{avg}(\%)$	$\delta_{max}(\%)$	$\mu_T(s)$	$\delta_{min}(\%)$	$\delta_{avg}(\%)$	$\delta_{max}(\%)$	$\mu_T(s)$
ABZ [2]	5	4.20	4.51	4.94	1558.03	3.81	3.96	4.29	1041.57
ORB [6]	10	0.26	0.65	1.16	474.09	0.37	0.57	0.94	130.62
FT/MT [50]	3	0.48	0.92	1.16	361.16	0.42	1.06	1.46	312.63
LA [88]	40	0.59	0.81	1.02	526.91	1.15	1.38	1.62	655.84
SWV [134]	20	6.98	7.73	8.40	2478.58	4.66	4.94	5.33	1877.26
YN [160]	4	6.10	7.22	8.00	3526.73	1.34	1.28	1.51	1373.46

the proposed BCO algorithm solves 30 instances to optimum, which is equivalent to 36.59% of the 82 instances. The average of δ_{avg} , δ_{min} and δ_{max} in Table 7.19 are 3.020%, 2.597% and 3.421% respectively.

The results shown in Table 7.19 are further analyzed according to a classification technique where the 82 JSSP OR-Library benchmark problems are categorized into six different series. The analysis results are reported in Table 7.20. Taking the ORB series as an example, the BCO variant Λ_4 solves all the problem instances in that series to an average of 0.65% from optimum within 474.09s. Besides the ORB series, it is also noticed that the BCO variant Λ_4 solves the FT/MT and LA series to $\leq 1\%$ deviation from known optimum. For the ABZ, SWV and YN series, the average of δ_{avg} are 4.51%, 7.73% and 7.22% from known optimum. Instances in the ABZ, SWV and YN series are difficult problems. They are classified as difficult/hard problems according to the criteria mentioned in [75] as follows: $n \times m \geq 200$, $n \geq 15$, $m \geq 10$ and $n < 2.5m$. For SWV11-15 instances, even though the criterion of $n < 2.5m$ is not met, they are still considered intractable because of the following constraint: a machine sequence for a job is generated by passing all machines of one set using uniformly distributed assignments before it turns into a machine of the next set [75, 134]. The proposed BCO algorithm is able to get good results ($\leq 1\%$ deviation from known optimum/upper bound) for the instances in ORB, FT/MT and LA series which violate the criteria for hard problems classification. This is because the BCO algorithm is able to select the bottleneck machine/processor when the

coefficient of work load variation increases [75]. This reduces the likelihood of becoming trapped in local minima and thus good solutions can be found.

In the subsequent section (refer to Section 7.7), comparison studies which assess the performance of the proposed BCO algorithm compared to various approaches reviewed in Section 3.4 are described.

7.7 Comparison Studies

This section presents some comparison studies of this research. The aim is to assess the performance of the proposed BCO algorithm when compared to various algorithms reviewed in Section 3.4.3. Ideally, all the benchmark algorithms used in the comparison study should be re-implemented and executed on the same hardware platform for a fair comparison on the accuracy and the performance. However, such a re-implementation can be bias due to various factors such as programmer's skills, parameters tuning and difficulty to obtain accurate information from the reported publication. Thus, this thesis takes an alternative approach for the comparison study whereby the comparison is carried out according to the reported results in the existing publications.

Table 7.21 highlights the comparison of the proposed BCO algorithm against various ACO, PSO and ILS algorithms. The comparison results are grouped according to the COP being solved. For TSP, QAP and JSSP, seven, four and five respective benchmark algorithms are included in such comparison study respectively. For each benchmark algorithm, the average of δ_{min} and δ_{avg} for all benchmark problem instances reported in the literatures are listed.

The comparison analysis is carried out from two different perspectives: performance evaluation based on the same amount of allowed solution construction as indicated in Table 7.22 and the peak/best performance evaluation with longer algorithm execution of the proposed BCO algorithm with respect to the solution quality of the final solutions

Table 7.21: Performance comparison of the BCO algorithm versus various ACO, PSO and ILS algorithms. Average of δ_{min} and δ_{avg} for all the instances reported in each literature are shown.

Approaches	Total Instances	Average δ (%)					
		δ_{min}	δ_{avg}	BCO(A)		BCO(B)	
				δ_{min}	δ_{avg}	δ_{min}	δ_{avg}
GACS+2-opt [89] ^T	45	0.35	0.39	0.55	0.82	0.06	0.09
TSACS+2-opt [119] ^T	11	1.78	n/a	0.91	1.14	0.11	0.13
ACO+GA [164] ^T	9	0.10	0.20	0.32	0.43	0.08	0.11
PSO+LS [61] ^T	20	0.01	0.12	1.26	1.55	0.27	0.44
HDPSO [95] ^T	6	4.54	5.38	0.00	0.00	0.00	0.00
C3DPSO [165] ^T	6	0.22	1.74	0.00	0.02	0.00	0.00
RAACS [120] ^T	4	0.98	1.14	n/a	n/a	0.26	0.33
PHAS [124] ^Q	109	0.42	n/a	2.26	4.05	0.14	0.27
PSOFA [62] ^Q	8	n/a	3.49	0.13	0.24	0.13	0.24
ILS [123] ^Q	131	0.66	n/a	n/a	n/a	0.15	0.33
AntSimulated [36] ^Q	38	n/a	0.21	n/a	n/a	0.26	0.44
ACOFIT [73] ^J	21	0.11	0.29	7.10	8.34	2.12	2.65
MPSO [98] ^J	43	0.15	n/a	1.43	1.97	0.59	0.82
HPSO [131] ^J	43	0.03	0.12	0.59	0.82	0.59	0.82
ACOPSS [130] ^J	20	0.93	2.25	n/a	n/a	0.00	0.05
HTSACO [49] ^J	12	n/a	1.20	n/a	n/a	8.34	9.46

Table 7.22: Algorithm execution settings used by the benchmark approaches and the proposed BCO algorithm (BCO(A)) reported in Table 7.21.

Approaches	Algorithm Execution Settings	
	$n \equiv$ number of cities; $M \equiv$ number of machines.	
	Benchmark	BCO(A)
GACS+2-opt [89] ^T	$5n$ iterations, 10 ants.	n iterations, 50 bees.
TSACS+2-opt [119] ^T	1000 iterations, 10 ants.	200 iterations, 50 bees.
ACO+GA [164] ^T	10000 iterations, 35 ants.	700 iterations, 50 bees.
PSO+LS [61] ^T	2000 iterations, 20 particles.	800 iterations, 50 bees.
HDPSO [95] ^T	$2500n$ tour constructions.	$50n$ iterations, 50 bees.
C3DPSO [165] ^T	$10n$ iterations, 30 particles.	$6n$ iterations, 50 bees.
PHAS [124] ^Q	$n/2$ iterations, $2n$ ants.	$n^2/50$ iterations, 50 bees.
PSOFA [62] ^Q	5000 or 10000 iterations, 100 particles.	10000 iterations, 50 bees.
ACOFIT [73] ^J	320 iterations, $\lfloor M/3 \rfloor$ ants.	$\lfloor M/3 \rfloor \times 320/25$ iterations, 25 bees.
MPSO [98] ^J	300 or 500 iterations, 30 particles.	360 iterations, 25 bees.
HPSO [131] ^J	10000 iterations, 30 particles.	10000 iterations, 25 particles.

when compared to other benchmark algorithms. These two approaches are generally used in evaluating different application domains (e.g. [73] uses the first approach and [95, 164] use the latter approach). They are indicated by BCO(A) and BCO(B) columns in Table 7.21 respectively. Taking the GACS+2-opt algorithm as an example, it solved 45 benchmark problem instances to an average of $\delta_{avg} = 0.39\%$ and $\delta_{min} = 0.35\%$ after $5n$

iterations (n denotes the number of cities of a TSP) with 10 ants in its algorithm execution (equivalent to a maximum of $5n \times 10 = 50n$ allowed solution constructions). When it is compared to the proposed BCO algorithm with the same amount of allowed solution constructions, it shows a slight improvement of $0.82\% - 0.39\% = 0.43\%$. On the other hand, based on the 11 benchmark problem instances reported in [119], the TSACS+2-opt algorithm achieves $\delta_{min} = 1.78\%$ after $1000 \times 10 = 10000$ allowed solution constructions. The BCO algorithm outperforms the TSACS+2-opt algorithm by $1.78\% - 0.91\% = 0.87\%$ with the same amount of allowed solution constructions. If only the best performance is considered (the BCO(B) column), the BCO algorithm outperforms both the GACS+2-opt and TSACS+2-opt algorithms.

In the TSP category where the comparison is carried out based on the same amount of allowed solution constructions, the BCO algorithm outperforms the TSACS+2-opt, HDPSO and C3DPSO algorithms and is comparable to the GACS+2-opt, ACO+GA and PSO+LS algorithms. A gap of 1.43% in terms of δ_{avg} between the proposed BCO algorithm and the PSO+LS is observed. We believe that the superior results shown by the PSO+LS algorithm as compared to the proposed BCO algorithm is because of the integration of the Lin-Ker heuristic. With the integration of an efficient local optimization technique, the overall performance of the proposed BCO algorithm can be improved. In the QAP category, the BCO algorithm outperforms the PSOFA algorithm by 3.25% but underperforms the PHAS algorithm by 1.84% . The PHAS is an extension of the HAS-QAP algorithm [53] which is a specific algorithm to solve the QAP. Although the proposed BCO algorithm slightly underperforms the PHAS algorithm, the BCO algorithm is able to solve other COPs due to its generic feature. In the JSSP category, the proposed BCO algorithm underperforms the ACOFT algorithm and is comparable to the HPSO and MPSO algorithms. The ACOFT algorithm involves a hybrid of the ACO algorithm with the Fast Taboo Search method [108]. The Fast Taboo Search is

one of the advanced methods which is able to solve the JSSP to a satisfactory level. The Fast Taboo Search performance very much depends on the provided initial solution [76]. Therefore, in the ACOFT algorithm, the ACO algorithm with exploration and information learning abilities is used to generate initial solutions for the Fast Taboo Search. This research motivates us to explore the possibility to implement such hybridization as part of the proposed BCO framework. The hybridization should be developed as a domain independent feature so that the effect would be applicable across all the COPs being solved.

The comparison between the proposed BCO algorithm and the BS algorithm [102, 103] is shown in Table 7.23. Ten TSPLIB benchmark instances were employed in the comparison. The dimensions of the selected benchmark instances are in the range of [51, 1002]. As reported in [102, 103], the results for the BS algorithm were obtained with the following settings: (i) a total of 20 executions for each problem instance were conducted on an IBM computer with a 533MHz PIII processor; and (ii) each algorithm execution was limited to 100 cycles. The speed of the processor that runs the proposed BCO algorithm is 2.00GHz (refer to Section 7.3). This indicates that speed of the processor that runs the proposed BCO algorithm is about $\frac{2000}{533} = 3.75$ times faster than the processor used in the BS experiments².

In Table 7.23, if the average of δ_{avg} and μ_T are computed based on the first eight instances (i.e. EIL51, BERLIN52, ST70, PR76, KROA100, EIL101, TSP225 and A280), the BS+2-opt algorithm requires 570.3s to solve the eight problem instances to an average of 3.021% from known optimum. For the BS+3-opt algorithm, δ_{avg} and μ_T are 0.440% and 2253.8s respectively. If a 2.00GHz processor was used to run the BS+2-opt and BS+3-opt algorithms, the running time to obtain similar outcome should be reduced to $\frac{570.3s}{3.75} = 152.1s$ and $\frac{2253.8s}{3.75} = 601.0s$ respectively. The proposed BCO algorithm requires

²Note that other factors such as memory hierarchy and system architecture are not considered.

Table 7.23: Performance comparison of the BCO algorithm versus the BS algorithm.

Instances	Optimal	BS+2-opt [102, 103]			BS+3-opt [102, 103]			BCO		
		δ_{min} (%)	δ_{avg} (%)	μ_T (s)	δ_{min} (%)	δ_{avg} (%)	μ_T (s)	δ_{min} (%)	δ_{avg} (%)	μ_T (s)
EIL51	426	0.530	1.140	44	0.000	0.000	29	0.000	0.000	2
BERLIN52	7542	0.000	1.190	18	0.000	0.000	0	0.000	0.000	0
ST70	675	0.220	1.060	238	0.000	0.000	7	0.000	0.000	1
PR76	108159	0.580	1.190	127	0.000	0.000	2	0.000	0.000	2
KROA100	21282	0.730	1.360	58	0.000	0.000	10	0.000	0.000	2
EIL101	629	0.350	3.970	146	0.000	0.450	61	0.000	0.000	18
TSP225	3916	5.350	6.600	2076	1.060	1.310	11651	0.000	0.000	128
A280	2579	5.950	7.660	1855	0.830	1.760	6270	0.000	0.000	106
PCB442	50778	-	-	-	1.150	1.920	4384	1.534	1.860	866
PR1002	259045	-	-	-	3.190	3.820	28101	2.475	3.624	8511
Avg of the first eight instances:		1.714	3.021	570.3	0.236	0.440	2253.8	0.000	0.000	32.3
		Avg of all ten instances:			0.623	0.926	5051.5	0.401	0.548	963.6

32.3s to solve the eight problem instance to optimum. The results show that the BCO algorithm takes shorter time to generate better solutions.

The average of δ_{avg} and μ_T for the BS+3-opt algorithm based on the ten problem instances in Table 7.23 are 0.926% and 5051.5s respectively. If the experiment was conducted using a 2.00GHz processor, the computational time should be reduced to $\frac{5051.5s}{3.75} = 1347.1s$. The BCO algorithm still outperforms the BS+3-opt algorithm in terms of the results quality and the computational time. The BCO algorithm takes 963.6s to solve all the benchmark problem instances to an average of 0.548% from known optimum.

Table 7.24 shows another comparison study that is based on the DIMACS Implementation Challenge for TSP [77]. 20 instances with dimension more than 1000 cities are used in such comparison. Taking PR1002 as an example, 132 different implementations of various algorithms has been tested on it. Among them are the metaheuristic technique (e.g. Tabu Search), classic tour construction algorithms (e.g. Nearest Neighbourhood heuristic, Boruvka Algorithm) and iterated local search techniques (e.g. Iterated Lin-Kernighan, Iterated 2/3-opt). Out of these 132 approaches, 25% of them (Q_1) record less than 0.61% as the deviation percentage from known optimum, 50% of the approaches (Q_2) are less than 2.52% and 75% of the approaches (Q_3) are less than 12.87%. When

the proposed BCO algorithm is compared against these approaches, it is ranked at the position of 37, which is within the 30th percentile. Out of 20 problem instances:

- 1 problem instance (equivalent to 5%) is ranked within 10th percentile;
- 2 problem instances (equivalent to 10%) are ranked within 20th percentile;
- 8 problem instances (equivalent to 40%) are ranked within 30th percentile;
- 1 problem instance (equivalent to 5%) is ranked within 40th percentile;
- 6 problem instances (equivalent to 30%) are ranked within 50th percentile; and
- 2 problem instances (equivalent to 10%) are ranked within 60th percentile.

It is also noted that 90% of the problem instances (all the problem instances except VM1748 and U2319) are ranked within the second quartile (Q_2). 25% of the problem instances (SI1032, D1291, RL1304, RL1323 and FL1577) are ranked within the first quartile (Q_1).

7.8 Summary

In this chapter, the findings and results for the proposed BCO algorithm are presented. The performance of the BCO algorithm is evaluated based on three selected COPs, namely: the TSP, QAP and JSSP. For each COP, a set of parameter tuning experiments was conducted. The parameter tuning experiments were conducted using the factorial design. The outcomes from the factorial design are further analyzed using the bit-by-bit mode comparison so that the BCO parameters for each COP are tuned effectively.

To examine the flexibility of enhancing the BCO algorithm and the applicability of the enhancement across all the three selected COPs, four different BCO variants are created as follows: Λ_1 (BCO_SSTR+LS), Λ_2 (BCO_FSTR+LS), Λ_3 (BCO_FSTR+LS+SFBPS) and Λ_4 (BCO_FSTR+LS+DFBPS). The local search procedure of each BCO variants is specific to a problem domain being solved. Hence, the LS in the list refers to either

Table 7.24: Performance comparison of the BCO algorithm against the results reported in the DIMACS challenge.

Instances	Variation of methods	% over optimum reported in the DIMACS challenge	Quartiles Q_1, Q_2, Q_3	δ_{Avg} of BCO, (%)	Rank
PR1002	132	[0.00, 98.24]	0.61, 2.52, 12.87	0.85 (within 30 th percentile)	37
SI1032	60	[0.00, 235.87]	0.15, 0.25, 0.47	0.00 (within 10 th percentile)	6
U1060	95	[0.00, 33.66]	0.23, 0.95, 3.64	0.76 (within 50 th percentile)	41
VM1084	95	[0.00, 25.17]	0.11, 0.90, 3.13	0.50 (within 50 th percentile)	42
PCB1173	132	[0.00, 98.10]	0.62, 2.31, 13.49	0.92 (within 30 th percentile)	38
D1291	96	[0.00, 25.50]	0.52, 2.11, 5.56	0.45 (within 30 th percentile)	24
RL1304	132	[0.00, 118.76]	0.96, 3.03, 15.43	0.58 (within 20 th percentile)	26
RL1323	132	[0.00, 102.38]	0.53, 1.94, 14.00	0.50 (within 30 th percentile)	32
NRW1379	131	[0.00, 99.16]	0.36, 1.77, 12.15	1.36 (within 50 th percentile)	63
FL1400	96	[0.00, 33.01]	0.28, 1.21, 4.23	0.35 (within 30 th percentile)	26
U1432	131	[0.00, 99.43]	0.48, 2.53, 13.65	2.02 (within 50 th percentile)	63
FL1577	94	[0.00, 24.04]	1.01, 3.90, 8.89	0.54 (within 20 th percentile)	18
D1655	96	[0.00, 21.13]	0.45, 1.75, 4.33	1.06 (within 40 th percentile)	37
VM1748	94	[0.00, 27.19]	0.16, 0.70, 3.41	0.93 (within 60 th percentile)	50
U1817	95	[0.00, 25.74]	0.68, 1.80, 3.87	1.36 (within 50 th percentile)	41
RL1889	132	[0.00, 103.18]	0.56, 2.93, 13.58	0.87 (within 30 th percentile)	39
D2103	94	[0.00, 14.85]	0.48, 1.60, 2.79	0.77 (within 30 th percentile)	29
U2152	95	[0.00, 27.77]	0.69, 2.17, 4.57	1.50 (within 50 th percentile)	42
U2319	129	[0.00, 99.97]	0.25, 1.09, 7.75	1.77 (within 60 th percentile)	77
PR2392	132	[0.00, 97.60]	0.72, 3.11, 14.13	1.04 (within 30 th percentile)	40

FRNN_2-opt_TL (for the TSP), 2-opt_TL (for the QAP) or TESN5_TL (for the JSSP). Further details on the experimental setup can be found in Section 7.3. From the experimental results, it is noted that if an enhancement is made to the generic BCO framework, the enhancement effect is applicable to all different COPs.

Table 7.25 gives a summary of the performance of the proposed BCO algorithm. It shows that the proposed BCO algorithm is able to solve the TSPLIB benchmark problems to 0.27% from known optimum. For QAP and JSSP, the average δ_{avg} are 0.33% and 3.02% respectively. In terms of the number of instances that are solved to optimum, the proposed BCO algorithm solves 53% of the TSPLIB instances to optimum. For QAP and JSSP, 63% and 37% of the total instances are solved to optimum.

In the subsequent chapter, conclusions and future work will be presented.

Table 7.25: A summary of the performance of the proposed BCO algorithm.

Properties	TSP	QAP	JSSP
Number of instances	95	134	82
Source of benchmark instances	TSPLIB	QAPLIB	OR-Library
Average δ_{min}	0.19%	0.15%	2.60%
Average δ_{avg}	0.27%	0.33%	3.02%
Average δ_{max}	0.35%	0.59%	3.42%
Average μ_T	3938.71s	1099.79s	1199.63s
Number of instances that are solved to optimum	50($\approx 53\%$)	84($\approx 63\%$)	30($\approx 37\%$)
Number of instances that are solved to $\leq 1\%$ from known optimum/upper bound	88($\approx 93\%$)	122($\approx 91\%$)	44($\approx 54\%$)

Chapter 8

Conclusions and Future Work

A generic BCO framework has been proposed to solve different Combinatorial Optimization Problems (COPs). The underlying mechanism of the generic BCO framework is the bee foraging behaviour. In a bee colony, bees travel from one location to another location in order to discover new food source or to collect food. When a bee has found a new food source, it would perform waggle dance to other mates upon flying back to its hive. By doing so, important information about the newly discovered food source (e.g. direction and distance) is conveyed to its mates. If another bee is attracted by the waggle dance, it would fly towards the newly discovered food source. This eventually will cause more and more bees to be recruited towards the newly discovered food source.

This collective intelligence shown in the bee foraging behaviour has been computationally realized as a generic BCO framework to solve different COPs. The proposed generic framework is developed using the object-oriented approach such that it consists of a set of domain independent classes and a set of abstract classes. The domain independent classes implement the bee foraging behaviour while the abstract classes serves as an interface to interact with a set of domain specific classes. To solve a COP with the proposed framework, the implementation of domain specific tasks as what have been defined in the abstract classes needs to be supplied to the framework. The advantages of implementing the bee collective intelligence as a generic framework are two-fold as

follows. First, it is able to solve different COPs. The addition of a new COP can be done via inheriting and implementing the methods as defined in the abstract classes. Second, the generic BCO framework allows the domain independent classes to be amended or enhanced flexibly. Once an enhancement has been incorporated to any of the domain independent classes, the effect will be applicable across all the COPs being solved.

There are six major components in the proposed generic BCO framework: an initialization engine, a local search procedure, a state transition rule, a pruning strategy, a waggle dance mechanism and a parameters controller. The initialization engine and the local search procedure are domain specific where their implementation depends on a COP being solved. The state transition rule, the pruning strategy, the waggle dance mechanism and the parameters controller are implemented as part of the domain independent classes in the generic BCO framework. All the six components are integrated algorithmically to solve different COPs by producing a solution which is a permutation of numbers without repetition. As long as the solution of a COP can be represented and interpreted in this format, the COP can be ported to the proposed BCO framework.

The Fragmentation State Transition Rule (FSTR) is proposed to computationally realize the bee behaviour where a bee travels across different locations to discover new food source. The main feature of the FSTR is to allow a bee to construct a feasible solution incrementally by adding a fragment of nodes at a particular time. The selection of a fragment is performed under the influence of arc fitness and heuristic distance. The FSTR is proposed to overcome the limitation of the Step-by-step State Transition Rule (SSTR), which is proposed prior to the FSTR. As the SSTR constructs a solution via a step-by-step pattern by adding one node at one time, it might not be scalable for problems with a larger dimension. With the FSTR, the high computational overhead is reduced.

The Dynamic Frequency-based Pruning Strategy (DFBPS) which is based on the frequency of the smallest building block of every generated solution is proposed as part

of the BCO algorithm. The DFBPS allows only a subset of solutions to be locally optimized by a local search procedure. It has the ability to dynamically tune the two parameters used in the pruning mechanism. Prior to the DFBPS, the Static Frequency-based Pruning Strategy (SFBPS) has been proposed. In the SFBPS, the parameters are statically tuned. This becomes the limitation of the SFBPS. The proposal of the DFBPS removes the limitation of SFBPS.

The replacements of the SSTR with the FSTR and the SFBPS with the DFBPS serve as showcases that illustrate the generic BCO framework can be altered flexibly. The effect of the replacements is applicable to all the selected COPs, namely: the TSP, QAP and JSSP.

Waggle dance serves as an important communication medium among bee. The waggle dance model is implemented as follows. A bee would observe dances before starting to forage. Once the bee has decided to follow a dance, it would pick a dance from a pool of dances to follow. If the bee has successfully found a better solution, it would dance upon returning to its hive. The dance would last for a longer duration if the solution was much better than the average profitability of the entire colony. With this, more bees are recruited towards the newly discovered food source.

The initialization engine and the local search procedure are specifically implemented according to a COP being solved as defined in the abstract classes. The initialization strategies that have been developed are as follows: the Nearest Neighbourhood heuristic, a state transition heuristic, a random permutation method and a set of dispatching rules. The following local search procedures have been developed: the Fixed-radius Near Neighbour 2-opt (FRNN 2-opt), a 2-opt local search procedure and the Two-enhancement Scheme with Neighbourhood N5 Perturbation (TESN5). The supply of the methods for initialization and local search is done via a mix-and-match strategy according to the problem being solved.

Table 8.1: Performance summary of the proposed BCO algorithm.

Properties	TSP	QAP	JSSP
Number of instances	95	134	82
Source of benchmark instances	TSPLIB	QAPLIB	OR-Library
Average δ_{min}	0.19%	0.15%	2.60%
Average δ_{avg}	0.27%	0.33%	3.02%
Average δ_{max}	0.35%	0.59%	3.42%
Average μ_T	3938.71s	1099.79s	1199.63s
Number of instances that are solved to optimum	50($\approx 53\%$)	84($\approx 63\%$)	30($\approx 37\%$)
Number of instances that are solved to $\leq 1\%$ from known optimum/upper bound	88($\approx 93\%$)	122($\approx 91\%$)	44($\approx 54\%$)

Three COPs are selected as the testbed for the proposed BCO framework. They are the Traveling Salesman Problem (TSP), the Quadratic Assignment Problem (QAP) and the Job Shop Scheduling Problem (JSSP). The results are compared against other bio-inspired algorithms such as Ant Colony Optimization, Particle Swarm Optimization and other variants of bee related algorithms. Out of 95 TSP benchmark instances, 53% of them are solved to optimum and 93% of them record $< 1\%$ deviation from known optimum. On average of all 95 instances, they are solved to 0.27% deviation from known optimum. For QAP, a total of 134 benchmark instances are tested. 63% of them are solved to optimum. On average of 134 instances, they are solved to 0.33% deviation from known optimum. For JSSP, out of 82 benchmark instances, 37% of them are solved to optimum. On average of 82 instances, they are solved to 3.02% deviation from known optimum. Table 8.1 gives a summary of the performance of the proposed BCO algorithm.

Figure 8.1: Linear function for duration calculation.

8.1 Future Work

This section proposed some enhancements that could be done to the proposed BCO algorithm. One of these is the improvement on the waggle dance duration determination. In the current proposed algorithm, the duration of a waggle dance is determined by a linear equation (Eq. 5.6). A graph corresponds to the linear function is shown in Fig. 8.1. It determines the duration, D_i based on the following criteria:

- (i) when $Pf_i = Pf_{colony} \Rightarrow D_i = K$.
- (ii) when $Pf_i < Pf_{colony} \Rightarrow D_i < K$.
- (iii) when $Pf_i > Pf_{colony} \Rightarrow D_i > K$.

The linear equation might have a limitation to reward or penalize instances near the inflection point (noted as regions A and B in Fig. 8.1). Penalizing instances with Pf_i just slightly lower or slightly higher than Pf_{colony} (instances that falls in regions A and B)

should be avoided too. To overcome this limitation, the fuzzy logic concept can be applied such that the duration determination is developed around the concept of reasoning in degrees, rather than reasoning in absolute. Various types of membership functions such as triangular, trapezoidal, sigmoid and bell-shaped can be considered.

In order to enhance the proposed BCO algorithm, parameters configuration is an aspect which needs to be investigated further. In this thesis, we employ a 2^5 factorial design of experiment with ten replicates to tune the five user-defined parameters, namely β , λ , K , N_{Bee} and ϖ (refer to Section 7.2). The upper and lower bounds listed in Table 7.1, Section 7.2 are configured empirically after considering various factors such as accuracy performance and computational cost. An investigation on using an optimization algorithm [44, 93, 162] to optimally determine the five parameters will be conducted. Also, this can be extended to optimally tune the values listed in the lookup table 5.1 in Section 5.4.1. A sensitivity study of the parameters listed in the lookup table (Table 5.1) will be conducted.

We would also like to enhance the frequency-based pruning strategy in the proposed BCO algorithm. Currently, the strategy considers only the smallest building blocks (i.e. two-element blocks). The investigation on applying different sizes of building blocks will be one of our future studies. A trade-off between the computational cost and accuracy as well as the sensitivity analysis will be conducted.

The proposed BCO algorithm has been integrated with a local search procedure and a pruning strategy. The local search procedure, which is implemented as a domain specific class, still shows a high computational overhead especially for larger problems. For TSP, other strategies such as the Lin-Kernighan heuristic will be considered. The Lin-Kernighan heuristic is another effective local search that is able to generate optimal or near-optimal solutions for symmetrical TSP. It is a dynamic k -opt heuristic. One of the successful implementations is available in the Concorde package developed by GATECH

[7]. An investigation on how to optimize the resulting solutions from BCO using the Lin-Kernighan heuristic in the Concorde package will be conducted. For QAP, the 2-opt local search procedure can be developed such that a set of don't look bits are incorporated [13]. This ensures that the compare and swap procedure will not consider the don't look bits and hence minimize the high computational overhead. For JSSP, an investigation on how to integrate the proposed BCO algorithm with the Fast Taboo method [108, 109] will be conducted.

The proposed BCO algorithm will be tested on other benchmark problems. For TSP, two different sets of benchmark problems are identified: National and VLSI TSPs. The collection of these dataset can be obtained from GATECH website (refer to Section 2.4 for details). National TSPs consists of 27 problems, ranging in size from 28 cities in Western Sahara to 71,009 cities in China. To date, of these 27 problems, 13 remained unsolved. VLSI TSPs consists of 102 problems (ranging from 131 cities to 744710 cities) and to date, 62 of them are still open to new algorithms for improvement. This provides a challenge to our proposed BCO algorithm. For JSSP, the Taillard benchmark problem instances as explained in Table 2.2 will be tested.

The proposed BCO framework will be tested with other COPs besides the three selected COPs in this thesis. COPs such as Knapsack problem, Sequential Ordering Problem can be ported into the BCO framework by providing the implementations of the domain specific tasks as explained in Chapter 4. The successful application on the mentioned COPs will make the proposed BCO framework an effective tool to solve real-world applications.

Appendix A

Author's Related Publications

Conference Papers

- (i) L.P. Wong, M.Y.H. Low, and C.S. Chong, "A bee colony optimization algorithm for traveling salesman problem," in *Proceedings of the Second Asia International Conference on Modelling & Simulation*, pp. 818-823, IEEE Computer Society, 2008.
- (ii) L.P. Wong, M.Y.H. Low, and C.S. Chong, "Bee colony optimization with local search for traveling salesman problem," in *Proceedings of the 6th IEEE International Conference on Industrial Informatics (INDIN 2008)*, pp. 1019-1025, IEEE, 2008.
- (iii) L.P. Wong, C.Y. Puan, M.Y.H. Low and C.S. Chong, "Bee colony optimization algorithm with big valley landscape exploitation for job shop scheduling problems," in *Proceedings of the 2008 Winter Simulation Conference*, pp. 2050-2058, 2008.
- (iv) L.P. Wong, M.Y.H. Low, and C.S. Chong, "An efficient bee colony optimization algorithm for traveling salesman problem using frequency-based pruning," in *Proceedings of the 7th IEEE International Conference on Industrial Informatics (INDIN 2009)*, pp. 775-782, 2009.
- (v) L.P. Wong, M.Y.H. Low, and C.S. Chong, "A bee colony optimization algorithm with the fragmentation state transition rule for traveling salesman problem," in

Proceedings of the 2009 Conference on Innovative Production Machines and Systems (IPROMS 2009), pp. 399-404, 2009.

- (vi) L.P. Wong, M.Y.H. Low, and C.S. Chong, "A generic bee colony optimization framework for combinatorial optimization problems," in *Proceedings of the Fourth Asia International Conference on Mathematical/Analytical Modelling & Computer Simulation*, pp. 144-151, IEEE Computer Society, 2010.

Journal Papers

- (i) L.P. Wong, M.Y.H. Low, and C.S. Chong, "Bee colony optimization with local search for traveling salesman problem," *International Journal on Artificial Intelligence Tools (IJAIT)*, vol. 19, no. 3, pp. 305-334, World Scientific Publishing, 2010.
- (ii) L.P. Wong, C.Y. Puan, M.Y.H. Low, C.S. Chong and Y.W. Wong, "Bee colony optimization algorithm with big valley landscape exploitation for job shop scheduling problems," *International Journal of Bio-Inspired Computation (IJBIC)*, vol. 2, no. 2, pp. 85-99, Inderscience Publishers, 2010.

Appendix B

Parameters Tuning Results for TSP

Table B.1: Results of the BCO parameters tuning experiment for TSP. The RAT575 is used as the testbed.

Config.	Rep 1	Rep 2	Rep 3	Rep 4	Rep 5	Rep 6	Rep 7	Rep 8	Rep 9	Rep 10	Average, μ_i
$i = 29$	6821	6820	6835	6822	6814	6825	6822	6804	6835	6847	6824.5
$i = 25$	6815	6819	6823	6836	6832	6827	6854	6835	6825	6816	6828.2
$i = 16$	6846	6824	6843	6804	6870	6832	6819	6856	6801	6830	6832.5
$i = 14$	6827	6814	6828	6847	6819	6834	6847	6831	6827	6852	6832.6
$i = 09$	6831	6848	6818	6853	6833	6842	6839	6840	6840	6853	6839.7
$i = 31$	6851	6848	6832	6827	6870	6825	6867	6832	6826	6835	6841.3
$i = 13$	6839	6837	6851	6849	6845	6817	6839	6821	6869	6855	6842.2
$i = 28$	6834	6828	6834	6851	6850	6849	6834	6827	6852	6866	6842.5
$i = 15$	6838	6850	6834	6827	6852	6840	6862	6841	6831	6855	6843.0
$i = 30$	6829	6841	6821	6847	6822	6848	6848	6872	6859	6852	6843.9
$i = 10$	6824	6872	6838	6849	6843	6836	6834	6834	6849	6864	6844.3
$i = 12$	6853	6855	6838	6829	6829	6840	6859	6831	6852	6869	6845.5
$i = 11$	6858	6855	6863	6852	6839	6858	6838	6826	6830	6864	6848.3
$i = 27$	6845	6857	6859	6835	6861	6831	6880	6844	6848	6851	6851.1
$i = 32$	6820	6844	6872	6827	6844	6873	6864	6819	6924	6830	6851.7
$i = 26$	6871	6888	6836	6848	6850	6854	6834	6824	6860	6863	6852.8
$i = 05$	6860	6886	6898	6892	6868	6895	6867	6899	6848	6909	6882.2
$i = 21$	6886	6914	6918	6905	6892	6872	6862	6887	6894	6887	6891.7
$i = 22$	6916	6921	6888	6907	6894	6885	6876	6868	6919	6888	6896.2
$i = 17$	6882	6912	6895	6900	6903	6882	6910	6904	6916	6887	6899.1
$i = 06$	6883	6893	6889	6938	6942	6947	6883	6913	6857	6873	6901.8
$i = 24$	6866	6884	6900	6896	6921	6889	6883	6936	6966	6881	6902.2
$i = 08$	6946	6916	6889	6940	6926	6904	6887	6895	6888	6865	6905.6
$i = 07$	6894	6914	6893	6963	6952	6930	6888	6939	6908	6898	6917.9
$i = 01$	6947	6901	6945	6927	6918	6912	6946	6902	6877	6915	6919.0
$i = 23$	6959	6915	6925	6942	6891	6936	6861	6956	6896	6924	6920.5
$i = 04$	6921	6903	6897	6959	6914	6932	6937	6962	6928	6918	6927.1
$i = 19$	6911	6940	6940	6905	6949	6966	6942	6954	6913	6925	6934.5
$i = 03$	6943	6925	6936	6912	6962	6954	6916	6965	6914	6922	6934.9
$i = 02$	6976	6921	6895	6924	6945	6927	6973	6947	6951	6912	6937.1
$i = 20$	6905	6957	6943	6925	6967	6903	6978	6926	6960	6967	6943.1
$i = 18$	6950	6951	6948	6942	6954	6922	6941	6932	6989	6950	6947.9

Table B.2: P-value of the F-Test and T-Test together with the 95% CI for difference between μ_i and $\mu_{min} = 6824.5$ obtained from the RAT575 parameters tuning experiment.

Hypothesis	F-Test	T-Test	95% CI	Hypothesis	F-Test	T-Test	95% CI
$\mu_{25} = 6828.2$	0.937	0.495	(-14.85, 7.45)	$\mu_{21} = 6891.7$	0.288	0.000	(-81.24, -53.16)
$\mu_{16} = 6832.5$	0.089	0.325	(-24.60, 8.60)	$\mu_{22} = 6896.2$	0.211	0.000	(-86.41, -56.99)
$\mu_{14} = 6832.6$	0.908	0.157	(-19.63, 3.43)	$\mu_{17} = 6899.1$	0.951	0.000	(-86.02, -63.18)
$\mu_{09} = 6839.7$	0.722	0.008	(-25.87, -4.53)	$\mu_{06} = 6901.8$	0.009	0.000	(-100.70, -53.90)
$\mu_{31} = 6841.3$	0.334	0.019	(-30.53, -3.07)	$\mu_{24} = 6902.2$	0.011	0.000	(-100.40, -55.00)
$\mu_{13} = 6842.2$	0.469	0.010	(-30.70, -4.70)	$\mu_{08} = 6905.6$	0.032	0.000	(-100.77, -61.43)
$\mu_{28} = 6842.5$	0.852	0.005	(-29.68, -6.32)	$\mu_{07} = 6917.9$	0.027	0.000	(-113.53, -73.27)
$\mu_{15} = 6843.0$	0.860	0.002	(-29.48, -7.52)	$\mu_{01} = 6919.0$	0.070	0.000	(-111.65, -77.35)
$\mu_{30} = 6843.9$	0.389	0.007	(-32.80, -6.00)	$\mu_{23} = 6920.5$	0.010	0.000	(-119.00, -73.00)
$\mu_{10} = 6844.3$	0.565	0.004	(-32.40, -7.20)	$\mu_{04} = 6927.1$	0.100	0.000	(-118.93, -86.27)
$\mu_{12} = 6845.5$	0.658	0.002	(-33.26, -8.74)	$\mu_{19} = 6934.5$	0.138	0.000	(-125.62, -94.38)
$\mu_{11} = 6848.3$	0.677	0.001	(-36.00, -11.60)	$\mu_{03} = 6934.9$	0.140	0.000	(-125.99, -94.81)
$\mu_{27} = 6851.1$	0.643	0.000	(-38.92, -14.28)	$\mu_{02} = 6937.1$	0.031	0.000	(-132.35, -92.85)
$\mu_{32} = 6851.7$	0.007	0.031	(-51.30, -3.10)	$\mu_{20} = 6943.1$	0.025	0.000	(-138.91, -98.29)
$\mu_{26} = 6852.8$	0.193	0.001	(-43.20, -13.40)	$\mu_{18} = 6947.9$	0.281	0.000	(-137.50, -109.30)
$\mu_{05} = 6882.2$	0.144	0.000	(-73.24, -42.16)				

Appendix C

Parameters Tuning Results for QAP

Table C.1: Results of the BCO parameters tuning experiment for QAP. The SKO100A is used as the testbed.

Config.	Rep 1	Rep 2	Rep 3	Rep 4	Rep 5	Rep 6	Rep 7	Rep 8	Rep 9	Rep 10	Average, μ_i
$i = 13$	152534	152486	152382	152406	152512	152374	152324	152214	152356	152222	152381.0
$i = 10$	152456	152486	152562	152404	152322	152378	152442	152326	152282	152160	152381.8
$i = 09$	152530	152286	152488	152072	152590	152178	152470	152340	152512	152442	152390.8
$i = 14$	152584	152256	152310	152622	152236	152514	152276	152478	152362	152488	152412.6
$i = 25$	152466	152578	152358	152638	152332	152388	152444	152290	152472	152432	152439.8
$i = 29$	152470	152372	152646	152518	152590	152594	152498	152454	152312	152440	152489.4
$i = 26$	152462	152346	152512	152714	152538	152566	152570	152414	152446	152344	152491.2
$i = 15$	152494	152302	152300	152378	152734	152448	152336	152664	152668	152620	152494.4
$i = 16$	152514	152216	152490	152568	152722	152424	152582	152570	152380	152560	152502.6
$i = 30$	152524	152482	152480	152460	152600	152612	152438	152456	152572	152436	152506.0
$i = 12$	152464	152684	152598	152616	152316	152270	152482	152576	152580	152526	152511.2
$i = 02$	152496	152588	152662	152152	152786	152520	152574	152578	152524	152532	152541.2
$i = 32$	152396	152424	152566	152834	152362	152318	152916	152644	152330	152668	152545.8
$i = 11$	152360	152484	152656	152286	152580	152616	152642	152554	152764	152526	152546.8
$i = 31$	152706	152764	152474	152420	152446	152484	152796	152668	152494	152518	152577.0
$i = 05$	152742	152736	152548	152432	152392	152538	152760	152432	152754	152592	152592.6
$i = 27$	152694	152702	152600	152596	152548	152730	152732	152482	152464	152464	152601.2
$i = 06$	152600	152508	152618	152630	152778	152838	152612	152508	152460	152532	152608.4
$i = 01$	152760	152512	152658	152538	152524	152812	152832	152548	152512	152480	152617.6
$i = 28$	152700	152778	152534	152410	152770	152648	152554	152652	152538	152836	152642.0
$i = 22$	152930	152602	152566	152636	152744	152304	153042	152650	152710	152656	152684.0
$i = 17$	152802	152580	152774	152682	152652	152848	152528	152692	152814	152696	152706.8
$i = 21$	152796	152812	152694	152594	152592	152786	152856	152848	152766	152638	152738.2
$i = 18$	152520	152856	152484	152818	152662	152952	152604	152988	152826	152884	152759.4
$i = 08$	152888	152620	153010	152584	153048	152666	153094	152736	152672	152730	152804.8
$i = 04$	152876	152442	152572	152988	152872	153184	152684	153124	152482	152858	152808.2
$i = 07$	152794	152850	152720	152842	152854	152664	153064	152730	152694	152928	152814.0
$i = 03$	152562	152940	152772	152652	152714	153188	152504	152940	152898	153112	152828.2
$i = 23$	152890	152664	152674	152940	152982	152894	152552	152822	153164	152744	152832.6
$i = 24$	153148	152820	152508	152626	152802	153114	153302	152798	152954	152820	152889.2
$i = 20$	153138	152878	152652	152890	153410	152900	152836	153076	152940	152898	152961.8
$i = 19$	152940	153212	152922	152868	152784	153172	153192	153212	152916	153172	153039.0

Table C.2: P-value of the F-Test and T-Test together with the 95% CI for difference between μ_i and $\mu_{min} = 152381.0$ obtained from the SKO100A parameters tuning experiment.

Hypothesis	F-Test	T-Test	95% CI	Hypothesis	F-Test	T-Test	95% CI
$\mu_{10} = 152381.8$	0.903	0.987	(-106.60, 105.00)	$\mu_{06} = 152608.4$	0.799	0.000	(-335.80, -119.00)
$\mu_{09} = 152390.8$	0.228	0.879	(-143.00, 123.40)	$\mu_{01} = 152617.6$	0.539	0.000	(-353.00, -120.20)
$\mu_{14} = 152412.6$	0.465	0.585	(-150.90, 87.70)	$\mu_{28} = 152642.0$	0.579	0.000	(-375.90, -146.10)
$\mu_{25} = 152439.8$	0.938	0.242	(-161.00, 43.40)	$\mu_{22} = 152684.0$	0.089	0.001	(-455.10, -150.90)
$\mu_{29} = 152489.4$	0.847	0.036	(-208.70, -8.10)	$\mu_{17} = 152706.8$	0.862	0.000	(-426.40, -225.20)
$\mu_{26} = 152491.2$	0.934	0.041	(-215.20, -5.20)	$\mu_{21} = 152738.2$	0.801	0.000	(-456.50, -257.90)
$\mu_{15} = 152494.4$	0.239	0.089	(-245.70, 18.90)	$\mu_{18} = 152759.4$	0.164	0.000	(-518.10, -238.70)
$\mu_{16} = 152502.6$	0.520	0.043	(-238.70, -4.50)	$\mu_{08} = 152804.8$	0.123	0.000	(-569.20, -278.40)
$\mu_{30} = 152506.0$	0.152	0.007	(-210.60, -39.40)	$\mu_{04} = 152808.2$	0.019	0.000	(-620.10, -234.30)
$\mu_{12} = 152511.2$	0.599	0.028	(-244.50, -15.90)	$\mu_{07} = 152814.0$	0.780	0.000	(-541.90, -324.10)
$\mu_{02} = 152541.2$	0.274	0.018	(-289.90, -30.50)	$\mu_{03} = 152828.2$	0.043	0.000	(-619.50, -274.90)
$\mu_{32} = 152545.8$	0.060	0.045	(-325.10, -4.50)	$\mu_{23} = 152832.6$	0.161	0.000	(-591.70, -311.50)
$\mu_{11} = 152546.8$	0.462	0.009	(-285.20, -46.40)	$\mu_{24} = 152889.2$	0.027	0.000	(-692.20, -324.20)
$\mu_{31} = 152577.0$	0.473	0.003	(-315.00, -77.00)	$\mu_{20} = 152961.8$	0.079	0.000	(-735.30, -426.30)
$\mu_{05} = 152592.6$	0.407	0.002	(-333.50, -89.70)	$\mu_{19} = 153039.0$	0.230	0.000	(-791.00, -525.00)
$\mu_{27} = 152601.2$	0.977	0.000	(-323.20, -117.20)				

Appendix D

Parameters Tuning Results for JSSP

Table D.1: Results of the BCO parameters tuning experiment for JSSP. The SWV01 is used as the testbed.

Config.	Rep 1	Rep 2	Rep 3	Rep 4	Rep 5	Rep 6	Rep 7	Rep 8	Rep 9	Rep 10	Average, μ_i
$i = 31$	1506	1493	1506	1529	1536	1528	1523	1511	1491	1487	1511.0
$i = 30$	1514	1522	1522	1529	1488	1528	1515	1527	1519	1530	1519.4
$i = 13$	1503	1542	1521	1516	1502	1484	1528	1526	1544	1539	1520.5
$i = 32$	1529	1516	1527	1513	1528	1517	1528	1523	1530	1525	1523.6
$i = 16$	1502	1544	1533	1503	1528	1526	1533	1532	1531	1515	1524.7
$i = 15$	1522	1539	1526	1526	1503	1522	1520	1535	1543	1519	1525.5
$i = 12$	1526	1511	1532	1534	1534	1518	1517	1527	1528	1532	1525.9
$i = 26$	1527	1532	1516	1533	1510	1540	1524	1534	1531	1515	1526.2
$i = 29$	1535	1528	1524	1527	1524	1513	1523	1516	1544	1528	1526.2
$i = 10$	1542	1527	1537	1527	1511	1505	1532	1526	1526	1533	1526.6
$i = 11$	1532	1528	1520	1520	1526	1511	1542	1542	1523	1526	1527.0
$i = 14$	1531	1523	1536	1517	1528	1524	1542	1521	1524	1526	1527.2
$i = 25$	1530	1530	1526	1523	1539	1526	1528	1527	1527	1522	1527.8
$i = 06$	1503	1526	1516	1538	1526	1549	1533	1542	1508	1543	1528.4
$i = 27$	1535	1523	1513	1526	1527	1527	1527	1535	1542	1531	1528.6
$i = 09$	1524	1539	1544	1538	1511	1526	1526	1526	1523	1531	1528.8
$i = 04$	1519	1544	1527	1517	1506	1540	1530	1537	1527	1542	1528.9
$i = 07$	1509	1526	1535	1532	1548	1531	1540	1507	1541	1539	1530.8
$i = 08$	1536	1527	1519	1528	1530	1536	1526	1531	1540	1545	1531.8
$i = 28$	1544	1537	1523	1543	1526	1528	1526	1526	1538	1530	1532.1
$i = 23$	1535	1540	1526	1528	1531	1537	1528	1538	1547	1527	1533.7
$i = 24$	1513	1546	1534	1541	1527	1535	1539	1528	1551	1530	1534.4
$i = 22$	1543	1509	1544	1548	1545	1526	1538	1531	1540	1521	1534.5
$i = 03$	1520	1540	1527	1531	1535	1543	1540	1541	1534	1540	1535.1
$i = 19$	1535	1548	1525	1542	1531	1552	1527	1528	1536	1540	1536.4
$i = 05$	1548	1539	1551	1544	1525	1540	1531	1542	1525	1520	1536.5
$i = 20$	1528	1529	1526	1543	1535	1548	1558	1541	1548	1542	1539.8
$i = 21$	1554	1548	1532	1529	1546	1527	1539	1534	1537	1555	1540.1
$i = 02$	1548	1526	1555	1529	1548	1536	1552	1549	1536	1545	1542.4
$i = 18$	1541	1537	1555	1542	1541	1542	1533	1542	1547	1544	1542.4
$i = 01$	1548	1541	1555	1541	1548	1550	1537	1548	1539	1538	1544.5
$i = 17$	1538	1556	1542	1560	1548	1537	1543	1539	1541	1547	1545.1

Table D.2: P-value of the F-Test and T-Test together with the 95% CI for difference between μ_i and $\mu_{min} = 1511.0$ obtained from the SWV01 parameters tuning experiment.

Hypothesis	F-Test	T-Test	95% CI	Hypothesis	F-Test	T-Test	95% CI
$\mu_{30} = 1519.4$	0.326	0.230	(-22.61, 5.81)	$\mu_{07} = 1530.8$	0.457	0.011	(-34.44, -5.16)
$\mu_{13} = 1520.5$	0.736	0.266	(-26.90, 7.90)	$\mu_{08} = 1531.8$	0.021	0.005	(-33.89, -7.71)
$\mu_{32} = 1523.6$	0.005	0.054	(-25.45, 0.25)	$\mu_{28} = 1532.1$	0.023	0.004	(-34.23, -7.97)
$\mu_{16} = 1524.7$	0.489	0.067	(-28.44, 1.04)	$\mu_{23} = 1533.7$	0.011	0.003	(-35.73, -9.67)
$\mu_{15} = 1525.5$	0.228	0.041	(-28.36, -0.64)	$\mu_{24} = 1534.4$	0.170	0.002	(-37.02, -9.78)
$\mu_{12} = 1525.9$	0.030	0.030	(-28.11, -1.69)	$\mu_{22} = 1534.5$	0.341	0.003	(-37.76, -9.24)
$\mu_{26} = 1526.2$	0.097	0.027	(-28.46, -1.94)	$\mu_{03} = 1535.1$	0.016	0.002	(-37.12, -11.08)
$\mu_{29} = 1526.2$	0.055	0.024	(-28.17, -2.23)	$\mu_{19} = 1536.4$	0.066	0.001	(-38.45, -12.35)
$\mu_{10} = 1526.6$	0.203	0.028	(-29.36, -1.84)	$\mu_{05} = 1536.5$	0.156	0.001	(-39.06, -11.94)
$\mu_{11} = 1527.0$	0.096	0.021	(-29.25, -2.75)	$\mu_{20} = 1539.8$	0.132	0.000	(-42.24, -15.36)
$\mu_{14} = 1527.2$	0.018	0.019	(-29.24, -3.16)	$\mu_{21} = 1540.1$	0.121	0.000	(-42.48, -15.72)
$\mu_{25} = 1527.8$	0.001	0.015	(-29.52, -4.08)	$\mu_{02} = 1542.4$	0.112	0.000	(-44.74, -18.06)
$\mu_{06} = 1528.4$	0.733	0.030	(-32.89, -1.91)	$\mu_{18} = 1542.4$	0.003	0.000	(-44.34, -18.46)
$\mu_{27} = 1528.6$	0.027	0.013	(-30.77, -4.43)	$\mu_{01} = 1544.5$	0.004	0.000	(-46.34, -20.66)
$\mu_{09} = 1528.8$	0.088	0.011	(-31.00, -4.60)	$\mu_{17} = 1545.1$	0.024	0.000	(-47.23, -20.97)
$\mu_{04} = 1528.9$	0.314	0.016	(-32.07, -3.73)				

References

- [1] H. A. Abbass. A single queen single worker honey-bees approach to 3-SAT. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, pages 807–814. Morgan Kaufmann, 2001.
- [2] J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3):391–401, 1988.
- [3] A. Afshar, O. B. Haddad, M. A. Mariño, and B. J. Adams. Honey-bee mating optimization (HBMO) algorithm for optimal reservoir operation. *Journal of the Franklin Institute*, 344(5):452–462, 2007.
- [4] S. Al-Shihabi. Backtracking ant system for the traveling salesman problem. In M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, and T. Stützle, editors, *Ant Colony, Optimization and Swarm Intelligence*, Lecture Notes in Computer Science, pages 318–325. Springer, Berlin / Heidelberg, 2004.
- [5] E. Angel. A survey of approximation results for local search algorithms. In E. Bampis, K. Jansen, and C. Kenyon, editors, *Efficient Approximation and Online Algorithms, Recent Progress on Classical Combinatorial Optimization Problems and New Applications*, Lecture Notes in Computer Science, pages 30–73. Springer Verlag, Berlin Heidelberg, 2006.
- [6] D. Applegate and W. Cook. Computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3(2):149–156, 1991.
- [7] D. L Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton Series in Applied Mathematics. Princeton University Press, Princeton, 2006.

REFERENCES

- [8] E. Balas and A. Vazacopoulos. Guided local search with shifting bottleneck for job shop scheduling. *Management Science*, 44(2):262–275, 1998.
- [9] A. Baykosoglu, L. Ozbakir, and P. Tapkan. Artificial bee colony algorithm and its application to generalized assignment problem. In T. S. Felix Chan and M. K. Tiwari, editors, *Swarm Intelligence: Focus on Ant and Particle Swarm Optimization*, volume 5, chapter 5, pages 532–564. Itech Education and Publishing, Austria, 2007.
- [10] J. E. Beasley. OR-Library: Distributing test problems by electronic mail. *The Journal of the Operational Research Society*, 41(11):1069–1072, 1990.
- [11] J. E. Beasley. Obtaining test problems via internet. *Journal of Global Optimization*, 8(4):429–433, 1996.
- [12] K. Benatchba, L. Admane, and M. Koudil. Using bees to solve data mining problem expressed as a Max-Sat one. In *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, Lecture Notes in Computer Science, pages 212–220. Springer, Berlin/Heidelberg, 2005.
- [13] J. L. Bentley. Fast algorithms for geometric traveling salesman problems. *ORSA Journal on Computing*, 4(4):387–441, 1992.
- [14] J. C. Biesmeijer and T. D. Seeley. The use of waggle dance information by honey bees throughout their foraging careers. *Behavioral Ecology and Sociobiology*, 59(1):133–142, 2005.
- [15] J. H. Blackstone, D. T. Phillips, and G. L. Hogg. A state-of-the-art survey of dispatching rules for manufacturing job shop operations. *International Journal of Production Research*, 20(1):27–45, 1982.
- [16] J. Blazewicz, W. Domschke, and E. Pesch. The job shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research*, 93(1):1–33, 1996.

REFERENCES

- [17] P. Brucker, B. Jurisch, and B. Sievers. A branch-and-bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, 49(1-3):107–127, 1994.
- [18] R. E. Burkard. Quadratic assignment problems. *European Journal of Operational Research*, 15(3):283–289, 1984.
- [19] R. E. Burkard, E. Cela, P. M. Pardalos, and L. S. Pitsoulis. The quadratic assignment problem. In D. Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization*, pages 241–337. Kluwer Academic Publishers, 1999.
- [20] R. E. Burkard, S. Karisch, and F. Rendl. QAPLIB - A quadratic assignment problem library. *European Journal of Operational Research*, 55(1):115–119, 1991.
- [21] R. E. Burkard, S. E. Karisch, and F. Rendl. QAPLIB - A quadratic assignment problem library. *Journal of Global Optimization*, 10(4):391–403, 1997.
- [22] J. Carlier. The one-machine sequencing problem. *European Journal of Operational Research*, 11(1):42–47, 1982.
- [23] F. Carrabs, R. Cerulli, and J. F. Cordeau. Parallel branch and bound model using logarithmic sampling (PBLs) for symmetric traveling salesman problem. *INFOR*, 45(4):223–238, 2007.
- [24] E. Cela. *The quadratic assignment problem: Theory and algorithms*. Kluwer Academic Publishers, Boston, MA, 1998.
- [25] B. Chandra, H. Karloff, and C. Tovey. New results on the old k -opt algorithm for the traveling salesman problem. *SIAM Journal on Computing*, 28(6):1998–2029, 1999.
- [26] C. S. Chong, M. Y. H. Low, A. I. Sivakumar, and K. L. Gay. A bee colony optimization algorithm to job shop scheduling. In L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, editors, *Proceedings of the 2006 Winter Simulation Conference*, pages 1954–1961. Winter Simulation Conference, 2006.

- [27] C. S. Chong, M. Y. H. Low, A. I. Sivakumar, and K. L. Gay. Using a bee colony algorithm for neighborhood search in job shop scheduling problems. In *Proceedings of the 21st European Conference on Modeling and Simulation (ECMS2007)*, 2007.
- [28] J. Clausen and M. Perregaard. Solving large quadratic assignment problems in parallel. *Computational Optimization and Applications*, 8(2):111–127, 1997.
- [29] J. Clausen and M. Perregaard. Parallel branch-and-bound methods for the job-shop scheduling problem. *Annals of Operations Research*, 83(0):137–160, 1998.
- [30] R. W. Conway, D. S Johnson, and W. L. Maxwell. An experimental investigation of priority dispatching. *Journal of Industrial Engineering*, 11:221–229, 1960.
- [31] I. A. Couzin, J. Krause, R. James, G. D. Ruxton, and N. R. Franks. Collective memory and spatial sorting in animal groups. *Journal of Theoretical Biology*, 218(1):1–11, 2002.
- [32] 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*, 250:245–250, 1990.
- [33] G. A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812, 1958.
- [34] M. Dell’Amico, J. C. D. Díaz, M. Iori, and R. Montanari. The single-finger keyboard layout problem. *Computers & Operations Research*, 36(11):3002–3012, 2009.
- [35] M. Dell’Amico and M. Trubian. Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research*, 41(3):231–252, 1993.
- [36] N. C. Demirel and M. D. Toksari. Optimization of the quadratic assignment problem using an ant colony algorithm. *Applied Mathematics and Computation*, 183(1):427–435, 2006.
- [37] E. Demirkol, S. Mehta, and R. Uzsoy. A computational study of shifting bottleneck procedures for shop scheduling problems. *Journal of Heuristics*, 3(2):111–137, 1997.

- [38] J. C. Deneubourg and S. Goss. Collective patterns and decision-making. *Ethology, Ecology & Evolution*, 1:295–311, 1989.
- [39] J. L Deneubourg, J. M. Pasteels, and J. C Verhaeghe. Probabilistic behaviour in ants: A strategy of errors? *Journal of Theoretical Biology*, 105:259–271, 1983.
- [40] M. Dorigo, M. Birattari, and T. Stützle. Artificial ants as a computational intelligence technique. *IEEE Computational Intelligence Magazine*, 2006.
- [41] M. Dorigo, V. Maniezzo, and A. Coloni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on System, Man and Cybernetics*, 26(1):29–41, 1996.
- [42] M. Dorigo and T. Stützle. The ant colony optimization metaheuristic: algorithms, applications, and advances. In *Handbook of Metaheuristics*, International Series In Operations Research & Management Science, pages 250–285. Springer, New York, 2002.
- [43] F. C. Dyer. The biology of the dance language. *Annual Review of Entomology*, 47:917–949, 2002.
- [44] A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.
- [45] H. A Eiselt, M. Gendreau, and G. Laporte. Arc routing problems, Part I: The chinese postman problem. *Operations Research*, 43(2):231–242, 1995.
- [46] H. A Eiselt, M. Gendreau, and G. Laporte. Arc routing problems, Part II: The rural postman problem. *Operations Research*, 43(3):399–414, 1995.
- [47] A. N. Elshafei. Hospital layout as a quadratic assignment problem. *Operational Research Quarterly*, 28(1):167–179, 1977.
- [48] G. Erdogan and B. Tansel. A branch-and-cut algorithm for quadratic assignment problems based on linearizations. *Computers & Operations Research*, 34(4):1085–1106, 2007.

REFERENCES

- [49] V. P. Eswaramurthy and A. Tamilarasi. Hybridizing tabu search with ant colony optimization for solving job shop scheduling problems. *The International Journal of Advanced Manufacturing Technology*, 40(9-10):1004–1015, 2009.
- [50] H. Fisher and G. L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In J. F. Muth and G. L. Thompson, editors, *Industrial Scheduling*, pages 225–251. Prentice Hall, Englewood Cliffs, New Jersey, 1963.
- [51] M. M. Flood. The traveling-salesman problem. *Operations Research*, 4(1):61–75, 1956.
- [52] L. M. Gambardella and M. Dorigo. Solving symmetric and asymmetric tsps by ant colonies. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, pages 622–627, 1996.
- [53] L. M. Gambardella, E. D. Taillard, and M Dorigo. Ant colonies for the quadratic assignment problem. *The Journal of the Operational Research Society*, 50(2):167–176, 1999.
- [54] S. Gao, L. Zhang, F. Zhuang, and C. Zhang. Solving traveling salesman problem by ant colony optimization algorithm with association rule. In *Proceedings of the Third International Conference on Natural Computation (ICNC 2007)*, volume 3, pages 693–698, 2007.
- [55] M. R Garey and D. S Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman Co., San Francisco, 1979.
- [56] M. Gendreau, A. Hertz, and G. Laporte. New insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, 40(6):1086–1094, 1992.
- [57] M. Gendreau, A. Hertz, G. Laporte, and M. Stan. A generalized insertion heuristic for the traveling salesman problem with time windows. *Operations Research*, 46(3):330–335, 1998.

- [58] J. S. Gero and V. A. Kazakov. Evolving design genes in space layout planning problems. *Artificial Intelligence in Engineering*, 12(3):163–176, 1998.
- [59] J. S. Gero, V. A. Kazakov, and T. Schnier. Genetic engineering and design problems. In D. Dasgupta and Z. Michalewicz, editors, *Evolutionary Algorithms in Engineering Applications*, pages 47–68. Springer, Berlin, 1997.
- [60] P. C. Gilmore. Optimal and suboptimal algorithms for the quadratic assignment problem. *Journal of the Society for Industrial and Applied Mathematics*, 10(2):305–313, 1962.
- [61] E. F. G. Goldberg, G. R. de Souza, and M. C. Goldberg. Particle swarm for the traveling salesman problem. In J. Gottlieb and Raidl.G.R, editors, *Evolutionary Computation in Combinatorial Optimization*, Lecture Notes in Computer Science, pages 99–110. Springer, Berlin / Heidelberg, 2006.
- [62] T. Gong and A. L. Tuson. Particle swarm optimization for quadratic assignment problems - A forma analysis approach. *International Journal of Computational Intelligence Research*, 4(2):177–185, 2008.
- [63] J. Grabowski, E. Nowicki, and S. Zdrzalka. A block approach for single-machine scheduling with release dates and due dates. *European Journal of Operational Research*, 26(2):278–285, 1986.
- [64] S. C. Greenway. Hormones in human metabolism and disease. In K. B. Storey, editor, *Functional Metabolism: Regulation and Adaptation*, volume 10, chapter 10, pages 271–294. John Wiley & Sons, 2005.
- [65] A. K. Gupta and A. I. Sivakumar. Job shop scheduling techniques in semiconductor manufacturing. *International Journal of Advanced Manufacturing Technology*, 27(11-12):1163–1169, 2006.
- [66] P. Gupta, A. B. Kahng, and S. Mantik. Routing-aware scan chain ordering. *ACM Transactions on Design Automation of Electronic Systems*, 10(3):546–560, 2005.

- [67] G. Gutin. Traveling salesman problems. In J. L. Gross and J. Yellen, editors, *Handbook of Graph Theory*, volume 4.6 of *Discrete Mathematics and Its Applications*, chapter 4.6, pages 279–299. CRC Press, New York, 2007.
- [68] G. Gutin and A. Yeo. Polynomial approximation algorithms for the TSP and the QAP with a factorial domination number. *Discrete Applied Mathematics*, 119(1-2):107–116, 2002.
- [69] O. B. Haddad, A. Afshar, and M. A. Mariño. Honey-bees mating optimization (HBMO) algorithm: A new heuristic approach for water resources optimization. *Water Resources Management*, 20(5):661–680, 2006.
- [70] P. Hahn, T. Grant, and N. Hall. A branch-and-bound algorithm for the quadratic assignment problem based on the Hungarian method. *European Journal of Operational Research*, 108(3):629–640, 1998.
- [71] R. Haupt. A survey of priority rule-based scheduling. *OR Spektrum*, 11(1):3–16, 1989.
- [72] A. J. Hoffman and P. Wolfe. History in the traveling salesman problem. In E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, volume 1 of *Wiley Series in Discrete Mathematics and Optimization*, chapter 1, pages 1–17. John Wiley & Sons, New York, 1984.
- [73] K. L. Huang and C. J. Liao. Ant colony optimization combined with taboo search for the job shop scheduling problem. *Computers & Operations Research*, 35(4):1030–1046, 2008.
- [74] Cor A. J. Hurkens and G. J. Woeginger. On the nearest neighbor rule for the traveling salesman problem. *Operations Research Letters*, 32(1):1–4, 2004.
- [75] A. S. Jain and S. Meeran. Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research*, 113(2):390–434, 1999.

- [76] A.S. Jain, B. Rangaswamy, and S. Meeran. New and stronger job-shop neighbourhoods: A focus on the method of Nowicki and Smutnicki (1996). *Journal of Heuristics*, 6(4):457–480, 2000.
- [77] D. S Johnson and L. A. McGeoch. Experimental analysis of heuristic for the STSP. In G. Gutin and A. P Punnen, editors, *The Traveling Salesman Problem and Its Variations*, chapter 9, pages 369–443. Kluwer Academic Publishers, 2002.
- [78] D. S Johnson and C. H Papadimitriou. Performance guarantees for heuristics. In E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, volume 5 of *Wiley Series in Discrete Mathematics and Optimization*, chapter 5, pages 145–181. John Wiley & Sons, New York, 1985.
- [79] L. Jourdan, M. Basseur, and E. G. Talbi. Hybridizing exact methods and meta-heuristics: A taxonomy. *European Journal of Operational Research*, 199(3):620–629, 2009.
- [80] D. Karaboga and B. Akay. A survey: Algorithms simulating bee swarm intelligence. *Artificial Intelligence Review*, 31(1-4):61–85, 2009.
- [81] D. Karaboga and B. Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of Global Optimization*, 39(3):459–471, 2007.
- [82] D. Karaboga and B. Basturk. On the performance of artificial bee colony (ABC) algorithm. *Applied Soft Computing*, 8(1):687–697, 2008.
- [83] A. Karci. Imitation of bee reproduction as a crossover operator in genetic algorithms. In *Proceedings of the 8th Pacific Rim International Conference on Artificial Intelligence - Trends in Artificial Intelligence (PRICAI 2004)*, Lecture Notes in Computer Science, pages 1015–1016. Springer Verlag, Berlin / Heidelberg, 2004.
- [84] J. Kennedy. Some issues and practices for particle swarms. In *Proceedings of the IEEE Swarm Intelligence Symposium (SIS 2007)*, pages 162–169. IEEE, 2007.

- [85] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of 1995 IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.
- [86] W. Kocay and D. L. Kreher. Hamilton cycles. In *Graph, Algorithms and Optimization*, volume 9 of *Discrete Mathematics and Its Applications*, chapter 9, pages 187–221. Chapman & Hall / CRC Press, New York, 2005.
- [87] T. C. Koopmans and M. Beckmann. Assignment problems and the location of economic activities. *Econometrica*, 25(1):53–76, 1957.
- [88] S. Lawrence. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement). Technical report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania., 1984.
- [89] F. X. Le Louarn, M. Gendreau, and J. Y. Potvin. GENI ants for the traveling salesman problem. *Annals of Operations Research*, 131(1-4):187–201, 2004.
- [90] J. Lee. *A First Course in Combinatorial Optimization*. Cambridge Texts in Applied Mathematics. The Press Syndicate of the University of Cambridge, Cambridge, United Kingdom, 2004.
- [91] N. Lemmens, S. de jong, K Tuyls, and A. Nowé. Bee system with inhibition pheromones. In *Proceedings of the European Conference on Complex Systems (ECCS'07)*, 2007.
- [92] N. Lemmens, S. de jong, K Tuyls, and A. Nowé. Bee behaviour in multi-agent systems (A bee foraging algorithm). In K Tuyls, A. Nowé, Z. Guessoum, and D. Kudenko, editors, *Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning*, Lecture Notes in Computer Science, pages 145–156. Springer, Berlin / Heidelberg, 2008.
- [93] S. Lessmann, M. Caserta, and I. M. Arango. Tuning metaheuristics: A data mining based approach for particle swarm optimization. *Expert Systems with Applications*, 38(10):12826–12838, 2011.

- [94] L. Y. O Li and Z. Fu. The school bus routing problem: A case study. *The Journal of the Operational Research Society*, 53(5):552–558, 2002.
- [95] X. Li, P. Tian, J. Hua, and N. Zhong. A hybrid discrete particle swarm optimization for the traveling salesman problem. In T. D. Wang, X. Li, S. H. Chen, X. Wang, H. Abbass, H. Iba, G. Chen, and X. Yao, editors, *Simulated Evolution and Learning*, Lecture Notes in Computer Science, pages 181–188. Springer, Berlin / Heidelberg, 2006.
- [96] M. Lihoreau, L. Chittka, and N. E. Raine. Travel optimization by foraging bumblebees through readjustments of traplines after discovery of new feeding locations. *The American Naturalist*, 176(6):744–757, 2010.
- [97] S. Lin and B. W. Kerningham. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21(2):498–516, 1973.
- [98] T. L. Lin, S. J. Horng, T. W. Kao, Y. H. Chen, R. S. Run, R. J. Chen, J. L. Lai, and I. H. Kuo. An efficient job-shop scheduling algorithm based on particle swarm optimization. *Expert Systems with Applications*, 37(3):2629–2636, 2010.
- [99] J. D. Little, K. G. Murty, D. W. Sweeney, and C. Karel. An algorithm for the traveling salesman problem. *Operations Research*, 11(6):972–989, 1963.
- [100] E. M. Loiola, N. M. M. de Abreu, P. O. Boaventura-Netto, P. Hahn, and T. Querido. A survey for the quadratic assignment problem. *European Journal of Operational Research*, 176(2):657–690, 2007.
- [101] M. Lozano and C. Garcia-Martinez. An evolutionary ILS-perturbation technique. In M. J. Blesa, C. Blum, C. Cotta, A. J. Fernández, J. E. Gallardo, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics, 5th International Workshop, HM 2008, Mlaga, Spain, October 8-9, 2008. Proceedings*, Lecture Notes in Computer Science, pages 1–15. Springer Verlag, Berlin Heidelberg, 2008.
- [102] P. Lučić. *Modeling transportation problems using concepts of swarm intelligence and soft computing*. Ph.D. Thesis, Virginia Polytechnic Institute and State University, 2002.

REFERENCES

- [103] P. Lučić and D. Teodorović. Computing with bees: Attacking complex transportation engineering problems. *International Journal on Artificial Intelligence Tools*, 12(3):375–394, 2003.
- [104] P. Lučić and D. Teodorović. Vehicle routing problem with uncertain demand at nodes: The bee system and fuzzy logic approach. In J. L. Verdegay, editor, *Fuzzy Sets in Optimization*, pages 67–82. Springer-Verlag, Berlin / Heidelberg, 2003.
- [105] O. Matsuo, C. J. Suh, and R. S. Sullivan. A controlled search simulated annealing method for the general job-shop scheduling problem. Technical Report 03-04-88, Graduate School Business, University of Texas at Austin, Austin, Texas, USA, 1988.
- [106] J. Mittenenthal and C. E. Noon. An insert/delete heuristic for the travelling salesman subset-tour problem with one additional constraint. *The Journal of the Operational Research Society*, 43(3):277–283, 1992.
- [107] S. Nakrani and C. Tovey. On honey bees and dynamic server allocation in Internet hosting centers. *Adaptive Behavior*, 12(3-4):223–240, 2004.
- [108] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42(6):797–813, 1996.
- [109] E. Nowicki and C. Smutnicki. An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling*, 8(2):145–159, 2005.
- [110] S. S. Panwalkar and W. Iskander. A survey of scheduling rules. *Operations Research*, 25(1):45–61, 1977.
- [111] J. L. Pasquier, I. K. Balich, D. W. Carr, and C. Lopez-Martin. A comparative study of three metaheuristics applied to the traveling salesman problem. In *Proceedings of the Sixth Mexican International Conference on Artificial Intelligence - Special Session (MICA 2007)*, pages 243–254, 2007.
- [112] F. Pezzela and E. Merelli. Tabu search method guided by shifting bottleneck for the job shop scheduling problem. *European Journal of Operational Research*, 120(2):297–310, 2000.

- [113] D. T. Pham, A. Ghanbarzadeh, E. Koc, S. Otri, S. Rahim, and M. Zaidi. The bees algorithm - a novel tool for complex optimisation problems. In D. T. Pham, E. E Eldukhri, and A. J. Soroka, editors, *Proceedings of the 2006 Conference on Innovative Production Machines and Systems (IPROMS 2006)*, pages 454–459, Cardiff University, Cardiff , UK, 2006.
- [114] D. T. Pham, A. Haj Darwish, and E. E Eldukhri. Optimisation of a fuzzy logic controller using the bees algorithm. *International Journal of Computer Aided Engineering and Technology*, 1(2):250–264, 2009.
- [115] D. T. Pham, J. Y. Lee, A. Haj Darwish, and A. J. Soroka. Multi-objective environmental/economic power dispatch using the bees algorithm with pareto optimality. In D. T. Pham, E. E Eldukhri, and A. J. Soroka, editors, *Proceedings of the 2008 Conference on Innovative Production Machines and Systems (IPROMS 2008)*, pages 422–429, Cardiff University, Cardiff , UK, 2008.
- [116] D. T. Pham, S. Otri, and A. Haj Darwish. Application of the bees algorithm to pcb assembly optimisation. In D. T. Pham, E. E Eldukhri, and A. J. Soroka, editors, *Proceedings of the 2007 Conference on Innovative Production Machines and Systems (IPROMS 2007)*, pages 511–516, Cardiff University, Cardiff , UK, 2007.
- [117] C. N. Potts and V. A. Strusevich. Fifty years of scheduling: A survey of milestones. *Journal of the Operational Research Society*, 60(S1):S41–S68, 2010.
- [118] A. P Punnen. The traveling salesman problem: Applications, formulations and variations. In G. Gutin and A. P Punnen, editors, *The Traveling Salesman Problem and Its Variations*, volume 1 of *Combinatorial Optimization*, chapter 1, pages 1–28. Kluwer Academic Publishers, New York, 2004.
- [119] A. Puris, R Bello, Y. Martinez, and A. Nowe. Two-stage ant colony optimization for solving the traveling salesman problem. In J. Mira and J. R. Alvarez, editors, *Nature Inspired Problem-Solving Methods in Knowledge Engineering*, Lecture Notes in Computer Science, pages 307–316. Springer, Berlin / Heidelberg, 2007.

- [120] C. Qi. An ant colony system hybridized with randomized algorithm for TSP. In *Proceedings of the 2007 International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel / Distributed Computing (SNPD 2007)*, volume 3, pages 461–465, 2007.
- [121] L. D. Qin, Q. Y. Jiang, Z. Y. Zou, and Y. J. Cao. A queen-bee evolution based on genetic algorithm for economic power dispatch. In *Proceedings of the 39th International Universities Power Engineering Conference (UPEC 2004)*, pages 453–456, 2004.
- [122] R. Radharamanan and L. I. Choi. A branch and bound algorithm for the traveling salesman and the transportation routing problems. *Computers & Industrial Engineering*, 11(4):236–240, 1986.
- [123] A. S. Ramkumar, S. G. Ponnambalam, and N. Jawahar. A new iterated fast local search heuristic for solving QAP formulation in facility layout design. *Robotics and Computer-Integrated Manufacturing*, 25(3):620–629, 2009.
- [124] A. S. Ramkumar, S. G. Ponnambalam, and N. Jawahar. A population-based hybrid ant system for quadratic assignment formulations in facility layout design. *The International Journal of Advanced Manufacturing Technology*, 44(5-6):548–558, 2009.
- [125] S. S. Ray, S. Bandyopadhyay, and S. K. Pal. Genetic operators for combinatorial optimization in TSP and microarray gene ordering. *Applied Intelligence*, 26(3):183–195, 2007.
- [126] E. Ridge and D. Kedenko. Tuning the performance of the MMAS heuristic. In T. Stützle, M. Birattari, and H. H. Hoos, editors, *Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics*, Lecture Notes in Computer Science, pages 46–60. Springer, Berlin / Heidelberg, 2007.
- [127] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing*, 6(3):563–581, 1977.

REFERENCES

- [128] R. Ruiz and T. Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049, 2007.
- [129] N. W. Sawaya and I. E. Grossmann. A cutting plane method for solving linear generalized disjunctive programming problems. *Computers & Chemical Engineering*, 29(9):1891–1913, 2005.
- [130] M. Seo and D. Kim. Ant colony optimisation with parameterised search space for the job shop scheduling problem. *International Journal of Production Research*, 48(4):1143–1154, 2010.
- [131] D. Y. Sha and C. Y. Hsu. A hybrid particle swarm optimization for job shop scheduling problem. *Computers & Industrial Engineering*, 51(4):791–808, 2006.
- [132] S. P Singh and R. K. K. Sharma. A review of different approaches to the facility layout problems. *The International Journal of Advanced Manufacturing Technology*, 30(5-6):425–433, 2006.
- [133] L. Steinberg. The backboard wiring problem: A placement algorithm. *SIAM Review*, 3(1):37–50, 1961.
- [134] R. H. Storer, S. D. Wu, and R. Vaccari. New search spaces for sequencing instances with application to job shop scheduling. *Management Science*, 38(10):1495–1501, 1992.
- [135] T. Stützle. Iterated local search for the quadratic assignment problem. *European Journal of Operational Research*, 174(3):1519–1539, 2006.
- [136] T. Stützle and H. H. Hoos. Improvements on the ant system: Introducing the MAX-MIN ant system. In *Proceedings the Third International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA 1997)*, University of East Anglia, Norwich, UK, Wien, 1997. Springer Verlag.
- [137] T. Stützle and H. H. Hoos. The MAX-MIN ant system and local search for combinatorial optimization problems. In *Proceedings of the 2nd Metaheuristics International Conference (MIC-97)*, 1997.

REFERENCES

- [138] H. J. Sung. Queen-bee evolution for genetic algorithms. *Electronic Letters*, 39(6):575–576, 2003.
- [139] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.
- [140] E. D. Taillard. Comparison of iterative searches for the quadratic assignment problem. *Location Science*, 3(2):87–105, 1995.
- [141] L. Tassiulas. Worst case length of nearest neighbor tours for the euclidean traveling salesman problem. *Siam Journal on Discrete Mathematics*, 10(2):171–179, 1997.
- [142] D. Teodorović. Swarm intelligence systems for transportation engineering: Principles and applications. *Transportation Research Part C: Emerging Technologies*, 16(6):651–657, 2008.
- [143] D. Teodorović. Bee Colony Optimization (BCO). In C. P. Lim, L. C. Jain, and S. Dehuri, editors, *Innovations in Swarm Intelligence*, Studies in Computational Intelligence, pages 39–60. Springer-Verlag, 2009.
- [144] D. Teodorović and M. Dell’Orco. Bee colony optimization - A cooperative learning approach to complex transportation problems. In *Proceedings of the 16th Mini EURO Conference and 10th Meeting of EWGT*, pages 51–60, 2005.
- [145] S. Tschoke, R. Lubling, and B. Monien. Solving the traveling salesman problem with a distributed branch-and-bound algorithm on a 1024 processor network. In *Proceedings of the 9th International Parallel Processing Symposium*, pages 182–189, 1995.
- [146] S. Tsubakitani and J. R. Evans. Optimizing tabu list size for the traveling salesman problem. *Computers & Operations Research*, 25(2):91–97, 1998.
- [147] N. Urquhart, P. Ross, B. Paechter, and K. Chisholm. Improving street based routing using building block mutations. In S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, and G. R. Raidl, editors, *Applications of Evolutionary Computing*, Lecture Notes in Computer Science, pages 334–341. Springer, Berlin / Heidelberg, 2002.

REFERENCES

- [148] P. J. M. van Laarhoven, E. H. L. Aarts, and J. K. Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40(1):113–125, 1992.
- [149] J. van Rijswijck. Are bees better than fruitflies? Experiments with a hex playing program. In *Advances in Artificial Intelligence*, Lecture Notes in Computer Science, pages 13–25. Springer, Berlin / Heidelberg, 2000.
- [150] T. Volgenant and R. Jonker. A branch and bound algorithm for the symmetric traveling salesman problem based on the 1-tree relaxation. *European Journal of Operational Research*, 9(1):83–89, 1982.
- [151] K. von Frisch. Decoding the language of the bee. *Science*, 185(4152):663–668, 1974.
- [152] F. H. Wedde, M. Farooq, and Y. Zhang. Beehive: An efficient fault-tolerant routing algorithm inspired by honey bee behavior. In *Ant Colony, Optimization and Swarm Intelligence*, Lecture Notes in Computer Science, pages 83–94. Springer, Berlin / Heidelberg, 2004.
- [153] H. Wenqi and Y. Aihua. An improved shifting bottleneck procedure for the job shop scheduling problem. *Computers & Operations Research*, 31(12):2093–2110, 2004.
- [154] L. P. Wong, M. Y. H. Low, and C. S. Chong. A bee colony optimization algorithm for traveling salesman problem. In *Proceedings of the Second Asia International Conference on Modelling & Simulation*, pages 818–823. IEEE Computer Society, 2008.
- [155] L. P. Wong, M. Y. H. Low, and C. S. Chong. Bee colony optimization with local search for traveling salesman problem. In *Proceedings of the 6th IEEE International Conference on Industrial Informatics (INDIN 2008)*, pages 1019–1025. IEEE, 2008.
- [156] L. P. Wong, M. Y. H. Low, and C. S. Chong. An efficient bee colony optimization algorithm for traveling salesman problem using frequency-based pruning. In *Proceedings of the 7th IEEE International Conference on Industrial Informatics (INDIN 2009)*, pages 775–782. IEEE, 2009.

- [157] L. P. Wong, C. Y. Puan, M. Y. H. Low, C. S. Chong, and Y. W. Wong. Bee colony optimization algorithm with big valley landscape exploitation for job shop scheduling problems. *International Journal of Bio-Inspired Computation*, 2(2):85–99, 2010.
- [158] A. S. Wu and R. K. Lindsay. A comparison of the fixed and floating building block representation in the genetic algorithm. *Evolutionary Computation*, 4(2):169–193, 1996.
- [159] T. Yamada. *Studies on metaheuristics for jobshop and flowshop scheduling problems*. Ph.d. thesis, Kyoto University, 2003.
- [160] T. Yamada and R. Nakano. A genetic algorithm applicable to large-scale job-shop instances. In R. Manner and B. Manderick, editors, *Proceedings of the Second International Workshop on Parallel Problem Solving from Nature (PPSN'2), Brussels, Belgium*, pages 281–290, 1992.
- [161] J. F. Yu and B. R. Sarker. Directional decomposition heuristic for a linear machine-cell location problem. *European Journal of Operational Research*, 149(1):142–184, 2003.
- [162] B. Yuan and M. Gallagher. A hybrid approach to parameter tuning in Genetic Algorithms. In *Proceedings of the 2005 IEEE Congress*, pages 1096–1103, 2005.
- [163] C. Zhang, P. Li, Z. Guan, and Y. Rao. A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. *Computers & Operations Research*, 34(11):3229–3242, 2007.
- [164] F. Zhao, J. Dong, S. Li, and J. Sun. An improved ant colony optimization algorithm with embedded genetic algorithm for the traveling salesman problem. In *Proceedings of the 7th World Congress on Intelligent Control and Automation (WCICA 2008)*, pages 7902–7906, 2008.
- [165] W. L. Zhong, J. Zhang, and W. N. Chen. A novel discrete particle swarm optimization to solve traveling salesman problem. In *Proceedings of the 2007 IEEE Congress on Evolutionary Computation (CEC 2007)*, pages 3283–3287, 2007.