KNOWLEDGE SHARING IN COOPERATIVE COMPACT GENETIC ALGORITHM

Miss Orakanya Gateratanakul

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering in Computer Engineering
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2018

การแบ่งปันความรู้ในขั้นตอนวิธีเชิงพันธุกรรมอย่างย่อแบบมีส่วนร่วม

น.ส.อรกัญญา เกตุรัตนกุล

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2561

Thesis Title	KNOWLEDGE SHARING IN COOPERATIVE COMPACT GENETIC ALGORITHM
By	Miss Orakanya Gateratanakul
Field of Study	Computer Engineering
Thesis Advisor	Professor PRABHAS CHONGSTITVATANA, Ph.D.

Accepted by the Faculty of Engineering, Chulalongkorn University in Partial Fulfillment of the Requirement for the Master of Engineering

.... Dean of the Faculty of Engineering
(Professor SUPOT TEACHAVORASINSKUN, Ph.D.)

THESIS COMMITTEE

.... Chairman
(Associate Professor SETHA PAN-NGUM, Ph.D.)

.... Thesis Advisor
(Professor PRABHAS CHONGSTITVATANA, Ph.D.)

.... External Examiner
(Associate Professor Worasait Suwannik, Ph.D.)

อรกัญญา เกตุรัตนกุล : การแบ่งปันความรู้ในขั้นตอนวิธีเชิงพันธุกรรมอย่างย่อแบบมีส่วนร่วม. (KNOWLEDGE SHARING IN COOPERATIVE COMPACT GENETIC ALGORITHM) อ.ที่ปรึกษาหลัก : ศ. ดร.ประภาส จงสถิตย์วัฒนา

อัลกอริทึมขั้นตอนวิธีเชิงพันธุกรรมอย่างย่อแบบมีส่วนร่วมนั้นมาจากอัลกอริทึมขั้นตอนวิธีเชิงพันธุกรรมที่ประชากรแทนด้วยเวกเตอร์ความน่าจะเป็น เพื่อพัฒนาความสามารถในการค้นหาคำตอบและหลีกเลี่ยงคำตอบเฉพาะที่ดังนั้นการทำงานแบบขนานจึงถูกใช้เมื่อโพรเซสหลายโพรเซสทำงานพร้อมกัน เนื่องจากการทำงานไปพร้อมกันของโพรเซสหลายโพรเซสการแบ่งปันข้อมูลจึงเป็นสิ่งจำเป็น โดยโพรเซสจะมีการแบ่งปันข้อมูลกันเป็นระยะ ทั้งนี้เพื่อจะหลีกหนีจากคำตอบเฉพาะที่การรีสตาร์ทจึงถูกนำเสนอ ซึ่งการทดลองเปรียบเทียบอัลกอริทึมที่นำเสนอกับอีกสองอัลกอริทึมโดยใช้ ปัญหาการเดินทางของพนักงานขาย ปัญหาการบรรจุผลิตภัณฑ์ ปัญหาผลรวมของสับเซต และปัญหาถุงกระสอบ ผลการทดลองพบว่าอัลกอริทึมนี่นำเสนอมีประสิทธิภาพในการค้นหาคำตอบได้ดีกว่าอีกสองอัลกอริทึมที่เปรียบเทียบ การวิเคราะห์ของการเกิดรีสตาร์ทแสดงให้เห็นถึงพฤติกรรมของอัลกอริทึมที่ถูกนำเสนอ

สาขาวิชา	วิศวกรรมคอมพิวเตอร์	ลายมือชื่อนิสิต ....
ปีการศึกษา	2561	ลายมือชื่อ อ.ที่ปรึกษาหลัก ....

6070372621 : MAJOR COMPUTER ENGINEERING
KEYWORD: compact genetic algorithm, local optima

Orakanya Gateratanakul : KNOWLEDGE SHARING IN COOPERATIVE COMPACT GENETIC ALGORITHM. Advisor: Prof. PRABHAS CHONGSTITVATANA, Ph.D.

The compact genetic algorithm is derived from the genetic algorithm in which the population is represented by the probabilistic vector. To improve the search capability and avoiding local minima, parallelization has been employed where many search processes are deployed concurrently. In order to coordinate the work of multiple processes, knowledge sharing is necessary. Multiple processes share their probabilistic vectors partially. To escape from local minima the restart step is introduced. The experiment compares the proposed algorithm with two other competitive algorithms using Traveling Salesman problem, Bin Packing problem, Subset Sum problem, and Knapsack problem. The results show that the proposed algorithm is more efficient in finding solutions than the competing algorithms. The detailed analysis of the restart step provides insight into the behaviour of the proposed algorithm.

Field of Study:	Computer Engineering	Student's Signature ....
Academic Year:	2018	Advisor's Signature ....

ACKNOWLEDGEMENTS

Foremost, I would like to express the deepest appreciation to my thesis advisor (Professor Prabhas Chongstitvatana, Ph.D.). This research can be achieved because of supports and advises from Professor Prabhas Chongstitvatana, Ph.D. All encouragement, inspiration, and guidelines I have received during the period of studying have supported me to pass through many obstacles.

Secondly, I would like to thank the thesis committees for their insightful comments and encouragement, but also for the hard question which incented me to widen my research from various perspectives.

Thirdly, I would like to thank all members of Intelligent Systems Laboratory (ISL) including friends from the department of computer engineering for all supports and I received many good friendships throughout my studies.

Last but not least, I would like to express my gratitude to my family that always stay beside me and give me a lot of love and encouragement

Orakanya Gateratanakul

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

Chapter 1 Introduction

1.1. Background

1.1.1. Motivation

In the present days, there are many groups of problems that are difficult to solve because complexity is high and time required to solve these problems increases very quickly as the sizes of the problems grow[1]. As a result, the evolutionary algorithms have been developed and designed to fix these problems and one of the most popular evolutionary algorithms is the genetic algorithm [2, 3] due to its efficient ability to precisely search the best answers.

However, the genetic algorithm still cannot avoid premature convergence leading to it being stuck in local optima [4]. These issues are the general problems in evolutionary algorithms. In addition, although the genetic algorithm is an effective algorithm, there is a major problem with the size of memories used to keep the whole possible solution or solution space [5].

Consequently, the compact genetic algorithm is an interesting algorithm that can reduce the size of memories by using probabilistic vector instead of collecting all possible solutions. Moreover, the compact genetic algorithm is also equivalent to the simple genetic algorithm with uniform crossover [5].

As the premature convergence still cannot be prevented in the compact genetic algorithm, parallelization is widely used to support the problems. In addition to the parallelization, the probabilistic vector of the compact genetic algorithm has an important advantage that affects the process of exploring solution space in the paralleled method [6].

This research focuses on designing a new algorithm supporting paralleled works in the compact genetic algorithm emphasizing on migration processes of some specific variables. We are also interested in studying the process of searching the high-quality solutions in search space. In detailed, we will use traveling salesman problem [7], bin packing problem [8], knapsack problem [9] and subset sum problem [10] in the experiment because they are in the group of the challenging problems. Their complexity are quite high due to the variety of the high-quality solutions and long time required to find the best solution.

1.1.2. Problem statement

The compact genetic algorithm is one of the effective heuristic search algorithms to solve many challenging problems. However, there are some hard problems that the compact genetic algorithm cannot solve them effectively. Because of the highly deceptive local optima, it is inevitable to getting stuck in the local optima. Moreover, it is very complicated to escape from local optima. Although the parallelization is used to improve the performance of many algorithms, it cannot completely prevent getting trapped in local optima. Thus, the purposes of this research are to design and improve the mechanism for the parallelization of the compact genetic algorithm to escape from the local optima and focus on the migration process of some specific data to increase the performance of the compact genetic algorithm and avoid premature convergence problem.

1.1.3. Scope

- Interested in migration methods and migrated parameters.
- Use Traveling salesman problem (11, 15, 17 cities), bin packing problem (50 items), knapsack problem (maximum of 200 items) and subset sum problem (maximum of 21 numbers) in the experiment
- Network issues are not concerned because all algorithms are tested on one computer.
- The number of iterations executed by a cGA node must not exceed 200,000 iterations.
- The numbers of iterations executed and the percentage errors are used to measure the performance.
- The runtime of iterations is not concerned.
- The experiment has one master node and four slave nodes

1.2. Objectives

The objective of the research is to improve the existing parallel compact genetic algorithm to escape from local optima and increase the performance by focusing on the shared parameters and the sharing mechanism.

Chapter 2 Literature Reviews

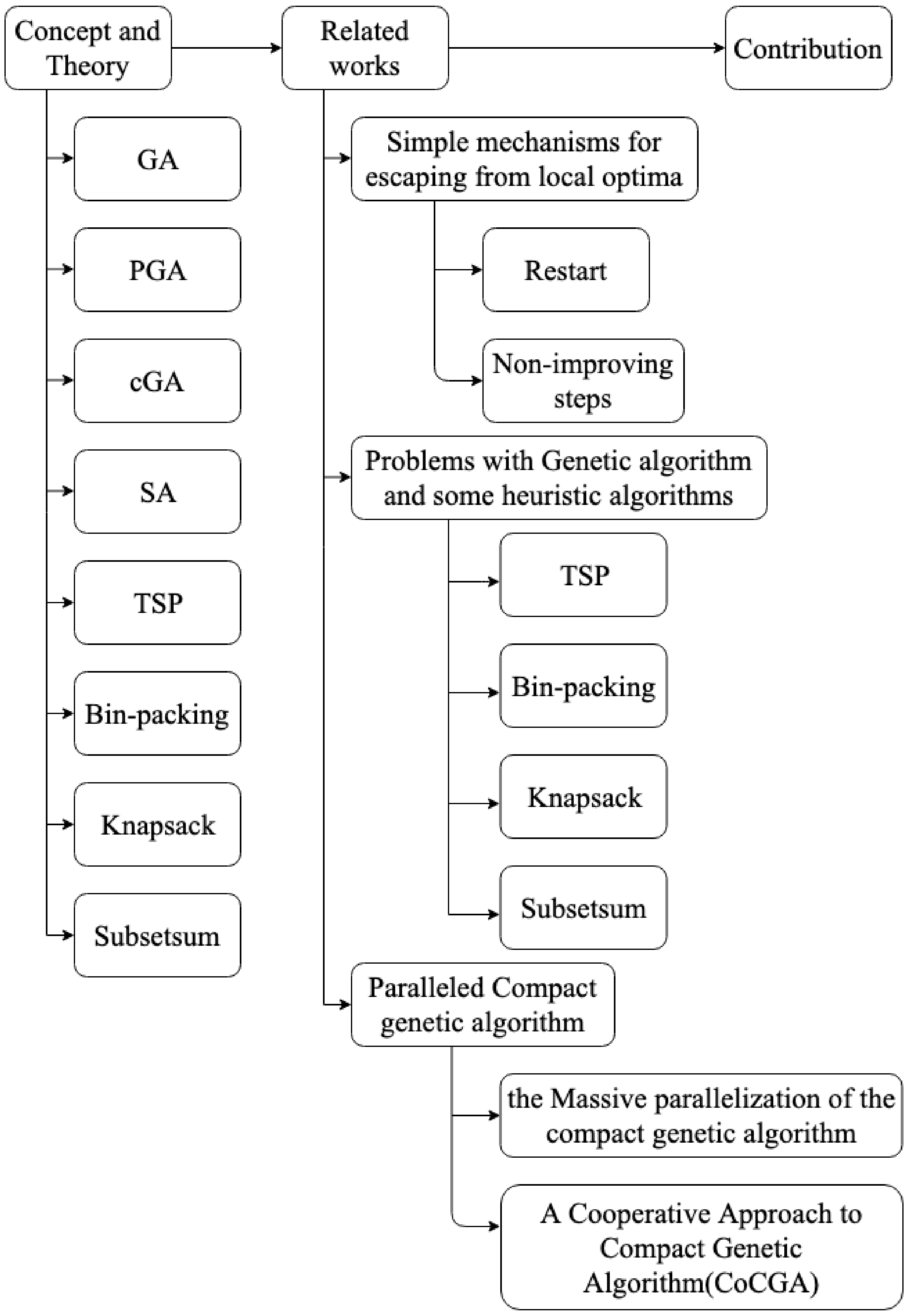

Concept and Theory
Related works
Contribution
GA
PGA
cGA
SA
TSP
Bin-packing
Knapsack
Subsetsum
Simple mechanisms for escaping from local optima
Restart
Non-improving steps
Problems with Genetic algorithm and some heuristic algorithms
TSP
Bin-packing
Knapsack
Subsetsum
Paralleled Compact genetic algorithm
the Massive parallelization of the compact genetic algorithm
A Cooperative Approach to Compact Genetic Algorithm(CoCGA)

Figure 2.1: An overview of literature reviews

2.1. Concept and Theory

2.1.1. Genetic algorithm (GA) [2]

Genetic algorithm is a search algorithm inspired by natural evolution. The main idea is to improve a set of solutions called population instead of only one solution. To initialize a population in GA, it starts from a set of random possible solutions to keep diversity. Moreover, the diversity of the population must be concerned, and the size of the population must be large enough to find the best result efficiently but not too large to slow down the GA. Then, in each iteration, some solutions in the set will be replaced by their children which are produced by some rules. Moreover, the algorithm produces offspring in the next generation by three main operations; selection, mutation, and crossover.

For selection, it is the first step to produce offspring by selecting a set of expected high-quality solutions from the current population. There are many ways of parent selections. For example, Fitness Proportionate Selection -- exploit fitness scores of each solution in population to indicate the chances to be selected, Tournament Selection -- randomly select some solutions from population and choose the best one, Rank Selection -- choose solutions from the population by the rank of their fitness scores, and Random Selection.

For Mutation, its idea is based on simple random change a solution. Mutation maintains the diversity in the solution pool and prevent premature convergence. It makes change or flip values of some fragments of a solution.

Crossover exchanging parts between two solutions by swapping of some bits with randomly choosing crossover points.

To update the population, all possible solutions in the current generation will be evaluated by the fitness function to calculate how good they are and give scores to them. Furthermore, the diversity of the population should be maintained. There are also many ways to update the population such as replacing all population by solutions from the next generation, replacing the solutions that have the worst scores by the solutions from the next generation, etc.

```
Algorithm: Genetic Algorithm
 1: Inputs: L is chromosome Length, N is population size
 2: Initialize: population = Generate_population(N, L), fitness =
Fitness_function(individual)
 3: Repeat
 4:         Parents = Parent_Selection(population)
 5:         Offspring = CrossOver(Parents)
 6:         Offspring = Mutaion(Offspring)
 7:         Fitness_Offspring = Fitness_function(Offspring)
 8:         Update Population
 9:         Find current best solution
11: Until termination criteria are reached
12: Outputs: best solution
```

Figure 2.2: Pseudocode of Genetic Algorithm

2.1.2. Parallel Genetic algorithm

A parallel genetic algorithm is a group of the genetic algorithm that cooperates with each other to discover the high-quality solutions. There are many types of the parallel genetic algorithm such as master-slave, fine-grained and coarse-grained [11],[12]. In the master-slave model, Master has duties to control and share data to all slaves, while slaves have the same duties. For example, Master controls parent selection and fitness assignment, and Slaves receive a set of solution and do the remaining steps of GA. In Coarse-grained or island model, the population is divided into subpopulations as the initial population of each island. Each island runs a genetic algorithm and sometimes share some data to other islands. In Fine-grained model, there is only one population, however, it has special structure for each node to only share data to its neighbors. Considering cooperation among nodes, we can clearly divide these groups into two types, non-migration, and migration. Moreover, in the migration process, there are many significant factors that should be carefully considered such as migration size, migration topology, migration frequency and migration strategies [13]

2.1.3. Compact genetic algorithm (cGA)

The compact genetic algorithm is proposed by Harik, Lobo, and Goldberg [5]. The main idea of the compact genetic algorithm is similar to the concept of the genetic algorithm and also proved that it is equivalent to the simple genetic algorithm with single point crossover. In detailed, the compact genetic algorithm attempts to search for high-quality solutions in the group of all possible solution or called solution space by imitating natural evolution. Firstly, a probabilistic vector is defined to represent the solution space. The size of the probabilistic vector indicates the number of bits required to represent a solution.

For example [14]:

Square roots problem:

Input: Given a real number

Output: Square roots of the input

Probabilistic vector: size of 30

Table 2.1: The example of the initialized probabilistic vector

0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

From the example above, the easiest way is to define a real number by binary digits 1 and 0. A method suggested that 30 bits (the size of the vector is 30) are divided into two parts equally, so the best number we can represent is a 2^{15} integer (32768) with a decimal precision of 2^{15}. To illustrate, 000000000000011 110000000000000, 000000000000011 is the sum of 2^0 and 2^1 which is equal to 3 and 110000000000000 is the sum of 2^{-1} and 2^{-2} which is equal to 0.75, so 000000000000011 110000000000000 represents 3.75. In addition, Each element of the probabilistic vector can possibly be a real number in the range to 0 to 1 because each element of the probabilistic vector means the probability that the i-th bit will be 1. After designing the probabilistic vector, the process begins with initializing values to the vector using a various method such as uniform distribution that all elements in the vector are set to 0.5. Then, a group of numbers or called individuals are sampling from the probabilistic vector. After that, two individuals will be evaluated its quality or called score by a function called fitness function which is designed depending on problems. In the case of square roots, the fitness function can be the difference between the power of two of an individual and the input number, so the less the difference is, the higher quality the individual has. Two individuals will compete with each other to find winners and losers which will be used for updating the probabilistic vector. To update the probabilistic vector, each position of the losers and the winners will be considered. If they are not equal, the probability of the vector in that position will be increased or decreased following the winner. All of these steps will be repeated until finding individuals that their scores are satisfactory.

```
Algorithm: Compact Genetic Algorithm
 1: Inputs: L is chromosome Length, N is population size
 2: Initialize: Prob_vector = Generate_vector(),
 3:             a = Generate_candidate(Prob_vector),
 4:             b = Generate_candidate(Prob_vector)
 5: Repeat
 6:         winner, loser = evaluate(Fitness_function(a), Fitness_function(b))
 7:         For i = 1 to L Do // Update probabilistic vector
 8:             If winner[i] != 1 Then Prob_vector[i] = Prob_vector[i] + 1/N
 9:             Else Prob_vector[i] = Prob_vector[i] - 1/N
10:         Find current best solution
11: Until termination criteria are reached
12: Outputs: best solution
```

Figure 2.3: Pseudocode of Compact Genetic Algorithm [5]

2.1.4. Simulated Annealing (SA)

Simulated annealing is an optimizing algorithm proposed by Kickpatrick, Gelett, and Vecchi (1983) [15] and Cerny (1985) [16]. The idea of the algorithm is inspired by the cooling process of forging metals that atoms in metals move unpredictably in high temperature. In the simulated annealing algorithm, in each temperature that is decreasing, it searched neighboring solution of the current solution. In the case that the neighboring solutions are better, they will always be accepted. However, if the neighboring solutions are worse, it may be accepted according to the current temperature-dependent probability given by

$$PT = \begin{cases} 1 & if\ Cost(x_i) \le Cost(x_j) \\ e^{\frac{-(Cost(x_i)-Cost(x_j))}{T}} & if\ Cost(x_i) > Cost(x_j) \end{cases}$$

Where T is the current temperature, x_i is the current solution, x_j is neighboring solutions

```
Algorithm: Simulated Annealing
1: Inputs: Problem_size, $iterations_{max}$, $temp_{max}$
2: Initialize: $S_{current}$ = CreateInitialSolution(Problem_size), $S_{best} = S_{current}$
3: Repeat i = 0 to $iterations_{max}$
4:       $S_i$ = createNeighborSolution($S_{current}$)
5:       $temp_{curr}$ = CalculateTempurature(i, $temp_{max}$)
6:           If(Cost($S_i$) <= Cost($S_{current}$))
7:             $S_{current} = S_i$
8:             If(Cost($S_i$) <= Cost($S_{best}$))
9:               $S_{best} = S_i$
10:          End
11:          Elseif(Exp( (Cost($S_{current}$- Cost($S_i$)))/$temp_{curr}$ ) > Rand())
12:                $S_{current} = S_i$
13:          End
12: Outputs: $S_{best}$
```

Figure 2.4: Pseudocode of Simulated Annealing [17]

2.1.5. Traveling Salesman Problem

Traveling Salesman Problem (TSP) is a difficult problem. It consists of cities and a salesman. The problem is to find the shortest path that the salesman can visit all cities just once and come back to the first city [7].

For example [18]:

Input :

cities = {A, B, C, D, E}

distances =

Table 2.2: The adjacency matrix showing the example of the input distances of the traveling salesman problem

Cities	A	B	C	D	E
A	0	2	0	12	5
B	2	0	4	8	0
C	0	4	0	3	3
D	12	8	3	0	10
E	5	0	3	10	0

To find a path of the problem is equivalent to find a Hamiltonian cycle which is NP-complete. For example, path {A, B, C, D, E, A} is a total length of 24

2.1.6. Bin Packing Problem

Bin packing problem is an NP-hard problem which is to find the minimum number of bins of fixed capacity that can contain all items having various weights. Formally, given n item types with weights W = $\{w_1,..., w_n\}$. There is the unlimited number of bins of fixed capacity c. The problem is [19], [20]

Minimize k where k is the number of bins used to contain all items

Subject to

$\sum_{i=1}^{k} c \leq \sum_{i=1}^{k}\sum_{j=1}^{n} x_{ij}w_j$ where x_{ij} = 1 if bin i contains item j, otherwise $x_{ij} = 0$

$\sum_{j=1}^{k} x_{ij} = 0 \; or \; 1$ and $\forall w_j \leq c$ where $w_j \in W$

For example [18] :

Input:

Weight W = {4, 8, 1, 4, 2, 1}

Bin Capacity c = 10

Output: 2

First bin = {4, 4, 2} and second bin = {8, 1, 1}

2.1.7. Knapsack Problem

Knapsack problem is a NP-hard problem that fills the knapsack with items chosen from n items with various weights and values in order to get the highest sum of values or profits without exceeding the weight capacity of the knapsack. Formally, it is given n items with weights W = $\{w_1,..., w_n\}$ and values V = $\{v_1,..., v_n\}$ and assumed that all of them are positive integers. X = $\{x_1,..., x_n\}$ shows which items are chosen and maximize the sum of profit without crossing the capacity c. x_i = 1 if item i is chosen, otherwise x_i = 0. The problem is [21]

Maximize $\sum_{i=1}^{n} x_i\ v_1$

Subject to

$$\sum_{i=1}^{n} x_i\ w_i\ \leq c$$

For example [22]:

Input:

Value V = {60, 100, 120}

Weight W = {10, 20, 30}

Knapsack capacity c = 50

Output: 220

Solution = {10} , sum of value = 60 ; Solution = {20} , sum of value = 100

Solution = {30} , sum of value = 120 ; Solution = {20, 10} , sum of value = 160

Solution = {30, 10} , sum of value = 180 ; Solution = {30, 20} , sum of value = 220

Solution = {30, 20, 10} > 50

2.1.8. Subset Sum Problem

In subset sum problem, a set of numbers S = $\{s_1, \ldots, s_n\}$ and a fixed number c are given. The problem is to find a subset of the given set and its sum is closest to the fixed number without exceeding it. Formally [23],

Maximize $\sum_{i=1}^{n} x_i \ s_i$

Subject to

$\sum_{i=1}^{n} x_i \ s_i \le c$, $x_i \in \{0,1\}$ where $x_i = 1$ if the ith number is chosen, otherwise $x_i = 0$

For example [24]:

Input:

S = {3, 34, 4, 12, 5, 2}

c = 9

Output: {4, 5}

2.2. Related works

2.2.1. Simple mechanisms for escaping from local optima

2.2.1.1. Restart

Restart is a simple and straightforward way to escape from local optima by reinitializing search process when it gets stuck in a local optimum. While it works effectively in the case that the number of local optima is not high and the cost of restarting is low, in other cases, it may not be suitable [25], [26], [27].

2.2.1.2. Non-improving steps

Non-improving step is another simple way to escape from local optima. The idea is to allow choosing neighboring solutions when a local optimum is encountered. There are many ways to choose neighboring solutions such as a random selection from all neighbors (uninformed Random Walk step) or from all neighbors that have the lowest increase (mildest ascent step). The example of the algorithms that use this mechanism is the stimulated annealing [25],[28].

2.2.2. Problems with Genetic algorithm and some heuristic algorithms

2.2.2.1. Traveling Salesman Problem

A Hybrid Heuristic for the Traveling Salesman Problem is proposed by R. Baraglia, J.I. Hidalgo, and R. Perego. [29] It is the combination of compact genetic algorithm and the Lin-Kernighan local search. For the compact genetic algorithm, the initialized probabilistic matrix of i*i, where i is the number of the input cities, is assigned by EL model. The concept is to assign high probability to short edges by the following equation:

$$p_{i,j} = \begin{cases} 0 & if\ i = j \\ \dfrac{\overline{L_\iota} - d(c_i, c_j)}{\overline{L_\iota} - \overline{l_\iota}} & ,\quad otherwise \end{cases}$$

Where $\overline{L_\iota} = max\left\{d(c_i, c_j): j\ \in\ \{1,2,\dots,i-1\}\right\}$,

$\overline{l_\iota} = min\left\{d(c_i, c_j): j\ \in\ \{1,2,\dots,i-1\}\right\}$

And $d(c_i, c_j)$ = distance from i-th city to j-th city

To generate a solution, the first visited city is randomly chosen, while the other is selected by the ranks of their probabilities as the first priority and distances from the currently selected city.

To update the probabilistic matrix, two solutions are considered which one has a higher score (winner). The idea is to update the probabilistic matrix by following the winner and escaping from the loser according to the equation:

$$p_{i,j}^{k+1} = \begin{cases} p_{i,j}^{k} + \dfrac{1}{n}\ if\ \left((c_i, c_j)\ or\ (c_j, c_i)\ \in winner\right)\ and\ \left((c_i, c_j)\ or\ (c_j, c_i)\ \notin loser\right) \\ p_{i,j}^{k} - \dfrac{1}{n}\ if\ \left((c_i, c_j)\ or\ (c_j, c_i)\ \notin winner\right)\ and\ \left((c_i, c_j)\ or\ (c_j, c_i)\ \in loser\right) \\ p_{i,j}^{k}\quad otherwise \end{cases}$$

Where $p_{i,j}^{k}$ = the probability of i-th city to j-th city in the k-th iteration

Algorithm: Compact Genetic Algorithm for Traveling Salesman Problem

```
 1: Inputs: L is chromosome Length, N is population size
 2: Initialize: Prob_vector = Generate_vector(), F_best = INT_MAX
 4: Repeat
 5:        S[1] = Generate_candidate(Prob_vector)
 6:        F[1] = Tour_Length(S[1])
 7:        idx_best = 1
 8:        For k = 2 to s do
 9:            S[k] = Generate_candidate(Prob_vector)
10:            F[k] = Tour_Length(S[k])
11:            If (F[k] < F[idx_best]): idx_best = k
12:        For k = 1 to s do
13:            If (F[idx_best] < F[k]) then Update(Prob_vec,S[idx_best],S[i])
14:                If (F[idx_best] < F_best) :
15:                        count = 0;
16:                        F_best = F[idx_best];
17:                        S_best =  S[idx_best];
18:                Else
19:                        Update(Prob_vec, S_best, S[idx_best]);
20:                        count = count + 1;
21:                end if
22:
23: Until Convergence(Prob_vec) OR count > CONV_LIMIT
24: Outputs: S_best, F_best
```

Figure 2.5: Pseudocode of Compact Genetic Algorithm for Traveling Salesman Problem [29]

2.2.2.2. *Bin Packing Problem*

Junkerneier proposed how to apply a genetic algorithm to the Bin Packing problem [19]. The concept is to randomly generate solutions and use the First-fit algorithm as the fitness function to calculate their scores. Given a set of numbers (the weights of items) and the fixed capacity of bins. First, it starts by randomly generate permutations of the same size as the set of numbers. Each permutation represents the index of the items that will be considered by the First-fit algorithm. Then, the First-Fit algorithm is applied to all permutations in the population. The First-Fit algorithm considers each item according to the order of indexes in the permutation. Because bins are also ordered by the time they are initialized, Items are assigned to the first bin that fits. If there are no current bins can fit the item, the new bin is initialized. The scores are the number of initialized bins, the less the better. Here is an example:

A solution in population = {4,5,6,3,9,2}, Bin_capacity = 13

Bin1 = {4}

Bin1 = {4,5}

Bin1 = {4,5}, Bin2 = {6}

Bin1 = {4,5,3}, Bin2 = {6}

Bin1 = {4,5,3}, Bin2 = {6}, Bin3 = {9}

Bin1 = {4,5,3}, Bin2 = {6,2}, Bin3 = {9}

For the parent selection, it uses the tournament selection. In the crossover step, it uses two parents to create an offspring by copying elements in the parents ordered from the first to the last one and from either of the parents alternatively. Next, the genetic algorithm randomly swaps two elements in a solution for the mutation step. To update the population, all solutions in the population will be replaced by all current offspring.

2.2.2.3. Knapsack Problem

For knapsack problem, Ken-Li li, Guang-Ming Dau and Qing-Hua Li proposed a genetic algorithm for the unbounded knapsack [22]. P.C. chu and J.E. Beasley proposed a genetic algorithm for the multidimensional knapsack [30]. Both of them use the total profits as scores for each solution. They use n-bit string, where n is the number of input items, to represent each solution (1 means selected item).

To generate and repair infeasible solutions from the mutation and crossover steps, they use Repair-operator to fix the infeasible solutions by deleting some selected items from the solutions and adding some items that should increase the total profits to the solutions according to the proportion of each item's profits and weights under the rule of not exceeding the bin capacity. For parent selection, mutation and crossover steps, they both use the same algorithms which are the tournament selection for parent selection, random bits converting from 0 to 1 and 1 to 0 for mutation and random copying elements from two parents for crossover.

2.2.2.4. Subset Sum Problem

For the subset sum problem, Rong Long Wang proposed a genetic algorithm to solve the subset sum problem [31]. The 0-1 vector was used to represent the solutions with the condition that the total sum of the selected numbers must not exceed the fixed value. Moreover, the following equation was used as the fitness function.

Fitness function f(x) = $k \times \left(C - A(x)\right) + (1 - k) \times A(x)$

Where $A(x) = \sum_{i=1}^{n} x_i s_i$, x is a candidate, C is the fixed value, and k = 1 if and only if $\left(C - A(x)\right) \geq 0$, otherwise k = 0.

For other operations like parent selection, crossover, and mutation, the processes were not different much, but the proposed algorithm used the proportion of the length of solutions and the number of the different genes of each pair of parent chromosome to control these operations.

2.2.3. Parallel Compact genetic algorithm

In theory, the compact genetic algorithm is equivalent to the genetic algorithm with crossover. Moreover, the compact genetic algorithm which uses the probabilistic vector instead of the collection of the whole populations will offer the alternative way to share the probabilistic vector which can affect to performance. There are two examples that use this alternative method to solve problems.

2.2.3.1. The Cooperative Compact Genetic Algorithm (CoCGA)

The first example is the Cooperative Approach to Compact Genetic Algorithm [32], this algorithm uses the cellular model. In detail, there are two types of nodes, the leader as a center, and the normal CoCGA are neighbours (4 cells). A variable called confidence counter (cc) is used to consider which probabilistic vector of each normal node will be updated to the probabilistic vector of the leader node. When all confidence counters of CoCGA around a leader are updated, the leader will choose the probabilistic vector from a CoCGA that has the highest confidence count at that time and then broadcast to the other CoCGA around it to update their probabilistic vectors. The result shows that the CoCGA with two normal nodes and one leader is at least three times better than a single compact genetic algorithm in term of execution time.

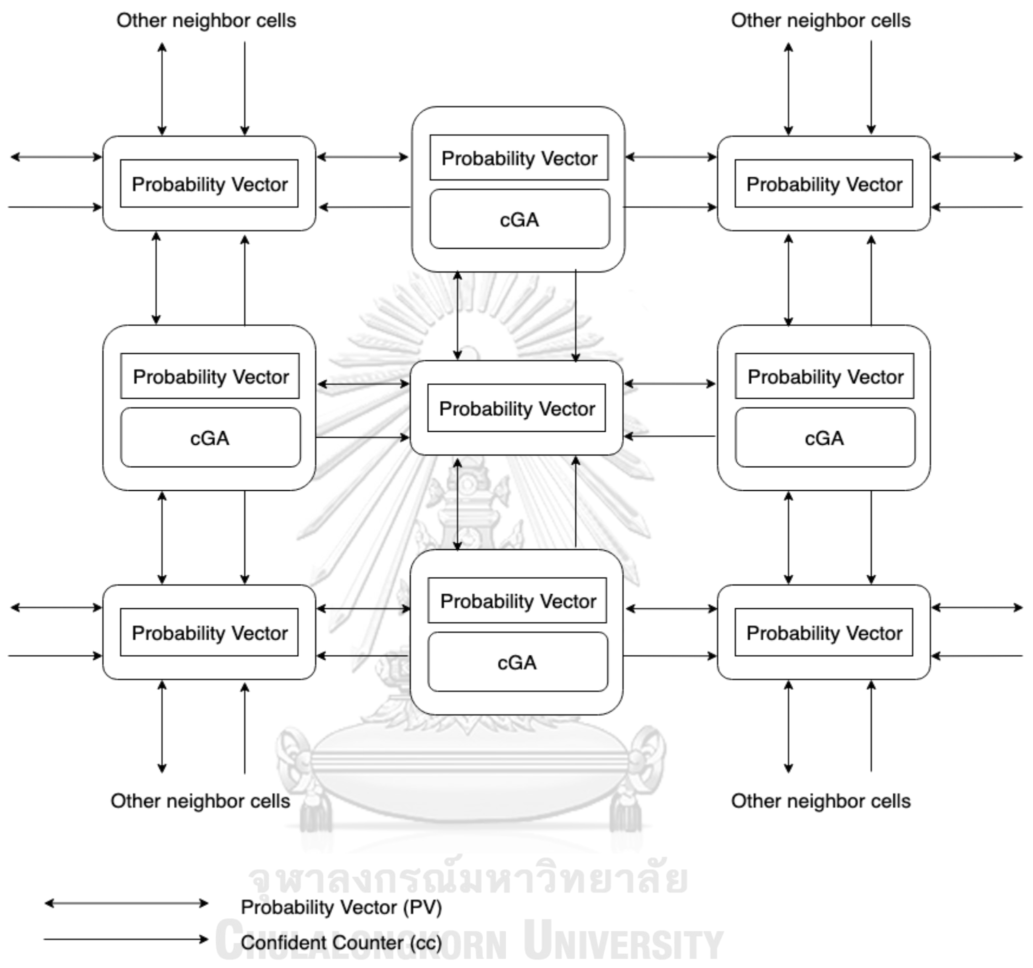


Figure 2.6: The structure of A Cooperative Approach to Compact Genetic Algorithm (CoCGA)

2.2.3.2. Massive parallelization of the compact genetic algorithm

The second example is that Lobo, Lima, and Martires [33]. They propose a parallel compact genetic algorithm using the master-slave model. Firstly, a probabilistic vector of each slave will be sent to master when the number of time that fitness function executed reaches the time interval of migration. Then, the master will calculate probabilistic vector obtained and resend a new probabilistic vector back to the slave. The point is that the master node may be updated many times, while a slave is working, so it is the way for all slaves to share information with each other. Moreover, this research measured the performance of the algorithm by counting the number of time that the fitness function are executed and shows the comparison of the performance with the various number of slaves and migration rates.

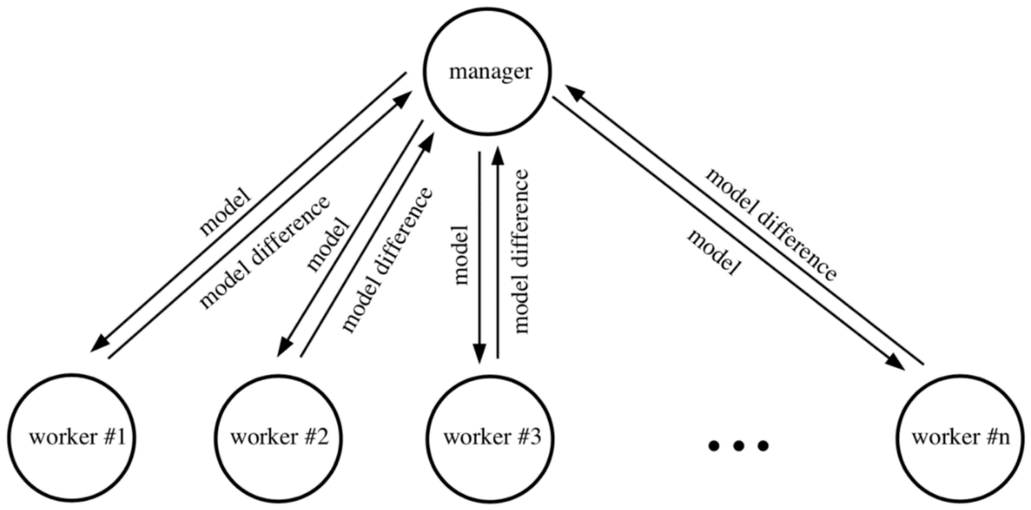


Figure 2.7 : The structure of the Massive parallelization of the compact genetic algorithm [33]

2.3. Contribution

This research focuses on the parallel compact genetic algorithm as to the Massive parallelization of the compact genetic algorithm and the Cooperative Compact Genetic Algorithm. The cooperative compact genetic algorithm is based on master-slave but this research uses simulated annealing and restart mechanism to escape from local optima. The simulated annealing is used to consider whether the probabilistic vector should be restarted or not and the number of iterations executed after the newest best solution is found is used as the temperature in the simulated annealing algorithm. Moreover, the restart mechanism is applied to two levels. The first level is the average the current probabilistic vector and the restart point because to reinitialize the probabilistic vector have a high cost from restarting to search for solutions from the beginning again. The simulated annealing is used for this level. However, if applying restart at the first level cannot escape from local optima and find the expected solution after k number of iterations executed by the slave, the restart is applied to the second level. The probabilistic vector must be reinitialized to the restart point. The experiment is different from the Massive parallelization of the compact genetic algorithm and the Cooperative Compact Genetic Algorithm. This research uses traveling Salesman Problem, Bin Packing Problem, Knapsack Problem, Subset Sum Problem in the tests, while the Massive parallelization of the compact genetic algorithm presents the experiment on a bounded deceptive function consisting of the concatenation of 10 copies of a 3-bit trap function with the deceptive-to-optimal ratio of 0.7 and the Cooperative Compact Genetic Algorithm uses One-Max and the De jong test functions (F1,F2,F3) [34].

Chapter 3 Material and Methodology

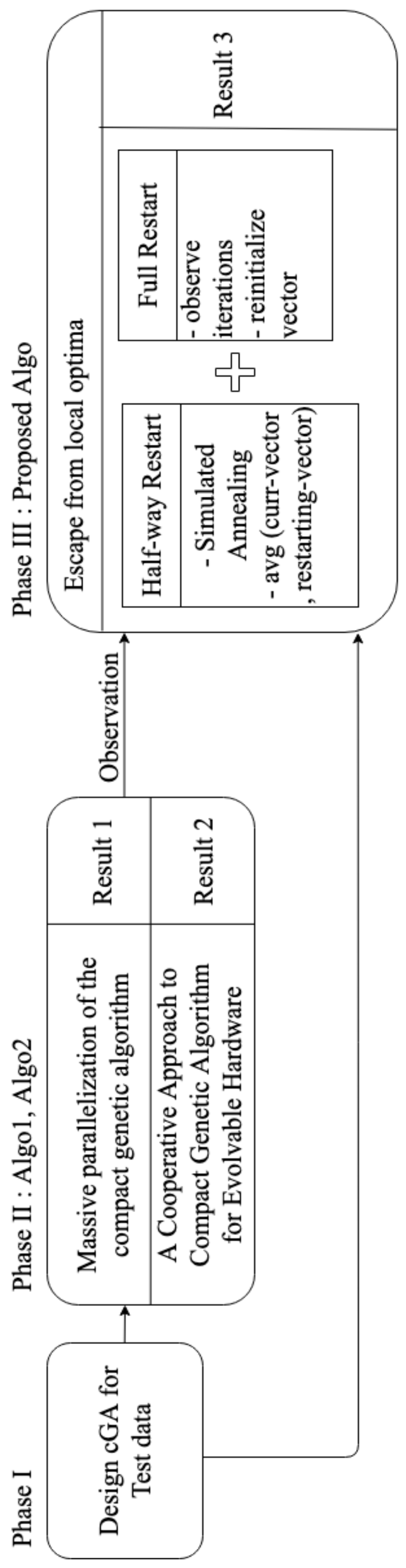


Figure 3.1: Flow chart of Material and Methodology

This chapter is about material and methodology. The process started by choosing the setting/test data that were suitable for the experiment. From the scope of the research, the experiment used the traveling salesman problem, bin packing problem, knapsack problem, and subset sum problem as the setting/test data. The compact genetic algorithm was designed for each problem to use in phase II (Implement and test Massive parallelization of the compact genetic algorithm and the Cooperative Compact Genetic Algorithm with the test data). In phase II, the test data started from small input and, from the observation, the sizes of input data were chosen. After that, the two algorithms were tested again and collected results. From observation, the proposed algorithm was designed in phase III.

3.1. The definition of attributes and Setting/Test Data

3.1.1. The definition of attributes

Table 3.1: The definition of attributes

Attributes	**Definition**
cGA	Compact genetic algorithm node
master node/leader node	The node that control all cGA nodes around it
iteration/cc	The node that control all cGA nodes around it
count	The number of iterations since the latest finding of good solution
bin_cap	Bin capacity
pop_size	The number of individuals or feasible solutions in the population generated from the probabilistic vector/matrix
state	The shared variables in the master node

3.1.2. Setting Data (Test)

3.1.2.1. Traveling Salesman Problem

The problems in the experiment has 3 sizes of cites (11, 15, and 17 cities and 5 problems per each). A problem of the 17-cities problems was from TSPLIB, a collection of traveling salesman problem datasets maintained by Gerhard Reinelt [35]. A problem of the 15-cities problems was created by John Burkardt [36], Florida State University and One problem of the 11-cities problems was from StackOverflow [37]. Other problems in each group were created by transforming from the problem in each group that was from other sources.

3.1.2.2. Bin Packing Problem

The experiment uses 4 problems of the bin packing problem from Prof. Dr. Armin Scholl and Dr. Robert Klein [38]. All problems have 50 items with average weight is "bin capacity/3". The first problem has the maximum deviation of all weight is 20 percent from the average weight, while the others has 50 percent.

3.1.2.3. Knapsack Problem

The 4 knapsack problems from Johny A. Ortega R. (Jao Ruiz) [39] are used as the test data. The second problem is in low-dimensional group with 20 items and 878 for bin capacity. The first, the third the fourth bin capacity around 1000 but the third one has 200 items, while the others have 100 items.

3.1.2.4. Subset Sum Problem

In the experiment, the 4 subset sum problems are used from John Burkardt, Florida State University [40]. The first and the fourth problems are a set of 10 numbers for a target of 50. The second problem consists of 21 numbers for an expected solution of 2463098. Then, the third one also has 10 numbers but for a target of 5842.

3.2. Phase I: Design cGa for Test data

All problems have similar structure of cGA: initialize probabilistic vector, generate individuals, evaluate populations and update the probabilistic vector. Solutions or individuals of each problem are different because of the different purposes of representation. Thus, the initialization of probabilistic vector may be different. However, the way to update the probabilistic vector is the same for all problems. The concept is the same as the update process of normal cGA which attempts to move searching areas torward to the winners and away from the losers by increasing and decreasing the probability in the vector. In the case of unsuccess to discover the expected solutions, the maximum iterations of each cGA node is 250,000 iterations

3.2.1. Traveling Salesman Problem

Initialize probabilistic vector

The algorithm uses a probabilistic matrix of size n × n instead of a probabilistic vector of size n by defining an element in row i-th column j-th to be a probability that city i-th will go to city j-th.

Generate individuals

A population will be generated by assigning the first city as the first visit and then randomizing the next city that has the chance to be visited over a constant. If the other cities have lower chances than the constant, the next city to be visited will be randomly selected. Moreover, each city must be visited only once.

Evaluate populations

Each population is assessed by its tour length. The goal is to discover the minimize tour length.

3.2.2. Bin Packing Problem

Initialize probabilistic vector

The probabilistic matrix initialization assigned 0.5 to each element of a matrix of size n which n is the number of givens items. The definition of the probability is that there are 50 percents of items i being in position j which i is a row and j is a column of the matrix.

Generate individuals

To generate a population from the probabilistic matrix, we start from randomizing an item that must have its probability to be in the first position over a constant. Then, for the other positions, the randomization is similar but, in the case that there is no other item having the probability to be in the specific position more than the constant, any of them will be randomly chosen.

Evaluate populations

To evaluate the value of a population, the fitness function for this problem is using a heuristic algorithm called the first-fit [19] and the less its score is, the better solution it is. In detailed, the first-fit algorithm is to attempt to sequentially put items into a group of bins that they first fit.

3.2.3. Knapsack Problem

Initialize probabilistic vector

For the knapsack problem, 0.5 is assigned to all elements in a probabilistic vector of size n which n is the number of items. It means that each item has 50 percents chance to be put into a knapsack with a fixed capacity.

Generate individuals

The algorithm generates populations from the probabilistic vector by randomizing each item to put into a knapsack. Each chosen item must have higher probabilistic than a constant. However, if the other items that can put into the knapsack without exceeding the capacity have lower probabilities, items will be randomly chosen to fill the knapsack as many as possible [22].

Evaluate populations

For the evaluation, the sum of profits in a knapsack is used as a score of a population. The higher the score is, the better the population is.

3.2.4. Subset Sum Problem

Initialize probabilistic vector

All elements of a probabilistic vector in this problem is assigned to 0.5 which means that all number have 50 percent chances to be selected for a population.

Generate individuals

Each population is generated by randomizing a number. A number S in the given set will be chosen, if the randomized number is smaller than a probability of S in the probabilistic vector.

Evaluate populations

To evaluate a population X = $\{x_1,..., x_n\}$, a set of numbers S = $\{s_1, ..., s_n\}$ and an expected value C are given. We use an equation below to calculate the population's score [31]:

Fitness function f(x) = $k \times (C - A(x)) + (1 - k) \times A(x)$

Where $A(x) = \sum_{i=1}^{n} x_i s_i$ and k = 1 if and only if $(C - A(x)) \geq 0$, otherwise k = 0

3.3. Phase II: Implement and test Massive parallelization of the compact genetic algorithm and Cooperative Compact Genetic Algorithm (CoCGA) with the test data

3.3.1. Massive parallelization of the compact genetic algorithm

The algorithm had master-slave topology. In this research, there were 4 normal cGA nodes with 1 master node. The performance of the algorithm was measured by the number of iterations executed until finding the expected solution or the number of iterations reached the maximum.

3.3.1.1. Normal cGA node

Firstly, Normal cGA nodes received the initialized probabilistic vector from the master node. Then, the 4 cGA nodes run cGA algorithm in parallel. There were 4 main steps as to normal cGA: generate 8 individuals from the probabilistic vector, evaluate all generated individuals and find the best individual of the current generation, compete the best individual with the others, update the probabilistic vector toward the best individual. For sharing data, when the number of iterations executed after finding the latest best solution reached the time interval of migration, the difference of the current probabilistic vector and the previous probabilistic vector was sent to the master node. Next, the master node calculated the received data and sent the calculated data back to the cGA node. After that, the received data from the master node replaces the current probabilistic vector of the normal cGA.

Algorithm: Massive parallelization of the compact genetic algorithm (Normal cGA node)

```
 1: Inputs: L is chromosome Length, N is population size
 2: Initialize: Prob_vector = $vector_{master}$ from Master node, iteration = 0, k = 0
 3:             best_candidate = []
 4: Repeat
 5:     If there is an update from Master node:
 6:         Prob_vector = $vector_{master}$ from Master node
 7:         Prev_vector = Prob_vector
 8:     Candidates = Generate_candidate(Prob_vector, N)
 9:     Scores = Fitness_function(Candidates)
10:     Update Prob_vector
11:     iteration++ , k++
12:     diff_vector = Prob_vector - Prev_vector
13:     If the best individual from Candidates is better than best_candidate :
14:         best_candidate = the best individual from Candidates
15:         counter = 0
16:     Send iteration, diff_vector to the Master node
17: Until a cGA node discovers the expected solution or reach the limit
18: Outputs: best_candidate, iteration
```

Figure 3.2: Pseudocode of Massive parallelization of the compact genetic algorithm (Normal cGA node)

3.3.1.2. Master node

Master node initialized the probabilistic vector and broadcasted to all normal cGA nodes. The master node had its own probabilistic vector which also was initialized with the same values as other normal cGA. When a normal cGA sent its data to the master node, it would add the data to its probabilistic vector and then sent the calculated variables back to the normal cGA. This was the way to share data among normal cGA nodes because the number of iterations executed by each normal cGA nodes reach the interval time of migration at the different points of time. Thus, during the time a normal cGA run the normal cGA steps, the master node's probabilistic vector might be updated by other normal cGA nodes.

```
Algorithm: Massive parallelization of the compact genetic algorithm (Master node)
1: Inputs: $diff\_vector^{i}_{cGA}$ , $k^{i}$ ; i = {1,2,3,4} // interrupted and  sent by $cGA^{i}$ node
2: Initialize: $vector_{master}$ = Generate_vector()
3: Repeat
4:        If $k^{i} = 8$ :
5:            $vector_{master} = vector_{master} + diff\_vector^{i}_{cGA}$
6:            Send $vector_{master}$ to $cGA^{i}$ node
11: Until a cGA node discovers the expected solution or reach the limit
```

Figure 3.3: Pseudocode of Massive parallelization of the compact genetic algorithm (Master node)

3.3.2. A Cooperative Approach to Compact Genetic Algorithm (CoCGA)

A Cooperative Approach to Compact Genetic Algorithm has cellular model topology which master nodes must have not more than 4 normal cGA nodes in control and normal cGA nodes must be connected with less than 5 master nodes. The main idea was the higher number of the iterations is executed, the better the cGA node was. Moreover, the number of iterations executed by each normal cGA node must not be more than 250,000 iterations.

3.3.2.1. Normal cGA node (Normal CoCGA)

All cGA nodes had their own variables called confident counter (cc) which was the number of iterations executed. However, other steps were still similar to normal cGA. For the first step, all cGA nodes generated 8 individuals from the initialized probabilistic vector from the master node. Then, all individuals were evaluated by the fitness function and given scores. Next, the best individual with the highest fitness score was competed with the others and updated the probabilistic vector toward the best one and to escape the others. To share data, in each iteration, each node sent its probabilistic vector and confident counter (cc) to its leader node. Finally, it went back to the first step and so on until it discovered the expected solutions, or the number of iterations reached the limit

Algorithm: A Cooperative Approach to Compact Genetic Algorithm for Evolvable Hardware (Normal CoCGA node)

```
 1: Inputs: L is chromosome Length, N is population size
 2: Initialize: Prob_vector = $vector_{master}$ from Leader node, cc = 0,
 3:             best_candidate = []
 4: Repeat
 5:        If there is an update from Master node:
 6:            Prob_vector = $vector_{master}$ from Master node
 7:        Candidates = Generate_candidate(Prob_vector, N)
 8:        Scores = Fitness_function(Candidates)
 9:        Update Prob_vector
10:        cc++
11:        If the best individual from Candidates is better than best_candidate :
12:            best_candidate = the best individual from Candidates
13:        Send cc, Prob_vector to the Leader node
14: Until a cGA node discovers the expected solution or reach the limit
15: Outputs: best_candidate, cc
```

Figure 3.4: Pseudocode of A Cooperative Approach to Compact Genetic Algorithm for Evolvable Hardware (Normal CoCGA node) [32]

3.3.2.2. *Leader node*

The leader node received cc and probabilistic vectors from all under controlling cGA nodes. Then, the leader node checked that all confident counters (cc) were updated from the previous time that updating data by the leader had happened. If all confident counters (cc) were changed, the master node chose the probabilistic vector from the cGA node that had the highest value of the confident counters (cc) and broadcast it to all under controlling cGA nodes. Moreover, this updated would replace the current probabilistic vectors of the cGA nodes.

Algorithm: the Cooperative Compact Genetic Algorithm for Evolvable Hardware (Leader node)

```
1: Inputs: $vector^{i}_{cGA}$ , $cc^{i}$ ; i = {1,2,3,4} // interrupted and  sent by $cGA^{i}$ node
2: Initialize: $prev\_cc^{i}$ = [0, 0, 0, 0], $vector_{master}$ = Generate_vector()
3: Repeat
4:        $vector_{master}$ = $vector^{max\_cc}_{cGA}$
5:        boardcast to all cGA nodes
6: Until a cGA node discovers the expected solution or reach the limit
```

Figure 3.5: Pseudocode of A Cooperative Approach to Compact Genetic Algorithm for Evolvable Hardware (Leader node) [32]

3.4. Phase III: Implement the cooperative compact genetic algorithm

The algorithm was improved upon the Massive parallelization of the compact genetic algorithm. Where the concept of sharing data is the same, the idea of simulated annealing accepting low-qualified solutions was used to consider an efficient sharing method. Then, if the sharing method cannot lead the current searching area away from local optima, the restart mechanism was exploited because it was more suitable to the probabilistic vector than a random selection from all neighbors of the Non-improving mechanism.

3.4.1. Restart half way

The concept of restart was exploited to escape local optima. To prevent high cost of reinitializing the probabilistic vector to restart the searching, there are two steps to escape from local optima. Restart halfway is the first step that exploited the idea of simulated annealing to control the sharing method. The number of iterations executed after finding the latest best local solutions called "counter" was used as the temperature in the equation of simulated annealing. In the simulated annealing, the high temperature had more probability to accept the worse solutions, but for this experiment, the lower value of the counter tended to accept the worse solutions.

Furthermore, after reinitializing the probabilistic vector, these new values were assigned to the master node's probabilistic vector. Thus, the next sharing data, other cGA nodes could acquire some effects from the previous restart halfway.

```
Algorithm: the Sharing Knowledge Compact Genetic Algorithm (SA-step I)
 1: Inputs: counter, $solution_{cGA}$, $solution_{master}$
 2: Initialize: result = False
 3: Calculate: diff_val = $solution_{cGA} - solution_{master}$
 4: If diff_val < 0 or Exp(-diff_val/counter) < random():
 5:      result = True // SA accept the $solution_{cGA}$
 6: End
12: Outputs: result
```

Figure 3.6: Pseudocode of A Cooperative Compact Genetic Algorithm (SA-step I)

3.4.2. Restart to the origin

However, the restart halfway may not completely escape from local optima. The observation of the number of iterations after finding the latest best solution was used to consider when the fully restart should be used. The observation is from the collected results of testing the restart halfway. The results show that the number of iterations of the restart halfway is around 80 iterations, but it still could not discover the better solutions. The "80 iterations" is from the maximum of the average iterations executed since the latest best solution was discovered which was calculated from the problems of 15-cities TSP, 17-cities TSP, Bin packing and Subset sum problems that had the highest results of each problem. The calculation considered only the rounds that had the closest total number of iterations to the average. Moreover, the values that were assigned to the probabilistic vector to restart were from the average of all probabilistic vectors from all cGA nodes in the first iteration. The consensus of their probabilistic vector was used to decide the start point for "Fully restart" .

Algorithm: the Sharing Knowledge Compact Genetic Algorithm (Master node)

```
 1: Inputs: $solution^i_{cGA}$, $vector^i_{cGA}$, $diff\_vector^i_{cGA}$ , $counter^i$
 2: Initialize: $vector_{master} = Generate\_vector()$, $solution_{master} = Max\_float$, $vector_{restart} = avg(vector^i_{cGA}; i = 1,..,4$ , $iteration = 1)$
 3: Repeat:
 4:         If result from SA-step1 is False :
 5:             $vector^i_{cGA} = avg(vector_{restart}, vector^i_{cGA})$
 6:         Elif $counter^i > 80$:
 7:             $vector^i_{cGA} = vector_{restart}$
 8:         Else :
 9:             $vector^i_{cGA} = diff\_vector^i_{cGA} + vector_{master}$
10: Until: a cGA node discovers the expected solution or reach the limit
```

Figure 3.7: Pseudocode of the Sharing Knowledge Compact Genetic Algorithm (Master node)

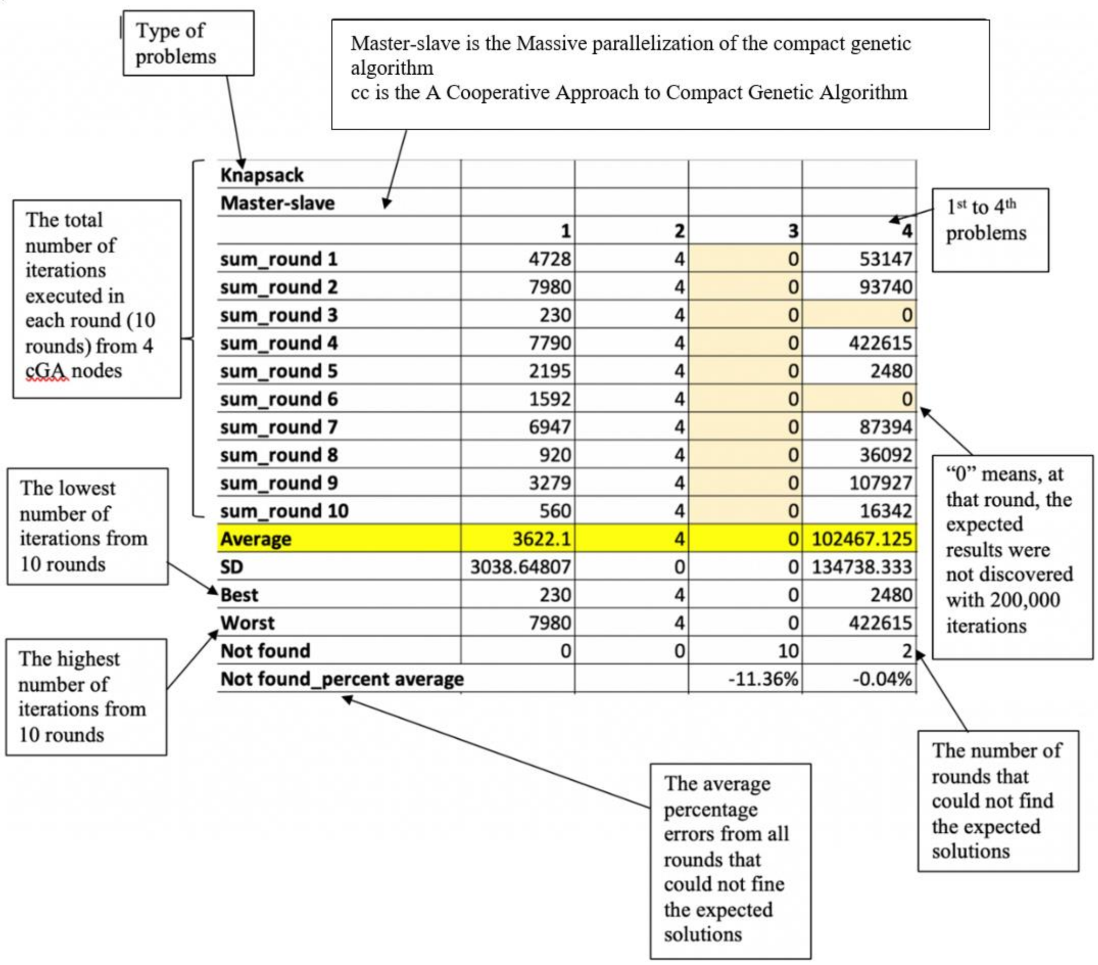
3.5. Evaluation

To evaluate the proposed method, the average number of the total iterations from all cGA nodes executed in the test for 10 rounds was used to measure the performance of the algorithm in the case that the algorithm could discover the expected solutions for all rounds (the first group). On the other hand, if it could not find the expected solutions (the second group), the best solution that they can find and the number of rounds that they cannot find the expected solutions were taken into account to measure the performance

Chapter 4 Results and discussion

This chapter is about results and discussion. The first figure illustrates the description of all result tables below. Then, the result of the experiment in phase II (Massive parallelization of the compact genetic algorithm and the Cooperative Compact Genetic Algorithm) are shown and compared to each other. Next, the results of the experiment in phase III (the proposed algorithm) for each problem were compared to two algorithms from phase II. Finally, the results summary and the overall comparison of each problem for the three algorithms are explained.

4.1. The description of the result tables and the summary chart



Type of problems

Master-slave is the Massive parallelization of the compact genetic algorithm
cc is the A Cooperative Approach to Compact Genetic Algorithm

Knapsack				
Master-slave				
	1	2	3	4
sum_round 1	4728	4	0	53147
sum_round 2	7980	4	0	93740
sum_round 3	230	4	0	0
sum_round 4	7790	4	0	422615
sum_round 5	2195	4	0	2480
sum_round 6	1592	4	0	0
sum_round 7	6947	4	0	87394
sum_round 8	920	4	0	36092
sum_round 9	3279	4	0	107927
sum_round 10	560	4	0	16342
Average	3622.1	4	0	102467.125
SD	3038.64807	0	0	134738.333
Best	230	4	0	2480
Worst	7980	4	0	422615
Not found	0	0	10	2
Not found_percent average			-11.36%	-0.04%

The total number of iterations executed in each round (10 rounds) from 4 cGA nodes

1st to 4th problems

The lowest number of iterations from 10 rounds

The highest number of iterations from 10 rounds

"0" means, at that round, the expected results were not discovered with 200,000 iterations

The number of rounds that could not find the expected solutions

The average percentage errors from all rounds that could not fine the expected solutions

Figure 4.1: The description of the result tables

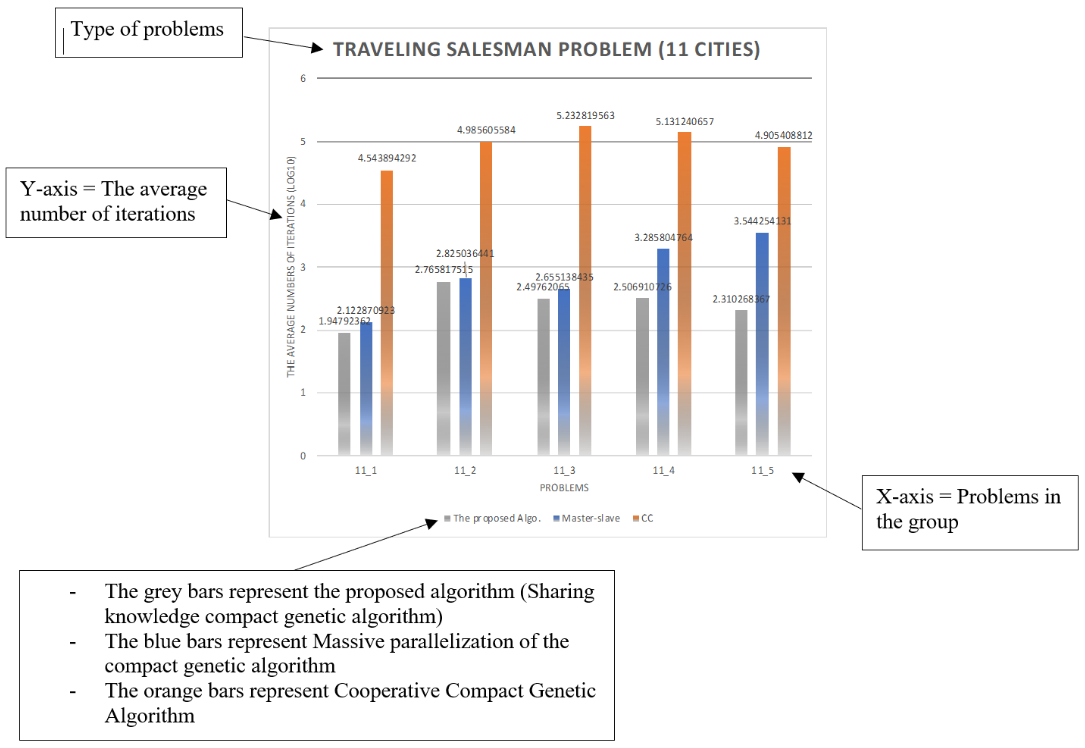


Figure 4.2: The description of the summary chart

4.2. Phase II

4.2.1. Traveling Salesman Problem (TSP)

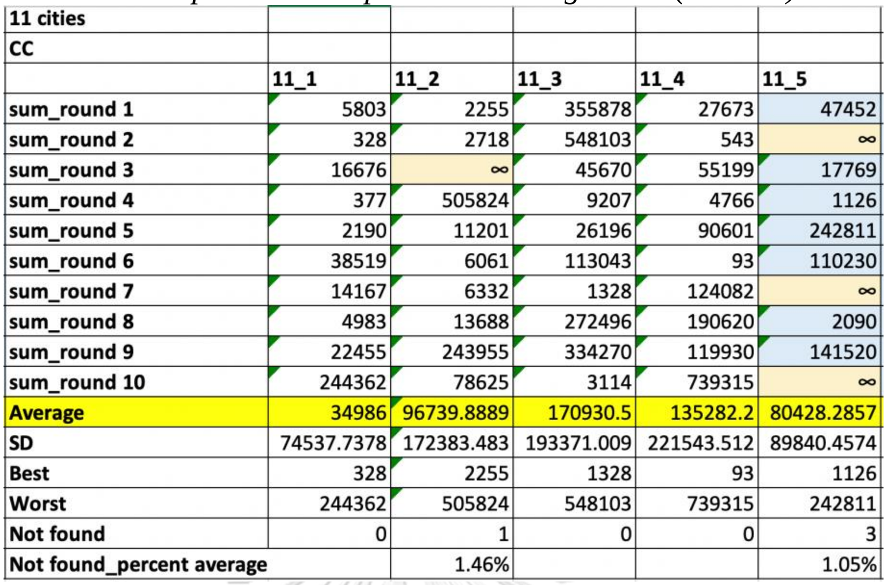
Overall, from the results of the experiment, the Massive parallelization of the compact genetic algorithm was more efficient than the Cooperative Compact Genetic Algorithm. Not only the average numbers of iterations executed were lower, but, in some rounds of some problems, the Cooperative Compact Genetic Algorithm could not find the expected solutions. Moreover, from all results, all the lowest numbers of iterations of each problem were from the Massive parallelization of the compact genetic algorithm, but the worst ones were from the Cooperative Compact Genetic Algorithm.

In 11 cities of TSP problem, Massive parallelization of the compact genetic algorithm had better results than Cooperative Compact Genetic Algorithm. The table below shows the total number of iterations in each round of 10 experiments. The results illustrate that, for all problems for 11 cities, the iterations executed by the Massive parallelization of the compact genetic algorithm are more than 100 times less than the iterations executed by the Cooperative Compact Genetic Algorithm. Moreover, for the second and the fifth problems, Cooperative Compact Genetic Algorithm could not find the expected solution in some rounds. The best solutions that it found had an average percentage error at 1.46 and 1.05, respectively.

Table 4.1: The performance on the traveling salesman problem (11 cities) by using the Massive parallelization of the compact genetic algorithm

11 cities					
Master-slave					
	11_1	11_2	11_3	11_4	11_5
sum_round 1	211	110	234	1508	979
sum_round 2	271	1662	166	87	97
sum_round 3	214	193	576	2966	262
sum_round 4	229	216	209	139	1538
sum_round 5	5	205	136	23	122
sum_round 6	41	412	345	355	3043
sum_round 7	89	466	1946	3443	1274
sum_round 8	220	868	220	15	55
sum_round 9	41	565	363	9742	139
sum_round 10	6	1987	325	1033	27506
Average	132.7	668.4	452	1931.1	3501.5
SD	105.33022	653.23183	540.05761	3014.0655	8487.2264
Best	5	110	136	15	55
Worst	271	1987	1946	9742	27506
Not found	0	0	0	0	0
Not found_percent average					

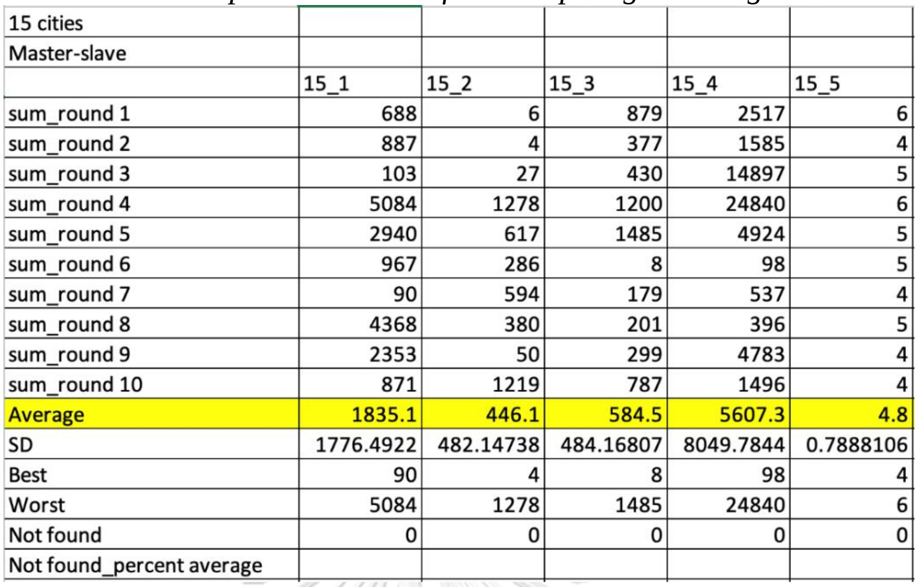
Table 4.2: The performance on the traveling salesman problem (11 cities) by using the Cooperative Compact Genetic Algorithm (CoCGA)



11 cities					
CC					
	11_1	11_2	11_3	11_4	11_5
sum_round 1	5803	2255	355878	27673	47452
sum_round 2	328	2718	548103	543	∞
sum_round 3	16676	∞	45670	55199	17769
sum_round 4	377	505824	9207	4766	1126
sum_round 5	2190	11201	26196	90601	242811
sum_round 6	38519	6061	113043	93	110230
sum_round 7	14167	6332	1328	124082	∞
sum_round 8	4983	13688	272496	190620	2090
sum_round 9	22455	243955	334270	119930	141520
sum_round 10	244362	78625	3114	739315	∞
Average	34986	96739.8889	170930.5	135282.2	80428.2857
SD	74537.7378	172383.483	193371.009	221543.512	89840.4574
Best	328	2255	1328	93	1126
Worst	244362	505824	548103	739315	242811
Not found	0	1	0	0	3
Not found_percent average		1.46%			1.05%

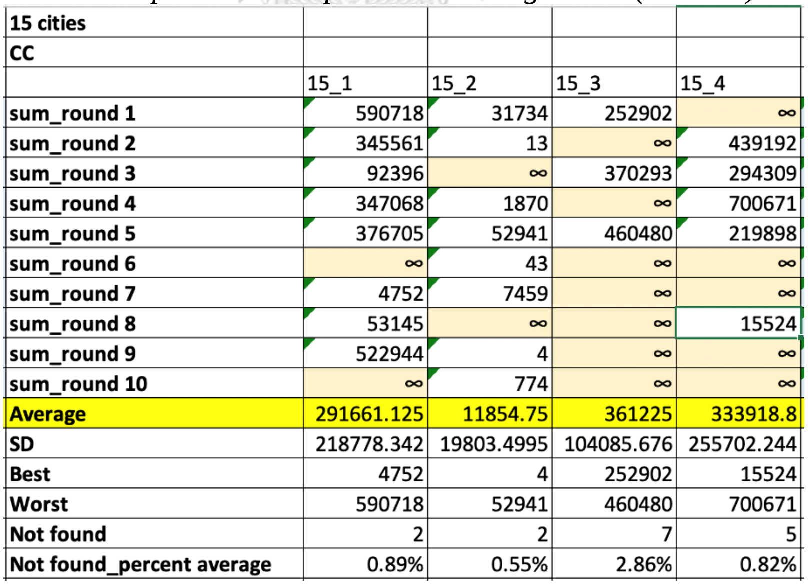
In 15 cities, the Massive parallelization of the compact genetic algorithm still had better performance, while the Cooperative Compact Genetic Algorithm discovered the expected solution only for the fifth problem. Furthermore, for the other problems, the number of rounds that the Cooperative Compact Genetic Algorithm could not find the expected outcomes were at least 2 rounds with the average percentage error at 0.89, 0.55,2.86 and 0.83, sequentially. Overall, the fourth and the third problems had the highest difficulty for the Massive parallelization of the compact genetic algorithm and the Cooperative Compact Genetic Algorithm, respectively, while the fifth problem was very easy for both algorithms to find the expected solutions.

Table 4.3 The performance on the traveling salesman problem (15 cities) by using the Massive parallelization of the compact genetic algorithm



15 cities					
Master-slave					
	15_1	15_2	15_3	15_4	15_5
sum_round 1	688	6	879	2517	6
sum_round 2	887	4	377	1585	4
sum_round 3	103	27	430	14897	5
sum_round 4	5084	1278	1200	24840	6
sum_round 5	2940	617	1485	4924	5
sum_round 6	967	286	8	98	5
sum_round 7	90	594	179	537	4
sum_round 8	4368	380	201	396	5
sum_round 9	2353	50	299	4783	4
sum_round 10	871	1219	787	1496	4
Average	1835.1	446.1	584.5	5607.3	4.8
SD	1776.4922	482.14738	484.16807	8049.7844	0.7888106
Best	90	4	8	98	4
Worst	5084	1278	1485	24840	6
Not found	0	0	0	0	0
Not found_percent average					

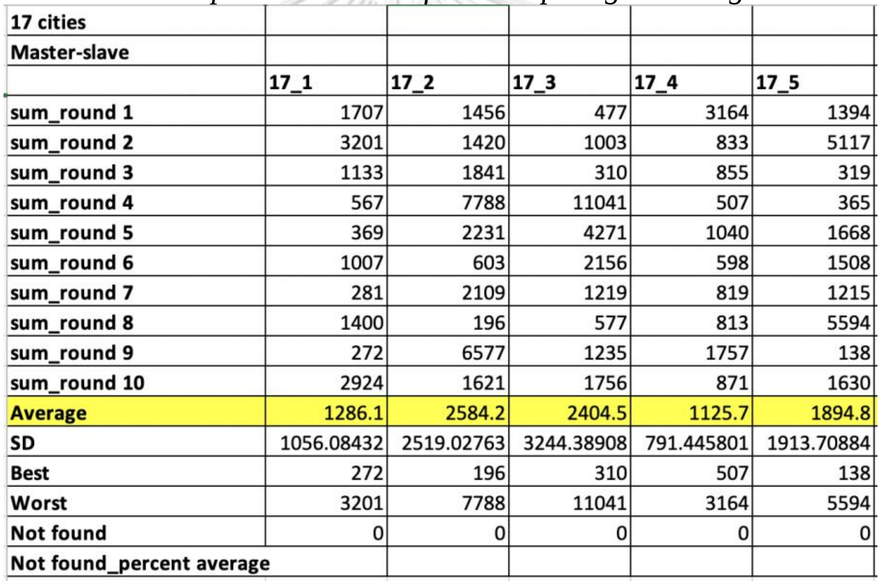
Table 4.4: : The performance on the traveling salesman problem (15 cities) by using the Cooperative Compact Genetic Algorithm (CoCGA)



15 cities				
CC				
	15_1	15_2	15_3	15_4
sum_round 1	590718	31734	252902	∞
sum_round 2	345561	13	∞	439192
sum_round 3	92396	∞	370293	294309
sum_round 4	347068	1870	∞	700671
sum_round 5	376705	52941	460480	219898
sum_round 6	∞	43	∞	∞
sum_round 7	4752	7459	∞	∞
sum_round 8	53145	∞	∞	15524
sum_round 9	522944	4	∞	∞
sum_round 10	∞	774	∞	∞
Average	291661.125	11854.75	361225	333918.8
SD	218778.342	19803.4995	104085.676	255702.244
Best	4752	4	252902	15524
Worst	590718	52941	460480	700671
Not found	2	2	7	5
Not found_percent average	0.89%	0.55%	2.86%	0.82%

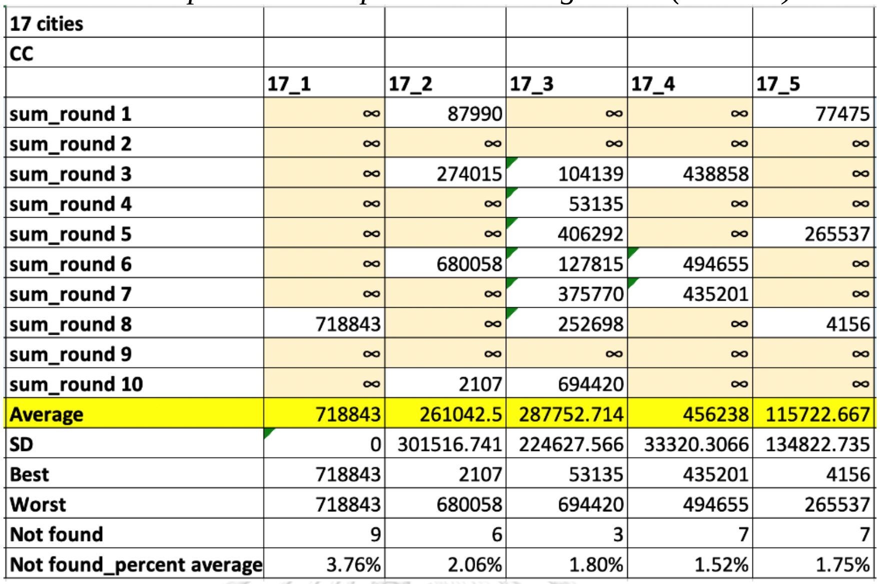
In the 17 cities of the TSP, the overall number of iterations executed by the Massive parallelization of the compact genetic algorithm still was much less than the results of the Cooperative Compact Genetic Algorithm which could not find the expected solutions more than two rounds of each problem. While the second problem seemed to be the most difficult for the Massive parallelization of the compact genetic algorithm, the Cooperative Compact Genetic Algorithm has the highest number of rounds in the experiment that could not reach the expected results. It was from the first problem (17_1) which also has the highest percentage error of 3.76.

Table 4.5: The performance on the traveling salesman problem (17 cities) by using the Massive parallelization of the compact genetic algorithm



17 cities					
Master-slave					
	17_1	17_2	17_3	17_4	17_5
sum_round 1	1707	1456	477	3164	1394
sum_round 2	3201	1420	1003	833	5117
sum_round 3	1133	1841	310	855	319
sum_round 4	567	7788	11041	507	365
sum_round 5	369	2231	4271	1040	1668
sum_round 6	1007	603	2156	598	1508
sum_round 7	281	2109	1219	819	1215
sum_round 8	1400	196	577	813	5594
sum_round 9	272	6577	1235	1757	138
sum_round 10	2924	1621	1756	871	1630
Average	1286.1	2584.2	2404.5	1125.7	1894.8
SD	1056.08432	2519.02763	3244.38908	791.445801	1913.70884
Best	272	196	310	507	138
Worst	3201	7788	11041	3164	5594
Not found	0	0	0	0	0
Not found_percent average					

Table 4.6: : The performance on the traveling salesman problem (17 cities) by using the Cooperative Compact Genetic Algorithm (CoCGA)



17 cities					
CC					
	17_1	**17_2**	**17_3**	**17_4**	**17_5**
sum_round 1	∞	87990	∞	∞	77475
sum_round 2	∞	∞	∞	∞	∞
sum_round 3	∞	274015	104139	438858	∞
sum_round 4	∞	∞	53135	∞	∞
sum_round 5	∞	∞	406292	∞	265537
sum_round 6	∞	680058	127815	494655	∞
sum_round 7	∞	∞	375770	435201	∞
sum_round 8	718843	∞	252698	∞	4156
sum_round 9	∞	∞	∞	∞	∞
sum_round 10	∞	2107	694420	∞	∞
Average	718843	261042.5	287752.714	456238	115722.667
SD	0	301516.741	224627.566	33320.3066	134822.735
Best	718843	2107	53135	435201	4156
Worst	718843	680058	694420	494655	265537
Not found	9	6	3	7	7
Not found_percent average	3.76%	2.06%	1.80%	1.52%	1.75%

4.2.2. Bin Packing Problem

From four bin-packing problems, according to the table below, both the massive parallelization of the compact genetic algorithm and The Cooperative Compact Genetic Algorithm had no different performance and both methods could discover the expected solutions of all four problems in every round. In details, the result of the massive parallelization of the compact genetic algorithm in the first problem was better than the result of The Cooperative Compact Genetic Algorithm, while the others were slightly worse. Furthermore, the difficulty of each problem for both algorithms was similar. According to the results from the table, the simplest problem was the second problem, while the hardest one was the fourth problem.

Table 4.7: The performance on the bin-packing problem by using the Massive parallelization of the compact genetic algorithm

Bin-packing				
Master-slave				
	1	**2**	**3**	**4**
sum_round 1	780	40	76	14517
sum_round 2	402	94	1040	4684
sum_round 3	383	24	947	4277
sum_round 4	1241	19	176	37356
sum_round 5	28	9	360	4524
sum_round 6	62	21	433	669
sum_round 7	446	7	2656	9035
sum_round 8	141	32	716	11789
sum_round 9	43	23	1019	5232
sum_round 10	621	70	231	7761
Average	414.7	33.9	765.4	9984.4
SD	387.001019	27.698576	755.174182	10428.0445
Best	28	7	76	669
Worst	1241	94	2656	37356
Not found	0	0	0	0
Not found_percent average				

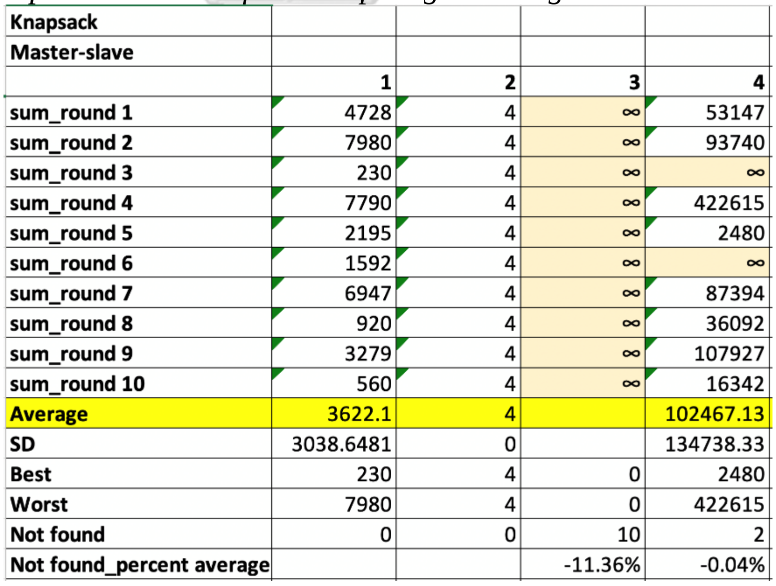
Table 4.8: : The performance on the bin-packing problem by using the Cooperative Compact Genetic Algorithm (CoCGA)

Bin-packing				
CC				
	1	**2**	**3**	**4**
sum_round 1	165	15	105	3354
sum_round 2	1554	13	432	6735
sum_round 3	287	35	90	9378
sum_round 4	549	51	2818	10681
sum_round 5	99	6	670	12527
sum_round 6	920	26	999	695
sum_round 7	1249	39	391	4929
sum_round 8	62	69	499	11774
sum_round 9	423	11	412	22050
sum_round 10	551	10	437	512
Average	585.9	27.5	685.3	8263.5
SD	505.052132	20.7431381	793.025718	6519.89586
Best	62	6	90	512
Worst	1554	69	2818	22050
Not found	0	0	0	0
Not found_percent average		0		

4.2.3. Knapsack Problem

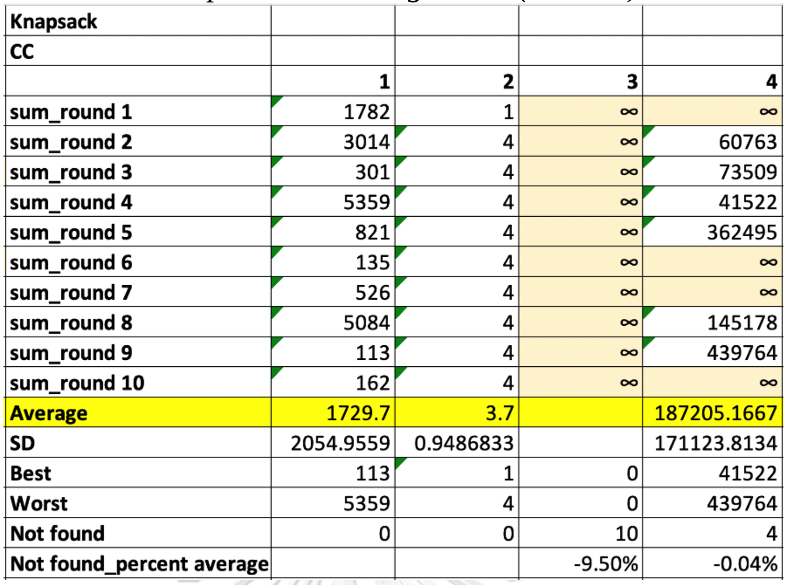
For Knapsack problem, the Cooperative Compact Genetic Algorithm and the Massive parallelization of the compact genetic algorithm had similar performance. They could discover the expected solutions for the first and second problems. Moreover, for the third problem, both algorithms could not find the expected solution for all ten rounds and some rounds from the fourth problem. The Cooperative Compact Genetic Algorithm had slightly better results for the first and the second problems, and, although both algorithms could not find the solution, the percentage error of the Cooperative Compact Genetic Algorithm was smaller. Interestingly, while, for the fouth problems, the number of rounds that the Massive parallelization of the compact genetic algorithm could not find the expected solution (2) were less than that of the Cooperative Compact Genetic Algorithm (4). The percentages error of both algorithms was equal.

Table 4.9: The performance on the knapsack problem by using the Massive parallelization of the compact genetic algorithm



Knapsack				
Master-slave				
	1	2	3	4
sum_round 1	4728	4	∞	53147
sum_round 2	7980	4	∞	93740
sum_round 3	230	4	∞	∞
sum_round 4	7790	4	∞	422615
sum_round 5	2195	4	∞	2480
sum_round 6	1592	4	∞	∞
sum_round 7	6947	4	∞	87394
sum_round 8	920	4	∞	36092
sum_round 9	3279	4	∞	107927
sum_round 10	560	4	∞	16342
Average	3622.1	4		102467.13
SD	3038.6481	0		134738.33
Best	230	4	0	2480
Worst	7980	4	0	422615
Not found	0	0	10	2
Not found_percent average			-11.36%	-0.04%

Table 4.10: The performance on the knapsack problem by using the Cooperative Compact Genetic Algorithm (CoCGA)

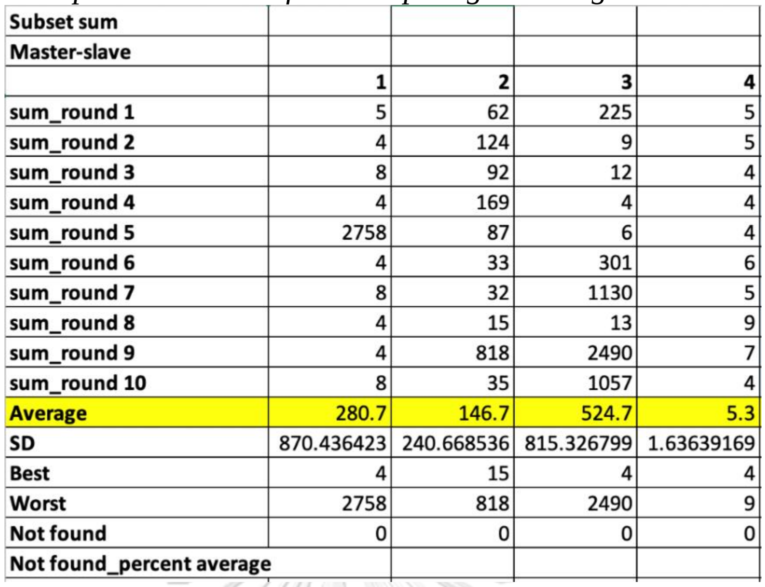


Knapsack				
CC				
	1	**2**	**3**	**4**
sum_round 1	1782	1	∞	∞
sum_round 2	3014	4	∞	60763
sum_round 3	301	4	∞	73509
sum_round 4	5359	4	∞	41522
sum_round 5	821	4	∞	362495
sum_round 6	135	4	∞	∞
sum_round 7	526	4	∞	∞
sum_round 8	5084	4	∞	145178
sum_round 9	113	4	∞	439764
sum_round 10	162	4	∞	∞
Average	1729.7	3.7		187205.1667
SD	2054.9559	0.9486833		171123.8134
Best	113	1	0	41522
Worst	5359	4	0	439764
Not found	0	0	10	4
Not found_percent average			-9.50%	-0.04%

4.2.4. Subset Sum Problem

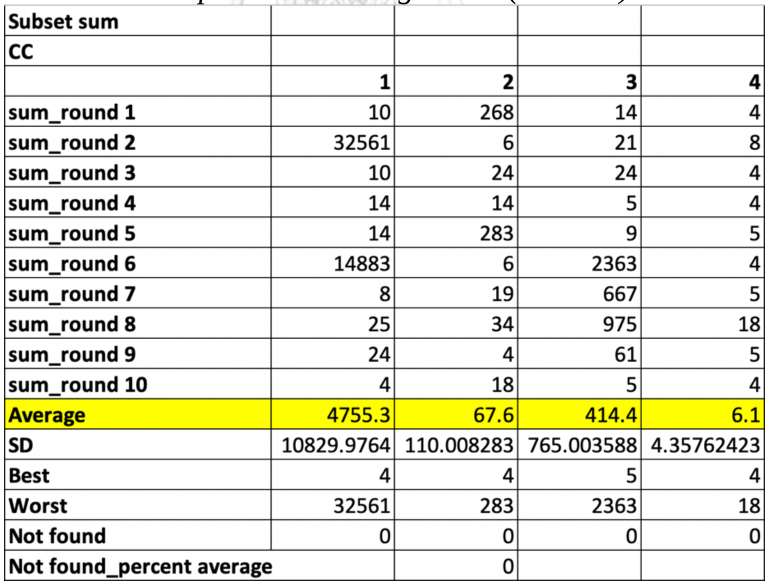
For subset sum problem, from the table below, the Massive parallelization of the compact genetic algorithm had a better performance than the Cooperative Compact Genetic Algorithm. It is significant in the first and in the fifth problems. However, for the second and third problems, the Cooperative Compact Genetic Algorithm were better. Overall, both algorithms could solve the problems.

Table 4.11: The performance on the subset sum problem by using the Massive parallelization of the compact genetic algorithm



Subset sum				
Master-slave				
	1	2	3	4
sum_round 1	5	62	225	5
sum_round 2	4	124	9	5
sum_round 3	8	92	12	4
sum_round 4	4	169	4	4
sum_round 5	2758	87	6	4
sum_round 6	4	33	301	6
sum_round 7	8	32	1130	5
sum_round 8	4	15	13	9
sum_round 9	4	818	2490	7
sum_round 10	8	35	1057	4
Average	280.7	146.7	524.7	5.3
SD	870.436423	240.668536	815.326799	1.63639169
Best	4	15	4	4
Worst	2758	818	2490	9
Not found	0	0	0	0
Not found_percent average				

Table 4.12: The performance on the subset sum problem by using the Cooperative Compact Genetic Algorithm (CoCGA)



Subset sum				
CC				
	1	2	3	4
sum_round 1	10	268	14	4
sum_round 2	32561	6	21	8
sum_round 3	10	24	24	4
sum_round 4	14	14	5	4
sum_round 5	14	283	9	5
sum_round 6	14883	6	2363	4
sum_round 7	8	19	667	5
sum_round 8	25	34	975	18
sum_round 9	24	4	61	5
sum_round 10	4	18	5	4
Average	4755.3	67.6	414.4	6.1
SD	10829.9764	110.008283	765.003588	4.35762423
Best	4	4	5	4
Worst	32561	283	2363	18
Not found	0	0	0	0
Not found_percent average		0		

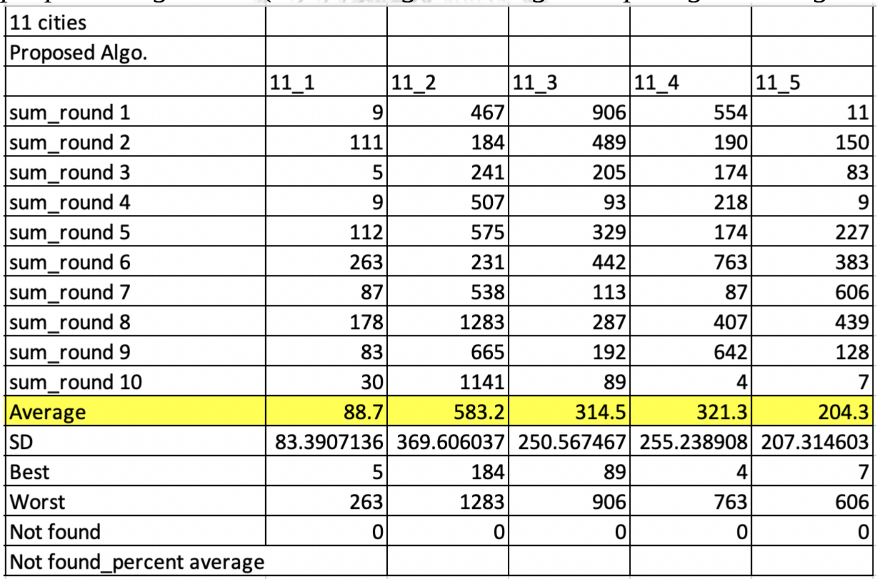
4.3. Phase III

The results of the experiment in phase III are shown below. In phase III, the results of the proposed algorithm were compared with the results from other algorithms in phase II.

4.3.1. Traveling Salesman Problem (TSP)

In the 11-cities problems, from the table below, the proposed algorithm could find the expected solutions for all rounds in the experiment. Moreover, the average numbers of iterations executed in each problem were not very different. The highest average number of iterations was from the second problem, while the lowest one was from the first problem.

Table 4.13: The performance on the traveling salesman problem (11 cities) by using the proposed algorithm (the sharing knowledge compact genetic algorithm)



11 cities					
Proposed Algo.					
	11_1	11_2	11_3	11_4	11_5
sum_round 1	9	467	906	554	11
sum_round 2	111	184	489	190	150
sum_round 3	5	241	205	174	83
sum_round 4	9	507	93	218	9
sum_round 5	112	575	329	174	227
sum_round 6	263	231	442	763	383
sum_round 7	87	538	113	87	606
sum_round 8	178	1283	287	407	439
sum_round 9	83	665	192	642	128
sum_round 10	30	1141	89	4	7
Average	88.7	583.2	314.5	321.3	204.3
SD	83.3907136	369.606037	250.567467	255.238908	207.314603
Best	5	184	89	4	7
Worst	263	1283	906	763	606
Not found	0	0	0	0	0
Not found_percent average					

Moreover, Figure 4.3 shows the comparison of three algorithms for all five problems. All average number of iterations in the results (the grey bars) were lower than the results from the Massive parallelization of the compact genetic algorithm (the blue bars) and the Cooperative Compact Genetic Algorithm (the orange bars). However, although the results from the proposed algorithm may be better than the results from the Massive parallelization of the compact genetic algorithm, they were not very different except the results from the fourth problem that the average number of iterations was 1931.1 (refer to the table 4.1) in the Massive parallelization of the compact genetic algorithm, while it was only 204.3, refer to the table 4.13,in the proposed algorithm.

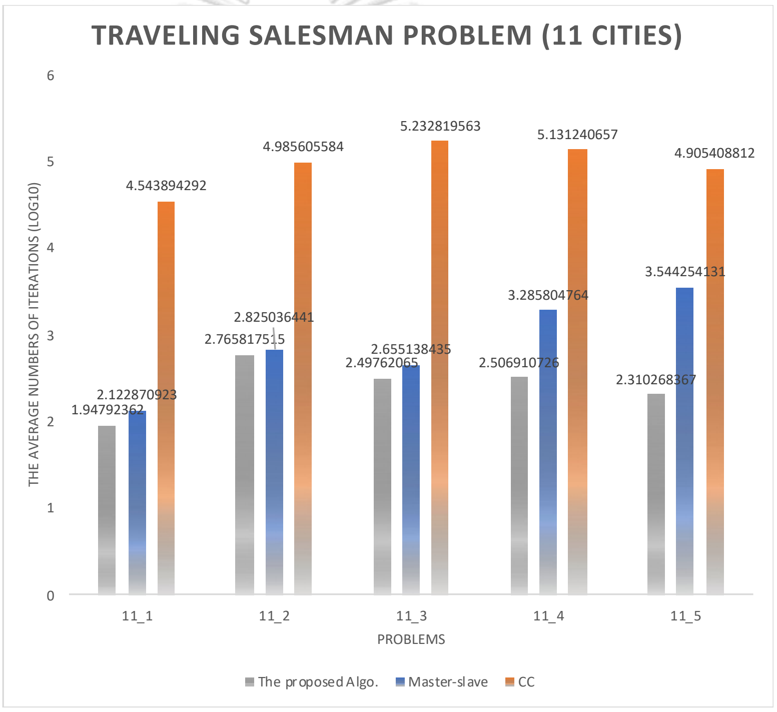


Figure 4.3: A bar chart shows the comparison of the results from the three algorithms on the traveling salesman problem (11 cities)

In the 15-cities problems, the proposed algorithm could find the expected solutions in every round. The fourth problem was the hardest and the fifth one was the simplest.

Table 4.14: The performance on the traveling salesman problem (11 cities) by using the proposed algorithm (the sharing knowledge compact genetic algorithm)

15 cities					
Proposed Algo.					
	15_1	15_2	15_3	15_4	15_5
sum_round 1	749	187	368	604	4
sum_round 2	269	18	368	160	4
sum_round 3	427	731	512	3879	5
sum_round 4	1957	932	139	619	5
sum_round 5	648	8	890	1299	6
sum_round 6	510	5	714	289	5
sum_round 7	765	465	90	1821	7
sum_round 8	450	8	942	261	4
sum_round 9	951	5	20	3008	4
sum_round 10	205	8	257	396	6
Average	693.1	236.7	430	1233.6	5
SD	500.646127	348.724039	327.807871	1289.37937	1.05409255
Best	205	5	20	160	4
Worst	1957	932	942	3879	7
Not found	0	0	0	0	0
Not found_percent average					

Furthermore, the bar chart below illustrates the comparison of the results from the three algorithms on the traveling salesman problem (15 cities). While the results of the Massive parallelization of the compact genetic algorithm (the blue bars) were better than the results of Cooperative Compact Genetic Algorithm (the orange bars) because the average number of iterations were lower, the average numbers of iterations in the results of the proposed algorithm (the grey bars) were lower than the results of the Massive parallelization of the compact genetic algorithm especially in the first and fourth problems which the differences of the results were more than 50 percent better, 1835.1 from table 4.3 and 693.1 from the table 4.14, and 5607.3 from the table 4.3 and 1233.6 from the table 4.14.

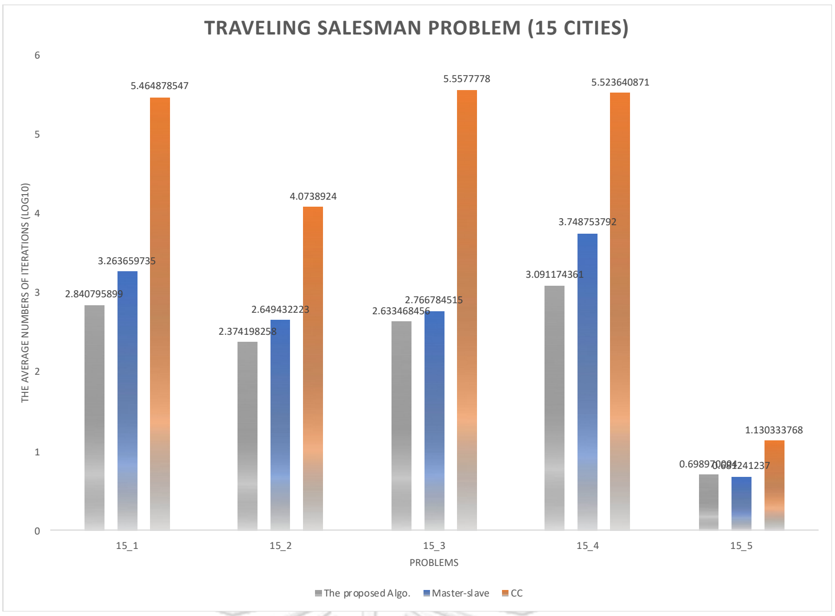


Figure 4.4: A bar chart shows the comparison of the results from the three algorithms on the traveling salesman problem (15 cities)

In the 17-cities problems, the proposed algorithm still could find the expected solutions for all problems as to the Massive parallelization of the compact genetic algorithm.

Table 4.15: The performance on the traveling salesman problem (17 cities) by using the proposed algorithm (the sharing knowledge compact genetic algorithm)

17 cities					
Proposed Algo.					
	17_1	**17_2**	**17_3**	**17_4**	**17_5**
sum_round 1	863	2126	2864	656	1125
sum_round 2	418	970	722	203	1732
sum_round 3	246	1441	1920	594	1011
sum_round 4	125	592	1626	657	1633
sum_round 5	1344	282	331	94	338
sum_round 6	969	1780	1129	74	973
sum_round 7	268	5083	948	544	1042
sum_round 8	547	1796	2064	149	746
sum_round 9	522	1288	864	117	702
sum_round 10	403	8637	2585	688	437
Average	570.5	2399.5	1505.3	377.6	973.9
SD	378.909913	2557.75206	841.064015	268.678742	454.174819
Best	125	282	331	74	338
Worst	1344	8637	2864	688	1732
Not found	0	0	0	0	0
Not found_percent average					

From the bar chart below that the comparison of the results from the three algorithms on the traveling salesman problem (17 cities), all results (the grey bars) were the best among the three algorithms. Although the result of the second problem (problem 17_2 in the Fig. 4.5) is not clearly better than the result from the Massive parallelization of the compact genetic algorithm, the average numbers of iterations were lower than 50 percent for the other problems.

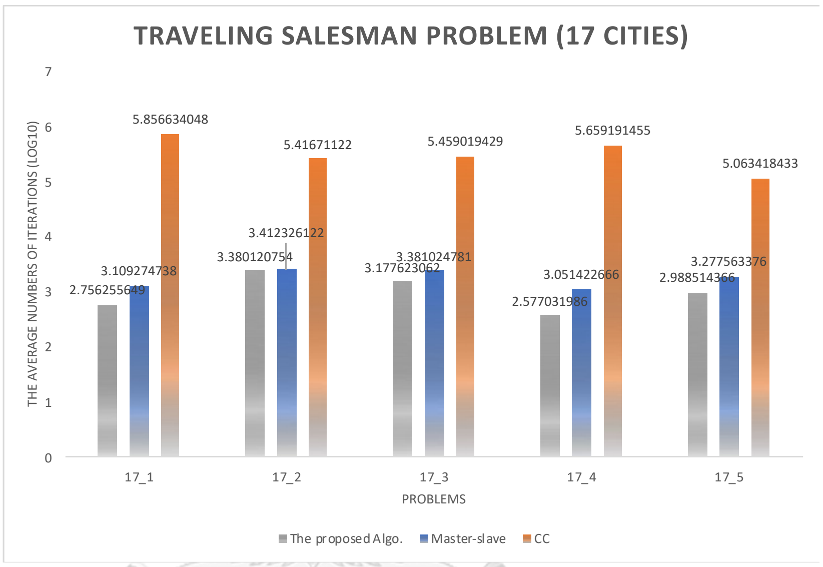
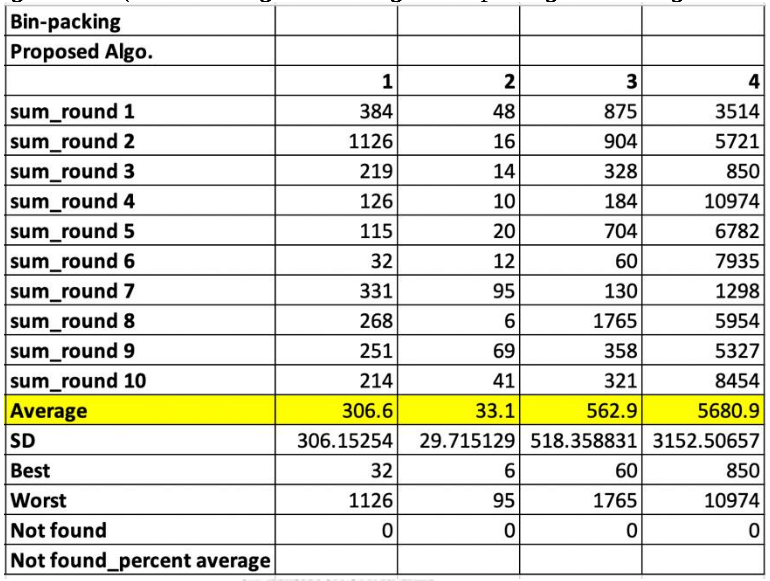


Figure 4.5: A bar chart shows the comparison of the results from the three algorithms on the traveling salesman problem (17 cities)

4.3.2. Bin Packing Problem

For the Bin Packing Problem, according to the table below, the proposed algorithm could discover the expected solutions within 200,000 iterations for all experiments.

Table 4.16: The performance on the bin-packing problem by using the proposed algorithm (the sharing knowledge compact genetic algorithm)



Bin-packing				
Proposed Algo.				
	1	**2**	**3**	**4**
sum_round 1	384	48	875	3514
sum_round 2	1126	16	904	5721
sum_round 3	219	14	328	850
sum_round 4	126	10	184	10974
sum_round 5	115	20	704	6782
sum_round 6	32	12	60	7935
sum_round 7	331	95	130	1298
sum_round 8	268	6	1765	5954
sum_round 9	251	69	358	5327
sum_round 10	214	41	321	8454
Average	306.6	33.1	562.9	5680.9
SD	306.15254	29.715129	518.358831	3152.50657
Best	32	6	60	850
Worst	1126	95	1765	10974
Not found	0	0	0	0
Not found_percent average				

Furthermore, The Fig. 4.6 the comparison of the results from the three algorithms on the bin-packing problem. The results from the proposed algorithm (the grey bars) were quite similar to the results from the other algorithms. The numbers of iterations executed were the lowest among the results from all algorithms except the result from the second problem (the problem number 2 in the Fig. 4.5) that the number of iterations was slightly higher than the result from the Cooperative Compact Genetic Algorithm. Moreover, the trend of the difficulty to solve the problem of the proposed algorithm was the same as the others' trends. The second problem was the easiest and the fourth one was the most difficult.

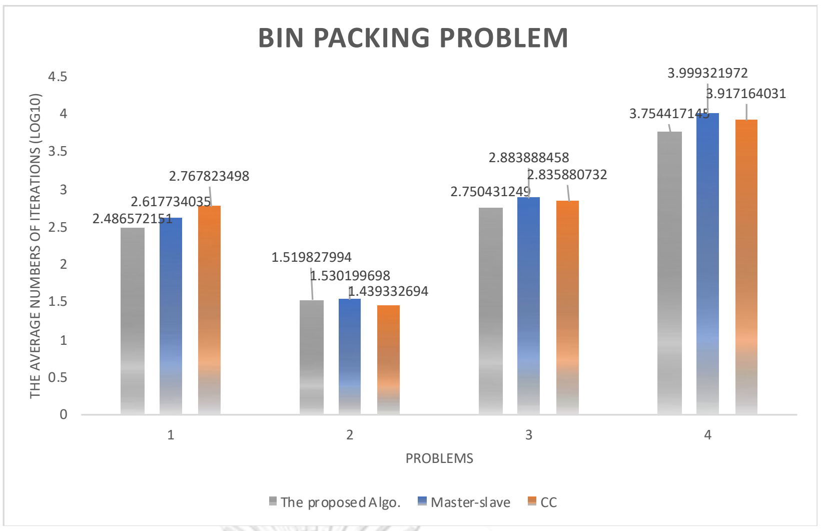
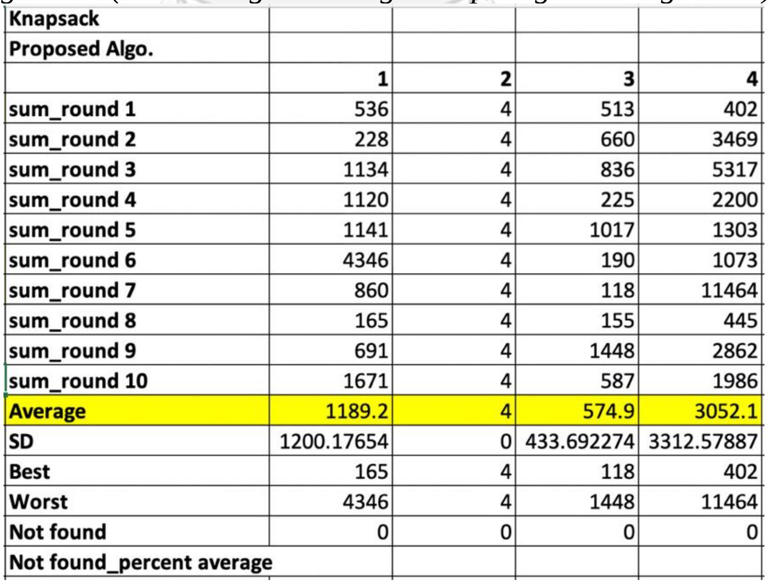


Figure 4.6: A bar chart shows the comparison of the results from the three algorithms on the bin-packing problem

4.3.3. Knapsack Problem

For the Knapsack problem, the proposed algorithm could find the expected solution for four problems. Like the other algorithms, the second problem was the easiest for the proposed algorithm. While the fourth problem was the hardest.

Table 4.17: The performance on the knapsack problem by using the proposed algorithm (the sharing knowledge compact genetic algorithm)



Knapsack				
Proposed Algo.				
	1	**2**	**3**	**4**
sum_round 1	536	4	513	402
sum_round 2	228	4	660	3469
sum_round 3	1134	4	836	5317
sum_round 4	1120	4	225	2200
sum_round 5	1141	4	1017	1303
sum_round 6	4346	4	190	1073
sum_round 7	860	4	118	11464
sum_round 8	165	4	155	445
sum_round 9	691	4	1448	2862
sum_round 10	1671	4	587	1986
Average	1189.2	4	574.9	3052.1
SD	1200.17654	0	433.692274	3312.57887
Best	165	4	118	402
Worst	4346	4	1448	11464
Not found	0	0	0	0
Not found_percent average				

For the first problem (see Fig. 4.7, the problem number 1), the result from the proposed algorithm (the blue bar) was better than the results from the Massive parallelization of the compact genetic algorithm (the orange bar) and the Cooperative Compact Genetic Algorithm (the gray bar). While the difference may not be clear between the results from the proposed algorithm and the Cooperative Compact Genetic Algorithm, it is noticeable. For the second problem (the problem number 2 in the Fig. 4.7), the performances from the three algorithms were similar. Unlike the others, the differences were large for the third fourth problems that the proposed algorithm could find the solutions, and its results were less than 5,000 iterations.

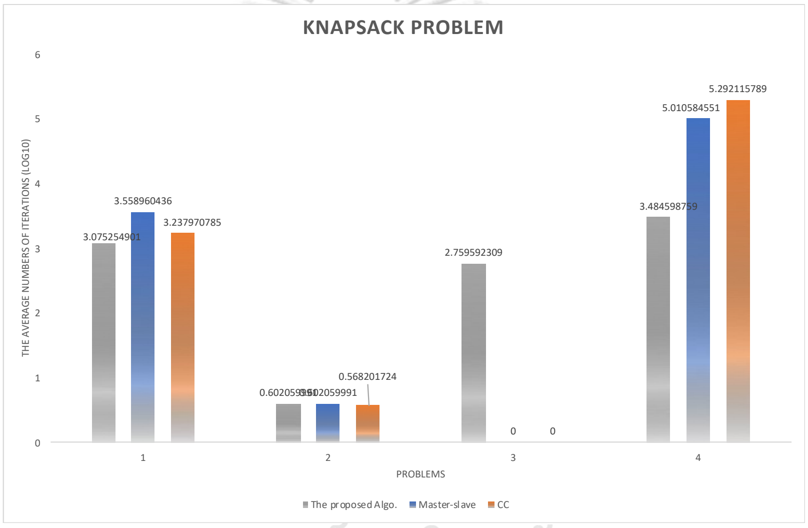
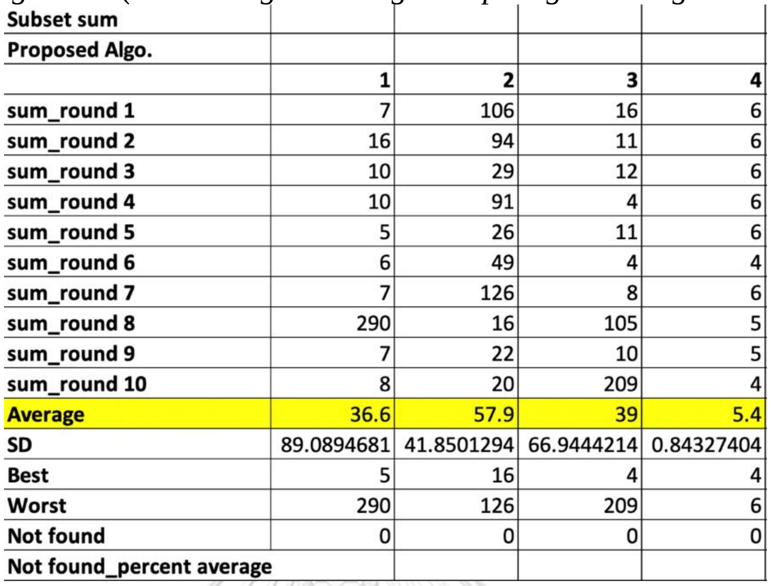


Figure 4.7: A bar chart shows the comparison of the results from the three algorithms on the knapsack problem

4.3.4. Subset Sum Problem

For the Subset Sum Problem, as the Fig. 4.8 that shows the comparison of the results from the three algorithms on the subset sum problem. The proposed algorithm (the grey bars) had the best results among the three algorithms. The average numbers of iterations for each problem were lower than 50 percent of the results from the Massive parallelization of the compact genetic algorithm (the blue bars) and the Cooperative Compact Genetic Algorithm (the orange bars) except the result of the fourth problem and the second problem that compared to the result from the Cooperative Compact Genetic Algorithm, 57.9 from the table 4.18 and 67.6 iterations, refer to the table 4.12.

Table 4.18: The performance on the subset sum problem by using the proposed algorithm (the sharing knowledge compact genetic algorithm)



Subset sum				
Proposed Algo.				
	1	**2**	**3**	**4**
sum_round 1	7	106	16	6
sum_round 2	16	94	11	6
sum_round 3	10	29	12	6
sum_round 4	10	91	4	6
sum_round 5	5	26	11	6
sum_round 6	6	49	4	4
sum_round 7	7	126	8	6
sum_round 8	290	16	105	5
sum_round 9	7	22	10	5
sum_round 10	8	20	209	4
Average	36.6	57.9	39	5.4
SD	89.0894681	41.8501294	66.9444214	0.84327404
Best	5	16	4	4
Worst	290	126	209	6
Not found	0	0	0	0
Not found_percent average				

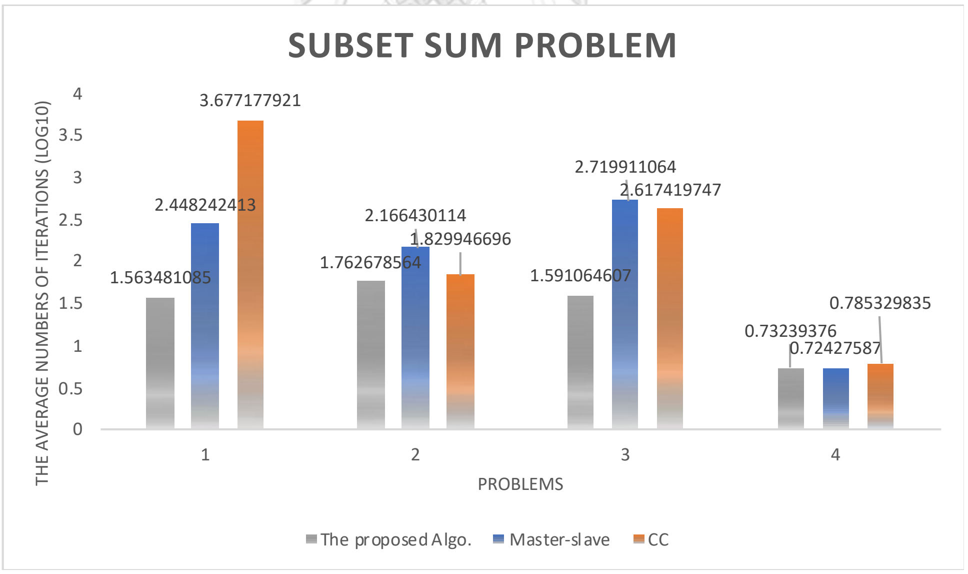


Figure 4.8: A bar chart shows the comparison of the results from the three algorithms on the subset sum problem

4.4. Results Summary

For Traveling Salesman Problem, the Cooperative Compact Genetic Algorithm could not find some expected solutions for some problems and had the worst result. While the Massive parallelization of the compact genetic algorithm and the proposed algorithm had similar results, the proposed algorithm was better in some problems. For example, the third and fourth problems of the 17-cities group, the result from the proposed algorithm were less than 50 percent of the results from the Massive parallelization of the compact genetic algorithm. For the Bin Packing Problem, all three algorithms had very similar results. Although the proposed algorithm had the best performance and the Cooperative Compact Genetic Algorithm was the second best, the difference in the results of all three algorithms were not distinct. For the Subset Sum Problem, the proposed algorithm had the best performance. The Cooperative Compact Genetic Algorithm had better results compared to the Massive parallelization of the compact genetic algorithm for the second and the third problems. For the fourth problem, the performance of the three algorithms were very close. To conclude, for small data set of Traveling Salesman Problem, Bin Packing Problem, Knapsack Problem, and Subset Sum Problem, the proposed algorithm was slightly better than the other two problems. Although the difference in the results from some problems were small, the proposed algorithm could find the expected solution of all test data and had good results. Thus, the proposed algorithm that employed the simulated annealing and the restart mechanism improves the overall performances in some cases such as the third and fourth problems of the Knapsack problem.

4.5. Analysis

The proposed algorithm employed the "Restart" mechanism to support escaping from local optima. Because the proposed algorithm developed from the Massive parallelization of the compact genetic algorithm by adding the restart mechanism and the simulated annealing, to show that the restart mechanism can decrease the average number of iterations executed, the number of half-restart and full-restart were analyzed. Table 1-6 in Appendix show the number of half-restart and full-restart on each instance of the problems for ten rounds. To illustrate the benefits of having the restart mechanism in the process, all instances of the problems could be divided into two groups by the average number of iterations from the Massive parallelization of the compact genetic algorithm from tables in phase II (the fourth and fifth problems of the traveling salesman problems (11 cities), the first and the fourth problems of the traveling salesman problems (15 cities), the fourth of the bin-packing problem, the first, third and fourth problems of the knapsack problems as hard group and the other problems as easy group). Overall, the restart mechanism worked effectively in the case that the average number of iterations executed of the Massive parallelization of the compact genetic algorithm were high such as the fourth problem of the traveling salesman problem(15 cities) and the third problem of the knapsack problem (hard group) according to Fig. 4.4 and 4.7 because the costs of the restart mechanism may have negative results in the case that traps in the problems were not difficult for the Massive parallelization of the compact genetic algorithm to escape such as the first to the third problems of the traveling salesman problem (11 cities) from Fig. 4.3. According to Fig. 4.4, 4.7 and 4.8, for the fifth problem of the traveling salesman problem(15 cities), the second of the knapsack problem and the fourth problem of the subset sum problem, there was no half-restart and full-restart occurred during the process so the average number of the iterations executed so the results of the Massive parallelization of the compact genetic algorithm and the proposed algorithm were very similar.

Although hard problems may not always have a high number of half-restart and full-restart, the number of half-restart and full-restart themselves can be employed to analyze the difficulty of the problems. The problems that had low numbers of half-restart and full-restart such as the first problem of traveling salesman problems (11 cities) from the table 1 in Appendix, the second problem of the bin-packing problem from the table 4 in Appendix and the second and third problems of the subset sum problem from the table 6 in Appendix also were easy to the Massive parallelization of the compact genetic algorithm (easy group) according to the results from the table 4.1, 4.7 and 4.11. However, the difficult problems may have to use the behaviors of the half-restart and full-restart to analyze.

The behaviors of the half restarts and full restarts on each problem were shown below in figure 4.9 to 4.14 but some problems did not appear below because there was no restart occurring during their execution. Overall, from Fig. 4.9 to 4.14, the half- restart usually occurred at the beginning of the execution. Then, the full-restart would start to occur in a later period when the half restart mechanism difficultly discovered better solutions. Thus, the number of full restarts might be higher than the number of half-restarts in the case that the traps in problems were very complicated and the half restart could not handle them. For example, from Fig. 4.13, for the fourth problem of the knapsack problem which the Massive parallelization of the compact genetic algorithm could not find the expected solution in some rounds, the half restarts occurred frequently at the early iterations, while the full restarts started to occur after that and continuously and intensively occurred until the expected solutions were discovered.

Moreover, the characteristic of traps of each problem also affects the behaviors of the restart. For the traveling salesman problem that the differences of its solutions (tour length) are high, the frequency of the half restart at the early iterations was not much because its traps were too complicated to find the very high-quality solutions in the early iterations. Thus, the half restart did not occur frequently in the early stages to escape from local optima. However, for the bin-packing problem, the feasible solutions are serial numbers so the half restart occurred more frequently due to the simulated annealing strategy and the chances to find the expected solutions by the half restart were higher. However, In the case that the very high-quality solution (the solutions next to the expected solution) was discovered and the number of iterations after finding the very high-quality solution was high, the half restart would stop and the full restart would take action and continuously occur until the expected solution was discovered according to Fig 4.12.

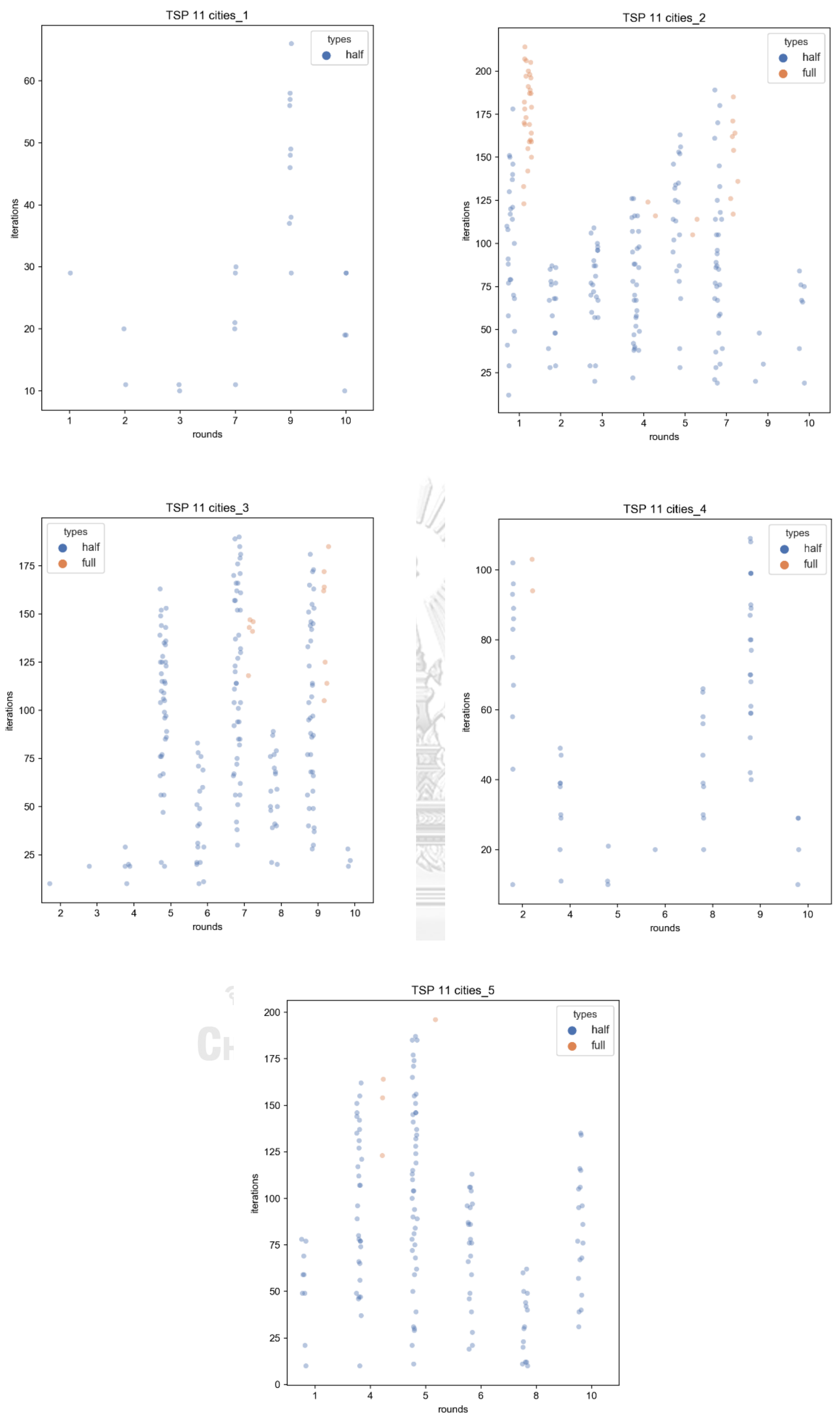


Figure 4.9: The scatter plot shows the behaviors of the half-restart and full-restart mechanism on the traveling salesman problem(11 cities) for ten rounds

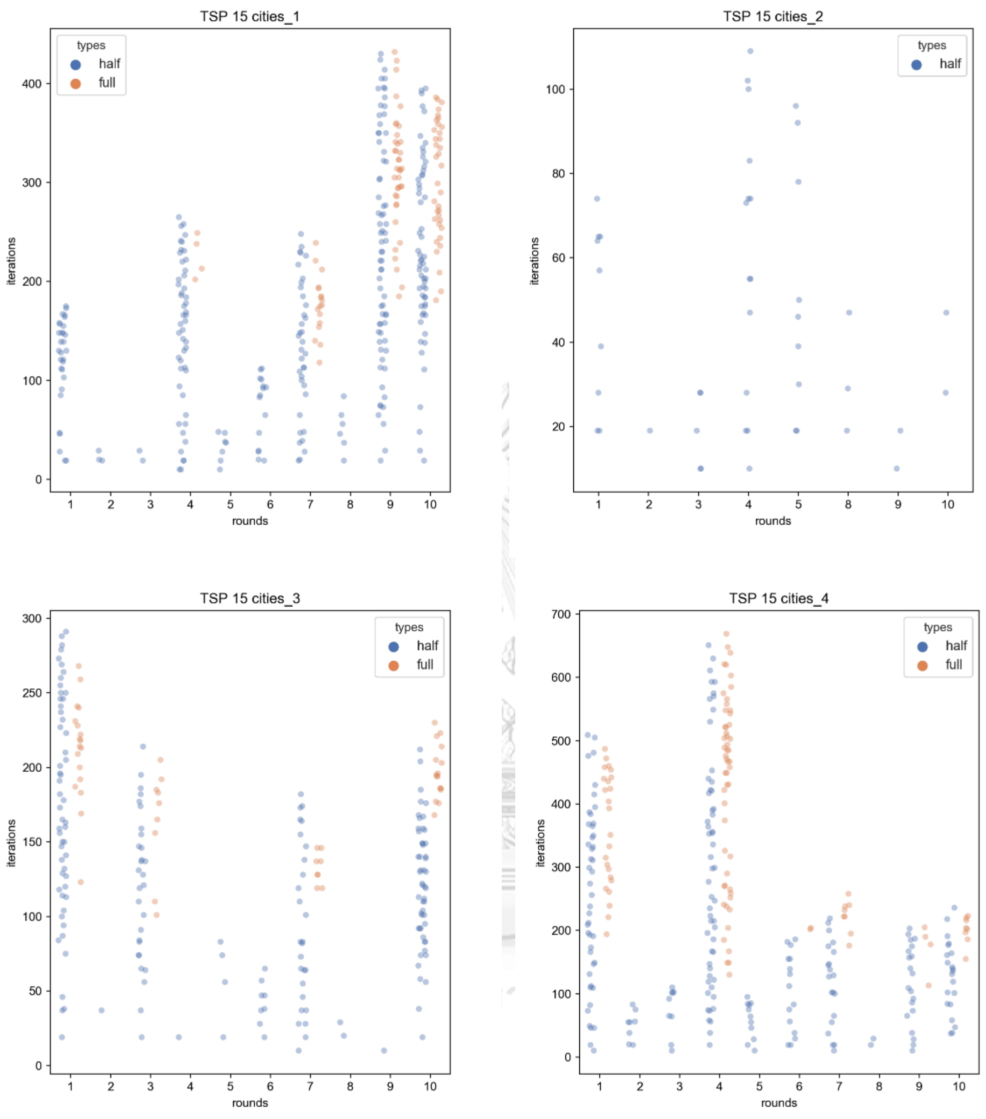


Figure 4.10: The scatter plot shows the behaviors of the half-restart and full-restart mechanism on the traveling salesman problem(15 cities) for ten rounds

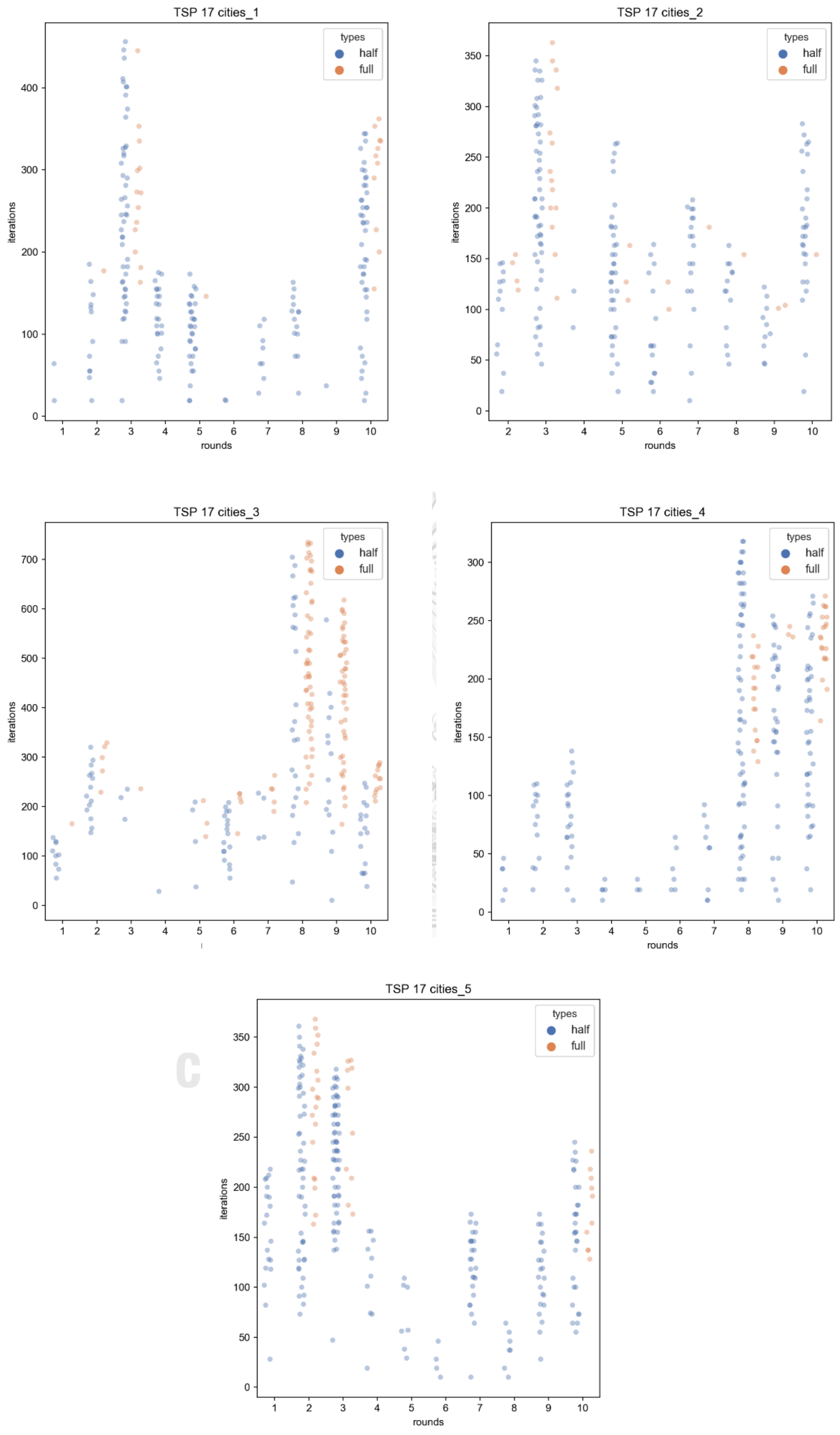


Figure 4.11: The scatter plot shows the behaviors of the half-restart and full-restart mechanism on the traveling salesman problem(17 cities) for ten rounds

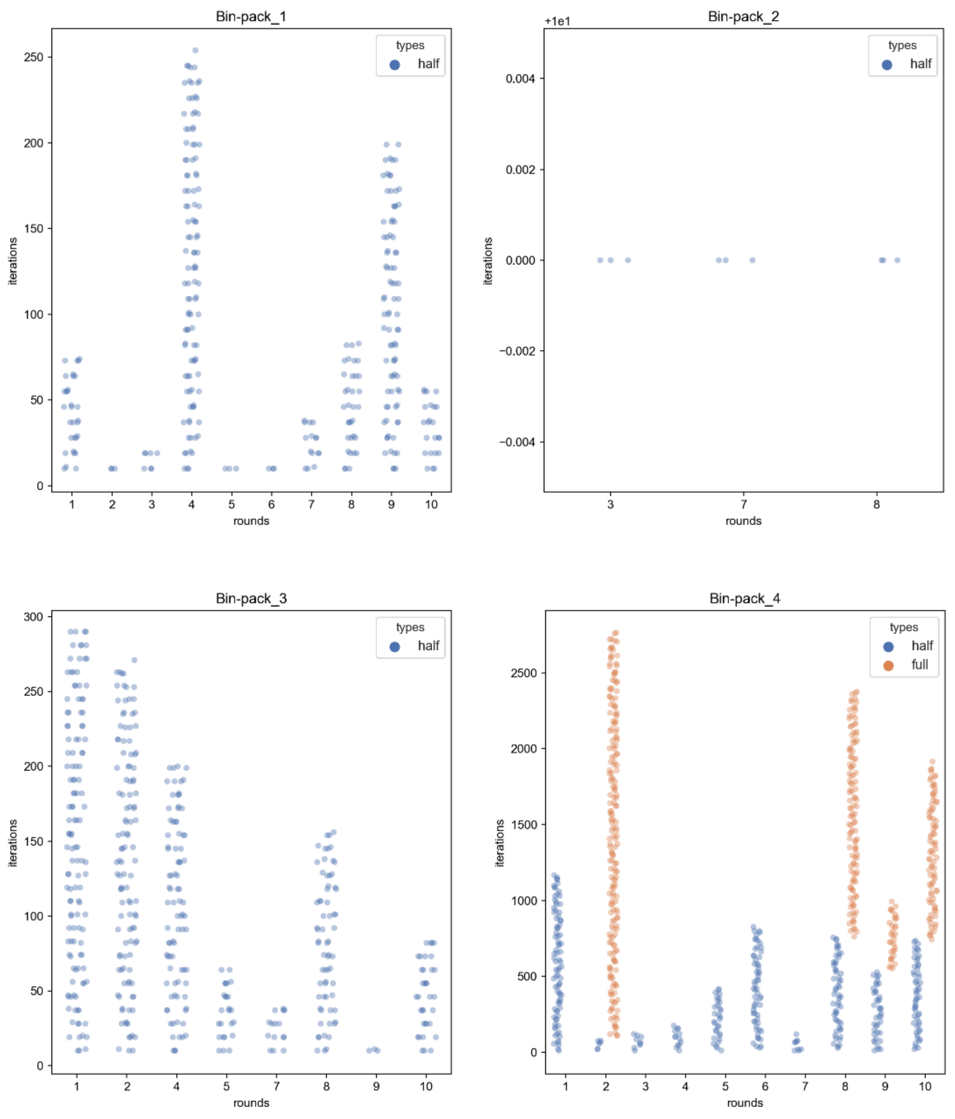


Figure 4.12: The scatter plot shows the behaviors of the half-restart and full-restart mechanism on the bin-packing problem for ten rounds

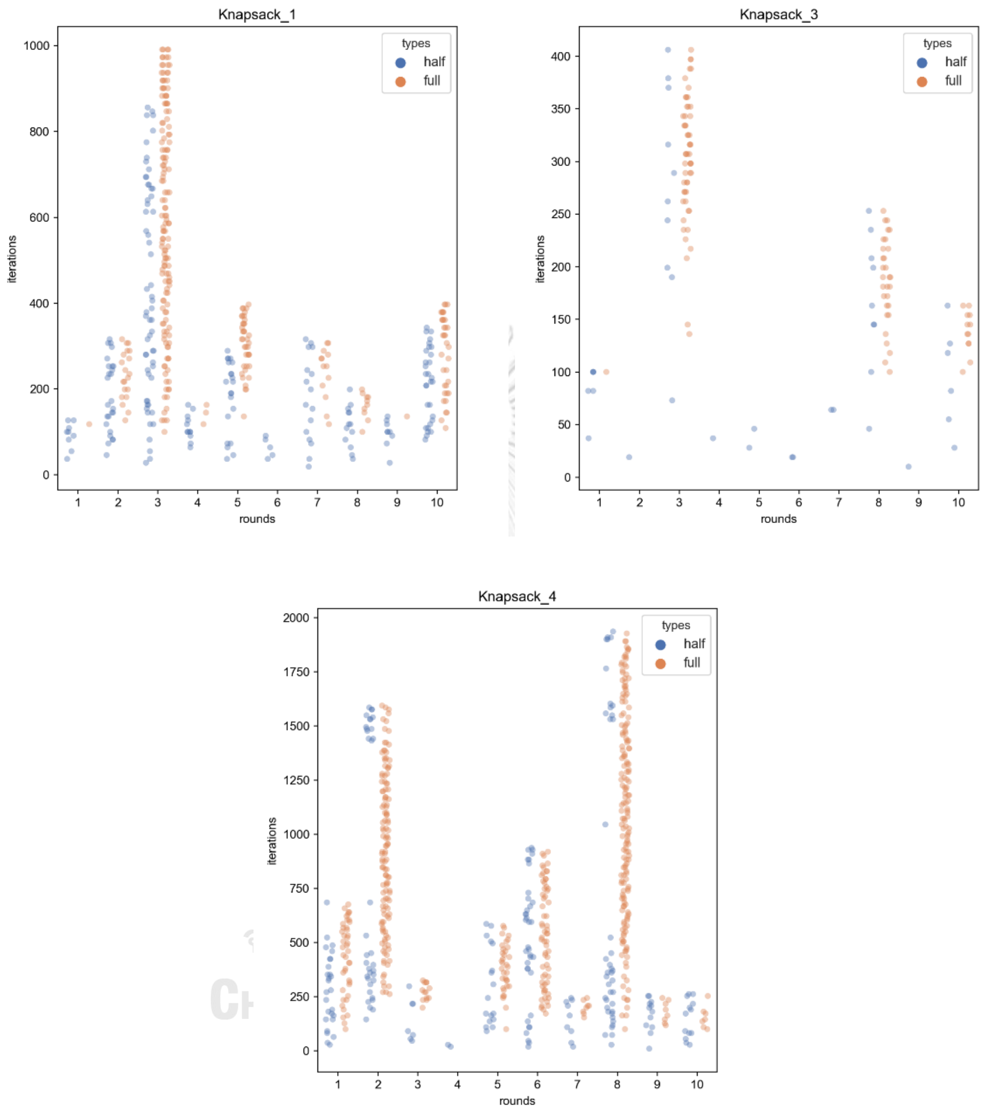


Figure 4.13: The scatter plot shows the behaviors of the half-restart and full-restart mechanism on the knapsack problem for ten rounds

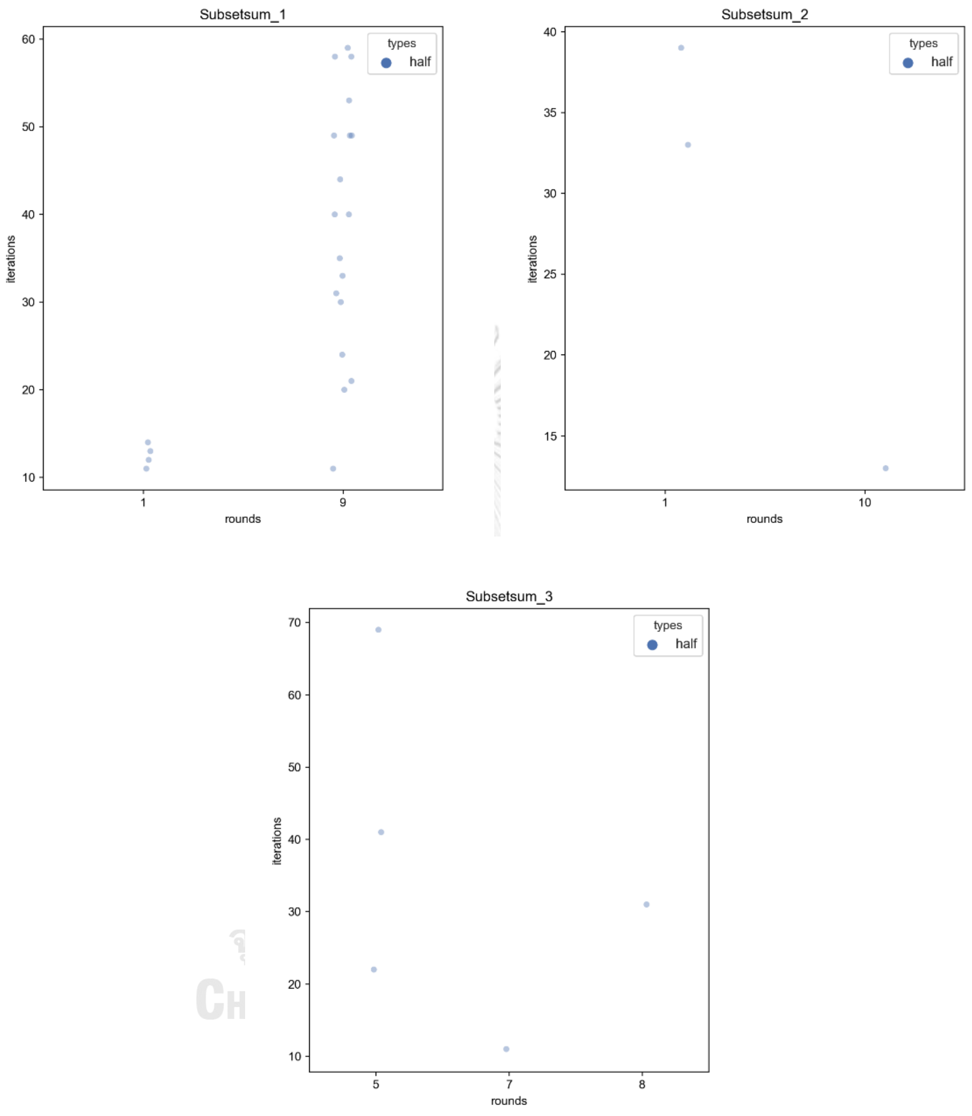


Figure 4.14: The scatter plot shows the behaviors of the half-restart and full-restart mechanism on the subset sum problem for ten rounds

Chapter 5 Conclusion

5.1. Conclusion

The compact genetic algorithm was an effective heuristic algorithm to solve hard problems. It still cannot prevent or avoid local optima. There are much research in search optimization attempting to escape from the local optima. Thus, the parallelization was used to improve the performance of the algorithm.

In this thesis, the research attempted to improve the performance of the existing parallel compact genetic algorithm by focusing on the shared parameters and sharing methods. The proposed algorithm exploited the property of the simulated annealing to consider the shared parameters whether they would be accepted or not. Moreover, the proposed algorithm used the "Restart" to help escape from local optima. In the case that the condition applying from the simulated annealing did not work, the restart was used. However, to prevent the high cost from rediscovering solution space, the probabilistic vector should be reinitialized by the average value. The observation of the results from the existing algorithm was employed to consider when it should restart the search.

In the experiment, the Massive parallelization of the compact genetic algorithm and the Cooperative Compact Genetic Algorithm were implemented and tested with Traveling salesman problem (11, 15, 17 cities), bin packing problem (50 items), knapsack problem (maximum of 200 items) and subset sum problem (maximum of 21 numbers). The proposed algorithm was designed, implemented and tested. The results of the experiment show that the proposed algorithm that employs the simulated annealing, the observation, and the restart can improve the performance. Although some results were not convincing due to the cost of the restart that may decrease the performance, there were no results from the proposed algorithm that was worse than the results from the other two algorithms.

However, there was some limitation of the proposed algorithm. Although the test data were varied, the input data of each test were not large. Thus, the observation was from the small group of the dataset. This parameter may not be suitable for large dataset. This challenge can be studied in the future.

Another point was, while the performance was effective, the number of shared variables of the proposed algorithm were higher than the other algorithms.

5.2. Future works

This research can be developed further by expanding the structure to more than four cGAs nodes to be able to exploit the benefits of topology and the observation number may be improved to be adaptive to each problem instead of using static value because each problem has different traps. The advantage of topology and the adaptive value for full restart may decrease the number of iterations executed to find the expected solutions. Moreover, the network issue should be concerned to support the scalable aspect of the practical works and the larger sizes of problems in the future.

REFERENCES

1. T. H. Cormen, C.E.L., R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition.*
2. Goldberg, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*. 1989: Addison-Wesley Longman Publishing Co., Inc. 372.
3. Holland, J.H., *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. 1992: MIT Press. 228.
4. Michalewicz, Z., *Genetic algorithms + data structures = evolution programs (3rd ed.)*. 1996: Springer-Verlag. 387.
5. Harik, G.R., F.G. Lobo, and D.E. Goldberg, *The compact genetic algorithm.* Trans. Evol. Comp, 1999. **3**(4): p. 287-297.
6. Balakrishnan, K. *Parallel Genetic Algorithms, Premature Convergence and the nCUBE*. 1993.
7. Matai, R., M.L. Mittal, and S. Singh, *Traveling Salesman Problem: an Overview of Applications, Formulations, and Solution Approaches*. 2010: INTECH Open Access Publisher.
8. Korf, R.E., *A new algorithm for optimal bin packing*, in *Eighteenth national conference on Artificial intelligence*. 2002, American Association for Artificial Intelligence: Edmonton, Alberta, Canada. p. 731-736.
9. Martello, S. and P. Toth, *Knapsack problems: algorithms and computer implementations*. 1990: John Wiley \\&amp; Sons, Inc. 296.
10. Caprara, A., H. Kellerer, and U. Pferschy, *The Multiple Subset Sum Problem.* SIAM J. on Optimization, 2000. **11**(2): p. 308-319.
11. R. K. Nayak, B.S.P.M., Jnyanaranjan Mohanty, *An overview of GA and PGA.* International Journal of Computer Applications, 2017. **176**.
12. Goldberg, E.C.-P.a.D.E., *Efficient Parallel Genetic Algorithms: Theory and Practice*. Computer Methods in Applied Mechanics and Engineering. 2000: press.
13. Cantu-Paz, E., *Efficient and Accurate Parallel Genetic Algorithms*. 2000: Kluwer Academic Publishers. 162.
14. Gold, M. *AI: Using the Compact Genetic Algorithm to Compute Square Roots in C#*. 2005.
15. Kirkpatrick, S., C.D. Gelatt, and M.P. Vecchi, *Optimization by simulated annealing.* science, 1983. **220**(4598): p. 671-680.
16. Černý, V., *Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm.* Journal of optimization theory and applications, 1985. **45**(1): p. 41-51.
17. Brownlee, J., *Clever algorithms: nature-inspired programming recipes*. 2011: Jason Brownlee.
18. *The Traveling Salesman Problem* Available from: https://www.csd.uoc.gr/~hy583/papers/ch11.pdf.
19. Junkermeier, J., *A Genetic Algorithm for the Bin Packing Problem.*
20. Maxence Delorme, M.I., Silvano Martello, *BPPLIB: a library for bin packing and cutting stock problems.* 2017.
21. GeeksforGeeks, *0-1 Knapsack Problem | DP-10.*

22. Ken-Li Li, G.-M.D., Qing-Hua Li, *A genetic algorithm for the unbounded knapsack problem*. Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.03EX693). Vol. 3. 2003.
23. Kellerer, H., U. Pferschy, and M.G. Speranza. *An efficient approximation scheme for the subset-sum problem*. 1997. Berlin, Heidelberg: Springer Berlin Heidelberg.
24. GeeksforGeeks. *Dynamic Programming - Subset Sum Problem*. Available from: https://www.geeksforgeeks.org/subset-sum-problem-dp-25/.
25. Hoos, H.H. and T. Stützle, *Stochastic local search: Foundations and applications*. 2004: Elsevier.
26. Fukunaga, A.S. *Restart scheduling for genetic algorithms*. in *International Conference on Parallel Problem Solving from Nature*. 1998. Springer.
27. Palmigiani, D. and G. Sebastiani, *A new restart procedure for combinatorial optimization and its convergence.* arXiv preprint arXiv:1709.06449, 2017.
28. Michalewicz, Z. and D.B. Fogel, *How to solve it: modern heuristics*. 2013: Springer Science & Business Media.
29. Baraglia, R., J.I. Hidalgo, and R. Perego, *A hybrid heuristic for the traveling salesman problem.* IEEE Transactions on evolutionary computation, 2001. **5**(6): p. 613-622.
30. Chu, P.C. and J.E. Beasley, *A genetic algorithm for the multidimensional knapsack problem.* Journal of heuristics, 1998. **4**(1): p. 63-86.
31. Wang, R.L., *A genetic algorithm for subset sum problem.* Neurocomputing, 2004. **57**: p. 463-468.
32. Jewajinda, Y. and P. Chongstitvatana. *A cooperative approach to compact genetic algorithm for evolvable hardware*. in *2006 IEEE International Conference on Evolutionary Computation*. 2006. IEEE.
33. Lobo, F.G., C.F. Lima, and H. Mártires. *Massive parallelization of the compact genetic algorithm*. 2005. Vienna: Springer Vienna.
34. Jong, K.A.D., *An analysis of the behavior of a class of genetic adaptive systems*. 1975, University of Michigan. p. 266.
35. Reinelt, G., *TSPLIB—A traveling salesman problem library.* ORSA journal on computing, 1991. **3**(4): p. 376-384.
36. Burkardt, J., *TSP Data for the Traveling Salesperson Problem.*
37. *Data for simple TSP*. Available from: https://stackoverflow.com/questions/11007355/data-for-simple-tsp.
38. Armin Scholl, R.K. *Data set 2 for BPP-1*. Available from: https://www2.wiwi.uni-jena.de/Entscheidung/binpp/bin2dat.htm.
39. Ruiz), J.A.O.R.J. *Instances of 0/1 Knapsack Problem*. Available from: http://artemisa.unicauca.edu.co/~johnyortega/instances_01_KP/.
40. Burkardt, J. *SUBSET_SUM Data for the Subset Sum Problem*. Available from: http://people.sc.fsu.edu/~jburkardt%20/datasets/subset_sum/subset_sum.html.

APPENDIX

Table 1: The total number of half and full restart on the traveling salesman problem(11 cities) for 10 rounds

TSP11_1			TSP11_2			TSP11_3			TSP11_4			TSP11_5		
	Half restart	Full restart		Half restart	Full restart		Half restart	Full restart		Half restart	Full restart		Half restart	Full restart
1	1	0	1	26	28	1	0	0	1	0	0	1	9	0
2	2	0	2	15	0	2	1	0	2	11	2	2	0	0
3	2	0	3	22	0	3	1	0	3	0	0	3	0	0
4	0	0	4	30	2	4	5	0	4	9	0	4	31	3
5	0	0	5	21	2	5	40	0	5	3	0	5	44	1
6	0	0	6	0	0	6	19	0	6	1	0	6	22	0
7	5	0	7	32	8	7	46	5	7	0	0	7	0	0
8	0	0	8	0	0	8	18	0	8	10	0	8	15	0
9	10	0	9	3	0	9	39	7	9	20	0	9	0	0
10	5	0	10	7	0	10	3	0	10	4	0	10	18	0

Table 2: The total number of half and full restart on the traveling salesman problem(15 cities) for 10 rounds

TSP15_1			TSP15_2			TSP15_3			TSP15_4			TSP15_5		
	Half restart	Full restart		Half restart	Full restart		Half restart	Full restart		Half restart	Full restart		Half restart	Full restart
1	30	0	1	9	0	1	54	18	1	88	50	1	0	0
2	3	0	2	1	0	2	1	0	2	15	0	2	0	0
3	2	0	3	5	0	3	29	9	3	18	1	3	0	0
4	49	4	4	14	0	4	1	0	4	110	106	4	0	0
5	7	0	5	9	0	5	4	0	5	22	0	5	0	0
6	15	0	6	0	0	6	8	0	6	30	4	6	0	0
7	36	17	7	0	0	7	27	8	7	38	18	7	0	0
8	6	0	8	3	0	8	2	0	8	4	0	8	0	0
9	71	38	9	2	0	9	1	0	9	40	7	9	0	0
10	59	36	10	2	0	10	51	16	10	43	15	10	0	0

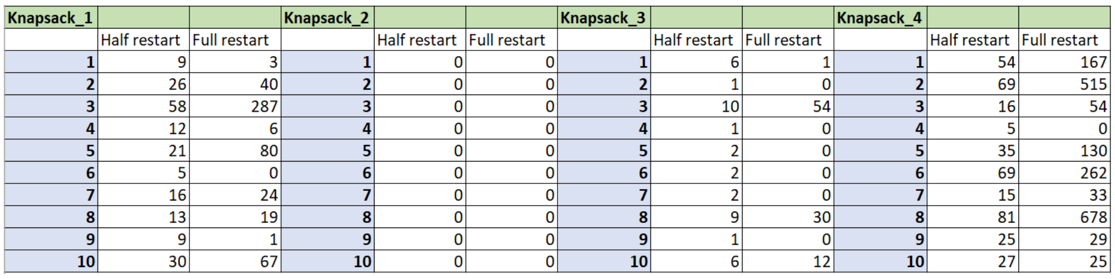
Table 3: The total number of half and full restart on the traveling salesman problem(17 cities) for 10 rounds

TSP17_1			TSP17_2			TSP17_3			TSP17_4			TSP17_5		
	Half restart	Full restart		Half restart	Full restart		Half restart	Full restart		Half restart	Full restart		Half restart	Full restart
1	2	0	1	0	0	1	26	2	1	5	0	1	19	0
2	12	2	2	12	7	2	45	13	2	13	0	2	51	19
3	48	25	3	52	24	3	10	1	3	20	0	3	59	10
4	22	0	4	2	0	4	5	0	4	5	0	4	10	0
5	30	2	5	37	5	5	10	10	5	3	0	5	7	0
6	2	0	6	15	4	6	49	16	6	6	0	6	4	0
7	8	1	7	19	2	7	12	15	7	9	0	7	25	0
8	14	1	8	13	2	8	71	151	8	62	17	8	7	0
9	1	0	9	10	3	9	44	130	9	33	3	9	22	0
10	43	23	10	27	1	10	46	38	10	40	20	10	29	10

Table 4: The total number of half and full restart on the bin-packing problem for 10 rounds

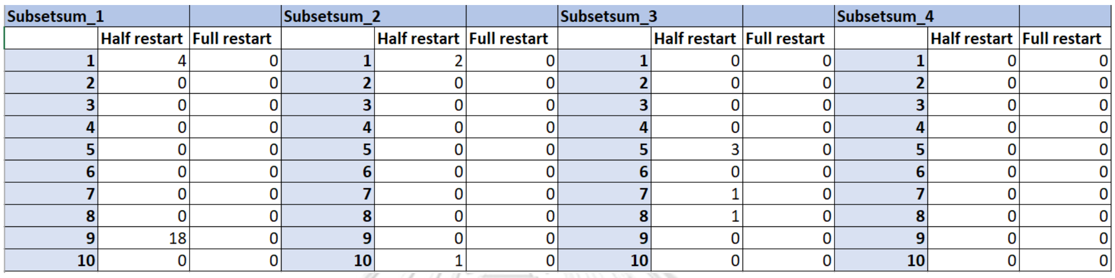
Bin-pack_1			Bin-pack_2			Bin-pack_3			Bin-pack_4		
	Half restart	Full restart		Half restart	Full restart		Half restart	Full restart		Half restart	Full restart
1	31	0	1	0	0	1	128	0	1	515	0
2	3	0	2	0	0	2	116	0	2	45	1175
3	7	0	3	3	0	3	0	0	3	55	0
4	109	0	4	0	0	4	87	0	4	81	0
5	3	0	5	0	0	5	25	0	5	186	0
6	3	0	6	0	0	6	0	0	6	360	0
7	15	0	7	3	0	7	15	0	7	50	0
8	34	0	8	3	0	8	66	0	8	326	728
9	85	0	9	0	0	9	3	0	9	237	201
10	23	0	10	0	0	10	35	0	10	325	510

Table 5: The total number of half and full restart on the knapsack problem for 10 rounds



Knapsack_1			Knapsack_2			Knapsack_3			Knapsack_4		
	Half restart	Full restart		Half restart	Full restart		Half restart	Full restart		Half restart	Full restart
1	9	3	1	0	0	1	6	1	1	54	167
2	26	40	2	0	0	2	1	0	2	69	515
3	58	287	3	0	0	3	10	54	3	16	54
4	12	6	4	0	0	4	1	0	4	5	0
5	21	80	5	0	0	5	2	0	5	35	130
6	5	0	6	0	0	6	2	0	6	69	262
7	16	24	7	0	0	7	2	0	7	15	33
8	13	19	8	0	0	8	9	30	8	81	678
9	9	1	9	0	0	9	1	0	9	25	29
10	30	67	10	0	0	10	6	12	10	27	25

Table 6: The total number of half and full restart on the subset sum problem for 10 rounds



Subsetsum_1			Subsetsum_2			Subsetsum_3			Subsetsum_4		
	Half restart	**Full restart**		**Half restart**	**Full restart**		**Half restart**	**Full restart**		**Half restart**	**Full restart**
1	4	0	1	2	0	1	0	0	1	0	0
2	0	0	2	0	0	2	0	0	2	0	0
3	0	0	3	0	0	3	0	0	3	0	0
4	0	0	4	0	0	4	0	0	4	0	0
5	0	0	5	0	0	5	3	0	5	0	0
6	0	0	6	0	0	6	0	0	6	0	0
7	0	0	7	0	0	7	1	0	7	0	0
8	0	0	8	0	0	8	1	0	8	0	0
9	18	0	9	0	0	9	0	0	9	0	0
10	0	0	10	1	0	10	0	0	10	0	0

VITA

NAME	Orakanya Gateratanakul
DATE OF BIRTH	25 August 1994
PLACE OF BIRTH	Bangkok
HOME ADDRESS	150/28-29 Boonyadis, Lanluang, Klongmahanak, Pomprab, Bangkok, 10100
PUBLICATION	Gateratanakul, O., & Chongstitvatana, P. (2018, April). Knowledge Sharing in Cooperative Compact Genetic Algorithm. In 2018 3rd International Conference on Computer and Communication Systems (ICCCS) (pp. 25-28). IEEE.