Genetic Learning Particle Swarm Optimization for Task Matching in Grid Environment

by

Eid Albalawi

A thesis submitted to
The Faculty of Graduate Studies of
The University of Manitoba
in partial fulfillment of the requirements
of the degree of

Doctor of Philosophy

Department of Computer Science
The University of Manitoba
Winnipeg, Manitoba, Canada
March 2020

© Copyright 2020 by Eid Albalawi
Abstract

Many scientific disciplines require processing computational-intensive applications with vast quantities of data. Large scale systems, such as Grid, provide shared resources on geographically distributed heterogeneous computing systems to meet the computational demands of the applications. One of the fundamental problems computing with the Grid system is assigning tasks of various users to the resources efficiently such that the execution time of the application is minimized while Grid resources are utilized optimally. This is called the task matching problem, an NP-hard problem, the focus of this thesis. While there are many heuristics for the task matching problem, particle swarm optimization (PSO) has been one of the latest heuristics in the literature adopted as a solution to this problem. However, PSO has few drawbacks, such as premature convergence to local optima. To circumvent these problems, we incorporate a learning approach inspired by the genetic algorithm to guide the PSO algorithm. We propose three novel genetic learning PSO algorithms: modified genetic learning PSO (MGLPSO), hybrid genetic learning PSO (HGLPSO), and modified genetic learning PSO with adaptive mutation ($\mu$-MGLPSO). The proposed algorithms
have been designed for both single and multi-objective task matching problem for tasks with and without dependencies. The proposed techniques are evaluated on standard benchmarks with large data sets and compared against state-of-the-art algorithms.
# Contents

Abstract ................................................................. ii  
Table of Contents ....................................................... vii  
List of Figures ........................................................... viii  
List of Tables ............................................................ x  
Acknowledgments ........................................................ xi  
Dedication ................................................................. xiii

## 1 Introduction
1.1 Grid Computing ..................................................... 1  
1.2 Grid Scheduling ..................................................... 5  
1.3 Task Matching Problem ............................................. 7  
1.4 Motivation ........................................................... 8  
1.5 Contributions ....................................................... 9  
1.6 Thesis Overview ................................................... 10

## 2 Meta-heuristic Approaches for Optimization Problems
2.1 Optimization Problem .............................................. 12  
2.2 Meta-heuristic Approach ........................................... 20  
2.3 Genetic Algorithm .................................................. 21  
2.4 Particle Swarm Optimization ...................................... 26  
2.5 Summary ............................................................ 35
## 3 Literature Review

3.1 Introduction ........................................... 37
3.2 Heuristic based approaches ............................... 38
   3.2.1 Task independent ........................................ 39
   3.2.2 Task dependent (workflow) ............................ 40
3.3 Meta-heuristic based approaches ......................... 45
   3.3.1 Genetic Algorithm (GA) ............................... 45
   3.3.2 Particle Swarm Optimization ......................... 50
3.4 Enhancing PSO algorithm ................................. 53
   3.4.1 Hybrid Approach ....................................... 53
   3.4.2 Learning Approach ..................................... 55
3.5 Summary .............................................. 59

## 4 Modified Genetic Learning Particle Swarm Optimization for Task Matching Problem

4.1 Drawbacks of the Existing Canonical PSO ............... 62
4.2 Behavioral Genetics in Biology .......................... 63
4.3 Modified Genetic Learning PSO (MGLPSO) ............... 65
   4.3.1 Task Matching Particle Solution Representation .... 68
   4.3.2 Phase 1: Genetic Learning .............................. 68
   4.3.3 Phase 2: PSO Algorithm ............................... 71
   4.3.4 Fitness Function ....................................... 73
   4.3.5 Complexity Analysis of the MGLPSO Algorithm .... 74
4.4 Experimental Design ..................................... 74
4.5 Results and Discussion ................................. 76
   4.5.1 Makespan ............................................. 77
   4.5.2 Convergence Rate ...................................... 79
   4.5.3 Exemplar Quality ...................................... 80
   4.5.4 Diversity ............................................. 82
   4.5.5 Mutation Rate ........................................ 86
   4.5.6 Population Size ...................................... 89
   4.5.7 Tournament Size ..................................... 90
4.6 Summary .............................................. 92

## 5 Genetic Learning PSO Algorithms for Single and Multi-objective Task Matching Problem

5.1 Hybrid Genetic Learning PSO (HGLPSO) ............... 95
5.2 Modified Genetic learning PSO with Adaptive Mutation ($\mu$-MGLPSO) ............... 97
5.3 Multi-objective Task Matching Problem .................. 100
   5.3.1 Multi-objective evolutionary algorithms (MOEAs) .... 103
5.4 Genetic Learning PSO Algorithms for Multi-objective Task Matching Problem ............... 105
5.4.1 Objective Function ........................................ 108
5.4.2 Multi-objective MGLPSO (MO-MGLPSO) algorithm .... 109
5.4.3 Multi-objective HGLPSO (MO-HGLPSO) algorithm ..... 112
5.4.4 Multi-objective $\mu$-MGLPSO (MO-$\mu$-MGLPSO) algorithm ... 118
5.5 Experimental Design ........................................ 119
5.5.1 Experimental Design for Single Objective task matching problem 122
5.5.2 Experimental Design for Multi-Objective Task Matching Problem 122
5.6 Results and Discussion for Single Objective Task Matching Problem .... 122
5.6.1 Comparison with PSO Variants for Large-scale Task Matching Problem ........................................ 123
5.6.2 Convergence Rate ........................................ 132
5.6.3 Effect of Sub-population Size ................................ 134
5.7 Results and Discussion for Multi-objective Task Matching Problem ... 136
5.7.1 Comparison with MOEA Variants for Large-scale Multi-objective Task Matching Problem ........................................ 138
5.7.2 Quality of Non-dominated Solutions .......................... 144
5.7.3 Convergence Rate ........................................ 146
5.7.4 Cardinality ........................................ 150
5.8 Discussion ........................................ 152
5.9 Summary ........................................ 154

6 Multi-objective Task Matching for Workflow Applications 156
6.1 Introduction ........................................ 157
6.2 Workflow Model and Preliminaries ................................ 159
6.2.1 Workflow Model ........................................ 160
6.2.2 Directed Acyclic Graph (DAG) ................................ 161
6.2.3 Resource Model ........................................ 163
6.2.4 Problem Definition: Workflow Task Matching on Grid ........................................ 163
6.2.5 Workflow Task Matching Example ................................ 167
6.3 Workflow-based Multi-objective Genetic Learning PSO .................. 170
6.3.1 Task Priority ........................................ 174
6.3.2 Match Generation ........................................ 175
6.3.3 Objective function ........................................ 177
6.4 Experimental Design ........................................ 179
6.4.1 Real-world Workflow Applications .......................... 181
6.4.2 Quality Indicator for MOP .................................. 183
6.5 Results and Discussion ........................................ 185
6.5.1 Comparison with MOEAs Variants for Extremely Large-scale Workflows ........................................ 185
6.5.2 Convergence Rate ........................................ 191
6.5.3 Quality of Non-dominated Solutions .......................... 194
6.5.4 Discussion ........................................ 195
6.6 Summary ......................................................... 197

7 Conclusions and Future Work ......................................... 199
  7.1 Future Work .................................................. 201

Bibliography ............................................................. 250
List of Figures

1.1 A Grid system with different VOs. ........................................ 3
1.2 Scheduling process on Grid environment. ............................ 6
1.3 Thesis organization. .......................................................... 11

2.1 Local and global minimum of $f(x)$. ................................. 14
2.2 Optimization problem modality. ......................................... 16
2.3 An example of minimization problem with two objectives. ....... 19
2.4 Crossover and mutation operations in GA. ............................ 23
2.5 The single-point crossover. ................................................. 25
2.6 The two-point crossover. .................................................. 25
2.7 Particle’s movement in the PSO algorithm. .......................... 29
2.8 Swarm topologies. ........................................................... 33

3.1 The taxonomy of task matching in grid systems. .................... 38

4.1 Oscillation phenomenon in PSO algorithm .......................... 64
4.2 Flowchart of the MGLPSO algorithm. .................................. 67
4.3 Particle Representation ..................................................... 68
4.4 Crossover example. ......................................................... 69
4.5 Mean makespan for PSO, GLPSO, and MGLPSO. .................. 78
4.6 Convergence rate of PSO, GLPSO, and MGLPSO. .................. 79
4.7 Exemplar’s quality of MGLPSO, and GLPSO in the inconsistent dataset. ........................................ 80
4.8 Exemplar quality of MGLPSO, and GLPSO in the Partially-consistent dataset. ........................................ 82
4.9 Exemplar’s quality of MGLPSO, and GLPSO in the consistent dataset. ........................................ 83
4.10 Swarm’s diversity of MGLPSO, GLPSO, and PSO for the inconsistent dataset. ........................................ 84
4.11 Swarm’s diversity of MGLPSO, GLPSO, and PSO for the Partially-consistent. ........................................ 85
4.12 Swarm’s diversity of MGLPSO, GLPSO, and PSO for consistent dataset. ........................................ 87
4.13 Effect of mutation rate on different datasets. ....................... 89
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.14</td>
<td>Effect of the population size on the makespan.</td>
<td>90</td>
</tr>
<tr>
<td>4.15</td>
<td>Effect of the tournament size on the makespan.</td>
<td>92</td>
</tr>
<tr>
<td>5.1</td>
<td>Different Pareto frontiers.</td>
<td>106</td>
</tr>
<tr>
<td>5.2</td>
<td>Bar plots of the mean makespan for all data sets.</td>
<td>128</td>
</tr>
<tr>
<td>5.3</td>
<td>Box plot of the mean makespan for all data sets.</td>
<td>129</td>
</tr>
<tr>
<td>5.4</td>
<td>Convergence rate of PSO variants for all data sets.</td>
<td>134</td>
</tr>
<tr>
<td>5.5</td>
<td>Mean makespan of different data sets for various sizes of subgroups.</td>
<td>135</td>
</tr>
<tr>
<td>5.6</td>
<td>Total mean makespan for different sizes of subgroups.</td>
<td>136</td>
</tr>
<tr>
<td>5.7</td>
<td>Hypervolume indicator.</td>
<td>137</td>
</tr>
<tr>
<td>5.8</td>
<td>Box plot of the HV for all data sets.</td>
<td>141</td>
</tr>
<tr>
<td>5.9</td>
<td>Quality of non-dominated solutions for the consistent data set.</td>
<td>145</td>
</tr>
<tr>
<td>5.10</td>
<td>Quality of non-dominated solution for the inconsistent data set.</td>
<td>147</td>
</tr>
<tr>
<td>5.11</td>
<td>Quality of non-dominated solution for the partially-consistent data set.</td>
<td>148</td>
</tr>
<tr>
<td>5.12</td>
<td>Convergence rate for all data sets.</td>
<td>149</td>
</tr>
<tr>
<td>5.13</td>
<td>Number of solutions for all data sets.</td>
<td>151</td>
</tr>
<tr>
<td>6.1</td>
<td>A high-level overview of the workflow execution in the grid environment.</td>
<td>160</td>
</tr>
<tr>
<td>6.2</td>
<td>Workflow example.</td>
<td>162</td>
</tr>
<tr>
<td>6.3</td>
<td>Task matching example.</td>
<td>170</td>
</tr>
<tr>
<td>6.4</td>
<td>Task partitioning of workflow in Figure 6.2.</td>
<td>176</td>
</tr>
<tr>
<td>6.5</td>
<td>Structure of real-world workflows.</td>
<td>182</td>
</tr>
<tr>
<td>6.6</td>
<td>Illustration of IGD indicator.</td>
<td>184</td>
</tr>
<tr>
<td>6.7</td>
<td>Box plot of the HV for the real-world workflows.</td>
<td>188</td>
</tr>
<tr>
<td>6.8</td>
<td>Box plot of the IGD for the real-world workflows.</td>
<td>189</td>
</tr>
<tr>
<td>6.9</td>
<td>Convergence rate of the HV for the real-world workflows.</td>
<td>192</td>
</tr>
<tr>
<td>6.10</td>
<td>Convergence rate of the IGD for the real-world workflows.</td>
<td>192</td>
</tr>
<tr>
<td>6.11</td>
<td>Solutions quality of the real-world workflows.</td>
<td>194</td>
</tr>
</tbody>
</table>
List of Tables

1.1 ETC matrix example. .............................................. 8
3.1 Pros and cons of different approaches. .......................... 55
4.1 MGLPSO Parameter settings. ........................................ 76
4.2 Statistical results of PSO, GLPSO and MGLPSO. .................. 77
5.1 Parameter settings. .................................................. 124
5.2 Statistical results of PSO variants on large data size. .......... 127
5.3 Wilcoxon signed rank test of the makespan obtained by HGLPSO for the consistent data set. ................................................ 132
5.4 Wilcoxon signed rank test of the makespan obtained by MGLPSO for the inconsistent data set. ......................................... 132
5.5 Wilcoxon signed rank test of the makespan obtained by $\mu$-MGLPSO for the partially-consistent data set. ................................. 133
5.6 Parameter settings. .................................................. 138
5.7 Statistical results of MOEA variants. ............................... 139
5.8 Friedman rank test for the consistent problem. ..................... 142
5.9 Friedman rank test for the inconsistent problem. ................. 142
5.10 Friedman rank test for the partially-consistent problem. ...... 143
6.1 Terms and notations used in workflow task matching problem. ........ 165
6.2 Sample execution time of all tasks on different resources based on Figure 6.2. ................................................ 168
6.3 Sample data transfer time based on Figure 6.2. ...................... 168
6.4 Cost of the resources. .............................................. 180
6.5 Transfer bandwidth between resources. ............................. 180
6.6 Statistical results of MOEA variants. .............................. 185
6.7 Friedman rank test of the MOEAs for HV metric. ................. 190
6.8 Friedman rank test of the MOEAs for IGD metric. ................ 191
Acknowledgments

First and foremost, O Allah, for you is praise until it pleases you, and for you is praise when You become pleased, and for you is praise after You have become pleased. O Allah, all praise is due only to you, praise which is abundant, much good, and blessed. O Allah, thank you for granting me the strength and patience to complete this research regardless of many impediments. O Allah, without your blessing, this work would not have been completed.

I would also like to take this opportunity to express my deepest and sincere gratitude to my supervisor, Prof. Parimala Thulasiraman, for her inexhaustible guidance and motivation throughout my research process. I am thankful for her immense knowledge that enriched my knowledge and ignited my interest in the field of evolutionary computation. I am also greatly indebted to my co-supervisor, Prof. Ruppa Thulasiram (Tulsi), for his ceaseless insightful advice and encouragement that helped me improve my research.

I would like to thank my committee members, Dr. Noman Mohammed and Dr. Yuvraj Gajpal, for serving my thesis and candidacy committees, and for their remarkable feedback and invaluable input into my thesis. My sincere thanks go to Dr. Amiya Nayak for allotting time to read and assist my thesis.

Heartfelt thanks to many friends and people with whom I have shared unforgettable memories in Canada. I am much obliged to the Saudi cultural bureau in Canada and King Faisal University for the full scholarship and support during my studies abroad.

I also extend my gratitude to my mother and siblings for their unconditional and loving support. I am truly fortunate to have such a gorgeous family. Finally, to my
wife, Eman, thanks for standing by side in this stressful yet rewarding PhD journey. Thank you for the endless support and understanding during the difficult times. After almighty Allah, a significant portion of my success is because of you. Words will never express how much I am grateful to you. To my children, Maya and Malik, thank you for being the source of joy and exhilaration in my life. I hope one day, this thesis motivates you to achieve your future endeavors.
This thesis is dedicated to the memory of my father, Mohammad albalawi, the one who instilled the love and enthusiasm of learning and knowledge acquisition in me. I miss you every day, and I know you would be proud of me as you used to tell me. May Allah grant you a peaceful rest in your grave, Ameen. Also, I dedicate this thesis to my beloved mother, Aziza albalawi, thanks for everything you have done for me. You are the source of support and encouragement in my life. I would not be the person I am today without your countless prayers for me. I will be forever thankful to you.
Chapter 1

Introduction

This chapter starts by providing a high-level overview of Grid computing. This is followed by a description of the task matching problem and the motivation for proposing meta-heuristic algorithms for the task matching problem. The chapter ends with identifying the contributions and the organization of the thesis.

1.1 Grid Computing

Many applications in scientific disciplines require processing of very large quantities of computationally intensive data. To meet the computation demand of these applications, large-scale systems are required. Such systems, e.g., Cloud and Grid, have emerged as important architectures due to their ability to solve large complex heterogeneous problems in less time than that is taken by a traditional single computer. A Cloud system provides procurement of on-demand services, such as infrastructure, platform and software (application), in a pay-as-you-go model such as the Amazon
Chapter 1: Introduction

Cloud. The advancement of cloud technology enables its provider to deliver highly scalable virtualized resources [249]. Grid, on the other hand, aggregates geographically distributed heterogeneous computing resources [95] to provide shared resources such as hardware, software and other services to accomplish a common objective [97]. These shared resources, to which no monetary values are assigned, are owned and managed by different organizations and institutions known as virtual organizations (VOs). Various VOs generally have different services, resources, and policies. Thus, a Grid environment can be regarded as a loosely coupled heterogeneous system. Due to the involvement of many VOs, the resources of the Grid are run under different policies. Based on the primary purpose (the main objective), VOs could be application service providers, storage service providers, or even cycle providers [97]. Thus, VOs diverge tremendously based on the scope of their work, their objectives, and the size of the community.

Figure 1.1 depicts a typical Grid system with various VOs. As can be from Figure 1.1, many different users can interact with different VOs. These users can be from the same VO, from different VOs. The sharing of the resources here is based on some predefined roles or policies. Both resource owners and consumers can define their own constraints. For example, a VO may only permit access to their resource for users from outside the VO if the submitted application (job) is small and not computationally intensive. Or the consumer may accept only resources that finish the job at a specified time. Also, a resource in one VO may be used or shared by other VOs in many different ways depending on their objective and the reasons for sharing the resource. Therefore, relationship for sharing resources is dynamic and
changes periodically in terms of the shared resources and users (participants) that can access the resources.

![Diagram of a Grid system with different VOs.](image)

Figure 1.1: A Grid system with different VOs.

The Grid was first originated in the mid of 1990’s to indicate a distributed computing architecture for advanced science and engineering computations [95]. The term *Grid* was chosen analogously to an electrical power Grid, as it provides access to stable, continuous, on-demand electricity regardless of the location of the source [96]. The Grid technologies were developed to allow resource sharing for scientific purposes. Resources in the Grid system are responsible for carrying out the assigned tasks. Different types of resources can be involved in the Grid system, such as computation resource, data resource, and storage resource [91]. A Grid can be classified into three
main categories [149; 259]:

- **Computational Grid.** Grid in this category aggregate resources with high computation capability. They are also suitable for executing compute-intensive applications which are formed by a set of independent tasks such as Bag of Tasks (BoT) applications [61]. Grid projects such as SETI@home [18] and Nimrod/G [40] were designed to work with such applications. Based on the utilization and characteristics of aggregated resources, computational Grid can be further categorized into distributed super-computing and high throughput Grids [149].

- **Data Grid.** Data Grids provide an architecture for accessing, transferring, and storing big geographically distributed data for scientific collaborations [54]. Thus, this architecture emphasizes on data management of distributed data repositories. Applications in areas of astronomy [133] and climate simulation [181] require access to huge data repositories. These repositories can be a digital library or warehouses and are usually distributed in different locations. Some projects have been introduced for Data Grid, e.g., International Virtual Observatory Alliance [7].

- **Service Grid.** They are also known as *Utility Grid*. These systems focus on delivering Grid services to computational and data Grid, on a pay-per-use basis. Hence, based on the needs of the end-user, resources are allocated on these Grids. Gridbus [41] is an example project for Service Grid.
1.2 Grid Scheduling

Grid scheduling involves making scheduling decisions over multiple resources that run on different domains [214]. Scheduling process includes finding a resource or multiple resources in single or different sites to scheduling requested jobs (applications). There are three essential components in the Grid system: Grid Resource Broker (GRB), Grid portal, and Grid Information System (GIS) [138]. The GRB is the scheduler. The Grid portal allows users to communicate with the scheduler. The scheduler then identifies and allocates resources by GIS based on users requests. Frequently, different users have different requirements. Therefore, the scheduler maps the submitted requests with suitable resources. Figure 1.2 shows the scheduling process on a Grid environment.

To access the grid’s resources, users need to submit their “job” or sometimes referred to as “application”. Jobs are programs to be executed on the grid’s resources. These jobs can be written in some programming languages such as Java. A job can be a program to compute something by executing some commands or collect some data. Jobs may be split into multiple tasks; these tasks are executed in parallel to increase the throughput of the grid system. Sending the tasks to be executed on the grid’s resource is the grid system responsibility. Scheduling on the grid can be specified by users. That is, users can select the appropriate resource and use the grid commands to send their tasks to the selected resources. However, most of the grid systems include “scheduler” to automate finding a suitable resource. The scheduling process may involve more than just finding suitable resources. For example, it is desired to schedule tasks with dependencies on the same or nearest resources to reduce the
network communication overhead.

Scheduling on the Grid consists of four different stages: matching, mapping, allocation, and scheduling. Each stage can be viewed as an independent research problem. The matching stage is an important stage. The latter three stages depend on the this stage. In other words, the matching process in the first stage will be used in the other stages. Efficient matching is therefore, essential to increase the efficiency of the Grid. This will allow submitted tasks to be completed in less time. Hence, user requirements have to be met and satisfied efficiently to map and complete the task on the existing resources. This problem is denoted as the task matching problem.

Figure 1.2: Scheduling process on Grid environment.
1.3 Task Matching Problem

Finding an optimal solution or optimal matching is not an easy task. The task matching problem has been identified as an NP-hard problem [260]. In addition, the heterogeneity environment of the Grid and different Quality of Service (QoS) from various users makes the task matching problem even more complex. Before going into details, few assumptions are made to address the task matching problem:

1. Tasks are independent (no interdependence between tasks).
2. Tasks cannot be pre-empted.
3. There is a one to one mapping on resources and tasks.

The task matching problem considers different independent tasks, \( T_i = \{T_1, T_2, ..., T_n\} \), and Grid heterogeneous resources, \( R_j = \{R_1, R_2, ..., R_m\} \). The length of each task is measured by Millions of Instructions (MI) while the speed of each resource is defined as Millions of Instructions per Second (MIPS). To evaluate the quality of the matching, makespan is used to determine the maximum completion time. The makespan is computed using the “Expected Time to Compute” (ETC) matrix. The Grid scheduler calculates this matrix before the scheduling algorithm takes over. Task profiling and analytic bench-marking are used to estimate the execution time for the ETC matrix. The number of tasks and resources defines the size of the ETC matrix. Thus, the ETC is represented by \( n \times m \) matrix, where \( n \) is the number of tasks, and \( m \) is the number of resources. Each row represents the estimated execution time of a given task on various resources, and each column represents the estimated execution time on a given resource for various tasks. For instance, \( \text{ETC}[T_2, R_4] \) in Table 1.1, gives the
Table 1.1: ETC matrix example.

<table>
<thead>
<tr>
<th></th>
<th>$R_1$</th>
<th>$R_2$</th>
<th>$R_3$</th>
<th>$R_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>4</td>
<td>12</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>$T_2$</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>$T_3$</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>$T_4$</td>
<td>6</td>
<td>9</td>
<td>10</td>
<td>8</td>
</tr>
</tbody>
</table>

The estimated execution time of task $T_2$ on resource $R_4$, which is 2. The completion time of resource $R_j$ is the sum of the execution time of all tasks assigned to it. Equation 1.1 shows the completion time of resource $R_j$.

$$C_j = \sum ETC[T_i, R_j], i = \{1, 2, ..., n\}, j = \{1, 2, ..., m\}$$

(1.1)

### 1.4 Motivation

The Grid is a large scale distributed system. The primary objective of a Grid system is to optimize the total execution time (makespan) of an application. To do this, it is important to efficiently map the tasks to the available resources. However, finding the maximum throughput with optimal resource utilization is a challenge. The characteristics of the Grid system such as heterogeneity make mapping of tasks to resources even more complicated. In the task matching problem, tasks can be arranged as a set of independent tasks (called *meta-task*), or as a *workflow*. Though, makespan is usually the metric optimized, other objectives such as cost or flow-time of the execution may be optimized, making the task matching problem a complex
multi-objective optimization problem.

In the last decade, various studies have tried to address the task matching problem (with one objective or more) using meta-heuristics. One such meta-heuristic that has gained popularity in the Grid scheduling community is Particle Swarm Optimization (PSO). Solomon et al. [222] showed that PSO is a viable technique to solve the task matching problem. However, there is a disadvantage to PSO. The technique stagnates at local optima. It was also noticed by [172] that diversity of the swarm is lost due to the stagnation. One possible solution is to add some learning mechanism to PSO to help the algorithm learn from the search space as the algorithm evolves. In this thesis, we incorporate genetic learning, a technique inspired from genetic algorithm, into the PSO technique. Genetic learning is inspired by the genetic algorithm (GA).

1.5 Contributions

The contributions to this thesis are as follows:

1. **Genetic learning PSO for task matching problem.**

   We incorporate the genetic learning approach to guide the particle swarm optimization technique. The algorithm’s complexity using the the dimensions of the problem, swarm’s size, and cost of the fitness function are presented. the weakness of the algorithm on some data sets are presented.

2. **Two extension to genetic learning algorithms.**

   The thesis proposes two extensions of the proposed genetic learning PSO to solve the weakness on some data sets. The first extension partitions the swarms
into two subpopulations (similar to an island model in genetic algorithm) and incorporates different learning techniques into these subpopulations. The second extension incorporates adaptive mutation for better convergence.

3. **Single and multi-objective task matching problem.**
   The independent task matching problem without dependencies are solved for both single and multi-objective optimization problems. We consider makespan and flowtime as two objectives to be optimized.

4. **Multi-objective optimization on workflow problems**
   We extend our work to study workflow applications represented as a direct acyclic graph to check the feasibility of problems the algorithms on these complex problems. We provide some ideas on the weakness of the algorithms for such applications and propose some modifications with some assumptions on the problems.

### 1.6 Thesis Overview

The rest of the thesis is organized as follows: Chapter 2 presents background on the optimization problem and meta-heuristic. This is followed by Chapter 3, which provides a detailed literature review about methods and techniques that have been done to solve the task matching problem. Moreover, Chapter 3 shows a useful taxonomy for task matching problem for better understanding. Then, the thesis discusses the current issue with PSO and proposes a modified genetic learning PSO in Chapter 4. Chapter 5 introduces two improved algorithms, which are hybrid genetic
learning PSO and Modified genetic PSO with adaptive mutation. In Chapter 5, all the three algorithms are experimentally tested against state-of-the-art algorithms for single and multi-objective task matching problem. Chapter 6 discusses the implementation of the three genetic learning PSO algorithms for task matching problem when the tasks are modeled as a workflow. Finally, the thesis concludes and presents some possible directions as future work in Chapter 7. Figure 1.3 depicts a pictorial representation of the thesis.

Figure 1.3: Thesis organization.
Chapter 2

Meta-heuristic Approaches for Optimization Problems

This chapter provides essential background on optimization and meta-heuristic approaches. The chapter begins by discussing some fundamental definitions with explanation on single objective and multi-objective optimization problems. This is followed by discussion on meta-heuristic methods for solving optimization problems. Finally, two well-known meta-heuristic methods adopted in this thesis are presented, namely, genetic algorithm (GA) and particle swarm optimization (PSO).

2.1 Optimization Problem

Given an objective function, an optimization problem refers to finding the best (optimized) solution from a set of feasible solutions. Optimization problems arise in many disciplines. An optimization problem can be represented as a minimization or
maximization problem. The goal, therefore, to the optimization problem, is to find
the optimal solution that either minimizes or maximizes the overall objective function
depending on the problem. Finding the so called ”best” optimal solution is difficult.
In general, the optimal solution to an optimization problem is described as a near to
the best solution or good enough solution [42]. For a set of feasible solutions \( X \), the
optimization problem can be expressed as follows:

\[
\begin{align*}
    f(x^*) &\leq f(x) & \forall \ x \in X \\
    f(x^*) &\geq f(x) & \forall \ x \in X
\end{align*}
\] (2.1) (2.2)

Equations 2.1 and 2.2 represent the minimization and maximization of the optimization
problem, respectively. In this thesis, we consider the minimization problem.
The function \( f(x) \) is denoted as the objective function. The objection function can
represent any parameter (such as cost, size, etc.) that needs to be optimized. The
variable \( x \) in the function \( f(x) \) is called a design variable or decision variable [20]. The
variable \( x^* \) is the global optimal solution of the optimization problem. Generally, optimi-
ization problems may contain a set of decision variables. The variable \( x \) is therefore
represented as a vector with set of decision variables, \( x = [x_1, x_2, ..., x_n] \). The decision
variables are located around the solution space. The decision variables can be either
continuous or discrete. Accordingly, the optimization problem can be continuous
where all decision variables are represented with real-valued numbers or combinatoric
with discrete values. The task matching problem is an example of a combinatorial
optimization problem. In any minimization (or maximization) problem, we could find
two values: the local minimum (maximum) or or the global minimum (maximum)
value. Since, the task matching problem is a minimization problem, we provide the
definition of local and global minimum [245]. We can likely do the same for the maximization problem.

**Definition 2.1 (Local Minimum)** A local minimum $x^* \in X$ as an input to $f(x^*) \leq f(x)$ for all $x$ near to $x^*$ is defined as:

\[
\forall x^* \exists \delta > 0 : f(x^*) \leq f(x) \forall x \in X, |x - x^*| < \delta
\]  

(2.3)

**Definition 2.2 (Global Minimum)** A global minimum $x^* \in X$ as an input to $f(x^*) \leq f(x)$ for all $x \in X$

The global minimum is the lowest value (solution) among all the feasible solutions. The local minimum is the lowest value among the neighbouring values at a distance $\delta$. Figure 2.1 illustrates an example of local and global minimum for a given function $f(x)$.

![Figure 2.1: Local and global minimum of $f(x)$.](image)
In general, different optimization problems have different problem characteristics. In other words, an optimization problem with a given objective function, \( f(x) \), may have single or multiple local optimum values. This is denoted as problem modality. A problem with a single local optimum is known as unimodal problem whereas optimization with many local optima is called multimodal problem. In unimodal problem, the search is a local optimizer search since there is just one local optimum. The search optimization in multimodal problem is in finding a global minimum. For many real-world problems, it is hard to identify whether the problem is unimodal or multimodal. Clearly, solving the multimodal problem is more difficult than the unimodal problem. Figure 2.2 depicts examples of unimodal and multimodal. Besides being a unimodal or multimodal problem, the objective that needs to be optimized can be single or multiple. That is, an optimization problem can be a single objective or multi-objective problem. This is an important classification for the optimization problem [238], due to the difficulty in solving such problems. In the sub-sections 2.1.1 and 2.1.2 we will discuss in detail these two types of problems.

### 2.1.1 Single Objective Optimization

In the single objective optimization, the goal is to find a single solution, namely, the global minimum in case of the minimization problem. For many single objective optimization problems, there is usually a unique optimal solution. As the name indicates, there is just one objective to be considered for optimization. Thus, the optimal solution found consists of a single vector \( x \) of the decision variable as shown in equation 2.1. Determining the global minimum solution(s) is denoted as global optimiza-
Chapter 2: Meta-heuristic Approaches for Optimization Problems

Figure 2.2: Optimization problem modality.

for the single objective optimization. Single objective optimization problems have been studied exhaustively in the literature wherein meta-heuristic algorithms have been applied to solve them. In this thesis, the single objective task matching problem considered uses makespan as the primary objective to be minimized. We have extended this to the multi-objective optimization problem.
2.1.2 Multi-objective Optimization

Many real-world optimization problems are formulated as many-objective optimization problems, where there are many conflicting objectives to be optimized simultaneously. In the context of multi-objective optimization, the notion of “optimum” solution may not be just for one objective but rather for more than two objectives. The formulation of the problem is therefore not in finding a single optimal solution, but rather a set of solutions for two or more objectives. In this thesis, we consider the minimization of the $k$ objective optimization problem. Equation 2.1 for single objective optimization can be extended to define the multi-objective optimization as follows [64; 73].

$$\text{Minimize} \quad f(x) = [f_1(x), f_2(x), f_3(x), f_k(x)] \quad (2.4)$$

where $f(x)$ is a vector with $k$ objective functions to be optimized and $x$ is an $n$-dimensional decision variable vector $x = [x_1, x_2, ..., x_n]$. When dealing with multi-objective optimization problems, it is hard to find a single solution that optimizes all objective functions. Therefore, there is a trade-off in finding the solutions. The most commonly use notion to deal with trade-off solutions was proposed by Francis Edgeworth and later generalized by Vilfredo Pareto, known as the Pareto Optimality [67].

**Definition 2.3 (Pareto Optimality)** A vector of decision variables $x^*$ is Pareto optimal if and only if there is no decision variables $x$ such that $f_i(x) \leq f_i(x^*)$ for all $i = 1, 2, 3, ..., k$ and $f_j(x) \leq f_j(x^*)$ for at least one $j$.

Definition 2.3 states that $x^*$ is Pareto optimal if and only if no feasible decision
variables $x$ exists such that increasing some criterion would decrease at least one other criterion, simultaneously. Also, when these two conditions hold for a decision variables $x^*$, then $x^*$ dominates $x$.

Definition 2.4 (Pareto Dominance) A decision variable $x^*$ dominates a decision variable $x$ (mathematically denoted $x^* \prec x$) if:

$$\forall i \in \{1, 2, 3, ..., n\} : x^*_i \leq x_i \land \exists i \in \{1, 2, 3, ..., n\} : x^*_i < x_i$$ (2.5)

In multi-objective optimization problem, a set of solutions are given instead of a single solution, which is called the Pareto Optimal Set.

Definition 2.5 (Pareto Optimal Set) For a given multi-objective problem, the Pareto optimal set $\mathcal{P}^*$ is defined as:

$$\mathcal{P}^* = \{x \in X | \nexists x^* \in X, f(x^*) \prec f(x)\}$$ (2.6)

$\mathcal{P}^*$ contains a set of vectors whose corresponding decision variables can not be improved for all objectives, simultaneously. That is, the optimal set (points) in the decision space form the Pareto optimal set $\mathcal{P}^*$. The vectors in $\mathcal{P}^*$ are called non-dominated vectors. Evaluating the vectors in $\mathcal{P}^*$ in the objective or response space yields Pareto optimal front $\mathcal{P}\mathcal{F}^*$ or Pareto frontier.

Definition 2.6 (Pareto Optimal Front) For a given multi-objective optimization problem and its Pareto optimal set $\mathcal{P}^*$, the Pareto optimal front is defined as:

$$\mathcal{P}\mathcal{F}^* = \{f(x) | x \in \mathcal{P}^*\}$$ (2.7)

To illustrate the definitions mentioned above, Figure 2.3 shows an example of multi-objective solution points in the objective space. For point $C$ in Figure 2.3, there
are three different sets. The first set is in the dominate area, which is a set of points that point \( C \) dominates. For instance, point \( C \) dominates point \( E \), \( f(C) \prec f(E) \). The second set contains points that dominate point \( C \). In this case, point \( A \) dominates point \( C \), \( f(C) \succ f(A) \). Neither point \( C \) nor the points in the third set dominate each other. That is to say, point \( C \) and points in the third set, such as \( G \), are non-dominated, \( f(C) \not\prec f(G) \) and \( f(C) \not\succ f(G) \). The Pareto optimal set in Figure 2.3 comprises the optimal set of solutions for the given optimization problem. Compared with other solutions, the Pareto optimal set solutions converge better. Moreover, all solutions in the Pareto optimal front are non-dominated to each other.

![Figure 2.3: An example of minimization problem with two objectives.](image)

To solve the multi-objective optimization problem, obtaining a set of solutions that minimizes the distance to Pareto set (convergence) and maximizes the diversity
(the distribution of the solutions on the Pareto front) is challenging. Furthermore, non-dominated solutions need to be chosen and maintained carefully to explore further better solutions and prevent the loss of non-dominated solutions [152]. Due to their population concept mechanism, meta-heuristic approaches can find multiple optimal solutions in a single run. In addition, meta-heuristic approaches need little or no knowledge about the characteristics of the problem. Hence, this makes meta-heuristic approaches ideal candidates to work on solving optimization problems (single and multi-objective optimization problems). Meta-heuristics such as Ant Colony Optimization (ACO) [82], Genetic Algorithm (GA) [115] and Particle Swarm Optimization (PSO) [139] have been used for single objective optimization problem where Non-dominated Sorting Genetic Algorithm 2 (NSGA2) [74], Pareto Archived Evolution Strategy (PAES) [146], Strength Pareto Evolutionary Algorithm 2 (SPEA2) [283], and Multiobjective Particle Swarm Optimization (MOPSO) [65] for multi-objective optimization problems.

2.2 Meta-heuristic Approach

Traditional exact algorithms are not sufficient to solve many objective optimization problems with properties, such as high-dimensionality and multi-modality. We need to rely on approximation algorithms such as heuristic and meta-heuristic approaches. Meta-heuristics have been adopted broadly to solve hard optimization problems in the past two decades [228]. The term meta-heuristic was first introduced by Fred Glover [105] in 1986. Meta-heuristics are approximate, non-deterministic algorithms that are guided in the search space to find near-optimal solutions to com-
Chapter 2: Meta-heuristic Approaches for Optimization Problems

plex optimization problems. Meta-heuristics are problem-independent frameworks. Therefor, meta-heuristics have been applied to a wide range of optimization problem domains, such as finance and engineering. Meta-heuristics have commonly been inspired by nature (i.e., biology and physics). A key to successfully solving a problem using meta-heuristics is to balance exploration with exploitation searching \[37\] techniques. Exploration searching is used to thoroughly locate promising regions with high-quality solutions, whereas exploitation focuses on (refines) promising areas. The balance of exploration with exploitation depends on the type of meta-heuristic being used. Various meta-heuristics have different criteria for balancing between exploration and exploitation search. That is, some meta-heuristics tend to prioritize exploration, whereas others prioritize exploitation searching. Meta-heuristics can be categorized into two types of search-based techniques: single-point search and population-based search techniques \[34\]. The single-point search method starts with an initial solution. Then it enhances the solution by exploring a set of distinct neighbors. Examples of this method include tabu search \[105\], iterated local search \[95\] and variable neighborhood search \[182\]. In contrast, population-based techniques operate on a group of solutions, simultaneously. Examples of these latter techniques include GA and PSO algorithms.

2.3 Genetic Algorithm

The genetic algorithm (GA) was developed by Holland based on processes of natural selection \[116; 115\]. GA algorithm is a popular evolutionary algorithm and has been applied to many optimization problems and machine learning applications \[107\].
GA algorithm applies three main operators: *selection*, *crossover*, and *mutation*. Each member called as an *individual* or *phenotype* in the population is represented as a *chromosome* or *genotype*. The phenotypes (candidate solutions) are evolved to find better solutions through the three GA operators. Originally, each individual is encoded as a binary (strings of 0’s and 1’s) to represent genotype. Other representations and encodings for genotype are also possible for the GA algorithm.

In GA, the population evolves through mating of chromosomes, and the best individuals survive. It uses an exploratory search technique that is directed by the chromosomes in the population. The evolution in GA usually begins with a random generation of the population, and in each iteration (generation), selection, crossover, and mutation operators are applied to the population. In the selection phase, fittest (better) individuals are selected for breeding. The fitness of each individual represents the value of the objective function. This phase ensures that the best individuals with high-quality chromosomes are selected for the next phase. At the end of the selection process, a mating pool is created, which contains the best individuals. In the crossover (recombination) phase, two individuals (parents) are selected from the mating pool. In this stage, both individuals exchange their chromosomes in creating an *offspring*. Next, in the mutation phase, a position (gene) is randomly selected and modified. Crossover and mutation are reproductive operators that ensure that high-quality genes are transmitted and reproduced within the population. Figure 2.4 visualises the crossover and mutation operators in GA.

The operation of evolving in GA continues until a predefined termination condition is reached, such as maximum number of generation (iterations) or no further
improvement has been noticed for certain number of generations. Algorithm 1 shows
the pseudocode of GA.

Algorithm 1 The pseudocode of GA

1: Initialize population randomly
2: Evaluate the fitness value
3: Select fittest individuals
4: \textbf{while} (number of generation \neq maximum) \textbf{do}
5: \quad Crossover operation
6: \quad Mutation operation
7: \quad Evaluate the fitness value
8: \quad Select fittest individuals
9: \textbf{end while}

2.3.1 Selection Operation

At the end of each successive generation, part of the population is selected for
breeding (crossover and mutation) to generate the next generations. Different selec-
tion methods have been introduced for this purpose. The selection is a fitness-based
process. The most common selection method is *generational genetic algorithm*. In this method, two parents are selected. Then, after the crossover and mutation phases, the offspring(s) are picked for the next generation. Hence, the entire population is replaced with the generated offspring. This method increases the diversity of the population, and prevents converging into local optimum. However, the downside of the method is the slow convergence of the population. That is, the population requires more iterations to locate the global optimum solution.

Another method that is similar to the generational genetic algorithm is called *steady-state genetic algorithm*. This selection method emphasizes on *elitism* concept. So, instead of replacing the entire population, some parents are preserved and carried to the next generation. A copy of the parents is saved before the crossover and mutation operations. Then, the algorithm chooses between the parents and the offspring based on the fitness value. There is a limited number of $k$ parents that can be preserved for the next generation.

### 2.3.2 Crossover Operation

After the selection, crossover operation may proceed to form the next generation. From the mating pool, two parents are randomly selected. Segments from each parent are exchanged to produce the offspring. Different crossover techniques such as *single-point*, *two-point* and *uniform* crossover [11] are commonly used in the GA. The single-point crossover chooses a locus randomly, and the segment after that locus is exchanged. In the two-point crossover, two random loci are selected, marking the boundary of the exchange segment. Figure 2.5 and Figure 2.6 show the single-point
and two-point crossover, respectively.

![Figure 2.5: The single-point crossover.](image)

![Figure 2.6: The two-point crossover.](image)

The uniform crossover is gene-based operator. Here, each gene in one parent is exchanged with the corresponding gene from the other parent with respect to a predefined probability, which is usually set to 0.5. When dealing with continuous optimization problems, arithmetic crossover is commonly used. The offspring is a basically linear combination of the parents. Equation 2.8 defines the offspring using the arithmetic crossover:

$$x_{offspring} = r \times x_{parent1} + (1.0 - r) \times x_{parent2}$$  

(2.8)

where \( r \sim U(0, 1) \) is a uniform random number. There are some worthwhile mentioning crossover methods such as partially-mapped crossover (PMX) [108], uniform partially-mapped crossover (UPMX) [60], and cycle crossover (CX) [188].
2.3.3 Mutation Operation

The mutation in GA is simple and straightforward. The operation is to invert the bit (in case of binary representation) of each offspring based on small probability, called mutation rate. For each bit, a random number \( r \) is generated, and if the random number \( r \) is less than the mutation rate, then the mutation is performed. While the crossover helps the population to converge (exploitation), the mutation rate in GA is to maintain the diversity of the population (exploration), thereby the algorithm is able to explore other areas in the solution space. Note that the mutation rate is problem-dependent, i.e., the value might work differently depending on the characteristics of the optimization problem. A very high mutation rate may disrupt the population, as many good solutions are altered. Thus, the search in the population is reduced to more random search, resulting in less convergence. A very low mutation rate is not preferred also, as the population may not be able to jump out of local optimum region. Hence, the mutation rate has to be determined carefully. Some suggest that a dynamic mutation rate is more effective than the fixed mutation rate during the optimization process [186; 93; 23]. For example, at the beginning of the optimization process, the mutation rate is high, then gradually decreases over time.

2.4 Particle Swarm Optimization

Particle swarm optimization is a stochastic population-based optimization technique introduced by Kennedy and Eberhart [139]. In contrast to the evolutionary algorithms, such as the genetic algorithm, the algorithm mimics the “intelligent col-
lective” behavior in a swarm, such as “birds’ flocking.” Members or individuals collaborate through exchanging information within the population to solve optimization problems. In the PSO algorithm, an individual is called a particle, and a set of particles form a swarm. Relying on their own best experience (pbest) and the best experience of the swarm (gbest), the particles “fly” in the solution space looking for regions with potentially high-quality solutions. For a given problem, each particle in the swarm represents a candidate solution. To verify the solution’s optimality, a “fitness function” is used as particles move in the solution space. The PSO algorithm does not require any gradient information for the objective function. Hence, the algorithm has been applied to many optimization problems in different domains.

2.4.1 The Canonical PSO

In the $D$-dimensional search space, the $i^{th}$ particle can be represented as a $D$-dimensional vector $x_i = \{x_i^1, x_i^2, x_i^3, ..., x_i^d\}$ for $d = \{1, 2, ..., D\}$ and $i = \{1, 2, ..., n\}$. Following these notations, velocity is calculated to determine the direction and the distance that each particle should move in the search space. The velocity and distance are calculated using the following equations [139]:

$$v_{i+1}^d = v_i^d + c_1 \cdot r_1^d (pbest_i^d - x_i^d) + c_2 \cdot r_2^d (gbest^d - x_i^d) \quad (2.9)$$

$$x_{i+1}^d = x_i^d + v_{i+1}^d \quad (2.10)$$

where $v_i^d$ and $x_i^d$ indicate the velocity and the position of the $i^{th}$ particle, respectively; $v_{i+1}^d$ and $x_{i+1}^d$ are the corresponding velocity and position of particle $i$ in the next instance of time. $pbest_i^d$ denotes the best position (solution) of the $i^{th}$ particle, and
$\text{gbest}^d$ is the best position in the swarm. $r_1^d$ and $r_2^d$ are independent random numbers generated within the range $[0,1]$. $c_1$ and $c_2$ are acceleration coefficients that determine the stochastic acceleration toward $\text{pbest}$ and $\text{gbest}$, respectively. Clearly from Equation 2.9, $c_1$ adjusts the maximum step size toward the best particle in the swarm and $c_2$ adjusts the maximum step size toward the personal best position of particle $i$.

In the initial version of the PSO, $c_1$ and $c_2$ are scaled to 2. The velocity is clamped to the range $[-v_{\text{max}}, v_{\text{max}}]$ so the particle does not go beyond the search space. The variable $v_{\text{max}}$ is set to $v_{\text{max}} = k \times x_{\text{max}}$ where $0.1 \leq k \leq 1.0$ and the boundaries of the solution space is defined by $[-x_{\text{max}}, x_{\text{max}}]$ [69].

We briefly describe the PSO algorithm. Initially, each particle evaluates its solution in the search space using the objective function. Then, the “best” particle’s position (solution) is identified as the best in the swarm. The rest of the swarm now moves in the direction of the best particle and their personal best position. Thus, some particles may exceed the “neighborhood” of the best particle exploring other areas in the solution space. Since most of the optimization problems have some continuity, there is a good chance that there are some good or better solutions around the best particle. As different particles come from different directions around the best position, different solutions are evaluated in the hope of discovering the global optimum solution. Figure 2.7 illustrates the movement of the particles in the PSO algorithm.

The Equation 2.9 has been subject to amendments in order to improve the convergence of the PSO algorithm. One of the most widely used modified version of PSO is the inertia weight version by Shi and Eberhart [215]. Shi and Eberhart investigated
Chapter 2: Meta-heuristic Approaches for Optimization Problems

the role of each term in Equation 2.9\(^1\) and found that the previous velocity term at iteration \(i\) influenced the balancing between the global and local search. They excluded the previous velocity term and found that the particles were unable to fly out of their region. However, when the value of the previous velocity term was high, the particles were more reluctant to converge to a good solution, exploring other areas in the solution space; the search became more random search. Therefore, Shi and Eberhart introduced the inertia weight to control the trade-off between exploration and exploitation tendencies of the particles, and the velocity equation was updated as follows:

\[
v_{i+1}^d = \omega v_i^d + c_1 \cdot r_1^d (pbest_i^d - x_i^d) + c_2 \cdot r_2^d (gbest_i^d - x_i^d)
\]

\(^1\)There are three terms in Equation 2.9, which are the previous velocity term, cognitive term, and social term.

Figure 2.7: Particle’s movement in the PSO algorithm.
The variable \( \omega \) controls how much the velocity term should be controlled. Clearly, when setting \( \omega = 1 \), the original PSO is obtained. Shi and Eberhart examined the impact of \( \omega \) in range \([0, 1.4]\) and when varying the \( \omega \) over time on the particles of PSO algorithm. The results showed that \( \omega \in [0.8, 1.2] \) result in faster convergence compared to other settings of \( \omega \). Shi and Eberhart concluded that having \( \omega \) close to 1.0 is suitable for good convergence. Another set of empirical experiments were conducted by Shi and Eberhart \([216; 217]\) for further investigation on \( \omega \). The best performance was obtained when \( \omega \) is decreased linearly from 0.9 to 0.4 over time.

Clerc \([63]\) proposed a constriction factor to ensure the convergence in PSO algorithm. Through correctly choosing the values of \( \omega, c_1, \) and \( c_2 \), there is no need to clamp the value of \( v_i \) in range \([-v_{\text{max}}, v_{\text{max}}] \). Updating the velocity in the proposed model defined as:

\[
v_{i+1}^d = \chi [v_i^d + c_1 * r_1(p_{\text{best}}^d - x_i^d) + c_2 * r_2(g_{\text{best}}^d - x_i^d)]
\]

\[
\chi = \frac{2}{|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}|}
\]

where \( \varphi = c_1 + c_2, \varphi > 4 \). Suppose \( c_1 = c_2 = 2.05 \), using Equation 2.13 results in \( \chi = 0.7298 \). Then, \( c_1 = c_2 = 0.7298 \times 2.05 = 1.4962 \). This is equivalent of using \( c_1 = c_2 = 1.4962 \) and \( \omega = 0.7298 \) in Equation 2.11. Eberhart and Shi \([83]\) compared the use of constriction factor and with the inertia weight in PSO algorithm. Both approaches produced the same performance. Moreover, the constriction factor struggled to solve some optimization problem without clamping \( v_{\text{max}} \). By adding the clamping to the constriction factor approach, the PSO algorithm performed better in most optimization functions.
In this thesis, Equation 2.11 is used for the PSO algorithm. The pseudocode of PSO is shown in Algorithm 2.

### 2.4.2 Social Behavior

Besides the velocity and the other two terms in the velocity Equation 2.9 the search behavior of the particles need to be considered. Kennedy [140] studied the function of each term for training a neural network for XOR problem classification. Recall Equation 2.9, the term $c_1*r_1(p_{best_i} - x_i)$ resembles the cognitive behavior since it considers only the particle’s best experiences (solutions). Kennedy found that using the cognition term is not sufficient for the PSO algorithm. One reason is that there was no communication between the particles, and each particle performing search on its own. The other term $c_2*r_2(g_{best} - x_i)$ presents the social interaction among the particles. The social term outperformed both the cognitive and the original models of PSO on some specific neural network problems [140]. Thus, the impact of the social term seems to be more substantial on the performance of the PSO algorithm.

### 2.4.3 Topology Structure

The topology structure of the PSO defines the communications between the particles. A particle $i$ is a neighbor of particle $j$ if there is a bidirectional edge paired with them. Hence, both particles can be considered to be in the same neighborhood. Communications between particles can affect the whole swarm performance. Thus, choosing the suitable topology for a given problem is crucial [235; 62]. Normally, there is one best particle in the neighborhood whereby other particles are influenced.
Algorithm 2 The pseudocode of PSO algorithm

1: /*Initialize N particles*/
2: for $i = 1$ to $N$ do
3:   randomly initialize $v_i$
4:   randomly initialize $x_i$
5:   calculate $f(x_i)$
6:   set $x_i$ to $pbest_i$
7:   if $pbest_i < gbest$ then
8:     set $gbest$ to $pbest_i$
9:   end if
10: end for
11: /*PSO loop*/
12: while number of iteration $\neq$ maximum do
13:   for $i = 1$ to $N$ do
14:     for $d = 1$ to $D$ do
15:       Update the velocity using Equation 2.11
16:       Update the position using Equation 2.10
17:     end for
18:     calculate $f(x_i)$
19:     if $f(x_i) < pbest_i$ then
20:       set $pbest_i$ to $x_i$
21:     end if
22:     if $pbest_i < gbest$ then
23:       set $gbest$ to $pbest_i$
24:     end if
25:   end for
26: end while
Figure 2.8: Swarm topologies.

The communication here is accomplished through the best position $p_{best}$ of the best particle in the neighborhood. In this thesis, an overview of three popular topologies is presented, which are fully connected, ring and pyramid topologies. The topologies are depicted in Figure 2.8.

From all variations of PSO, the fully connected PSO, which is also called $g_{best}$ or star topology, has been broadly considered as the “standard” topology [31]. One reason for its prevalence is the simplicity of the algorithm. That is, the fully connected PSO is considerably easy to code and implement since the particles in this topology are connected to each other. Moreover, the algorithm delivers adequate results for many optimization problems. Additionally, the PSO performs fast as there is no need for any look-up for neighbors; only the best position in the swarm needs to be tracked. Although successful results could be achieved with the fully connected topology, the fully connected topology suffers from premature convergence. This is because of the rapid influence of the best position in the swarm. Therefore, in case the initialization
is wrong or the particle is in the local optimum area, the whole swarm will not be able to explore the solution space effectively.

The ring topology, which is called \textit{lbest} topology, differs from the fully connected topology as the particles here are just connected to their immediate neighbors (the left and the right neighbors) [171]. Furthermore, the information flow in the ring topology is slower compared to the fully connected topology. However, this helps the particles in the ring topology to thoroughly explore the solution space. In other words, the information of the best position will be passed through different neighborhoods before the swarm starts to converge. This also helps the swarm to be less likely to get trapped in local minimum. Works [244; 226; 141], investigated different \textit{lbest} topologies. Equation 2.14 shows the velocity calculation for the ring topology:

\[
v_{i+1}^d = \omega \ast v_i^d + c_1 \ast r_1^d(pbest_i^d - x_i^d) + c_2 \ast r_2^d(lbest_i^d - x_i^d) \tag{2.14}
\]

where \textit{lbest} is the best position found in the neighborhood of the \textit{i}th particle.

The only difference between the ring and the fully connected topology is the use of the best particle in the neighborhood instead of the best particle in the swarm. In the fully connected topology, one big neighborhood contains all the particles of the swarm. Nevertheless, the ring topology has many neighborhood size of 3 particles. It is worth mentioning that there is no superior topology for all problems. Indeed, it has been shown in [31] that the ring topology outperformed the fully connected in some optimization problems, while in some other problems, the fully connected worked more efficiently than the ring. For hard problems, the ring topology showed better performance than the fully connected topology. In contrast, the fully connected out-
performed the ring when the optimization problem was easy [31]. Thus, knowing the type of optimization problem will help significantly in selecting the proper topology to solve the problem efficiently. The pyramid topology is a 3-dimensional topology that forms triangles. One particle represents the top point of the pyramid, and the other four form the base of the pyramid.

The neighbors of a particle can be determined by either the particle indices\(^2\) or the particle positions in the solution space. In this thesis, using the indices method has been adopted for two reasons: it requires low computation time since no clustering has to be performed to form the neighborhoods. The second reason is to help spread the information of good particles regardless of the actual position on the solution space. Hence, far particles are more likely to receive the information. Moreover, the fully connected (gbest) topology is selected based on our findings in [14].

### 2.5 Summary

This chapter presented the necessary background information needed for the thesis. We specifically focused on the two meta-heuristic algorithms used in the thesis in solving the task matching optimization problem. We also introduced the different topological structure of the particles in the the PSO algorithm, that will be discussed in length in the chapters ahead.

\(^2\)For the ring topology as an example, particle \(i + 1\) forms a neighborhood along with particle \(i\) and particle \(i + 2\).
Chapter 3

Literature Review

This chapter intends to provide a comprehensive survey of existing algorithms and techniques used for the task matching in the grid system. A classification of the current approaches and techniques for task matching problem is presented for better understanding of the problem. Based on the classification, the survey of the task matching problem reviews two essential approaches: (i) heuristic based algorithms for independent tasks and dependent tasks (workflow) in the grid system (section 3.2); and (ii) meta-heuristic algorithms (in particular, GA and PSO) for solving the task matching problem for independent and dependent tasks (section 3.3). Section 3.4 discusses the literature on the enhancement to the PSO algorithm such as hybridization and learning techniques to overcome of the weaknesses in the PSO algorithm.
3.1 Introduction

Large scale distributed systems have provided scientists the means to deal with large amounts of data. Along with these systems, includes plethora of computing technologies to deal with computing the data. However, finding an efficient algorithm to match tasks to these systems is still a challenge. Research is still underway on the task matching problem.

In the grid system, assigning a set of tasks to a set of resources is key to increase processor utilization to thereby increase parallelism and obtain overall high throughput. Several issues need to be considered when working on the task matching problem in grid systems:

- The tasks may not necessarily be of the same length. They may be heterogeneous affecting the performance on a processor.

- Tasks may be submitted by different users with different requirements which needs to handled in such a way that processor utilization is efficiently maintained.

- Tasks modeled as a workflow may have dependencies that has to be maintained. Communication between tasks may affect the performance on the grid system.

To address these issues, many heuristic and meta-heuristic approaches have been proposed. Figure 3.1 presents the taxonomy of the task matching problem in the literature.
3.2 Heuristic based approaches

Finding an optimal matching to the task matching problem is challenging. The task matching problem is NP-hard. Exact or deterministic methods [206] are not practical. They are computationally intensive [85]. Approximate methods on the other hand provide sub-optimal solutions in a reasonable amount of time. Heuristics are problem-dependent methods. The techniques generally take advantage of the domain knowledge (knowledge about the problem) to produce fast, near optimal results. Tasks could be independent of each other or dependent on each other. In both cases, heuristic approaches have been used. The next two subsections describe heuristic methods used in task independent and task dependent problems.
3.2.1 Task independent

The task matching problem has been studied extensively in the past. Numerous techniques and algorithms have been proposed to solve this issue efficiently on computational grids. A heuristic approach, such as problem-specific heuristic approach, has been used widely for the task matching problem. First-Come-First-Serve (FCFS) is one example, which handles tasks in order based on their arrival time [278]. In the event of an unavailable resource, the scheduler does not process the current task until a resource is available. Simplicity and short computational time are significant advantages to using FCFS. However, FCFS could lead to load imbalance as the scheduling is not based on the task length. Thus, large tasks could be assigned to low computational resources and vice-versa. The performance of FCFS could be enhanced by using a back filling technique [230]. Here, tasks that are waiting at the back of the queue are assigned to other available resources if a task with a high computational resource requirement is waiting to be executed. Consequently, the average idle time is decreased.

Another algorithm, Opportunistic Load Balancing (OLB) [147] keeps all resources as busy as possible by splitting the task on multiple resources. A drawback of this algorithm is not taking the computation length of the task into consideration. Therefore, the makespan can be high [175]. The Min-Min and Max-Min are worth-mentioning methods. Min-Min [121] begins with an unassigned set of tasks. Then, the minimum estimated finish time is calculated for all jobs. Finally, the task with the overall minimum estimated finish time is chosen for the corresponding resource. That is, small (short) tasks are assigned first so the resources are available more quickly to
execute the next task. Max-Min is similar to the Min-Min algorithm [46] except that the task with the maximum estimated finish time is assigned first.

Maheswaran et al. [176] used different heuristic algorithms, i.e., Min-Min, Max-Min, Sufferage [57], and XSufferage [46], for a dynamic environment in a grid system. They divided the algorithms into two different groups, immediate mode and batch mode algorithms. They found that the performance of all algorithms is affected by the structure of the heterogeneity between tasks and resources. Izakian et al. [129] proposed a Min-Max heuristic for grid systems. The proposed algorithm has two stages. The first stage is similar to Min-Min algorithm; for each task, the minimum finish times on other resources is determined. The second stage is to choose the task with the maximum value of the finish time of the task on the fastest resource divided by the finish time on the selected resource (from the first stage). Min-Max was compared with other heuristics such as workQueue [112], max-min, LJFR-SJFR [251], Sufferage, Min-Min, and the technique outperformed all algorithms with respect to makespan.

3.2.2 Task dependent (workflow)

There are various works in the literature that have been considered using tasks matching problem when a set of tasks are represented as a workflow. The problem here is more difficult as there are precedence constraints (dependencies) between tasks. Directed Acyclic Graph (DAG) is the most commonly used model to represent workflows. Finding the near-optimal matching for workflows are categorized into static and dynamic task matching. Static task matching assumes that all the infor-
mation is known aprior. Therefore, the decision is made before the execution of the tasks. In dynamic task matching, the decision is made during the execution of the tasks due to lack of the information about the workflow and the grid environment. List scheduling, cluster scheduling, and duplication based heuristic are well-known methods for static planning.

List scheduling maintains ordering list for all tasks using priority. Then, the tasks are selected based on their priorities to minimize the objective function. Topcuoglu et al. [234] proposed Heterogeneous Earliest Finish Time (HEFT) algorithm for workflow data sets. HEFT involves two steps: the first step is upward task prioritization; the second step is processor selection. In the second step, the tasks are selected for the processors based on the prioritization properties from the first step. Hence, the finishing time of the task is minimized. HEFT was compared [234] with Dynamic Level Scheduling (DLS) [220], Mapping heuristic (MH) [86], and Levelized-Min Time (LMT) [128] algorithms on random generated workflows and real-world workflows. HEFT outperformed all other algorithms with respect to performance, speedup, and average scheduling length ratio.

Kwok and Ahmad [151] introduced Dynamic Critical Path (DCL) algorithm. The algorithm assigns priority to the tasks in the critical path in which the finishing time of the task is minimized. Rahman et al. [199] extended the DCL to work in the grid environment and called it Dynamic Critical Path for Grids (DCL-G). Sakellariou et al. [209] proposed Hybrid Minimum Completion Time algorithm. The algorithm breaks the workflow into independent sub-tasks using standard list schedule ranking approach. Through the literature, there are some noteworthy heuristic algorithms for
workflow such as Extended Dynamic Critical Path (xDCP) [174], Fast Critical Path (FCP) [198], Critical Path On a Processor (CPOP) [233], Predict Earliest Finish Time (PEFT) [19], Heterogeneous Critical Parent Trees (HCPT) [113], and Performance Effective Task Scheduling (PETS) [122].

Cluster scheduling can be used to reduce the communication overhead between tasks by grouping these tasks into a single group and assigning them to the same resource. The task here could be a ready task or not ready task. At each iteration, a merging step is performed to merge two or more tasks into a single cluster. In the end, the number of clusters should be no more than the number of resources. The first attempt to use the clustering concept was done by Sarkar [213]. The algorithm proposed by Sarkar begins by sorting all edges of the workflow in descending order based on the weights (communication time). Then, the algorithm examines edges of the workflow one by one. Two tasks are merged into one cluster if there is no increase in the execution time. The process continues until all edges in the workflow are examined. Yang and Gerasoulis [258] proposed Dominant Sequence Clustering (DSC) algorithm using the clustering idea. The algorithm tries to identify tasks in the dominate sequence (the longest path) and merge them if the execution time is reduced. Kim and Park [142] proposed an enhancement for DSC to work upward and downward clustering, simultaneously. Some other algorithms in this group are Linear Clustering Algorithm [143], Mobility Directed [248], Clustering and Scheduling System [163], and Cluster Mapping Algorithm (CMA) [36].

Duplication based algorithms aim to utilize the idle time of the resources through duplicating predecessor (parent) tasks and thereby minimize the communication time
which consequently reduces the execution time. Kruatrachue et al. [150] introduced Duplication Scheduling Heuristic (DSH) to improve the resource utilization. The algorithm combines the list scheduling and duplication concepts. The algorithm assigns priorities to the tasks in descending order. Then, the algorithm identifies the start time for each task using their priorities. The first task is selected and scheduled on all resources. If the selected task creates an idle time, parents of the current task are duplicated to reduce the stating time. If the start time is reduced, then the parent of the previous task is considered and the algorithm trace back to its parents. The process is repeated until the entry task is reached or there are no more idle time slots. Chung and Ranka [59] extended the DSH algorithm to Bottom-Up Top-Down Duplication Heuristic (BTDH) algorithm. The only difference here is that BTDH keeps duplicating the parent of the current node even if the idle time is filled up and the start time of the current task is increased temporarily assuming that duplicating other predecessors may reduce the starting time. Bajai and Agrawal [24] introduced a task duplication based scheduling algorithm for network of heterogeneous systems (TANH). The algorithm combines concepts of clustering and duplication. For every task, the algorithm determines some useful parameters such as earliest start and completion time, and latest start and completion time. Then, by using these parameters, the algorithm starts clustering the tasks. Two tasks in the same cluster will be mapped to the same resource. Some classical algorithms in this category are: Lower Bound (LWB) algorithm [68], Linear Clustering with Task Duplication (LCTD) algorithm [218], Critical Path Fast duplication [12], and Duplication First and Reduction Next (DFRN) [190].

Some research works proposed to capture the performance dynamism of resources
in the grid systems. Ma et al. [174] suggested a dynamic approach (pM-S) algorithm, an extension for Master-Slave (M-S) algorithm [204]. In the pM-S algorithm, the master node utilizes two queues simultaneously; one is ready queue, and the other is unscheduled queue. The algorithm starts by placing all the tasks in unscheduled queue. Tasks with no parents can be placed in the ready queue. All subtasks in the ready queue can be mapped to resources to be executed. When the subtasks are completed, the master node will check for the children of the subtasks to be placed in the ready queue. Garg and Singh [102] proposed adaptive workflow scheduling (AWS) algorithm to deal with dynamic grid resources when scheduling workflow. The algorithm is based on static scheduling and considers some other factors such as resource loading and adding new resource during the run time. To show the effectiveness of their algorithm, they compared the AWS with other algorithms such as HEFT, Min-Min, Max-Min, and HEFT-based Adaptive Rescheduling (AHEFT) [266]. The algorithm showed superior performance for minimizing the makespan. There are other algorithms specifically for dynamic scheduling such as Round Robin (RR) for dynamic scheduling [99], dynamic adaptive Path Clustering Heuristi (PCH) [30], and Adaptive List Scheduling for Service (ALSS) [55].

Some comparative studies [246; 169; 35] showed that static scheduling performs better than dynamic scheduling. Algorithms were able to search globally more efficiently in the static scheduling environment due to information availability. Although these problem-specific heuristics are applied widely in scheduling, they tend to produce a lower quality solution for complex heterogeneous environments [17; 219]. The problem-specific heuristic mentioned above are greedy constructive approaches. They
start with an empty solution and try to find the best solution at the immediate moment in order to generate the final global solution. Since it is greedy, the selected solution at each step can not be changed and may turn to be the worst solutions in the future. Thus, the generated final solution is not generally optimal. Hence, these above heuristics are suitable for specific problem instances.

3.3 Meta-heuristic based approaches

Meta-heuristics are substantially different in the underlying concept compared with heuristics. The search for a near-optimal through a guided process is one of the notable differences from heuristics. They are also problem independent. Moreover, meta-heuristics produce and evaluate several solutions and find the best among them. Section 3.3 will cover important and recent works of meta-heuristics for task matching problem. The GA and PSO algorithms are included in this section since these are the techniques used in this thesis. They are also two algorithms that are commonly used in the task matching problem.

3.3.1 Genetic Algorithm (GA)

Braun et al. [39] conducted a comparison of eleven algorithms for matching independent tasks to resources. Among these eleven algorithms, GA preformed the best. Xhafa et al. [252] studied the various parameters in the task matching problem, considered different crossover operators with two representation methods (direct and permutation-based representations) and showed that their genetic algorithm produced better results for makespan and flow time metrics. Another work by Xhafa
et al. [253] enhanced GA by using the struggle technique. The key difference here is
the selection operation. Instead of replacing the worst individual, as in the standard
GA, the individual is replaced with a new similar individual only if the new indi-
vidual has a better fitness value. They also introduced two methods based on this
method: hash-based struggle GA and quadratic struggle GA. Quadratic struggle GA
has quadratic search computation time. This was reduced to linear search time us-
ing hash techniques in hash-based struggle GA. The hash-based technique performed
better than quadratic GA to minimize the makespan and flow time.

Abraham [9] considered GA, Simulated Annealing (SA), and Tabu Search (TS)
for task matching independent tasks. GA demonstrated better performance than SA
and TS. Moreover, the authors have presented a hybrid GA-SA and GA-TS. The
proposed hybrid GA-SA algorithm showed an improvement in the convergence rate
while GA-TS enhanced the GA’s efficiency on the grounds that TS and SA algorithms
work on improving one solution at a time whereas GA can produce multiple solutions
at a time. In other words, TS and SA work to refine the solutions produced by GA.
Carretero and Xhafa [44] investigated various GA algorithms for large-scale tasks.
The goal was to identify which parameters work best for large-scale applications on
gird to reduce the makespan.

Other works introduced new methods for task representation. Zomaya and Teh [286]
used GA to produce efficient scheduling with a good load-balance. They used 2D rep-
resentation to represent the strings. Columns represent number of resources in the
system while rows represent assigned tasks to a given resource. The authors indi-
cated that one point crossover does not perform well as some tasks may be assigned
to more than once while other tasks may not be assigned at all. To circumvent this problem, they experimented with cycle crossover. Cycle crossover takes two parents to find the so-called cycles in their genes. Initially, the values of the first position are copied to the children. That is, the gene at position 1 for parent A and B is copied to children A and B, respectively. Next, a cycle is found such that the selected gene of parent B has equivalent value in parent A and has not yet been selected. If an equivalent value is found in parent A, the values at the corresponding position are copied to the children. The procedure continues until no cycle is found. Any gap (empty positions) in the generated children is filled with gene values of the corresponding empty position in the parents such that child A and B are filled from parent B and A, respectively. Also, they used a slide-window technique to improve the efficiency where a window contains small number of tasks to be considered for execution each time. They concluded that their algorithm outperformed random allocate and first fit algorithms as the number of the tasks increased in terms of execution time and average utilization.

Other techniques such as niche method have been applied to develop adaptive GA [279]. In this method, a group of individuals with strong fitness values are selected to build a niche. They used segmentation hamming to identify if two individuals are in the same niche generating different niches. The authors stated that this technique can improve the diversity of the population. They showed that their algorithm performed better than the standard GA and Max-Min algorithm.

**GA for multi-objective optimization:** GA algorithm has been considered to solve multi-objective problems in the literature. Multi-objective is classified broadly
into *lexicographic* and *simultaneous* approaches. The lexicographic approach considers one single objective to be optimized at a time. Here, the primary objective is chosen to be optimized without considering any other objectives. Then, other objectives are selected to be optimized without deteriorating the primary objective. Xhafa et al. [254] implemented lexicographic to minimize makespan and flow time. The primary objective was the makespan and flow time set to be a secondary objective. They proposed hybrid GA (TS) algorithm to minimize the multi-objective function. To evaluate their algorithm, they considered pure GA and Tabu Search (TS) [106] algorithms. They implemented the algorithms on small, medium and large data size. The proposed algorithm outperformed the other algorithms in all data size for makespan and large data size for flow time.

Carretero and Xhafa [45] presented GA with different encoding and empirically studied different operators in GA. They found their proposed GA performed better than other existing GA [39].

The lexicographic approach can not handle all multi-objective problems efficiently as some conflict objectives cannot be enhanced without deteriorating other objectives. Simultaneous approach handles different conflict objectives concurrently. One such method is the weighted sum approach. The weighted sum method is a scalarization method that reduces different objective functions into one single objective function. Chitra et al. [56] integrated the weighted sum for GA to minimize three objective functions, which are makespan, flow time, and maximize the reliability. They tried to identify the suitable parameter for the weighted sum approach to obtain best results.

Another similar work was conducted by Kołodziej and Xhafa [148]. The authors
used Hierarchic Genetic Strategy (HGS) which can be considered as a variant of GA. HGS implements concurrent search through tree structure where each branch represents evolving operations of the population. Also, HGS utilizes a branch reduction for ineffective branches that work in the same region. The authors showed the effectiveness of their proposed algorithm against the pure GA for different data sizes reported in the literature.

**GA for workflow:** GA also has been studied for task matching workflow in the heterogeneous environment such as the grid. Wang [241] proposed a GA algorithm for task matching in the heterogeneous environment. Each chromosome in the GA is represented with two tuple that are matching string and scheduling string. Matching string represents the mapping tasks to resources and scheduling string represents the order of the execution of subtasks. The proposed algorithm was tested on different random generated DAGs against a random search, the baseline (BL), and leveled min-time (LMT). Yu and Buyya [263] proposed GA for workflow applications to optimize either monetary cost or execution time while meeting budget or deadline constraint, respectively. They evaluated the GA algorithm on both balanced and unbalanced workflows. The obtained result from GA was significantly better than Greedy Cost-Time Distribution (TD) and Greedy Time-Cost Distribution (CD) heuristics. The previous work has been extended by the same authors [264] for a multi-objective optimization problem on the grid. They focused on three multi-objective evolutionary algorithms, including Non-dominated Sorting Genetic Algorithm (NSGA2) [74], Strength Pareto Evolutionary Algorithm (SPEA2) [283], and Pareto Archived Evolution Strategy (PAES) [145]. Also, a local search algorithm was proposed to enhance
the solution quality of the three algorithms. The statistical results showed a tremendous enhancement in the solution quality, especially for the SPEA2 algorithm.

### 3.3.2 Particle Swarm Optimization

Salman et al. [210] demonstrated that PSO outperforms GA for task scheduling. In the PSO algorithm, given that there are $m$ number of tasks, $m$ dimensions are used to represent the solution space. The value of each dimension (location) represents the resource that the task is assigned to. Two variants of PSO have been proposed for task matching problem, namely, continuous (original) PSO and discrete PSO [84].

Solomon et al. [222] stated that continuous PSO performed better than discrete PSO. The task matching problem is a discrete problem. Thus, many methods have been introduced to transform continuous PSO values to discrete values such as integer-PSO [28], ranked-Order Value (ROV) [164] and rounding the continuous value to integer value PSO [197].

Tasgetiren et al. [231] proposed the Smallest Value Position (SPV) technique to transform continuous values to discrete values. Here, each dimension of PSO contains a real value. The algorithm takes the dimensions of each particle and associates them with an integer number based on ascending order. The smallest value will be assigned number 1. After using the ascending order of all dimensions, modulus operation ($X_i \mod M$) is performed. $X_i$ is the integer value and $M$ is the total resource number. Zhang et al. [273] used the SPV method for the task matching problem. They showed that PSO is better than GA by using this technique. Sadasivam and Rajendran [208] also used SPV with PSO. Compared to the random selection resource algorithm,
results showed that PSO with SPV provided a good load balance. Kang and He [136] used PSO to solve task assignment problem in heterogeneous environment. They incorporated variable neighborhood descent (VND) search algorithm to increase the efficiency of the exploitation process. Moreover, to balance between the exploitation and exploration of PSO, migration mechanism was introduced with the PSO. PSO was able to minimize the makespan more than a re-excited PSO (RPSO) [8], GA, SA, and TS.

Izakian [130] designed a new updating method for the particles. They used updated velocity matrix and probability for executing task $T_i$ on resources $R_j$. After getting the matrix, the random number is generated, and if it is less than the specified random number, the task is matched to the resources of high probability. Otherwise, roulette wheel selection is implemented to select the resources for the task. The proposed algorithm outperformed the original PSO and GA in terms of makespan. The same work has been extended [132] to handle multi-objective constraints. Authors used weighted sum approach to minimize makespan and flow time.

A recent work by Sarathambekai and Umamaheswari [212] adopted a new Hamming inertia weight for PSO. Three objectives are minimized such as makespan, flow time, and reliability cost. Variable neighborhood descent (VND) search algorithm was applied when there is no improvement on the $g_{best}$. There are some works that incorporate fuzzy approach with PSO to maintain multi-objective scheduling such as a fuzzy particle swarm optimization (FPSO) algorithm [166]

**PSO for workflow and multi-objective optimization:** PSO has been explored to work on workflow applications. Salman et al. [210] attempted to use PSO for task
matching of workflow applications. The implementation was straightforward, and they assessed PSO on randomly generated workflow and some other real-life problems, for instance, Tree, Fork-Join, Laplace Equation Solver, Gaussian Elimination and LU decomposition. GA was chosen for compression, and they showed that PSO exceeds GA in terms of solution quality for most test cases.

Chen et al. [48] stated that the work by Salman et al. did not include the heterogeneity of the resources and the precedence constraints among tasks. Thus, they included this in their work. The objective function was to minimize makespan. The proposed PSO was implemented on various DAGs. Compared with GA, the proposed PSO has shown its effectiveness.

Garg and Singh [101] proposed \( \epsilon \)-Fuzzy dominance sort based Discrete Particle Swarm Optimization (\( \epsilon \)-FDPSO) for multi-objective workflow scheduling problem. Dominant sort was used to obtain the Pareto front solution. The non-dominant solutions is a set of solutions where no solution is dominant with respect to all other solution. That is, enhancing the current solution in objective function can degrade the other objective function. \( \epsilon \)-FDPSO was used for bi-objective (makespan and economic cost) and tri-objective (makespan, economic cost and reliability). \( \epsilon \)-FDPSO was validated on randomly generated graphs and real-life problems such as Gaussian Elimination (GE) and Fast Fourier Transform (FFT). The authors used GA based non-dominated sort genetic algorithm (NSGA2) multi-objective particle swarm optimization (MOPSO) to measure the efficiency of the proposed algorithm. Results revealed that \( \epsilon \)-FDPSO was able to converge toward the Pareto front and better and produced high quality solutions compared with NSGA2 and MOPSO. Similarly,
Pei [194] used non-dominated sort with PSO for multi-objective workflow. The disadvantage of this technique is that it requires high computation time to compute the result [90].

Although PSO shows good performance in solving the task matching problem [237], it can converge prematurely \[^1\]. As stated earlier, this is because the movement of the particles and the solution improvement depend on the local and global best values. PSO can be stuck at the local minimum if both local and global best are located at the same local minimum point [25; 221]. Learning techniques and hybrid methods can enhance PSO performance to overcome this drawback.

### 3.4 Enhancing PSO algorithm

This section presents two popular methods to improve the performance of the PSO algorithm. The first method is to hybridize the PSO algorithm with other meta-heuristic algorithms, such as GA. The second method is to embed a learning mechanism into the PSO algorithm.

#### 3.4.1 Hybrid Approach

Over the past few years, numerous algorithms have been proposed. The algorithms, however, do not follow a specific standard meta-heuristic paradigm. These algorithms combine different algorithmic elements of different algorithms. This type of algorithm is called the *hybrid meta-heuristic algorithm* [33]. The motivation of the

\[^1\]More details about the premature convergence problem are presented in Section 4.1
hybridization method is to exploit various complementary characteristics of different algorithms whereby a synergy between different components of algorithms is established to achieve better performance. One such example is hybridizing the GA and PSO algorithms to work together where GA tends to exploit the search space, and PSO, in contrast, explores the solution space [274].

Zahedani and Dastghaibyfard [270] studied hybrid PSO-GA algorithm for the the task matching problem on grid. In their algorithm, the initial population was generated using the Min-Min algorithm. This was then passed to the GA and then back to PSO to improve the solutions. This is done iteratively in the algorithm. They show that this enhancement improves performance compared to Min-Min, Max-Min and discrete PSO.

Higashino et al. [114] incorporated GA only when there is no improvement in gbest for several iterations. Using GA can help the PSO to jump out of the local optima. Also, GA may create new particles in regions that may not have been explored by the swarm. The authors showed that their algorithm, GAPSO, outperformed the pure GA and fuzzy PSO [166].

Despite hybrid approaches can provide better results compared to GA and PSO, the hybrid method still has its limitations. One notable drawback of this approach is a conflict of the characteristics, i.e., GA and PSO have different features in nature. GA explores the solution space through the crossover operator while PSO uses past best experience mechanism to direct the search of the swarm. As a result, this may affect each other since they utilize different search mechanisms. For example, acceptable positions in PSO may become inefficient after the crossover, as the crossover in GA
Table 3.1: Pros and cons of different approaches.

<table>
<thead>
<tr>
<th>The approach</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heuristic algorithm</td>
<td>Simplicity</td>
<td>Problem specific</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mostly greedy constructive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tend to stuck at local minimum</td>
</tr>
<tr>
<td>Genetic Algorithm (GA)</td>
<td>Better global search</td>
<td>Long time to converge to global optimum</td>
</tr>
<tr>
<td></td>
<td>Less susceptible to be trapped at local minimum</td>
<td></td>
</tr>
<tr>
<td>Particle Swarm Optimization (PSO)</td>
<td>Converge fast to local optimum</td>
<td>Premature convergence</td>
</tr>
<tr>
<td>Hybridization with GA</td>
<td>Help PSO to jump out of local minimum</td>
<td>The crossover may delay the convergence</td>
</tr>
<tr>
<td></td>
<td>Better exploration search</td>
<td></td>
</tr>
</tbody>
</table>

is purely blind. Therefore, a careful design of how these algorithms should interact with each other is imperative.

Table 3.1 summarizes the strengths and weaknesses of the previously mentioned approaches in the literature for task matching problem.

### 3.4.2 Learning Approach

As mentioned earlier in Section 2.4, two information are utilized in the PSO algorithm to search the solution space. However, these two information are not needed when applying the learning approach. In other words, the learning approach tries to make use of the personal best mostly to create an *exemplar* for each particle whereby the particles are guided in the solution space. Liang et al. [159] designed comprehensive Learning PSO (CLPSO), where particles learn only from *pbest*. CLPSO generates a random number for each particle dimension. Then, it defines a probability learning value, \( P_{c_i} \), for each particle. If \( P_{c_i} \) is larger than the random number, the parti-
cle will learn from its $p_{best}$. Otherwise, a tournament selection method is applied to choose a particle to learn from. By learning from different particles, the swarm diversity increases.

Lynne and Suganthan [172] proposed Heterogeneous Comprehensive Learning PSO (HCLPSO) to enhance CLPSO by introducing an exploitation-enhanced sub-population. The whole population is divided into two different groups where each group solely focuses on either exploration or exploitation process. The information exchange between the subgroups is eliminated to avoid any redundant information between particles. Hence, the balance between the exploration and the exploitation is obtained.

Recently, Lin et al. [160] enhanced the CLPSO algorithm through adopting a dynamic probability learning $P_{ci}$. Moreover, to improve the diversity, a cooperative archive is used. The archive can provide more useful information from inferior particles, which helps to direct the swarm toward a promising region. The proposed algorithm, referred to as ACLPSO-CA, showed superiority compared to seven PSO variant algorithms, including HCLPSO and CLPSO algorithms. Several works have tried to improve the CLPSO algorithm, such as parallel comprehensive learning particle swarm optimizer (PCLPSO) [111], improved CLPSO (ICLPSO) [243], and fuzzy-controlled CLPSO (FC-CLPSO) [211].

Zhan et al. [272] applied orthogonal learning in PSO (OLPSO). Through the orthogonal experimental design, PSO can explore more within the search space to find more information. Also, orthogonal learning needs a small number of samples to be examined. Hence, it constructs more promising exemplars. They incorporated or-
orthogonal learning PSO algorithms such as OLPSO with global topology (OLPSO-G) and OLPSO with local topology (OLPSO-L). The algorithms are tested on sixteen numerical benchmark functions. The results showed that both algorithms are substantially better than other PSO algorithms in most of the test functions.

Goan et al. [109] presented genetic learning PSO (GLPSO), where a candidate solution is created for every particle in PSO by operators from the genetic algorithm. For every dimension, $d$ in particle $P_i$, a random particle, $P_j$, is chosen. If the fitness value of the given dimension at the random particle, $P_j$, is better than the dimension of that in the particle, $P_i$, under consideration, the value of the dimension of the random particle is chosen as the candidate solution. Otherwise, the particle, $P_i$ performs a crossover with $gbest$. A mutation operator is then applied to the candidate solution vector. Again, for each dimension, a random number is generated. If the random number is smaller than the mutation probability, then the dimension value is changed to another value. Finally, based on the fitness value, the offspring or candidate solution is chosen for the next generation. Ultimately, the resultant candidate solution is passed the PSO instead of the $pbest$, and $gbest$ values.

More recently, Lin et al. [161] proposed further improvements into the GLPSO algorithm, called GGL-PSOD. They adopted the ring topology to maintain the swarm’s diversity and adjust the parameters linearly. They propose three algorithms, GLPSO with ring topology, GLPSO with linear parameters adjustment, and the third is to incorporate both methods into GLPSO. The GLPSO with the two methods exhibited better performance against seven variants PSO algorithms.

A work by Xu et al. [255] presented a dimensional learning strategy (DLS) to

\footnote{Section 4.3 presents the GLPSO algorithm elaborately.}
discover and extract valuable information in the swarm. In DLS, each particle tries to learn from $gbest$ dimension by dimension. Thus, useful information in $gbest$ is extracted and to construct the candidate solution. The candidate solution combines dimensions of $pbest$ with some other dimensions learned from $gbest$. To improve the global search of the population, the authors introduced a two-swarm learning particle swarm optimization algorithm based on different learning strategies (TSLPSO), as learning from only $gbest$ lead all particles to move near $gbest$ with more exploitation search tendency. The TSLPSO is composed of two sub-populations with two different learning strategies. That is, One sub-population uses DLS while the other sub-population follows the comprehensive learning proposed by Liang et al. [159] to maintain the trade-off between exploitation and exploration search in the population. The TSLPSO manifested a promising performance compared to other PSO algorithms, such as GLPSO, HCLPSO, and OLPSO. There are variety of learning strategies presented in the literature, for instance [154; 202; 53; 173; 49].

**Learning strategies for multi-objective optimization:** Some learning strategy based PSO algorithms have been extended for multi-objective optimization problem. Huang et al. [119] adopted the CLPSO algorithm to handle multi-objective problems. Selecting $pbest$ and $gbest$ is changed as there are a set of non-dominated solutions instead of one global solution. To update $pbest$, each particle checks if the new solution after updating the velocity and the position dominates the $pbest$, and if so, the $pbest$ is set to the new solution. In case that both solutions are not dominated each other, a random number is generated and if the random number is great than 0.5, the $pbest$ is set to the new solution; otherwise, the $pbest$ remains with no updating. Since there is
no *gbest* in multi-objective problems, each particle randomly selects one solution from the external archive. The external archive is used to keep track of the non-dominated solutions found during the search process. The proposed algorithm converged faster while keeping a good diversity compared with MOPSO [65] and NSGA2 [74].

Recently, Liu et al. [168] developed co-evolutionary PSO (CPSO) where multiple swarms co-evolve concurrently. In other words, each swarm tries to optimize one objective at a time. To locate any missing areas in the Pareto front, CPSO employed learning strategy such that each swarm can learn from other swarms through crossover and mutation operators. The CPSO algorithm showed better performance compared to state-of-the-art multi-objective optimization algorithms in terms of diversity and convergence.

### 3.5 Summary

In this chapter, a comprehensive literature review about the current state of the task matching problem was presented. The chapter presented a taxonomy of the current approaches that have been used for task matching problem, namely, heuristic and meta-heuristic approaches. The chapter then reviewed some important works for task matching problem using heuristic methods. Heuristics usually are greedy and problem-specific, which limit their applicability. Next, meta-heuristic approaches, GA and PSO are presented. These standard techniques have been applied to many optimization problems such as task matching problem. Throughout the literature, PSO showed its superiority over GA in many optimization problems, including task matching problem; yet, the PSO algorithm still suffers some limitations, such as
premature convergence. Hybridizing the PSO algorithm with GA seems to improve the PSO through increasing exploration search. The drawback of the method is the lack of consistency between the natural characteristics of PSO and GA algorithms. Finally, the chapter reviewed in-depth learning strategies that have been incorporated into PSO algorithms. Many PSO algorithms with learning strategy have hitherto been used exclusively on numerical benchmark tests. Therefore, it can be identified that there is a huge gap in research using the learning strategy for other practical optimization problems, such as the task matching problem. This chapter, along with Chapter 2, establishes the base of the thesis by providing the necessary background information for the subsequent chapters.
Chapter 4

Modified Genetic Learning Particle Swarm Optimization for Task Matching Problem

As stated in Chapter 3, PSO algorithm has been applied to combinatorial optimization problems. However, the algorithm exhibits some limitations, such as premature convergence and loss of diversity around the local minimum. In this chapter, we propose a Modified Genetic Learning PSO (MGLPSO) technique to enhance the search ability of the PSO algorithm. The experimental parameter settings and data sets used to evaluate the proposed algorithm is described in Section 4.4. Section 4.5 demonstrates the results, and elaborately discusses the quality of the solutions. Finally, a summary of the chapter is presented in Section 4.6.
4.1 Drawbacks of the Existing Canonical PSO

The canonical PSO still suffers from significant drawbacks. The first issue are the parameters, \( p_{best} \) and \( g_{best} \) used to update the particle position. These parameters are either on opposite sides or on the same side of the current location of the particle. If \( p_{best} \) and \( g_{best} \) parameters are on opposite side, as depicted in Figure 4.1, a phenomena called “oscillation phenomenon” may occur [272; 202; 257]. In particular, the social term \((g_{best} - x_i)\) in Equation 2.11 is usually more significant than the cognitive term \((p_{best} - x_i)\), thereby \( g_{best} \) contributing to a substantial momentum to the particle’s position. Similarly, as the particle moves towards \( g_{best} \), the cognitive term gradually becomes larger causing the particle to move in its own direction. This delays convergence. On the other hand, if both \( g_{best} \) and \( p_{best} \) are on the same side, the particle will move in the same direction. Also, there is a chance that the parameters may be unable to move away from local optimum if they reach the same region simultaneously.

The second issue is the influence of \( g_{best} \) parameter on the entire swarm [237]. Variable \( g_{best} \) values gradually affects the search behavior of the swarm by assimilating with other particles. As a result, the swarm moves toward \( g_{best} \), and the swarm’s diversity is lost. The third issue is when the particle’s position is updated. This causes some dimensions of the particle to improve while others to deteriorate, causing a slow convergence. This phenomenon is called “two steps forward, one step back phenomenon” [237].

To overcome the above mentioned challenges, many variations of PSO have been proposed in the literature. Some works [185] focused on adaptive parameters, while
others explored the network topology of PSO [167]. Still other studies introduced and incorporated a learning strategy [53] into the PSO algorithm. This thesis focuses on the learning strategy. To our knowledge, incorporating learning technique to PSO for solving the task matching problem has not been proposed in the literature.

4.2 Behavioral Genetics in Biology

Most of the meta-heuristic algorithms have a foundation that is linked to biology directly. In biology, the genotype is composed of a set of genes that defines a particular trait. The phenotype represents the observable characteristics of traits, such as color, size, and behavior. The phenotype in an organism is influenced through its genotypes as well as environmental factors and experiences. In the past decade there has been tremendous amount of research [144] on the organism’s behavior either through genes or environment. This led to the “Nature versus Nurture” theory attributed to psychologist Francis Galton in 1869 [70; 178]. In this context, nature refers to biological/genetic influence on traits, while nurture expresses the influence of the environment through learning or other activities. Since then, research on the importance of genetics and environment on the organism’s behavior has continued until the first book published by Fuller and Thompson in 1960 [100] that announced the area of “Behavioral genetics.” There is a wide acceptance among genetic researchers on the role of both genes (genotypes) and the environment in shaping the differences in organism’s behavior [196; 27]. That is, genes by itself do not have any control over the behavior without an external interaction with the environment. Via genes, a framework is created wherein the environment influences morphological and physi-
Figure 4.1: Oscillation phenomenon in PSO algorithm

Figure 4.1: Oscillation phenomenon in PSO algorithm

ological developments. For example, genes create the platform for learning, memory, so on, that can be used in the environment by the organisms to search and acquire different information that, in turn, shape the behavior. Sterelny [225] stated that
learning might result in evolutionary consequences on organisms. The author gave an example of Woodpecker finches that learn how to use cactus spines to harvest food. Birds whose beak shape adopts to the use of the spines are more likely to be chosen through “natural selection”. As a result, birds do not only acquire the skill of using spines but also inherit a specific beak shape.

In genetic algorithm (GA) and particle swarm optimization (PSO), the phenotype expresses the candidate solution while the value in each dimension represents the genotype, as mentioned in Section 2.3. Modifying the phenotype in GA is different from that of PSO. GA uses different combinations of genotypes using crossover and mutation operators. PSO changes the phenotype via an interactive learning mechanism. In this thesis, we are bringing together the behavioral concept of GA into PSO. The next section explains the details of the proposes algorithm.

### 4.3 Modified Genetic Learning PSO (MGLPSO)

The proposed modified genetic learning PSO (MGLPSO) method is based on the work of GLPSO by Gong et al. [109]. GLPSO has two main phases. The first phase is to create an exemplar (candidate solution) for each particle. That is, there is one exemplar associated with each particle. For each dimension $d$ of exemplar $E_i$, a random particle $k$ is chosen randomly to be compared with the current particle. If $pbest$ of the current particle $i$ is better than $pbest$ of the random particle $k$, a crossover operation is performed between $gbest$ and $pbest$ of particle $i$. The crossover here is an arithmetic crossover, since particles use continuous values. Otherwise, the $d^{th}$ dimension of $pbest$ of particle $k$ is copied into the $d^{th}$ dimension of exemplar $E_i$. After the crossover oper-
Chapter 4: Modified Genetic Learning Particle Swarm Optimization for Task Matching Problem

...ation is completed, a mutation operation is implemented on the generated exemplar. The mutation operation is similar to that of GA. A random number is generated for each dimension of the exemplar. If the random number is less than the mutation probability, the mutation operation is applied to that dimension. The mutation operation here means re-initializing the value of the \( d^{th} \) dimension randomly. Otherwise, the value is kept unchanged. The last operation is the selection operation. Here, the fitness of the generated exemplar is compared with that of the current exemplar of the particle. The new exemplar then replaces the current exemplar if the fitness value of the latter is inferior to that of the new exemplar. Otherwise, the new exemplar is discarded. If there is no change in the current exemplar for a certain number of generations, a tournament selection operation is performed. Various particles are selected randomly, and the fittest among them are chosen to replace the current exemplar. Next, the exemplars are used to guide the particles in the PSO phase.

For our proposed method, MGLPSO, \( n \) particles and \( m \) tasks are assumed; the number of tasks \( m \) equals the number of dimensions \( d \). Hence, each particle is represented with \( m \) dimensions. Particles are initialized randomly. Similar to GLPSO, MGLPSO consists of two phases. In the first phase, for each dimension of a particle, crossover and mutation operations are performed to create exemplars. In the second phase, the exemplars obtained from the genetic learning phase are used in the PSO algorithm instead of \( gbest \) and \( pbest \). Figure 4.2 illustrates the MGLPSO algorithm.

In the following sections, we show how the algorithm is designed to represent the task matching problem.
Chapter 4: Modified Genetic Learning Particle Swarm Optimization for Task Matching Problem

Figure 4.2: Flowchart of the MGLPSO algorithm.
4.3.1 Task Matching Particle Solution Representation

The task matching problem is a discrete problem. Thus, many methods, such as integer PSO [28], ranked-order value (ROV) [164], rounding the continuous value to integer value PSO [197] and the smallest value position (SPV) [231], have been introduced to transform continuous PSO values into discrete values. In this thesis, the rounding method is adopted. Since each particle represents a candidate solution, we choose the direct solution representation for the task matching problem. Each particle $i$ is a vector of size $d$ dimensions, where $d$ is the number of tasks. That is, the index of the vector corresponds to a task $T_i$. The value of each dimension represents a particular resource $R_j$. Figure 4.3 illustrates the representation of one particle. In the illustrative example, task 1 (T1), task 4 (T4), and task 5 (T5) are matched to resource 1 (R1); task 3 (T3) and task 6 (T6) are matched to resource 2 (R2), and so on.

\[
\begin{array}{cccccccc}
T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & T_7 \\
R_1 & R_3 & R_2 & R_1 & R_1 & R_2 & R_3
\end{array}
\]

Figure 4.3: Particle Representation

4.3.2 Phase 1: Genetic Learning

The genetic learning scheme implements genetic operators (crossover, mutation and selection) to ensure that PSO does not converge prematurely. At the end of this phase, a pool of exemplars of high quality is created that is further refined through PSO. The arithmetic crossover in GLPSO is performed only by considering
the overall fitness function of $pbest_i$ and $pbest_k$. This may lead to crossover between poor dimensions, which may degrade the quality of good particles and result in a poor solution. The crossover part has been modified for use with the task matching problem as explained below. The values of dimension $d$ of both particles correspond to a resource. First, the continuous value in each dimension is converted to a discrete value by rounding it to represent the resources. Next, if the fitness value (the execution time) produced by $x_{i,d}$ is inferior to that of $x_{k,d}$, then the $d^{th}$ dimension of exemplar $E_{i,d}$ is the crossover of $x_{i,d}$ and $x_{k,d}$. If the fitness value is the same or better, then there is no need to learn from $x_{k,d}$, and the value of $x_{i,d}$ is copied into exemplar $E_{i,d}$. Equation 4.1 shows the crossover operation in our MGLPSO algorithm:

$$E_{i,d} = \begin{cases} 
\alpha * x_{i,d} + (1 - \alpha) * x_{k,d}, & f(x_{k,d}) < f(x_{i,d}) \\
\alpha, & \text{otherwise}
\end{cases}$$ (4.1)

where $\alpha$ is a random uniformly distributed value between $[0,1]$.

To illustrate, consider the example shown in Figure 4.4. At $d = 3$, $x_{i,d} = 5$ and $x_{k,d} = 2$ after the rounding process. Now, the fitness function is calculated for both
Chapter 4: Modified Genetic Learning Particle Swarm Optimization for Task Matching Problem

$x_{i,d}$ and $x_{k,d}$ for task 3. In other words, the execution times of task 3 on resource 5 and resource 2 are evaluated. If the execution time of task 3 on resource 5 is shorter (better) than that of resource 2, then the value of $x_{i,d}$ is copied into exemplar $E_{i,d}$. Otherwise, the crossover operation is performed using Equation 4.1, and the result is stored in exemplar $E_{i,d}$. The process is continued for each dimension $d$. By doing so, useful information about different aspects of matching is extracted for the tasks. As a result, better matches remain intact and are forwarded to the next generation. Moreover, the crossover in Equation 4.1 is comparatively guided and will be performed only if a particle needs to improve poor fitness in some $d$ dimensions.

The mutation process is similar to that in the standard GA. A random number between $[0, 1]$ is created for each dimension of the exemplar. If the random number is less than mutation probability $Pm$, then the dimension is reinitialized. The reinitialized value is a random value in the search space. The selection operation is the last operation. The newly created exemplar $E_i$ is compared with the best exemplar $e_i$ of particle $i$. Here, the fitness value (the makespan) is calculated for both exemplars. Each exemplar represents a complete solution. The makespan is calculated for all tasks. Exemplar $e_i$ will remain if its makespan is better than that of the exemplar $E_i$. Otherwise, it is replaced by exemplar $E_i$. The selection operation ensures that the particle improves in every generation. Moreover, if there is no change in the exemplar of a particle for a certain number of iterations, a stopping gap $sg$, of generations, then this implies that the particle is stuck at a local optimum. To overcome this problem, a selection tournament method is used. Some of the exemplars of other particles are selected for the tournament. the the winner is the exemplar with the best fitness
value (makespan) among those considered. By learning from this exemplar, the particle changes the search direction and moves away from the local optimum region. Algorithm 3 shows the genetic learning phase.

### 4.3.3 Phase 2: PSO Algorithm

The selected exemplars in phase 1 are used to guide the particles in PSO. Equations 2.11 and 2.10 are modified in the MGLPSO algorithm. In learning approach, the social term is discarded, as the social term may turn the particle’s direction significantly toward poor solution regions if $g_{best}$ is located far from the global optimum, and particles may become trapped in local optima. The modified version for updating positions of the particles are used frequently in learning strategy [109; 272; 158; 172]. Equations 4.2 and 4.3 describe the modified updating velocity that incorporates the selected exemplars $e_i$ in phase 1:

$$
v_i^{d+1} = \omega * v_i^{d} + c * r^d(e_i^{d} - x_i^{d}) \quad (4.2)$$

$$
x_i^{d+1} = x_i^{d} + v_i^{d+1} \quad (4.3)$$

where $v_i^{d}$ is the velocity value in $d^{th}$ dimension. $r^d$ is a random number between $[0, 1]$ generated for the $d^{th}$ dimension. Factor $c$ is the acceleration coefficient. $e_i^{d}$ is the value of the exemplar at $d^{th}$ dimension. $x_i^{d}$ is the position value of the particle in $d^{th}$ dimension. $\omega$ is the inertia weight. Algorithm 4 illustrates the PSO phase.
Algorithm 3 Genetic learning phase

1: /*Crossover operation*/
2: for \( d = 1 \) to \( D \) do
3: randomly pick a particle \( k \in \{1, 2, 3, ..., N\} \)
4: calculate \( f(x_{i,d}) \)
5: calculate \( f(x_{k,d}) \)
6: if \( f(x_{k,d}) < f(x_{i,d}) \) then
7: \( E_{i,d} = \alpha \times x_{i,d} + (1 - \alpha) \times x_{k,d} \)
8: else
9: \( E_{i,d} = x_{i,d} \)
10: end if
11: end for
12: /*Mutation operation*/
13: set \( pm \) to 0.001
14: for \( d = 1 \) to \( D \) do
15: generate random number \( r \)
16: if \( r < pm \) then
17: \( E_{i,d} = \text{rand}(1, R) \)
18: end if
19: end for
20: /*Selection operation*/
21: calculate \( f(E_i) \)
22: if \( f(E_i) < f(e_i) \) then
23: \( e_i = E_i \)
24: end if
25: if \( E_i \) is not improving for \( sg \) iterations then
26: set \( S \) based on the tournament selection method
27: \( e_i = S \)
28: end if
4.3.4 Fitness Function

Many criteria could be considered to measure the solution (matching) quality. Makespan is one such measure. It is the maximum completion time of a set of resources. That is, given a set of resources, the completion time of the last resource to finish is the makespan. It measures the throughput of the system. A small makespan implies that the scheduler has produced an excellent and efficient schedule. One method of calculating the makespan is by using the ETC matrix, as mentioned in Chapter 1. The makespan Equation 4.4 is given below:

\[
\text{Makespan} = \max\{C_j\}, j = \{1, 2, 3, ..., m\}
\]  

The primary objective in the task matching problem is minimizing the makespan.
4.3.5 Complexity Analysis of the MGLPSO Algorithm

In many meta-heuristic algorithms, the time complexity depends on four entities: the problem dimensions $D$, the size of population $N$, the cost of the fitness function $cf$, and the number of iterations $MAXit$. The proposed algorithm has two phases: the GA learning phase, and the PSO phase. During the genetic learning phase, the crossover is checked for every dimension. The execution time of a task on a given resource can be checked in constant time $O(1)$. This checking is done for all dimensions, so the complexity of crossover operation is $O(D)$. Similarly, the mutation is checked for every dimension; hence, the mutation operation takes $O(D)$. Finally, the selection operation takes $O(D)$ to calculate the fitness function $cf$, i.e., the makespan in this thesis, for exemplar $E_i$. Moreover, the tournament size method requires $O(ts)$, where $ts$ is the size of the tournament. Adding all these operations together yields $O(D + D + D + ts) = O(3D + ts) = O(D + ts)$ for the genetic learning phase. The computation part of the PSO phase involves updating the position and velocity of the particles, which takes $O(D)$. Additionally, the complexity of computing the fitness function $cf$ is $O(D)$. Then, the complexity of the PSO phase has the order of $O(D)$. Thus, MGLPSO’s time complexity is $O(D + ts) + O(D) = O(D + D + ts).N.MAXit = O(D + ts).N.MAXit$.

4.4 Experimental Design

The data set from Braun et al. [39] is used to evaluate the performance of MGLPSO algorithm. This data set has been frequently used as a benchmark in the literature.
The data set is composed of three different main data sets (scenarios). Each data set has four different categories; hence, there are twelve data sets in total. These data sets are represented through ETC matrices and they vary based on three different parameters: task heterogeneity, resource heterogeneity, and matrix consistency.

Task and resource heterogeneity can be either high or low; “high” implies that the variation in task and resource characteristics is high, while “low” implies that the variation in characteristics of tasks and resources is small. Additionally, for the table in Section 4.5.1, task and resource heterogeneity are represented in the form of tuples x-y. Here, “x” indicates task heterogeneity, and “y” denotes resource heterogeneity. For instance, “lo-lo” means a low task and low resource heterogeneity.

The ETC matrix can be consistent, inconsistent, or partially-consistent. The matrix is consistent if resource $R_i$ executes every task $T_j$ faster than all other resources $R_k$ for $k > i$. That is, resource $R_i$ is the fastest resource in the Grid system. The inconsistent ETC matrix has no such order in terms of performance (execution time) among the resources. Thus, any resource can be fast for some tasks and slow for some other tasks. The partially-consistent ETC matrix is a mixture of inconsistent and consistent orders, i.e., the ETC matrix preserves order for some resources on some tasks.

A variation of these twelve different data sets were generated, and each data set was generated twenty times, making the number of generated data sets more than 200 data sets for the experiments. The experiments were implemented in Python on a PC featuring an AMD A8-3870 APU operating at 3.0 GHz with 8 GB of RAM. The number of iterations (generation) was set to 500 for every data set. All initial
Chapter 4: Modified Genetic Learning Particle Swarm Optimization for Task Matching Problem

solutions were generated randomly. The population size for all algorithms was set to 40 particles. Table 4.1 shows the parameter settings for the MGLPSO algorithm.

Table 4.1: MGLPSO Parameter settings.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mutation Probability</td>
<td>0.001</td>
</tr>
<tr>
<td>Selection Operator</td>
<td>Tournament</td>
</tr>
<tr>
<td>Tournament Size</td>
<td>8</td>
</tr>
<tr>
<td>Cognitive Coefficient ((C))</td>
<td>1.4962</td>
</tr>
<tr>
<td>Inertia Weight</td>
<td>0.7298</td>
</tr>
</tbody>
</table>

4.5 Results and Discussion

To thoroughly evaluate the effectiveness of the MGLPSO algorithm, two different sets of experiments were conducted. The first set was to evaluate the solution quality obtained by the MGLPSO algorithm. For comparison, PSO algorithm [210] and GLPSO algorithm [109] were chosen. Moreover, in the first set, four criteria were evaluated: makespan, convergence rate, exemplar’s quality, and diversity. For the makespan experiment, the algorithms’ scalability were examined. Three data sizes were used in the experiment: small, medium, and large. 20 tasks on 3 resources, 64 tasks on 8 resources, and 512 tasks on 16 resources were chosen for the small, medium, and large sizes, respectively. The rest of the experiments considered only large data size. The second set of experiments measured the impact of different
parameter settings since these parameters are problem-dependent. These parameters include mutation rate, population size, and the tournament size.

### 4.5.1 Makespan

Table 4.2: Statistical results of PSO, GLPSO and MGLPSO.

<table>
<thead>
<tr>
<th>Parameter Setting</th>
<th>PSO</th>
<th>GLPSO</th>
<th>MGLPSO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>lo-lo</td>
<td>lo-lo</td>
<td>lo-lo</td>
</tr>
<tr>
<td><strong>Inconsistent</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Small</td>
<td>1388.16</td>
<td>1368.63</td>
<td>1268.15</td>
</tr>
<tr>
<td>Medium</td>
<td>1630.5</td>
<td>1436.46</td>
<td>1425.52</td>
</tr>
<tr>
<td>Large</td>
<td>858.17</td>
<td>960.41</td>
<td>3565.71</td>
</tr>
<tr>
<td><strong>Consistent</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Small</td>
<td>1378.83</td>
<td>1338.21</td>
<td>1425.52</td>
</tr>
<tr>
<td>Medium</td>
<td>1707.62</td>
<td>1657.74</td>
<td>1689.03</td>
</tr>
<tr>
<td>Large</td>
<td>8561.7</td>
<td>7466.05</td>
<td>8201.17</td>
</tr>
<tr>
<td><strong>Partially-consistent</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Small</td>
<td>1856.4</td>
<td>1565.34</td>
<td>1561.52</td>
</tr>
<tr>
<td>Medium</td>
<td>1871.18</td>
<td>1750.98</td>
<td>1796.01</td>
</tr>
<tr>
<td>Large</td>
<td>9563.15</td>
<td>780.08</td>
<td>4866.36</td>
</tr>
</tbody>
</table>

Table 4.2 presents the average makespan obtained by the three algorithms for all problem instances. For small data size, MGLPSO was able to produce less makespan compared to the PSO and GLPSO algorithms. MGLPSO was able to achieve around 5% and 3% better solution quality than the PSO and GLPSO, respectively. As the data size was increased, MGLPSO showed more notable results. For the medium data size, MGLPSO obtained better results than the PSO algorithm by 28%. Also, MGLPSO was able to minimize the makespan by 16% compared with GLPSO. With large data size, the makespan was decreased significantly by 52% and 43% in comparison with PSO and GLPSO, respectively. The inefficiency of the PSO was evident as data size increased. This is due to the fact that PSO stagnated at a local minimum. In the GLPSO and MGLPSO, the crossover operator is crucial. The crossover in the GLPSO takes place only when the particle has a good fitness value (good match),...
which may impact the improvement of the particle. MGLPSO allows every particle to learn from others dimensions when needed. Therefore, it results in better performance. Figure 4.5 depicts the mean results of all problem for PSO, GLPSO, and MGLPSO.

Although MGLPSO was able to minimize the makespan dramatically (for large data size), its performance was inferior to GLPSO in the consistent data set. As seen in Table 4.2, GLPSO was more efficient than MGLPSO for small consistent data set. With increasing data size, MGLPSO outperformed GLPSO in some cases when the resource heterogeneity was high. This is because of the nature of the consistent data set. In MGLPSO, fast resources get more tasks while slow resources get a few tasks, which yielded load resource imbalance.
4.5.2 Convergence Rate

It is important to measure the speed of obtaining the global optimum. As shown in Figure 4.6, the average makespan was observed in every 50 iterations for all algorithms. That is, MGLPSO required around 50 iterations to converge to global optimum whereas GLPSO and PSO needed more iterations. It is evident that MGLPSO needed fewer iterations to find accurate results compared with PSO and GLPSO. The PSO algorithm stagnated after 200 iterations. Also, from Figure 4.6, a significant difference was observed in the makespan between MGLPSO and the other algorithms (GLPSO and PSO). Hence, MGLPSO can reach to high-quality results with less number of iterations compared with PSO and GLPSO algorithms.
(a) high task heterogeneity and high re-
(b) low task heterogeneity and high re-
source heterogeneity.

(c) low task heterogeneity and low re-
(d) high task heterogeneity and low re-
source heterogeneity.

Figure 4.7: Exemplar’s quality of MGLPSO, and GLPSO in the inconsistent dataset.

4.5.3 Exemplar Quality

This experiment aims to identify the search behavior of MGLPSO. As mentioned earlier in section 4.3.2, the generated exemplars from the genetic learning phase are used to guide the particles in the search space. The quality of the exemplar is vital for particles to search for global optimum efficiently. Exemplar’s quality was observed for a single run. The generated exemplars from GLPSO and MGLPSO for inconsistent
data set are shown in Figure 4.7. With crossover implemented in MGLPSO, multiple particles learned from other particles to produce their exemplars. As a result, these produced exemplars were much more diverse in the search space. Exemplars generated by MGLPSO were significantly better than GLPSO for inconsistent problem. A further observation from Figure 4.7 is that exemplars were improving, as the number of the iterations increased. Incorporating different information from different particles helped to create better exemplars in MGLPSO compared to GLPSO.

Figures 4.8 and 4.9 show exemplars quality of MGLPSO and GLPSO for partially-consistent and consistent data sets, respectively. Apropos of high resource heterogeneity, the gap between MGLPSO and GLPSO was apparent. The obtained makespan of MGLPSO was better than GLPSO due to exemplars’ quality. Additionally, the quality of exemplars of MGLPSO was improving, as the number of iterations increased. The improvement rate, however, in the consistent data set was slower compared to the partially-consistent data set. When the resource heterogeneity was low, MGLPSO seemed to be inefficient, especially, in the consistent data set. As shown in Figures 4.8c and 4.8d, exemplars’ quality of MGLPSO after 100 iterations had a small enhancement. In 4.9c and 4.9d, there was just a small improvement at the beginning where the swarm diversity was high. Then, for about 200 iterations, there was no enhancement due to no improvement in the quality of the exemplar. MGLPSO was not suitable to work perfectly on the consistent data set. This is due to considering one dimension at a time to construct the exemplars. This may lead to load imbalance as mentioned previously, and as a result, worsen the final makespan.
Chapter 4: Modified Genetic Learning Particle Swarm Optimization for Task Matching Problem

(a) high task heterogeneity and high resource heterogeneity. 
(b) low task heterogeneity and high resource heterogeneity. 
(c) low task heterogeneity and low resource heterogeneity. 
(d) high task heterogeneity and low resource heterogeneity.

Figure 4.8: Exemplar quality of MGLPSO, and GLPSO in the Partially-consistent dataset.

4.5.4 Diversity

The algorithm’s ability to balance the trade-off between the exploration or exploitation search can affect the overall performance. To overcome premature convergence, balancing between these two is essential. In this experiment, the swarm’s diversity was observed for a single run. Equation 4.5 was used to calculate the swarm’s
Chapter 4: Modified Genetic Learning Particle Swarm Optimization for Task Matching Problem

(a) high task heterogeneity and high resource heterogeneity.

(b) low task heterogeneity and high resource heterogeneity.

(c) low task heterogeneity and low resource heterogeneity.

(d) high task heterogeneity and low resource heterogeneity.

Figure 4.9: Exemplar’s quality of MGLPSO, and GLPSO in the consistent dataset.

diversity \[172\]:

\[
Diversity(S) = \frac{1}{|N|} \cdot \sum_{i=1}^{|N|} \sqrt{\sum_{d=1}^{D} (x_i^d - \bar{x}^d)^2}
\]  
(4.5)

\[
\bar{x}^d = \frac{\sum_{i=1}^{|N|} x_i^d}{N}
\]  
(4.6)
(a) high task heterogeneity and high resource heterogeneity.

(b) low task heterogeneity and high resource heterogeneity.

(c) low task heterogeneity and low resource heterogeneity.

(d) high task heterogeneity and low resource heterogeneity.

Figure 4.10: Swarm’s diversity of MGLPSO, GLPSO, and PSO for the inconsistent dataset.

where $N$ is the population size, $d$ is the size of problem dimension, $x_i^d$ is the value of the $d^{th}$ dimension of particle $i$, $\bar{x}$ is the average $d^{th}$ dimension over all particles.

The swarm diversity of MGLPSO, GLPSO, and PSO for the inconsistent data set is shown in Figure 4.10. The three algorithms started at a high diversity, indicating the exploration search. In all data sets (consistent, inconsistent, and partial-consistent), the PSO was not able to maintain a good diversity. Approximately, af-
Chapter 4: Modified Genetic Learning Particle Swarm Optimization for Task Matching Problem

85

(a) high task heterogeneity and high resource heterogeneity.
(b) low task heterogeneity and high resource heterogeneity.
(c) low task heterogeneity and low resource heterogeneity.
(d) high task heterogeneity and low resource heterogeneity.

Figure 4.11: Swarm’s diversity of MGLPSO, GLPSO, and PSO for the Partially-consistent.

After 200 iterations, particles in PSO became clustered, and seemingly got stuck into a local minimum. Also, from Figure 4.10, MGLPSO and GLPSO were able to manage between exploration and exploitation, with MGLPSO being more diverse than GLPSO at early iterations due to the $g_{best}$ and $p_{best}$ employed in the learning scheme of GLPSO were getting closer to each other at every iteration.

With respect to partially-consistent data set, shown in Figure 4.11, MGLPSO...
and GLPSO showed a good balance when the resource heterogeneity was high. Both MGLPSO and GLPSO managed the control between exploration and exploitation after 200 iterations. Besides, with low resource heterogeneity, both MGLPSO and GLPSO had less diversity compared to PSO. However, after approximately 150-200 iterations, the search in the PSO became exploitative with small diversity, while in both MGLPSO and GLPSO, the search was more diverse. Another observation was a notable increase in the MGLPSO’s diversity, and it can be seen in 4.11a and 4.11c. These hikes can be interrupted as the MGLPSO tried to jump out of local minimum area.

Figure 4.12 elucidates the diversity of the three algorithms when the data set is consistent. It was evident that MGLPSO tends to converge faster than PSO and GLPSO. In spite of this, MGLPSO had control over the diversity after 70-80 iterations. Another significant increase in diversity was noticed in Figure 4.12d conducted by MGLPSO. Generally, the diversity experiments revealed a shred of clear evidence that MGLPSO can deal with tricky optimization problem instances such as partially-consistent and consistent problems efficiently. Also, MGLPSO could manage between exploration and exploitation search, which aids the MGLPSO in escaping out of the local minimum.

4.5.5 Mutation Rate

The crossover operation is more exploitative process, as the generated exemplars are in the same region of the particles. Contrasting, the mutation operation tends to be a comparatively explorative process, and its effect is more prominent. Hence,
Chapter 4: Modified Genetic Learning Particle Swarm Optimization for Task Matching Problem

(a) high task heterogeneity and high resource heterogeneity.
(b) low task heterogeneity and high resource heterogeneity.
(c) low task heterogeneity and low resource heterogeneity.
(d) high task heterogeneity and low resource heterogeneity.

Figure 4.12: Swarm’s diversity of MGLPSO, GLPSO, and PSO for consistent dataset.

in this experiment, the impact of mutation rate was investigated. The mutation rate started at 0.001 and was gradually increased to 0.5. As evidenced by Figure 4.13, the solution’s quality was affected dramatically, as the mutation rate increased. The inconsistent data set appeared to be solved efficiently with a low mutation rate. A further rise of the mutation rate in case of the inconsistent data set yielded a deterioration of the solution’s quality; however, increasing mutation rate slightly seemed to be better for solving consistent and partially-consistent data sets. In Figure 4.13a–4.13b,
when resource heterogeneity was high, increasing the mutation rate improved the results of MGLPSO by reducing the makespan by 7.55% and 11.07% for the consistent data set. Further, the same was noted for the partially-consistent; the makespan was enhanced by 6.15% and 4.67% for low and high task heterogeneity, respectively. When resource heterogeneity was low, as shown in Figures 4.13c–4.13d, the improvement in the makespan was clear. MGLPSO minimized the makespan by 4.74% for low task heterogeneity and 9.81% for high task heterogeneity. Similarly, for the consistent data set, MGLPSO performed better, and the obtained makespan was reduced by 15.13% for low task heterogeneity. Furthermore, the makespan was decreased by 17.51% for high task heterogeneity.
Chapter 4: Modified Genetic Learning Particle Swarm Optimization for Task Matching Problem

4.5.6 Population Size

The objective of this experiment is to identify the influence on the solution quality through varying the size of the swarm. This experiment started with a small size swarm, 20 particles, and then was increased linearly to 100 particles in the swarm. As depicted in Figure 4.14, expanding the swarm size enhanced the solution quality of the MGLPSO, as more particles explore more regions. Nonetheless, this came at a
price, as the computation time increased with a bigger swarm. The improvement was about 7% on the makespan for 80 particles, while the computation time grew to 298%. Thus, there is a trade-off between the quality of the solution and the computation time. In MGLPSO case, setting the population size between 40 and 60 seemed to be perfect balancing between solution’s quality and the computation time. Going beyond 80 particles impacted the MGLPSO adversely as shown in Figure 4.14.

Figure 4.14: Effect of the population size on the makespan.

### 4.5.7 Tournament Size

The tournament selection is used to help the particle to fly out of the local minimum region. It is imperative to have a proper size for the tournament, as a small size tournament turns the selection to be a random selection, and an immense size makes all particle to fly toward the best particle. The size of the tournament set to
20% of the population size as suggested in [109]. The idea here is to give a chance for other exemplars to participate in the tournament and subsequently increase the diversity more. In this experiment, the population size was 100 particles. The size of tournament started with 10 exemplars and incremented it up to 80 exemplars. Figure 4.15 visualizes the obtained results. For the size of 10, MGLPSO did not seem to benefit from the tournament selection. However, increasing the size to 20, which is 20% of the population size, reduced the makespan by 18.40% as the exemplars in the MGLPSO are more diverse compared with the size of 10 exemplars. Interestingly, for 40 exemplars, there was a small reduction in the makespan, less than 1%. Increasing the size further to 60 exemplars minimized the makespan further by 12.41%. Any increasing in the size of the tournament more than 60 exemplars appeared to be not efficient, as the probability of the best exemplar entering the tournament was high. Hence, many exemplars started to use it to guide their particles and loss of diversity may happen. The experiment suggested that the tournament size between 20% to 60% of the population is better to maintain a good diversity with an acceptable solution quality.
4.6 Summary

This chapter introduced a modified genetic learning PSO (MGLPSO) to solve the task matching problem in Grid environments. Using $g_{best}$ and $p_{best}$ parameters to guide the particles in the search space may lead to premature convergence, and the particles get trapped in a local minimum. Also, the canonical PSO does not incorporate any information about the problem domain, and the time to find the global optimum as a result is high, especially for large-scale problems. Genetic learning technique can solve these issues via extracting useful information from different particles by generating high-quality exemplars through guidance. In the Genetic learning technique, genetic operators, such as crossover, mutation, and selection, are implemented.
for exemplar creation. The empirical results showed that the MGLPSO algorithm is more effective and efficient in most of the task matching problems compared with GLPSO and PSO. The results also showed that MGLPSO could manage and produce small makespan for large problem size more efficiently. Moreover, MGLPSO required a few iterations to find better solutions compared with GLPSO and PSO. Finally, MGLPSO managed an acceptable diversity with the ability to jump out of local optimum.

Nevertheless, the MGLPSO algorithm suffered on a specific data set, the consistent data set. MGLPSO was not able to balance the load of the resources for the consistent data set. In the next chapter, we propose two algorithms to tackle the weakness of the MGLPSO algorithm. Please note that the work done in this chapter has been published in [16].
Chapter 5

Genetic Learning PSO Algorithms for Single and Multi-objective Task Matching Problem

MGLPSO presented in Chapter 4 showed competitive performance with other techniques such as GLPSO and PSO. However, the algorithm did not work well for consistent data set. In this chapter, two further extensions for the MGLPSO algorithm are proposed: hybrid genetic learning PSO (HGLPSO) (Section 5.1) and modified genetic learning with adaptive mutation ($\mu$-MGLPSO) (Section 5.2) to tackle the weakness of MGLPSO. We propose these two extensions to not only consider single objective task matching problem but also to consider the more complicated multi-objective task matching problem (Section 5.3). While HGLPSO considers the balance between exploration and exploitation, $\mu$-MGLPSO integrates an adaptive mutation rate into the algorithms. In Section 5.5, the data sets and parameter settings
are introduced for single and multi-objective problems. Results of single and multi-objective are discussed in Section 5.6 and Section 5.7, respectively, followed by a summary in Section 5.9.

5.1 Hybrid Genetic Learning PSO (HGLPSO)

The key challenge in designing a meta-heuristic algorithm is in finding a balance between exploration (diversification) and exploitation (intensification) search. Inspired by the work by Lynn and Suganthan [172], we propose hybrid genetic learning PSO (HGLPSO) that finds this balance. This technique works on two different sub-populations as will be explained later. In comparison to MGLPSO, the HGLPSO algorithm also consists of two main phases: the genetic learning phase and the PSO phase. In the genetic learning phase, GA operators (crossover, mutation, and selection) are used to create the exemplars to direct the particles in the second phase.

The swarm is divided into two fixed subgroups. The first subgroup uses the crossover technique used in MGLPSO. The other subgroup uses the crossover operator similar to GLPSO algorithm. Therefore, the two subgroups have different depths of exploration and exploitation. The particles of the first subgroup (MGLPSO) learn from different particles’ dimensions directly. This, in turn, makes the particles in this subgroup tend to be more exploratory. Meanwhile, the particles in the second subgroup (GLPSO) learn from the particle’s pbest and gbest values for each dimension, which results in the particles being more exploitative. Hence, the swarms in the proposed HGLPSO algorithm are formed by balancing the two search techniques, exploration and exploitation. We divide the swarms such that 75% of the swarm are
placed in the first subgroup while the other 25% are placed in the second subgroup. This ensures that the swarm can attain a broader diversity with a good convergence rate \(^1\).

During the crossover operation, all particles in the swarm are compared with each other. There is no restriction on learning from the each other subgroups. Consequently, there is rapid information exchange between the two subgroups to expedite the convergence. For every dimension \(d\) of the corresponding particle \(p\), a random particle \(r\) is selected to create an exemplar similar to the process mentioned in GLPSO and MGLPSO. All particles in the first subgroup do the following for a particular dimension \(d_i\): if the fitness value (execution time) in \(p\) is better than the exemplar’s fitness value, then \(p\)’s value is copied to the exemplar. Otherwise, the current particle \((p)\) learns from the selected particle \((r)\) through genetic crossover, and the result is stored in exemplar’s \(d_i\).

Similarly, particles in the other subgroup follow the same steps explained above through genetic learning. However, in contrast to the first subgroup, particles use their \(pbest\) and the best particle of the swarm \(gbest\) in the crossover operation. We modify the crossover operation slightly. In GLPSO, crossover occurs if the overall fitness value (complete makespan) of \(pbest\) of the current particle is better than that of the \(pbest\) of the selected particle. In HGLPSO, we perform crossover when the fitness value of \(pbest\) of the current particle is inferior to that of \(pbest\) of the selected particle to retain the best matches discovered in particles. The \(d^{th}\) dimension of \(pbest\) of the current particle is saved in the \(d^{th}\) dimension of the exemplar if \(pbest\) of the current particle has a better fitness value (less makespan). Algorithm 5 illustrates

\(^1\)This division is based on results in Section 5.6.3.
the proposed HGLPSO algorithm.

For the second phase (PSO phase), HGLPSO simply follows the MGLPSO algorithm and Equations 4.2 and 4.3 are used to update the velocity and position of each particle, respectively.

5.2 Modified Genetic learning PSO with Adaptive Mutation ($\mu$-MGLPSO)

In the literature, it has been understood that the mutation operation has a significant impact on the solution quality of the standard GA [223]. The mutation operator is typically used to prevent the population from getting stuck at the local minimum and to have unique individuals in each generation. Thus, the mutation operation emphasizes an exploration search. Determining the value of the mutation rate ($pm$) is tedious and has been investigated a lot in the literature. Having a small $pm$ may cause premature convergence of the population, while a high mutation rate changes the algorithm to a purely random search. The mutation rate can consequently change the exploitation search into exploration search and vice versa. The mutation rate is typically fixed during the entire process. Nevertheless, a fixed mutation rate’s value seems to be inefficient for many complex optimization problems. Indeed, several studies have revealed that using a dynamic mutation rate is more efficient than using a fixed mutation rate [80; 118; 256].

Based on empirical experiments on the MGLPSO algorithm in Section 4.5.5, a low mutation rate seems to be effective in some problems, whereas a high mutation
Algorithm 5 Hybrid genetic learning PSO (HGLPSO) - part 1

1: divide the swarm into two subgroups

2: for $i = 1$ to $N$ do

3:   /*Crossover operation*/

4:   if $\text{particle}_i$ is the first subgroup then

5:     for $d = 1$ to $D$ do

6:        randomly pick a particle $k \in \{1, 2, 3, ..., N\}$

7:        calculate $f(x_{i,d})$

8:        calculate $f(x_{k,d})$

9:        if $f(x_{k,d}) < f(x_{i,d})$ then

10:       $E_{i,d} = \alpha \ast x_{i,d} + (1 - \alpha) \ast x_{k,d}$

11:      else

12:       $E_{i,d} = x_{i,d}$

13:     end if

14:   end for

15:   else

16:     for $d = 1$ to $D$ do

17:        randomly pick a particle $k \in \{1, 2, 3, ..., N\}$

18:        calculate $f(pbest_i)$

19:        calculate $f(pbest_k)$

20:        if $f(pbest_k) < f(pbest_i)$ then

21:           $E_{i,d} = \alpha \ast pbest_{k,d} + (1 - \alpha) \ast gbest_d$

22:        else

23:           $E_{i,d} = pbest_{i,d}$

24:        end if

25:     end for

26: end if
Algorithm 5 Hybrid genetic learning PSO (HGLPSO) - Part 2

27: /*Mutation operation*/
28: set \( pm \) to 0.001
29: for \( d = 1 \) to \( D \) do
30: generate a random number \( r \)
31: if \( r < pm \) then
32: \( E_{i,d} = \text{rand}(1,R) \)
33: end if
34: end for
35: /*Selection operation*/
36: calculate \( f(E_i) \)
37: if \( f(E_i) < f(e_i) \) then
38: \( e_i = E_i \)
39: end if
40: if \( E_i \) is not improving for \( sg \) iterations then
41: set \( S \) based on the tournament selection method
42: \( e_i = S \)
43: end if
44: end for

rate worked better for other problems. This observation motivates us to introduce an adaptive mutation method to be integrated into the MGLPSO algorithm. The proposed adaptive mutation method adjusts the mutation rate based on feedback gathered from the current population, i.e., particles near \( gbest \) are assigned a low mutation rate, while faraway particles are given a high mutation rate. The rationale behind assigning different mutation rate adaptively is that particles nearby \( gbest \) should intensify searching in the vicinity, and the particles that are located far away
from the gbest explore in the hopes of finding promising regions. Accordingly, by varying the mutation rate for all particles adaptively, the trade-off between exploration and exploitation can be attended. The proposed adaptive mutation method is based on the work by Srinivas and Patnaik [223]. Equation 5.1 integrates the adaptive mutation into MGLPSO:

\[
pm = \begin{cases} 
  pm_1 \times \frac{(f_{p_{best}} - f_{g_{best}})}{(f_{p_{best}} - f_{g_{best}})} & \text{if } avg_{p_{best}} \leq f_{p_{best}} \\
  pm_2 & \text{otherwise}
\end{cases}
\]

(5.1)

where \( pm_1 = 0.04 \) and \( pm_2 = 0.001 \).

In the adaptive mutation method, the average of the \( p_{best} \) values of the population is calculated. Then, based on the position (value) of \( p_{best_i} \), a particle \( i \) either explores or exploits the solution space. A position \( p_{best_i} \) located close to \( g_{best} \) is set to a low mutation rate. On the other hand, particles whose \( p_{best_i} \) is larger than the mean of all \( p_{best} \) are regarded as being far from \( g_{best} \), and a higher mutation rate is assigned adaptively. Hence, the closer \( p_{best_i} \) is to \( g_{best} \), a smaller mutation rate will be set. In Equation 5.1, the purpose of the term \( (ave_{p_{best}} - f_{g_{best}}) \) is to observe the diversity of the population to prevent premature convergence. The other expression \( (f_{p_{best}} - f_{g_{best}}) \) is to vary the mutation rate among the population based on the population’s \( p_{best} \) position. Algorithm 6 shows the proposed \( \mu \)-MGLPSO algorithm.

### 5.3 Multi-objective Task Matching Problem

Chapter 2 introduced an overview background on multi-objective optimization problems (MOPs). Most of the MOPs are nontrivial, i.e., there is no single solution
Algorithm 6 MGLPSO with adaptive mutation

1: /*Crossover operation*/
2: for $d = 1$ to $D$ do
3: randomly pick a particle $k \in \{1, 2, 3, ..., N\}$
4: calculate $f(x_{i,d})$
5: calculate $f(x_{k,d})$
6: if $f(x_{k,d}) < f(x_{i,d})$ then
7: $E_{i,d} = \alpha * x_{i,d} + (1 - \alpha) * x_{k,d}$
8: else
9: $E_{i,d} = x_{i,d}$
10: end if
11: end for
12: /*Mutation operation*/
13: calculate $ave_{pbest}$
14: set $pm$ according to Equation 5.1
15: for $d = 1$ to $D$ do
16: generate a random number $r$
17: if $r < pm$ then
18: $E_{i,d} = \text{rand}(1,R)$
19: end if
20: end for
21: /*Selection operation*/
22: calculate $f(E_i)$
23: if $f(E_i) < f(e_i)$ then
24: $e_i = E_i$
25: end if
26: if $E_i$ is not improving for $sg$ iterations then
27: set $S$ based on the tournament selection method
28: $e_i = S$
29: end if
that optimizes all objectives simultaneously [207; 157]. Thus, in many real-world problems, included task matching problem, having a single objective that captures all aspects is hard. Multi-objective optimization defines a set of objectives rather than one single objective to be optimized. The search space becomes partially ordered if different objectives are concurrently considered. A set of trade-off solutions have to be determined instead of one global solution [281]. There are some classical methods for solving MOPs [38]. These include,

1. Scalarization methods:
   - Linear scalarization [104].
   - $\epsilon$-constraint [268].

2. No-preference methods:
   - Method of global criterion or compromise programming [271].
   - Neutral compromise solution [247].

3. Priori methods:
   - Value function [137].
   - Lexicographic ordering [92].
   - Goal programming [47].

4. Posteriori methods:
   - Normal boundary intersection (NBI) [72].
   - Successive Pareto optimization (SPO) [183].
• Multi-objective evolutionary algorithms (MOEAs) [94; 203; 224].

5. Interactive methods [180].

The latter three (priori, posteriori, interactive) are the classical methods for solving MOPs. These techniques involve interfering with a decision maker (DM) to find "the appropriate final solution". The appropriate final solution here means the Pareto Optimal solution that is deemed by the DM as the best option.

The role of the DM providing preference information for objectives of the optimization is different in the three methods. In priori methods, the preference information is given before the optimization problem, e.g., weights of the objectives [88]. In posteriori methods, the information preference is provided after the optimization process; for instance, providing the final Pareto optimal solutions allowing the DM to choose the most satisfactory solution. Finally, the DM involves progressively in interactive methods during the optimization process. That is, a set of solutions is provided at the optimization stage, and then DM presents the information preference in preference elicitation phase. These two phases alternate until the preferred solution is found by the DM.

This thesis focuses on posteriori methods, in particular, MOEAs which is the scope of this thesis.

5.3.1 Multi-objective evolutionary algorithms (MOEAs)

Due to their applicability in many multi-objective optimization problems, MOEAs have been broadly acclaimed in the past decade. In fact, evolutionary algorithms (EAs) are better suited for complex optimization problems than the other methods
for the following two reasons [66]: (i) EAs are less susceptible to the shape of the Pareto frontier. That is, EAs find Pareto optimal set even if cases where the Pareto frontier is concave or discontinuous; (ii) EA’s population-based mechanism.

Most of the EAs produce a set of solutions simultaneously, which allows locating several solutions of the Pareto optimal in a single run. Different techniques have been introduced to be integrated with EAs to solve multi-objective optimization problems. These include, Lexicographic order, Scalarization, Pareto-based techniques\(^2\). In the Lexicographic order method, the DM arranges the objectives based on their importance. Therefore, the most important objective is optimized first. If the result is satisfactory to DM, then the optimization terminates. Otherwise, the second objective is considered to be optimized, such that there is no deterioration in the first objective. The Lexicographic method is inept when dealing with conflicting objectives. Moreover, arranging the objectives in a specific order may become an arduous task for the DM [38].

The scalarization method aggregates the objectives into one single objective. One widely used scalarization is the weighted sum method. Using a weight vector, The weighted sum method [269] linearly aggregates all objectives of MOP. The weighted sum method has been used in other MOEAs, such as multiple single objective Pareto sampling (MSOPS) [120] and multi-objective evolutionary algorithm based on decomposition (MOEA/D) [275]. The weighted sum method is defined as follows:

\[
\text{Minimize } \sum_{i=1}^{k} \omega_i f_i(x) \\
\text{Subject to } x \in X
\]

\(^2\)These techniques have been previously introduced in Chapter 3.
where $\omega_i \geq 0$ for all objectives $i = 1, \ldots, k$, and $\sum_{i=1}^{k} \omega_i = 1$. The weighted sum method has been integrated with meta-heuristic algorithms for the task matching problem [131; 10; 265]. Nevertheless, the method suffers from two potential shortcomings. The first issue is in choosing the weight correctly for all objectives. This has given rise to research works such as [134] that introduce a dynamic weighted sum method. Another issue is that the method only works on convex Pareto front, which real world problems may not produce.

The concept of Pareto optimality concept\(^3\) together with EA was suggested early in 1990’s [203]. Since then, this approach has become popular and has rapidly been recognized in the filed of MOPs. Pareto-based evolutionary algorithms have proven their effectiveness in working with different multi-objective optimization problems. Most of the EAs are less sensitive to the shape of the Pareto frontier, and as a result, they are applicable to different Pareto frontiers, as shown in Figure 5.1. As they are adopted broadly for MOPs, the proposed genetic learning algorithms in this thesis follow the Pareto-based MOEAs for the multi-objective task matching problem.

### 5.4 Genetic Learning PSO Algorithms for Multi-objective Task Matching Problem

Earlier in the thesis, three genetic learning PSO algorithms were introduced for the single task matching problem. The three genetic learning PSO algorithms, MGLPSO, HGLPSO, and $\mu$-MGLPSO, are extended to consider multi-objective problems. To

\(^3\)The Pareto optimality concept was introduced previously in Chapter 2.
our knowledge, there is no work in the literature using the genetic learning strategy to solve multi-objective task matching problem.

We first present few important concepts. Dealing with MOPS implies that there exists a set of non-dominated solutions instead of one global solution. Moreover, there may not be \(pbest\) and \(gbest\) if two solutions are dominated by each other. Selecting the \(pbest\) of each particle has to be reconsidered for solving MOPs. Coello et al. [65] proposed a simple non-dominated based approach for updating the \(pbest\). In this approach, if the new position \(x_i\) dominates \(pbest\), then the \(pbest\) of particle \(i\) is updated. The \(pbest\) is kept in case the \(pbest\) dominates the new position \(x_i\). When both \(x_i\) and \(pbest\) are non-dominated to each other, a random selection process is
applied. For the proposed genetic learning PSO algorithms, the updating approach by Coello is adopted. This is shown in Algorithm 7.

**Algorithm 7** Updating the \( p_{best} \)

1. \textbf{if} \( x_i \prec p_{best_i} \) \textbf{then} \quad \text{// } x_i \text{ dominates } p_{best_i}. \\
2. \hspace{1em} p_{best_i} = x_i \quad \text{// Update } p_{best}. \\
3. \textbf{else if} \hspace{1em} p_{best_i} \prec x_i \hspace{1em} \textbf{then} \quad \text{// } p_{best_i} \text{ dominates } x_i. \\
4. \hspace{1em} \text{Pass} \quad \text{// Keep the } p_{best}. \\
5. \textbf{else} \quad \text{// } x_i \text{ and } p_{best_i} \text{ are non-dominated.} \\
6. \hspace{1em} r \sim U(0,1) \quad \text{// Generate a random number.} \\
7. \textbf{if} \hspace{1em} r < 0.5 \hspace{1em} \textbf{then} \\
8. \hspace{2em} p_{best_i} = x_i \\
9. \hspace{1em} \textbf{else} \\
10. \hspace{2em} \text{Pass} \\
11. \hspace{1em} \textbf{end if} \\
12. \textbf{end if}

It is important to keep the record of all non-dominated solutions that are discovered during the optimization process. This important concept is called \textit{elitism}. Elitism plays a vital role in MOPs. Elitism keeps non-dominated solutions that are close to the Pareto front with the best dispersion. Elitism, therefore, improves the diversity as well as the convergence speed [74]. Zitzler and Thiele [283] proposed using an external archive (repository) to maintain the non-dominated solutions and promote elitism. The external archive is updated at each generation (iteration), preserving the non-dominated solutions. At the end of each generation, the new solutions are compared with the solution of the external archive. There are three cases when updating the external archive. First, if the new solutions dominate some other solu-
tion in the external archive, then these solutions in the external archive are replaced with the new solutions. Secondly, the new solutions are rejected if some solutions in the external archive dominate them. Finally, the new solutions are inserted into the external archive when they are non-dominated with the solutions of the external archive.

To update the $gbest$, several methods have been suggested [65]. All the non-dominated solutions are regarded as good, so it is hard to choose the best one. For the genetic learning PSO algorithms, a simple approach [119] is used. This is a random selection from the external archive. If number of non-dominated solutions exceed the capacity of the external archive, some non-dominated solutions need to be removed. For the genetic learning PSO algorithms, estimating the density of crowding distance is adopted to maintain the diversity in the Pareto optimal set.

5.4.1 Objective Function

In the single objective optimization problem, the makespan was the only objective considered for the task matching problem. For multi-objective task matching problem, in this thesis, two objectives are considered to be optimized simultaneously: makespan and flowtime. The makespan is the completion time of the last task in the grid system. The flowtime is the sum of completion time of all tasks. While the makespan measures the throughput (performance), the flowtime indicates the response time of the Grid system or the elapsed time of a task in the Grid system. Having these criteria to be optimized improves the system significantly. Unfortunately, it is hard to find such a solution that optimizes one criterion without deteriorating the
Minimization of the makespan requires matching the largest tasks to be executed on the fastest resource. On the other hand, assigning the small tasks to the fastest resource is essential to minimize the flowtime. Thus, the minimization of the makespan may increase the flowtime. These two objectives are usually conflicting with each other [200].

The makespan is calculated using Equation 4.4. For the flowtime, let $ET_{j,i}$ be the execution time of the $i^{th}$ task on the $j^{th}$ resource. The flowtime can be estimated using equation 5.3 for $n$ tasks and $m$ resources:

$$flowtime = \sum_{j=1}^{m} \sum_{i=1}^{n} ET_{j,i}$$

(5.3)

The makespan and the flowtime have different value ranges. The flowtime has a significantly higher magnitude over the makespan. Thus, many works [131; 250] suggest considering the mean of the flowtime rather than the total flowtime to mitigate the big gap. The mean flowtime is simply the total flowtime divided by the number of resources. To summarize, the makespan and the mean of the flowtime are considered for the multi-objective task matching problem. To resolve any misconception, any reference to flowtime denotes the mean of the flowtime.

### 5.4.2 Multi-objective MGLPSO (MO-MGLPSO) algorithm

Similar to MGLPSO for a single objective, multi-objective MGLPSO (MO-MGLPSO) has two phases: genetic learning and PSO phases. The created exemplars from the genetic learning phase will guide the particles toward the Pareto optimal front. Here, makespan and flowtime are the two objectives to be minimized simultaneously for
Chapter 5: Genetic Learning PSO Algorithms for Single and Multi-objective Task Matching Problem

the task matching problem. In single objective task matching problem, the crossover operator only takes place when there is better (less completion time). To incorporate the flowtime in the crossover, previous matching of the exemplar $E_i$ is considered. At this point, the makespan and the flowtime can be calculated progressively as the exemplar $E_i$ is updated. For each dimension $d$ of particle $i$, a random particle $k$ is selected for comparison. The completion time of the $d^{th}$ task is calculated for the particles $i$ and $k$ and are added to the previous makespan of the exemplar $E_i$. If the resultant makespan and flowtime from considering the $d^{th}$ of particle $i$ dominate that of particle $k$, then the corresponded $d^{th}$ value of particle $i$ is copied into the exemplar $E_i$. Inversely, when the result (makespan and flowtime) of particle $k$ dominates the that particle $k$, the crossover (learning) is conducted for the corresponded $d^{th}$ of the exemplar $E_i$. Finally, if both solutions are non-dominated, then the random selection of that in updating the $pbest$ is performed here.

To illustrate the above procedure, consider the previous example in Figure 4.4. In the exemplar $E_i$, there are two previous matches, task 1 and task 2 with resources 4 and 3, after rounding the decimals, respectively. Now, there are two candidates matches, which are task 3 on resource 5 (from particle $i$) and task 3 on resource 2 (from particle $k$). The completion time of these two matches can be check in $O(1)$. If the resultant makespan and flowtime from adding the match of particle $i$ to exemplar $E_i$ at $d = 3$ dominates $^4$ of that matching of particle $k$, then the corresponded $d^{th}$ of particle $i$ is copied into the same corresponded $d^{th}$. In case that adding the matching from particle $k$, which in this case task 3 on resource 2, yields makespan and

$^4$That means the makespan and flowtime of task 1 and task 2 and task 3 to resources 4, 3, and 5, respectively.
flowtime that dominates that of matching from particle \( i \), the crossover is performed between particle \( i \) and particle \( k \). If both are non-dominated, then a random number is generated. If the random number is less than 0.5, then the value at \( d = 3 \) of particle \( i \) is copied into the exemplar \( E_i \). Otherwise, the crossover is performed, as explained previously. Note that after updating the exemplar \( E_i \), the new makespan and flowtime are calculated to be used for \( d = 4 \).

The mutation operator has no changes from that of a single objective MGLPSO. For each dimension \( d \), a random number is generated between 0 and 1. If the random number is less than the mutation rate \( pm \), then the \( d^{th} \) dimension is mutated. Otherwise, the \( d^{th} \) dimension is kept intact. The selection process is also similar to MGLPSO for a single objective. The generated exemplar \( E_i \) is compared to the best exemplar created so far \( e_i \) in the selection operation. Since there are multiple results, which are makespan and flowtime, the comparison is based on dominance. That is, the current best exemplar is updated when the generated exemplar dominates it. When the current exemplar is better (dominated), the generated exemplar is discarded. If both are non-dominated, then the random selected explained above is applied. A tournament selection is performed when the exemplar is ceased improving for a certain number of generations (iterations), which is denoted by stopping gap \( sg \). In the tournament selection, a set of exemplars is chosen. A binary comparison is used to find the best in terms of makespan and flowtime. The dominated exemplar are selected to compete with the other exemplars. In the case of two exemplars that are non-dominated, a random selection explained earlier is used. There is a small

\[5\text{The mutation here is reinitialization of the dimension } d \text{ randomly in the range of the number of resources.}\]
modification in the PSO phase of the MO-MGLPSO algorithm. Equations 4.2 and 4.3 are used to update the velocity and position, respectively. There is no updating for the \textit{gbest} since there are a set of non-dominated solutions to find. However, to update the \textit{pbest}, Algorithm 7 is adopted. Algorithm 8 presents the MO-MGLPSO algorithm.

### 5.4.3 Multi-objective HGLPSO (MO-HGLPSO) algorithm

The Multi-objective HGLPSO (MO-HGLPSO) is not much different from MO-MGLPSO algorithm (Section 5.4.2). The only difference is that it incorporates the GLPSO’s learning strategy in the crossover operation. As presented in Section 5.1, the whole swarm is divided into two sub-populations (subgroups). The first sub-population follows the MO-MGLPSO while the second sub-population uses the learning scheme (crossover) of the GLPSO algorithm. For each dimension \(d\), a random particle \(k\) is selected. Then, the \textit{pbest} of particle \(i\) is compared to the \textit{pbest} of particle \(k\). The crossover operator is carried out only if \(\text{pbest}_k\) dominates \(\text{pbest}_i\). The crossover is performed between the \textit{pbest} of particle \(k\) and the \textit{gbest}. The selection of \textit{gbest} is adopted from [119], which is simply a random selection from the external archive. The \(d^{th}\) dimension of \textit{pbest}_i is preserved and copied into \(E_i\) if \(\text{pbest}_i\) dominates \(\text{pbest}_k\). Again, the random selection method is implemented if \(\text{pbest}_i\) and \(\text{pbest}_k\) are non-dominated. A random number is generated and if the random number is less than 0.5, then the \textit{pbest}_i, \(d\) is chosen for \(E_{i,d}\). If the random number is more than 0.5, the crossover is executed between the \textit{pbest} and \textit{gbest}. The other procedures, such as mutation, selection, and PSO phase, are the same as the MO-MGLPSO algorithm.
Algorithm 8 Multi-objective MGLPSO (MO-MGLPSO) algorithm - Part 1

1: /*Crossover operation*/
2: for $d = 1$ to $D$ do
3: randomly pick a particle $k \in \{1, 2, 3, ..., N\}$
4: $\text{Match}_i = \emptyset$ // To include a match from particle $i$.
5: $\text{Match}_k = \emptyset$ // To include a match from particle $k$.
6: $\text{Match}_i = E_i \cup x_{i,d}$
7: $\text{Match}_k = E_k \cup x_{k,d}$
8: if $f(\text{Match}_k) \prec f(\text{Match}_i)$ then // $\text{Match}_k$ dominates $\text{Match}_i$.
9: $E_{i,d} = \alpha \ast x_{i,d} + (1 - \alpha \ast x_{k,d}$
10: else if $f(\text{Match}_i) \prec f(\text{Match}_k)$ then // $\text{Match}_i$ dominates $\text{Match}_k$.
11: $E_{i,d} = x_{i,d}$
12: else // $\text{Match}_i$ and $\text{Match}_k$ are non-dominated.
13: $r \sim U(0, 1)$ // Generate a random number.
14: if $r < 0.5$ then
15: $E_{i,d} = x_{i,d}$
16: else
17: $E_{i,d} = \alpha \ast x_{i,d} + (1 - \alpha \ast x_{k,d}$
18: end if
19: calculate $f(E_i)$ to be used for the next updating
20: end if
21: end for
22: /*Mutation operation*/
23: set $pm$ to 0.001
24: for $d = 1$ to $D$ do
25: generate random number $r$
26: if $r < pm$ then
27: $E_{i,d} = \text{rand}(1,R)$ // $R$ is number of resources.
28: end if
29: end for
Algorithm 8 Multi-objective MGLPSO (MO-MGLPSO) algorithm - Part 2

30: /*Selection operation*/
31: calculate $f(E_i)$  // Calculate the fitness value of $E_i$.
32: if $f(E_i) \prec f(e_i)$ then
33:   $e_i = E_i$
34:   $sg = 0$  // Reset $sg$.
35: else if $f(e_i) \prec f(E_i)$ then
36:   $E_i$ is discarded
37:   $sg = sg + 1$  // Increase $sg$.
38: else  // $E_i$ and $e_i$ are non-dominated.
39:   $r \sim U(0,1)$  // Generate a random number.
40: if $r < 0.5$ then
41:   $e_i = E_i$
42:   $sg = 0$
43: else
44:   $E_i$ is discarded
45:   $sg = sg + 1$
46: end if
47: end if
48: if $E_i$ is not improving for $sg$ iterations then
49:   set $S$ based on the tournament selection method and random selection
50:   $e_i = S$
51:   $sg = 0$
52: end if
53: /*PSO update*/
54: for $d = 1$ to $D$ do
55:   $v_{i+1}^d = \omega \ast v_i^d + c \ast r^d(e_i^d - x_i^d)$
56:   $x_{i+1}^d = x_i^d + v_{i+1}^d$
57: end for
58: calculate $f(x_i)$
59: use Algorithm 7 to update $pbest$
The MO-HGLPSO is presented in Algorithm 9.

**Algorithm 9** Multi-objective HGLPSO (MO-HGLPSO) algorithm - part 1

1: divide the swarm into two subgroups
2: for $i = 1$ to $N$ do
3: /*Crossover operation*/
4:   if particle$_i \in$ the first subgroup then // Apply MGLPSO learning strategy.
5:     for $d = 1$ to $D$ do
6:       randomly pick a particle $k \in \{1, 2, 3, ..., N\}$
7:       Match$_i = \emptyset$ // To include a match from particle $i$.
8:       Match$_k = \emptyset$ // To include a match from particle $k$.
9:       Match$_i = E_i \cup x_{i,d}$
10:      Match$_k = E_i \cup x_{k,d}$
11:     if $f$(Match$_k$) $<$ $f$(Match$_i$) then // Match$_k$ dominates Match$_i$.
12:        $E_{i,d} = \alpha \times x_{i,d} + (1 - \alpha) \times x_{k,d}$
13:     else if $f$(Match$_i$) $<$ $f$(Match$_k$) then // Match$_i$ dominates Match$_k$.
14:        $E_{i,d} = x_{i,d}$
15:     else // Match$_i$ and Match$_k$ are non-dominated.
16:       $r \sim U(0, 1)$ // Generate a random number.
17:     if $r < 0.5$ then
18:        $E_{i,d} = x_{i,d}$
19:     else
20:        $E_{i,d} = \alpha \times x_{i,d} + (1 - \alpha) \times x_{k,d}$
21:     end if
22:     calculate $f(E_i)$ to be used for the next updating
23:   end if
24: end for
Algorithm 9 Multi-objective HGLPSO (MO-HGLPSO) algorithm - Part 2

25: else  // Apply GLPSO learning strategy.
26:     for $d = 1$ to $D$ do
27:         randomly pick a particle $k \in \{1, 2, 3, ..., N\}$
28:         calculate $f(pbest_i)$  // Calculate the fitness value of $pbest_i$.
29:         calculate $f(pbest_k)$  // Calculate the fitness value of $pbest_k$.
30:         if $f(pbest_k) \prec f(pbest_i)$ then  // $pbest_k$ dominates $pbest_i$.
31:             $E_{i,d} = \alpha \times pbest_{k,d} + (1 - \alpha) \times gbest_d$
32:         else if $f(pbest_i) \prec f(pbest_k)$ then  // $pbest_i$ dominates $pbest_k$.
33:             $E_{i,d} = pbest_{i,d}$
34:         else  // $pbest_k$ and $pbest_i$ are non-dominated.
35:             $r \sim U(0,1)$  // Generate a random number.
36:             if $r < 0.5$ then
37:                 $E_{i,d} = pbest_{i,d}$
38:             else
39:                 $E_{i,d} = \alpha \times pbest_{k,d} + (1 - \alpha) \times gbest_d$
40:             end if
41:         end if
42:     end for
43: end if
44: /*Mutation operation*/
45: set $pm$ to 0.001
46: for $d = 1$ to $D$ do
47:     generate random number $r$
48:     if $r < pm$ then
49:         $E_{i,d} = \text{rand}(1,R)$  // $R$ is number of resources.
50:     end if
51: end for
Algorithm 9 Multi-objective HGLPSO (MO-HGLPSO) algorithm - Part 3

52: /*Selection operation*/
53: calculate $f(E_i)$ // Calculate the fitness value of $E_i$.
54: if $f(E_i) \prec f(e_i)$ then
55: $e_i = E_i$
56: $sg = 0$ // Reset $sg$.
57: else if $f(e_i) \prec f(E_i)$ then
58: $E_i$ is discarded
59: $sg = sg + 1$ // Increase $sg$.
60: else // $E_i$ and $ei$ are non-dominated.
61: $r \sim U(0, 1)$ // Generate a random number.
62: if $r < 0.5$ then
63: $e_i = E_i$
64: $sg = 0$
65: else
66: $E_i$ is discarded
67: $sg = sg + 1$
68: end if
69: end if
70: if $E_i$ is not improving for $sg$ iterations then
71: set $S$ based on the tournament selection method and random selection
72: $e_i = S$
73: $sg = 0$
74: end if
75: end for
5.4.4 Multi-objective $\mu$-MGLPSO (MO-$\mu$-MGLPSO) algorithm

The MGLPSO is improved by introducing an adaptive mutation rate. The $\mu$-MGLPSO algorithm is extended to handle the multi-objective task matching problem. The multi-objective $\mu$-MGLPSO (MO-$\mu$-MGLPSO) algorithm is similar to the MO-MGLPSO algorithm except for the mutation operator. In MO-$\mu$-MGLPSO, the adaptive mutation method is altered for the multi-objective problem.

In the single objective $\mu$-MGLPSO, the average $pbest$ is used to determine the mutation rate of each exemplar. In the multi-objective case there are two objectives to minimize. The average for each objective is calculated. That is, the average of makespan and the flowtime of all $pbest$ are estimated. The mutation rate is changed if the average point \(^6\) dominates the $pbest$ of the particle $i$. If the $pbest$ dominates or is non-dominated to the average point, then the $pm$ is set to 0.001. The $pm$ for each objective is estimated using Equation 5.4 as follows:

$$pm = pm_1 \times \frac{(f_{pbest} - f_{gbest})}{(ave_{pbest} - f_{gbest})}$$ (5.4)

where $pm_1 = 0.04$, $ave_{pbest}$ is the average of all $pbest$ for a given objective (makespan or flowtime), and $gbest$ is chosen randomly from the external archive. After estimating the $pm$ for each objective, the final $pm$ is the average of two $pm$. \(^7\) The MO-$\mu$-MGLPSO) algorithm is shown in Algorithm 10. The other operators (crossover and selection) and PSO phase are similar to that of the MO-MGLPSO algorithm.

---

\(^6\)The point contains makespan and flowtime on the objective space.

\(^7\)One $pm$ for the makespan and another $pm$ for the flowtime.


**Algorithm 10** Multi-objective $\mu$-MGLPSO (MO-$\mu$-MGLPSO) algorithm - Part 1

1: /*Crossover operation*/
2: for $d = 1$ to $D$ do
3: randomly pick a particle $k \in \{1, 2, 3, ..., N\}$
4: $Match_i = \emptyset$ // To include a match from particle $i$.
5: $Match_k = \emptyset$ // To include a match from particle $k$.
6: $Match_i = E_i \cup x_{i,d}$
7: $Match_k = E_i \cup x_{k,d}$
8: if $f(Match_k) \prec f(Match_i)$ then // $Match_k$ dominates $Match_i$.
9: $E_{i,d} = \alpha \cdot x_{i,d} + (1 - \alpha) \cdot x_{k,d}$
10: else if $f(Match_i) \prec f(Match_k)$ then // $Match_i$ dominates $Match_k$.
11: $E_{i,d} = x_{i,d}$
12: else // $Match_i$ and $Match_k$ are non-dominated.
13: $r \sim U(0, 1)$ // Generate a random number.
14: if $r < 0.5$ then
15: $E_{i,d} = x_{i,d}$
16: else
17: $E_{i,d} = \alpha \cdot x_{i,d} + (1 - \alpha) \cdot x_{k,d}$
18: end if
19: calculate $f(E_i)$ to be used for the next updating
20: end if
21: end for

### 5.5 Experimental Design

This section presents the experimental setup for single and multi-objective task matching problem. For both optimization problems, the same data set by Braun et al. [39] is used. As explained previously in Chapter 4, the data set contains twelve
Algorithm 10 Multi-objective $\mu$-MGLPSO (MO-$\mu$-MGLPSO) algorithm - Part 2

22: /*Mutation operation*/
23: calculate $\text{ave}_{\text{makespan}}$
24: calculate $\text{ave}_{\text{flowtime}}$
25: set $\text{ave}_{\text{pbest}} = \text{ave}_{\text{makespan}} \cup \text{ave}_{\text{flowtime}}$
26: if $\text{ave}_{\text{pbest}} \prec \text{pbest}_i$ then
27: select $\text{gbest}$ from the external archive randomly
28: $\text{pm}_{\text{makespan}} = \text{pm}_1 \times \left(\frac{f_\text{pbest} - f_\text{gbest}}{\text{ave}_{\text{makespan}} - f_\text{gbest}}\right)$
29: $\text{pm}_{\text{flowtime}} = \text{pm}_1 \times \left(\frac{f_\text{gbest} - f_\text{gbest}}{\text{ave}_{\text{flowtime}} - f_\text{gbest}}\right)$
30: set $\text{pm} = (\text{pm}_{\text{makespan}} + \text{pm}_{\text{flowtime}})/2.0$
31: else
32: set $\text{pm} = 0.001$
33: end if
34: for $d = 1$ to $D$ do
35: generate random number $r$
36: if $r < \text{pm}$ then
37: $E_{t,d} = \text{rand}(1,R)$ // $R$ is number of resources.
38: end if
39: end for

different data set, representing different scenarios (problems) in grid environments. The experiments of both optimization problems were implemented using Python on a PC featuring an AMD A8-3870 APU operating at 3.0 GHz with 8 GB of RAM. All initial solutions were generated randomly. The population size for all algorithms was set to 40 particles. More details on the experimental setup are presented in Section 5.5.1 and Section 5.5.2 for single and multi-objective task matching problems.

8Further details about the data set are given in Section 4.4.
Algorithm 10 Multi-objective $\mu$-MGLPSO (MO-$\mu$-MGLPSO) algorithm - Part 3

40: /*Selection operation*/
41: calculate $f(E_i)$ // Calculate the fitness value of $E_i$.
42: if $f(E_i) \prec f(e_i)$ then
43:     $e_i = E_i$
44:     $sg = 0$ // Reset $sg$.
45: else if $f(e_i) \prec f(E_i)$ then
46:     $E_i$ is discarded
47:     $sg = sg + 1$ // Increase $sg$.
48: else // $E_i$ and $e_i$ are non-dominated.
49:     $r \sim U(0,1)$ // Generate a random number.
50: if $r < 0.5$ then
51:     $e_i = E_i$
52:     $sg = 0$
53: else
54:     $E_i$ is discarded
55:     $sg = sg + 1$
56: end if
57: end if
58: if $E_i$ is not improving for $sg$ iterations then
59:     set $S$ based on the tournament selection method and random selection
60:     $e_i = S$
61:     $sg = 0$
62: end if

respectively.
5.5.1 Experimental Design for Single Objective task matching problem

In addition to the experimental setup in Section 5.5, all experiments for the single objective task matching problem considers a large data set with 512 tasks and 16 resources. Each data set was generated twenty times, making them more than 200 data sets included in the experiments. The number of iterations was set to 1000 iterations.

5.5.2 Experimental Design for Multi-Objective Task Matching Problem

The experiments for multi-objective task matching problem were carried out on large data set as well. The data sets of which included in the experiments were more than 400 data sets, where each data set was reproduced 35 times. The maximum iterations number was 500 iterations for each optimization process. The size of the external archive is 30.

5.6 Results and Discussion for Single Objective Task Matching Problem

Three experiments were performed to evaluate the performance of proposed algorithms. The first set of experiments was conducted to comprehensively compare the three proposed algorithms to ten state-of-the-art algorithms that are PSO vari-
Chapter 5: Genetic Learning PSO Algorithms for Single and Multi-objective Task Matching Problem

ants: PSO [210], PSO with a ring topology (PSO$_R$), PSO with a pyramid topology (PSO$_P$), unified PSO (UPSO) [191], PSO with time-varying acceleration coefficients (PSO-TVAC) [201], weighted fully informed particle swarm (wFIPS) [179], fitness-distance-ratio PSO (FDR-PSO) [195], comprehensive learning PSO (CLPSO) [159], heterogeneous comprehensive learning (HCLPSO) [172], and genetic learning PSO (GLPSO) [109]. Since there were no specific parameter settings for most of the mentioned PSO algorithms for the task matching problem, we used the same parameter settings as reported in the corresponding references, as shown in Table 5.1. Moreover, to measure any significant difference between the results of algorithms, a non-parametric statistical method, namely, the Wilcoxon signed rank test, was adopted.

Following the first experiment, the convergence rate is measured in experiment 2. Finally, in experiment 3, the impact of sub-population sizes in the HGLPSO algorithm is assessed.

5.6.1 Comparison with PSO Variants for Large-scale Task Matching Problem

In this experiment, ten state-of-the-art PSO variants are included in comparison with MGLPSO, HGLPSO, and $\mu$-MGLPSO. The statistical results of all algorithms are tabulated in Table 5.2. For each data set, Table 5.2 shows the best and the worst makespan, the mean makespan, the standard deviation (Std), and the rank of the best mean makespan for each algorithm. The final rank indicates the ranks of the algorithm based on the average rank from all data sets. The result in bold indicates the best average makespan among the algorithms. As it can be seen from Table 5.2,
Table 5.1: Parameter settings.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Parameters settings</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSO</td>
<td>( w = 0.9, C_1 = C_2 = 1.0 )</td>
<td>[210]</td>
</tr>
<tr>
<td>PSO_( r )</td>
<td>( w = 0.9 \sim 0.4, C_1 = C_2 = 1.796180 )</td>
<td>-</td>
</tr>
<tr>
<td>PSO_( p )</td>
<td>( w = 0.9 \sim 0.4, C_1 = C_2 = 1.796180 )</td>
<td>-</td>
</tr>
<tr>
<td>UPSO</td>
<td>( \chi = 0.729, C_1 = C_2 = 2.05 )</td>
<td>[191]</td>
</tr>
<tr>
<td>PSO-TVAC</td>
<td>( w = 0.9 \sim 0.4, C_1 = 2.5 \sim 0.5, C_2 = 0.5 \sim 2.5 )</td>
<td>[201]</td>
</tr>
<tr>
<td>FDR-PSO</td>
<td>( w = 0.9 \sim 0.4, C_1 = 1, C_2 = 1, C_3 = 2 )</td>
<td>[195]</td>
</tr>
<tr>
<td>wFIPS</td>
<td>( w = 0.7298, \sum C_i = 4.1 )</td>
<td>[179]</td>
</tr>
<tr>
<td>CLPSO</td>
<td>( w = 0.9 \sim 0.4, C = 1.49445, m = 7 )</td>
<td>[159]</td>
</tr>
<tr>
<td>HCLPSO</td>
<td>( w = 0.99 \sim 0.2, C_1 = 2.5 \sim 0.5, C_2 = 0.5 \sim 2.5, C = 3 \sim 1.5, m = 5 )</td>
<td>[172]</td>
</tr>
<tr>
<td>GLPSO</td>
<td>( w = 0.7298, C = 1.49618, pm = 0.01, sg = 7 )</td>
<td>[109]</td>
</tr>
<tr>
<td>MGLPSO</td>
<td>( w = 0.7298, C = 1.4962, pm = 0.001, sg = 10 )</td>
<td>-</td>
</tr>
<tr>
<td>HGLPSO</td>
<td>( w = 0.7298, C = 1.4962, pm = 0.001, sg = 10 )</td>
<td>-</td>
</tr>
<tr>
<td>( \mu )-MGLPSO</td>
<td>( w = 0.7298, C = 1.4962, pm_1 = 0.04, pm_2 = 0.001, sg = 10 )</td>
<td>-</td>
</tr>
</tbody>
</table>

the proposed algorithm ranked better than most of the algorithms. Overall, HGLPSO was ranked as statistically the best algorithm. Also, among all algorithms, MGLPSO ranked second behind HGLPSO.

For the consistent data set, HGLPSO showed its effectiveness compared to GLPSO. Moreover, \( \mu \)-MGLPSO manifested better performance compared to MGLPSO on the same data set. HGLPSO and \( \mu \)-MGLPSO improved by 17.31\% and 12.81\%, respectively, compared to MGLPSO for the consistent data set. Further, HGLPSO improved slightly by 1.59\% over MGLPSO for the inconsistent data set. \( \mu \)-MGLPSO was
ranked the third after MGLPSO and HGLPSO for all categories in the inconsistent data set. Seemingly, $\mu$-MGLPSO performed better than HGLPSO and MGLPSO in the partially-consistent data set. $\mu$-MGLPSO obtained better results by 14.91% over MGLPSO. Furthermore, HGLPSO surpassed MGLPSO and reduced the makespan by 12.56%. Generally, the two extension of the algorithms, i.e., HGLPSO and $\mu$-MGLPSO, showed a significant improvement over MGLPSO.

In Chapter 4, the results of MGLPSO for the consistent data set showed a performance deficiency, as the algorithm got stuck in a local optimum. However, HGLPSO and $\mu$-MGLPSO are proposed as enhancements for MGLPSO. As evidenced in Table 5.2, HGLPSO and MGLPSO had the ability to jump out of a local optimum’s region. GLPSO showed competitive results in consistent and partially-consistent data sets. Indeed, GLPSO worked better when resource heterogeneity is low, i.e., resources are more homogeneous. It is worth mentioning that the three data sets (consistent, inconsistent, and partially-consistent) in this thesis are different. In other words, algorithms that work well on the consistent data set may perform worse on the consistent data set and vice versa. Therefore, there is no perfect algorithm here for all data sets. Another possible observation from Table 5.2 is that algorithms with the learning technique were ranked better than other methods among the PSO variants, demonstrating the effectiveness of the learning strategy. CLPSO and HCLPSO showed comparable performance with CLPSO being a slightly better than HCLPSO on most of the data sets. The best PSO algorithm that does not implement the learning technique is PSO-TVAC. The PSO-TVAC algorithm attempted to balance between exploration and exploitation via changing the acceleration coefficients during
the search operation. As a consequence, balancing between exploration and exploitation search processes can be vital. The wFIPS algorithm seemed to work better on the consistent data set, particularly, when resource heterogeneity is low. However, for the inconsistent and the partially-consistent data sets, the performance of wFIPS degraded drastically. The results of FDR-PSO showed a stable performance on all data sets. PSO, PSO$_P$, and PSO$_R$ produced different results depending on the data set. Finally, the UPSO algorithm was observed to be the worst algorithm in this experiment, in which it ranked the last on all data sets.

For better visualization, Figure 5.2 presents the mean makespan of all PSO variants. For each subfigure, the “x-axis” represents PSO algorithms, and the “y-axis” shows the mean makespan in seconds. As it can be seen in Figure 5.2a, $\mu$-MGLPSO outperformed MGLPSO by 12.81% on the consistent data set. Moreover, HGLPSO produced better results by 17.31% compared to MGLPSO. This is a clear indication that the proposed methods in $\mu$-MGLPSO and HGLPSO to enhance MGLPSO are incredibly fruitful for the consistent data set. Furthermore, MGLPSO and HGLPSO gave nearly identical results; HGLPSO was slightly better by 1.60% for the inconsistent data set, as shown in Figure 5.2b.

Similarly, for the partially-consistent data set shown in Figure 5.2c, $\mu$-MGLPSO and HGLPSO were the best algorithms, and $\mu$-MGLPSO was marginally more efficient than HGLPSO by 2.68%. Despite the remarkable performance of $\mu$-MGLPSO for consistent and partially-consistent data sets, a degradation by 9.49% compared to MGLPSO was noted on the inconsistent data set. One possible reason for this could be due to balancing issue between exploration and exploitation search, i.e., the
Table 5.2: Statistical results of PSO variants on large data size.

<table>
<thead>
<tr>
<th>GLPSO</th>
<th>UP</th>
<th>PSO</th>
<th>R</th>
<th>Mean</th>
<th>Std</th>
<th>Mean</th>
<th>Std</th>
<th>Mean</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>791.3</td>
<td>607134</td>
<td>222809</td>
<td>1177087</td>
<td>435.1</td>
<td>62187</td>
<td>133347</td>
<td>788984</td>
<td>713.8</td>
<td>47824</td>
</tr>
<tr>
<td>1956</td>
<td>224678</td>
<td>47777</td>
<td>1108316</td>
<td>384.7</td>
<td>10642</td>
<td>52638</td>
<td>221126</td>
<td>2790.9</td>
<td>14566</td>
</tr>
<tr>
<td>791.3</td>
<td>607134</td>
<td>222809</td>
<td>1177087</td>
<td>435.1</td>
<td>62187</td>
<td>133347</td>
<td>788984</td>
<td>713.8</td>
<td>47824</td>
</tr>
<tr>
<td>2149</td>
<td>406933</td>
<td>154916</td>
<td>432517</td>
<td>256.3</td>
<td>1080</td>
<td>7985.9</td>
<td>495510</td>
<td>2189.5</td>
<td>14751</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BILPSO</th>
<th>GLPSO</th>
<th>UP</th>
<th>PSO</th>
<th>R</th>
<th>Mean</th>
<th>Std</th>
<th>Mean</th>
<th>Std</th>
<th>Mean</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>724.7</td>
<td>648896</td>
<td>252331</td>
<td>1272173</td>
<td>357.6</td>
<td>238300</td>
<td>112247</td>
<td>400497</td>
<td>782.2</td>
<td>47824</td>
<td>220875</td>
</tr>
<tr>
<td>1956</td>
<td>224678</td>
<td>47777</td>
<td>1108316</td>
<td>384.7</td>
<td>10642</td>
<td>52638</td>
<td>221126</td>
<td>2790.9</td>
<td>14566</td>
<td>23477</td>
</tr>
<tr>
<td>791.3</td>
<td>607134</td>
<td>222809</td>
<td>1177087</td>
<td>435.1</td>
<td>62187</td>
<td>133347</td>
<td>788984</td>
<td>713.8</td>
<td>47824</td>
<td>220875</td>
</tr>
<tr>
<td>2149</td>
<td>406933</td>
<td>154916</td>
<td>432517</td>
<td>256.3</td>
<td>1080</td>
<td>7985.9</td>
<td>495510</td>
<td>2189.5</td>
<td>14751</td>
<td>78864</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CILPSO</th>
<th>GLPSO</th>
<th>UP</th>
<th>PSO</th>
<th>R</th>
<th>Mean</th>
<th>Std</th>
<th>Mean</th>
<th>Std</th>
<th>Mean</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>397.4</td>
<td>683164</td>
<td>221258</td>
<td>1949627</td>
<td>591.5</td>
<td>270184</td>
<td>316312</td>
<td>1949627</td>
<td>591.5</td>
<td>270184</td>
<td>316312</td>
</tr>
<tr>
<td>1956</td>
<td>224678</td>
<td>47777</td>
<td>1108316</td>
<td>384.7</td>
<td>10642</td>
<td>52638</td>
<td>221126</td>
<td>2790.9</td>
<td>14566</td>
<td>23477</td>
</tr>
<tr>
<td>791.3</td>
<td>607134</td>
<td>222809</td>
<td>1177087</td>
<td>435.1</td>
<td>62187</td>
<td>133347</td>
<td>788984</td>
<td>713.8</td>
<td>47824</td>
<td>220875</td>
</tr>
<tr>
<td>2149</td>
<td>406933</td>
<td>154916</td>
<td>432517</td>
<td>256.3</td>
<td>1080</td>
<td>7985.9</td>
<td>495510</td>
<td>2189.5</td>
<td>14751</td>
<td>78864</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DFSPRD</th>
<th>GLPSO</th>
<th>UP</th>
<th>PSO</th>
<th>R</th>
<th>Mean</th>
<th>Std</th>
<th>Mean</th>
<th>Std</th>
<th>Mean</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>408.4</td>
<td>683164</td>
<td>221258</td>
<td>1949627</td>
<td>591.5</td>
<td>270184</td>
<td>316312</td>
<td>1949627</td>
<td>591.5</td>
<td>270184</td>
<td>316312</td>
</tr>
<tr>
<td>1956</td>
<td>224678</td>
<td>47777</td>
<td>1108316</td>
<td>384.7</td>
<td>10642</td>
<td>52638</td>
<td>221126</td>
<td>2790.9</td>
<td>14566</td>
<td>23477</td>
</tr>
<tr>
<td>791.3</td>
<td>607134</td>
<td>222809</td>
<td>1177087</td>
<td>435.1</td>
<td>62187</td>
<td>133347</td>
<td>788984</td>
<td>713.8</td>
<td>47824</td>
<td>220875</td>
</tr>
<tr>
<td>2149</td>
<td>406933</td>
<td>154916</td>
<td>432517</td>
<td>256.3</td>
<td>1080</td>
<td>7985.9</td>
<td>495510</td>
<td>2189.5</td>
<td>14751</td>
<td>78864</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P3O</th>
<th>GLPSO</th>
<th>UP</th>
<th>PSO</th>
<th>R</th>
<th>Mean</th>
<th>Std</th>
<th>Mean</th>
<th>Std</th>
<th>Mean</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>37179</td>
<td>906132</td>
<td>328052</td>
<td>2305679</td>
<td>577.4</td>
<td>290783</td>
<td>316312</td>
<td>2305679</td>
<td>577.4</td>
<td>290783</td>
<td>316312</td>
</tr>
<tr>
<td>1956</td>
<td>224678</td>
<td>47777</td>
<td>1108316</td>
<td>384.7</td>
<td>10642</td>
<td>52638</td>
<td>221126</td>
<td>2790.9</td>
<td>14566</td>
<td>23477</td>
</tr>
<tr>
<td>791.3</td>
<td>607134</td>
<td>222809</td>
<td>1177087</td>
<td>435.1</td>
<td>62187</td>
<td>133347</td>
<td>788984</td>
<td>713.8</td>
<td>47824</td>
<td>220875</td>
</tr>
<tr>
<td>2149</td>
<td>406933</td>
<td>154916</td>
<td>432517</td>
<td>256.3</td>
<td>1080</td>
<td>7985.9</td>
<td>495510</td>
<td>2189.5</td>
<td>14751</td>
<td>78864</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P3OTM</th>
<th>GLPSO</th>
<th>UP</th>
<th>PSO</th>
<th>R</th>
<th>Mean</th>
<th>Std</th>
<th>Mean</th>
<th>Std</th>
<th>Mean</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>37179</td>
<td>906132</td>
<td>328052</td>
<td>2305679</td>
<td>577.4</td>
<td>290783</td>
<td>316312</td>
<td>2305679</td>
<td>577.4</td>
<td>290783</td>
<td>316312</td>
</tr>
<tr>
<td>1956</td>
<td>224678</td>
<td>47777</td>
<td>1108316</td>
<td>384.7</td>
<td>10642</td>
<td>52638</td>
<td>221126</td>
<td>2790.9</td>
<td>14566</td>
<td>23477</td>
</tr>
<tr>
<td>791.3</td>
<td>607134</td>
<td>222809</td>
<td>1177087</td>
<td>435.1</td>
<td>62187</td>
<td>133347</td>
<td>788984</td>
<td>713.8</td>
<td>47824</td>
<td>220875</td>
</tr>
<tr>
<td>2149</td>
<td>406933</td>
<td>154916</td>
<td>432517</td>
<td>256.3</td>
<td>1080</td>
<td>7985.9</td>
<td>495510</td>
<td>2189.5</td>
<td>14751</td>
<td>78864</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure 5.2: Bar plots of the mean makespan for all data sets.

(a) The mean of the makespan for the consistent data set.
(b) The mean of the makespan for the inconsistent data set.
(c) The mean of the makespan for the partially-consistent data set.
(d) The averaged makespan of all data sets.

Particles tend to be more exploitative during the search process.

Figure 5.2d presents the average makespan of all data sets (consistent, inconsistent and partially-consistent). Overall, the three proposed algorithms were significantly
Figure 5.3: Box plot of the mean makespan for all data sets.

better than other PSO variants for the task matching problem.

Figure 5.3 depicts different box plots for each data set to analyze the variability of the results obtained from PSO algorithms. Undoubtedly, the proposed algorithms
exhibited the dispersion superior to that of other PSO variants; there was substantially more variation in the rest algorithms for all data sets. In Figure 5.3a, HGLPSO had a better distribution with a smaller box compared to those of $\mu$-MGLPSO and MGLPSO. Additionally, the result of MGLPSO was better than GLPSO, although GLPSO had a lower minimum value. The proposed algorithms produced dramatically better results for the inconsistent data set, as illustrated in Figure 5.3b. The medians of HGLPSO and MGLPSO were almost at the same level.

For the partially-consistent data set, as depicted in Figure 5.3c, the results of $\mu$-MGLPSO were the best, with a narrower box and a smaller median value compared with those of HGLPSO. Finally, Figure 5.3d shows the overall results. The effectiveness of the proposed algorithm was unequivocally demonstrated by a smaller makespan with a narrower distribution of results; consequently, the proposed algorithms are more robust and reliable in comparison to the rest of PSO variations for the task matching problem.

The Wilcoxon signed rank non-parametric test [71] was used for the statistical analysis; the test normally is used to identify whether there is a significant difference between the obtained results. In this experiment, the Wilcoxon signed rank test was performed at a significance level of $\alpha = 0.05$. If the null hypothesis is true, the probability of rejecting it is denoted by $\alpha$. Therefore, the level of significance indicates whether we can reject the null hypothesis. Tables 5.3–5.5 show results of the Wilcoxon signed rank test of the three different data sets. Each table shows a comparison between the best among the proposed algorithms, e.g., HGLPSO for consistent data set, for a given data set and the rest of the algorithms. The sign “+” means that the
corresponding proposed algorithm is significantly better than the compared algorithm, the sign “-” indicates that the compared algorithm is significantly better than the corresponding proposed algorithm, and the sign “≈” means that no significance is noticed.

For the consistent data set, HGLPSO was used to compare with the rest of PSO algorithms, as presented in Table 5.3. HGLPSO was significantly better than most of the PSO algorithms. Compared with GLPSO, MGLPSO showed a significant improvement only when the resource heterogeneity was high. Also, at the level of significance $\alpha = 0.05$, there was no statistically significant improvement observed between $\mu$-MGLPSO and HGLPSO when resource heterogeneity was high.

MGLPSO and HGLPSO have comparable performance when the data set was inconsistent. However, HGLPSO has already been selected for the consistent data set. Thus, for the inconsistent data set, MGLPSO was selected for comparison against the rest of PSO variants. As shown in Table 5.4, there was a statically significant difference in results of MGLPSO and other PSO variants, whereas there was no significant difference when other methods were compared with HGLPSO.

$\mu$-MGLPSO was chosen for the partially-consistent data set for the Wilcoxon signed rank test. As can be observed from Table 5.5, the results of $\mu$-MGLPSO were statically significant compared with most of the PSO algorithms. However, no significance was indicated compared with HGLPSO for all categories and with GLPSO only when resource heterogeneity is low. Therefore, in conclusion, the Wilcoxon signed rank test verified the superiority and the effectiveness of the proposed algorithms for the task matching problem.
Table 5.3: Wilcoxon signed rank test of the makespan obtained by HGLPSO for the consistent data set.

<table>
<thead>
<tr>
<th>Consistent</th>
<th>MGLPSO</th>
<th>GLPSO</th>
<th>PSO</th>
<th>CLPSO</th>
<th>HCLPSO</th>
<th>wFIPS</th>
<th>μ-MGLPSO</th>
<th>PSOFDR</th>
<th>PSO-P</th>
<th>PSO-R</th>
<th>PSO-TVAC</th>
<th>UPSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>lo-lo</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>lo-hi</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>≈</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>hi-lo</td>
<td>+</td>
<td>≈</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>hi-hi</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>+/-/≈/-</td>
<td>4/0/0</td>
<td>2/1/1</td>
<td>4/0/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
<td>3/1/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
</tr>
</tbody>
</table>

Table 5.4: Wilcoxon signed rank test of the makespan obtained by MGLPSO for the inconsistent data set.

<table>
<thead>
<tr>
<th>Inconsistent</th>
<th>HGLPSO</th>
<th>GLPSO</th>
<th>PSO</th>
<th>CLPSO</th>
<th>HCLPSO</th>
<th>wFIPS</th>
<th>μ-MGLPSO</th>
<th>PSOFDR</th>
<th>PSO-P</th>
<th>PSO-R</th>
<th>PSO-TVAC</th>
<th>UPSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>lo-lo</td>
<td>≈</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>lo-hi</td>
<td>≈</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>hi-lo</td>
<td>≈</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>hi-hi</td>
<td>≈</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>+/-/≈/-</td>
<td>0/4/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
</tr>
</tbody>
</table>

### 5.6.2 Convergence Rate

In this experiment, the speed of convergence was investigated for all the proposed algorithms. The mean makespan for each algorithm was tracked for every 100 iterations.

Figure 5.4 depicts the convergence rate for the three data sets. In the consistent data set, the proposed algorithms converged faster than other PSO variants. Moreover, compared with other PSO variants, the proposed algorithms required fewer iterations to converge to the resulting solutions of higher quality. MGLPSO converged well on for 100 iterations; there after, convergence became slower, as the particles
Table 5.5: Wilcoxon signed rank test of the makespan obtained by $\mu$-MGLPSO for the partially-consistent data set.

<table>
<thead>
<tr>
<th>Partially-consistent</th>
<th>HGLPSO</th>
<th>GLPSO</th>
<th>PSO</th>
<th>CLPSO</th>
<th>HCLPSO</th>
<th>wFIPS</th>
<th>MGLPSO</th>
<th>PSOFDR</th>
<th>PSO-P</th>
<th>PSO-R</th>
<th>PSO-TVAC</th>
<th>UPSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>lo-lo</td>
<td>$\approx$</td>
<td>$\approx$</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>lo-hi</td>
<td>$\approx$</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>hi-lo</td>
<td>$\approx$</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>hi-hi</td>
<td>$\approx$</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>+/-</td>
<td>0/4/0</td>
<td>2/2/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
<td>4/0/0</td>
</tr>
</tbody>
</table>

exploit in the solution space. By incorporating adaptive mutation to MGLPSO, the search in $\mu$-MGLPSO was further enhanced. HGLPSO seemed to be able to handle consistent data set efficiently due to aid from particles using the GLPSO learning method. The same was observed for the proposed algorithms in the case of the inconsistent data set. In fact, the inconsistent data set captures more the reality of the grid system as the environment of the grid was heterogeneous. Clearly, the proposed algorithms were immensely better than the rest of PSO variants.

The effectiveness of the proposed algorithms in working with the inconsistent data set is attributed to the learning strategy (the crossover method). MGLPSO and HGLPSO exhibited the same convergence behavior, outperforming $\mu$-MGLPSO. However, compared with MGLPSO and HGLPSO, $\mu$-MGLPSO had the best convergence to the solution. At last, the convergence of all data sets was averaged for all PSO variants to assess the overall performance. On average, the proposed algorithms attained the best overall convergence by managing the trade-off between exploration and exploitation search.
Figure 5.4: Convergence rate of PSO variants for all data sets.

5.6.3 Effect of Sub-population Size

To further evaluate the effectiveness of the proposed HGLPSO method, a comparison experiment was performed by varying the sizes of subgroups (sub-populations). Three different sizes were chosen as subgroups: 75%-25% indicates that 75% and 25% (30 and 10 particles) of the swarm utilize MGLPSO and GLPSO, respectively. The observations from these three experiments hold true for other subgroup sizes.

As depicted in Figure 5.5, it is obvious that different data sets require different
sizes to be solved efficiently. In other words, no one size fitted all data sets perfectly. For instance, 75%-25%, unlike the other sizes, was unsuitable for the consistent data set. In contrast, 75%-25% was able to deliver a better makespan for the other data sets. In addition, 50%-50% was more efficient in the inconsistent data set and the worst for the partially-consistent data set; opposite results were observed for 25%-75%.

Figure 5.6 presents the average makespan. Choosing 75%-25% as the size of the subgroups ameliorated the results of HGLPSO algorithm by 4.76% and 5.24% compared with results obtained using 50%-50% and 25%-75%, respectively.
Chapter 5: Genetic Learning PSO Algorithms for Single and Multi-objective Task Matching Problem

5.7 Results and Discussion for Multi-objective Task Matching Problem

The primary goal of the proposed MOEAs is to find a set of Pareto solutions which minimizes the distance to the true Pareto front (convergence) and maximizes a high spread distribution over the Pareto front (diversity). Few metrics have been adopted that measures the convergence as well as diversity simultaneously. One such is the hypervolume (HV) indicator [284]. This indicator measures the volume induced by the non-dominated solutions [87; 162]. The hypervolume metric has been used broadly in the area of MOPs, as it is the only known metric that is Pareto compliance [282]. That is to say, the indicator preserves a maximum hypervolume for the Pareto optimal front while lower hypervolume is given for all dominated sets. Intuitively, high hypervolume value means a better approximation of the true Pareto
Chapter 5: Genetic Learning PSO Algorithms for Single and Multi-objective Task Matching Problem

Figure 5.7: Hypervolume indicator.

front. To illustrate, Figure 5.7 shows an example of the hypervolume metric.

Given a set of points $X$ and a reference point $r$, the hypervolume indicator can be defined formally as follows [110; 153]:

$$HV(X) = \lambda \left( \bigcup_{p \in X \subset [p, r]} \right)$$

(5.5)

where $[p, r] = \{ q \in \mathbb{R} \mid p < q < r \}$ and $\lambda$ indicates the Lebesgue measure. Choosing the reference point requires calculating the hypervolume indicator. The reference point should be dominated by all Pareto optimal solutions to get a positive hypervolume value. A general method is to select a point that maximizes (in this thesis) all objectives [124; 21]. Thus, for each data set, a point which maximizes both the makespan and flowtime is selected. The value of hypervolume is normalized using the standard normalization method. In order to appraise the performance of the proposed algorithms, the following multi-objective algorithms are considered: multi-objective PSO (MOPSO) [65], Multi-objective comprehensive learning PSO (MOCLPSO) [119],
Table 5.6: Parameter settings.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Parameters settings</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOPSO</td>
<td>( w = 0.9 \sim 0.4, C_1 = C_2 = 2.0, nGrid = 10 )</td>
<td>[65]</td>
</tr>
<tr>
<td>MOCLPSO</td>
<td>( w = 0.9 \sim 0.2, Pc = 0.1, Pm = 0.4 )</td>
<td>[119]</td>
</tr>
<tr>
<td>NSGA-2</td>
<td>( pc = 0.9, pm = 1/n, \eta_c = 20, \eta_n = 20 )</td>
<td>[74]</td>
</tr>
<tr>
<td>MOGLPSO</td>
<td>( w = 0.7298, C = 1.49618, pm = 0.01, sg = 7 )</td>
<td>-</td>
</tr>
<tr>
<td>MO-MGLPSO</td>
<td>( w = 0.7298, C = 1.4962, pm = 0.001, sg = 10 )</td>
<td>-</td>
</tr>
<tr>
<td>MO-HGLPSO</td>
<td>( w = 0.7298, C = 1.4962, pm = 0.001, sg = 10 )</td>
<td>-</td>
</tr>
<tr>
<td>MO-( \mu )-MGLPSO</td>
<td>( w = 0.7298, C = 1.4962, pm_1 = 0.04, pm_2 = 0.001, sg = 10 )</td>
<td>-</td>
</tr>
</tbody>
</table>

non-dominated sorting genetic algorithm-2 (NSGA-2) [74], and multi-objective genetic learning PSO (MOCLPSO). As summarized in Table 5.6, for a fair comparison, the related parameters of the compared algorithms were adjusted as suggested in their references.

5.7.1 Comparison with MOEA Variants for Large-scale Multi-objective Task Matching Problem

The proposed algorithms for multi-objective task matching problem are compared with well-established algorithms for MOPs. The statistical results are presented in Table 6.6, in which the mean (the top value) and standard deviation of the HV are reported for each algorithm. As stated earlier, all results are normalized, and a higher value of HV indicates a better set of solutions approximating the true PF in terms of diversity and convergence. The best obtained mean of the HV value
### Table 5.7: Statistical results of MOEA variants.

<table>
<thead>
<tr>
<th>MOEA Variants</th>
<th>Consistent</th>
<th>Inconsistent</th>
<th>Partially-consistent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>lo-lo</td>
<td>lo-lo</td>
<td>lo-lo</td>
</tr>
<tr>
<td>MO-MGLPSO</td>
<td>8.72E-01±</td>
<td>9.06E-01±</td>
<td>9.45E-01±</td>
</tr>
<tr>
<td></td>
<td>5.29E-02±</td>
<td>3.45E-02±</td>
<td>2.00E-02±</td>
</tr>
<tr>
<td>MO-HGLPSO</td>
<td>8.70E-01±</td>
<td>9.18E-01±</td>
<td>9.51E-01±</td>
</tr>
<tr>
<td></td>
<td>5.00E-02±</td>
<td>3.36E-02±</td>
<td>2.00E-02±</td>
</tr>
<tr>
<td>MO-μ-MGLPSO</td>
<td>8.55E-01±</td>
<td>9.18E-01±</td>
<td>9.51E-01±</td>
</tr>
<tr>
<td></td>
<td>4.41E-02±</td>
<td>2.88E-02±</td>
<td>2.00E-02±</td>
</tr>
<tr>
<td>MOGLPSO</td>
<td>7.82E-01±</td>
<td>8.18E-01±</td>
<td>8.58E-01±</td>
</tr>
<tr>
<td></td>
<td>4.14E-02±</td>
<td>2.94E-02±</td>
<td>2.00E-02±</td>
</tr>
<tr>
<td>MOCLPSO</td>
<td>7.82E-01±</td>
<td>8.18E-01±</td>
<td>8.58E-01±</td>
</tr>
<tr>
<td></td>
<td>4.25E-02±</td>
<td>3.27E-02±</td>
<td>2.00E-02±</td>
</tr>
<tr>
<td>MOPSO</td>
<td>5.95E-01±</td>
<td>6.48E-01±</td>
<td>6.96E-01±</td>
</tr>
<tr>
<td></td>
<td>5.10E-02±</td>
<td>4.07E-02±</td>
<td>3.00E-02±</td>
</tr>
<tr>
<td>NSGA-2</td>
<td>4.21E-01±</td>
<td>5.01E-02±</td>
<td>6.48E-02±</td>
</tr>
<tr>
<td></td>
<td>8.35E-02±</td>
<td>4.41E-02±</td>
<td>5.90E-02±</td>
</tr>
</tbody>
</table>

As it can be observed from Table 6.6, MO-MGLPSO achieves the best results on 10 out of 12 comparisons, which indicates the superiority of the MO-MGLPSO for the multi-objective task matching problem. The other two proposed algorithms, which are MO-HGLPSO and MO-μ-MGLPSO, obtain the best results only on one comparison when the data set is consistent with high resource heterogeneity. For the consistent data set, MO-HGLPSO and MO-μ-MGLPSO show competitive results compared to the MO-MGLPSO algorithm. Although it seems from Table 6.6 that the MO-MGLPSO algorithm is the best for the consistent data set, MO-HGLPSO achieves the best results by 0.26% in comparison to MO-MGLPSO when all sub-data sets \(^9\) averaged. However, MO-MGLPSO is the best compared with MO-μ-MGLPSO by 0.69% for the consistent data set.

Compared with the other algorithms, the significance was noticed more. Compared with the MOGLPSO algorithm, the proposed algorithms obtained 5.3% at

---

\(^9\) The variation of task and resource heterogeneity.
least when the data set is consistent. Other PSO variants seemed to be more efficient for the consistent problem. The NSGA-2 algorithm was the worst algorithm against others. The proposed algorithms achieved at least 60% better approximating results than NSGA-2.

The preponderance of the proposed algorithms continued in the inconsistent data set. The performance of the proposed algorithms was dramatically superior to that of the other algorithms. All the proposed algorithms outperformed the other algorithms by more than 60%. The MO-MGLPSO algorithm was reported as the best algorithm for the inconsistent problem, achieving 3.76% and 3.18% better HV as compared with MO-HGLPSO and MO-µ-MGLPSO, respectively. Apparently, the MO-µ-MGLPSO algorithm worked better than MO-HGLPSO when the task heterogeneity is high. Moreover, NSGA-2 seems to work more efficiently in inconsistent data set than MOCLPSO and MOPSO algorithms. The MOCLPSO algorithm performs significantly worse, while other multi-objective PSO algorithms perform better. The same is observed for the partially-consistent data set. The MO-MGLPSO is the best algorithm, although the slight degradation in the HV values. MO-MGLPSO surpasses the MOGLPSO algorithm by 28%. Additionally, MO-MGLPSO significantly performs better than MCLPSO and MOPSO with 63% and 58%, respectively. Similarly, the other proposed algorithms notably outperformed competitive algorithms. Comparing the proposed algorithms with each other, their performance did not exceed 2.5%. 
Figure 5.8: Box plot of the HV for all data sets.

Figure 5.8 illustrates the range of the HV value obtained from different multi-objective algorithms for the three data sets. Certainly, the best performance was
Chapter 5: Genetic Learning PSO Algorithms for Single and Multi-objective Task Matching Problem

Table 5.8: Friedman rank test for the consistent problem.

<table>
<thead>
<tr>
<th></th>
<th>MO-MGLPSO</th>
<th>MO-HGLPSO</th>
<th>MO-µ-MGLPSO</th>
<th>MOGLPSO</th>
<th>MOCLPSO</th>
<th>MOPSO</th>
<th>NSGA-2</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Friedman ranks</td>
<td>1.97</td>
<td>1.80</td>
<td>2.29</td>
<td>3.94</td>
<td>5.00</td>
<td>6.00</td>
<td>7.00</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>Final rank</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.9: Friedman rank test for the inconsistent problem.

<table>
<thead>
<tr>
<th></th>
<th>MO-MGLPSO</th>
<th>MO-HGLPSO</th>
<th>MO-µ-MGLPSO</th>
<th>MOGLPSO</th>
<th>MOCLPSO</th>
<th>MOPSO</th>
<th>NSGA-2</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Friedman ranks</td>
<td>1.11</td>
<td>2.57</td>
<td>2.31</td>
<td>4.03</td>
<td>6.91</td>
<td>5.91</td>
<td>5.14</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>Final rank</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

obtained from the proposed algorithms. In the consistent data set, as shown in Figure 5.8a, the three proposed algorithm showed comparative results. The MO-HGLPSO algorithm may seem to be slightly better, as its median was the highest as compared with MO-MGLPSO and MO-µ-MGLPSO. Other algorithms exhibited different performance, as shown in Figure 5.8a. In the inconsistent problem, the proposed algorithms performed significantly better than the other algorithms, as depicted in Figure 5.8b, while the other algorithms struggled. This data set captures more the reality of the heterogeneity of the grid environment. Thus, the proposed algorithms unveiled their ability to be adopted for such an environment. The same was observed in Figure 5.8c. Algorithms, such as MOPSO and MCLPSO, seemed to be worse with the more heterogeneous environment, while NSGA-2 showed more robust performance.

To identify any significance in the HV results of the algorithms, the Friedman test [98] was conducted. The Friedman test is a nonparametric test that can be used to detect any significant differences among set of algorithms over given data
Table 5.10: Friedman rank test for the partially-consistent problem.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Friedman ranks</th>
<th>Final rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>MO-MGLPSO</td>
<td>1.77</td>
<td>1</td>
</tr>
<tr>
<td>MO-HGLPSO</td>
<td>1.97</td>
<td>2</td>
</tr>
<tr>
<td>MO-µ-MGLPSO</td>
<td>2.26</td>
<td>3</td>
</tr>
<tr>
<td>MOGLPSO</td>
<td>4.00</td>
<td>4</td>
</tr>
<tr>
<td>MCLPSO</td>
<td>6.54</td>
<td>7</td>
</tr>
<tr>
<td>MOPSO</td>
<td>5.94</td>
<td>6</td>
</tr>
<tr>
<td>NSGA-2</td>
<td>5.51</td>
<td>5</td>
</tr>
</tbody>
</table>

The null hypothesis in the Friedman test states that all algorithms perform equivalently. The test gives a rank for each algorithm; the best algorithm is assigned rank 1, the second best is 2, and so on. The Friedman test results are shown in Table 5.8, Table 5.9, and Table 5.10 for consistent, inconsistent, and partially-consistent, respectively. As can be seen from the tables, the proposed algorithms ranked as the best three algorithms in tests. The ranks of the proposed algorithms, though, were not the same for all data sets. The MO-MGLPSO scored the best in inconsistent and partially-consistent data sets, and for the consistent problem, the algorithm ranked as the second best behind the MO-HGLPSO. MO-µ-MGLPSO ranked third in consistent and partially-consistent data sets. However, the algorithms showed better performance, scoring second in the inconsistent data set. MOGLPSO ranked fourth in all data set. Similarly, MOPSO ranked the second worst algorithm in all data sets. MCLPSO was as noticed the worst algorithm in inconsistent and partially-consistent, whereas NSGA-2 was the worst only in the consistent problem. The p-value was < 0.001 in all data set, indicating significant differences between the performance of algorithms.
5.7.2 Quality of Non-dominated Solutions

Generally, when evaluating the resulting non-dominated set of solutions, it is important to measure how well it represents the Pareto front. There are few aspects that can help in the evaluation process, such as convergence, diversity, and cardinality. Normally, it is preferable to have a set of non-dominated solution that maximizes the above aspects. This section provides the solution quality obtained by the algorithms. Since there were different numbers of cardinality for each run, the best HV from each algorithm was selected, since the HV metric considers the aspects mentioned previously. Figure 5.9 demonstrates the solution quality of all algorithms for the consistent problem. The proposed algorithms showed good convergence with better solutions spread. When the resource heterogeneity was low, as shown in Figures 5.9a and 5.9c, most of the algorithms converged more compared to when the resource heterogeneity was high in Figures 5.9b and 5.9d. Moreover, the MOGLPSO algorithm appeared to perform well in the consistent data set. In all sub data sets, MOGLPSO converged more as compared with other algorithms. However, MOGLPSO could not discover other solutions that minimize the flowtime objective efficiently. Also, there are a few numbers of solutions produced by MOGLPSO, which may degrade the overall solution quality. The same was observed with the other algorithms except for the proposed one. The proposed algorithms exhibited better diversity and cardinality over the MOGLPSO algorithm. The proposed algorithms were able to generate non-dominated solutions that include both objectives more efficiently, making them better in terms of the HV indicator. This is because of incorporating the flowtime information in the crossover operator in the proposed algorithms. MO-HGLPSO seemed to
Chapter 5: Genetic Learning PSO Algorithms for Single and Multi-objective Task Matching Problem

(a) Low task heterogeneity and low resource heterogeneity.
(b) Low task heterogeneity and high resource heterogeneity.
(c) High task heterogeneity and low resource heterogeneity.
(d) High task heterogeneity and high resource heterogeneity.

Figure 5.9: Quality of non-dominated solutions for the consistent data set.

Figure 5.10 depicts the solution quality for the inconsistent data set. It is clear that the proposed algorithms outperformed the other algorithms significantly. MO-MGLPSO was the best compared with the other two proposed algorithms. From Figures 5.10a–5.10d, the proposed algorithms were able to converge to the PF with...
less diversity and cardinality. This was also observed with other algorithms, as they had small cardinality. This could be attributed to the nature of the data set, where the number of solutions that satisfy both objectives is small. Finally, Figure 5.11 shows the quality of the Pareto front in the partially-consistent data set. All algorithms were able to locate a good number of solutions. The proposed algorithms again were observed to be the best in terms of convergence, diversity, and cardinality. MOCLPSO was the worst algorithm in this data set also. Moreover, when the task heterogeneity is high, as shown in Figure 5.11c and Figure 5.11d, NSGA-2 and MOGLPSO were not able to locate the Pareto front properly. Another possible observation from Figure 5.10a and Figure 5.10c is that the proposed algorithms tuned to have a bit linear Pareto front when the resource heterogeneity is low. In Figure 5.11b, MO-MGLPSO and MO-µ-MGLPSO contributed more to the Pareto front over MO-HGLPSO. However, the MO-HGLPSO converged more as against to MO-MGLPSO and MO-µ-MGLPSO, as shown in Figure 5.11d. Thus, the proposed algorithms validated their efficiency and applicability for multi-objective task matching problem.

5.7.3 Convergence Rate

To get an intuitive view of the algorithms, the HV value was tracked for every 50 iterations. The objective of this experiment was to measure the needed number of iterations for each algorithm to attain acceptable results. Figure 5.12 shows the convergence of the algorithms in each data set. The proposed algorithms in Figure 5.12a converged rapidly in less than 100 iterations and became more steady during the optimization problem. As noted for the single objective problem, the learning
Chapter 5: Genetic Learning PSO Algorithms for Single and Multi-objective Task Matching Problem

(a) Low task heterogeneity and low resource heterogeneity.

(b) Low task heterogeneity and high resource heterogeneity.

(c) High task heterogeneity and low resource heterogeneity.

(d) High task heterogeneity and high resource heterogeneity.

Figure 5.10: Quality of non-dominated solution for the inconsistent data set.

Scheme in the proposed algorithm helped to preserve some dimensions in the particles, which in turn saved time (iteration) in the search operation. MO-HGLPSO showed the best convergence, followed by MO-MGLPSO and MO-µ-MGLPSO algorithms. MOGLPSO and MOCLPSO showed a better convergence rate than the other algorithms. The proposed algorithms were remarkably better than the other algorithms.
Chapter 5: Genetic Learning PSO Algorithms for Single and Multi-objective Task Matching Problem

(a) Low task heterogeneity and low resource heterogeneity.

(b) Low task heterogeneity and high resource heterogeneity.

(c) High task heterogeneity and low resource heterogeneity.

(d) High task heterogeneity and high resource heterogeneity.

Figure 5.11: Quality of non-dominated solution for the partially-consistent data set in the inconsistent data set. The gap is apparent in Figure 5.12b, demonstrating the efficiency over other algorithms. Another observation is that the proposed algorithms were improved as the number of iterations increased. In the inconsistent data set, the MO-MGLPSO was spotted as the best algorithm. Moreover, MO-μ-MGLPSO was the best over MO-HGLPSO. The NSGA-2 algorithm seemed to cope with the
Figure 5.12: Convergence rate for all data sets.

(a) Convergence rate in the consistent data set.

(b) Convergence rate in the inconsistent data set.

(c) Convergence rate in the partially-consistent data set.

inconsistent more appropriately than the consistent environment. The performance of MOCLPSO retrogressed as the worst algorithm among all algorithms.

Similarly, for the partially-consistent shown in Figure 5.12c, the proposed algorithms were unrivaled over other algorithms. Therefore, based on the results of the experiment, the proposed algorithms are evidently more effective against other well-established multi-objective algorithms.
5.7.4 Cardinality

This experiment aimed to measure the number of solutions (cardinality) generated by the algorithms. It is preferable to get a good number of solutions, so the DM can have more flexibility to choose the most desirable ones. Figure 5.13 shows the cardinality of each algorithm over 35 runs. The proposed algorithms had a more significant number of solutions in comparison to other algorithms for the consistent data set, as shown in Figure 5.13a. MO-HGLPSO and MO-μ-MGLPSO had the highest number of solutions generated, which was 29 solutions. Besides, there were no less than 20 solutions obtained from the proposed algorithms. Other algorithms, such as MOPSO and NSGA-2, had a broader range of solutions, which may be interpreted as less robust algorithms. For the inconsistent data set, the algorithms had a comparative cardinality, as depicted in Figure 5.13b. Also, the outliers of each algorithm are reduced compared with the consistent problem, except for MOPSO and NSGA-2 algorithms. In Figure 5.13c, the proposed algorithms maintained a good number of solutions with no less than 10 solutions for the partially-consistent problem. The experiment revealed that the proposed algorithms are efficacious in terms of the number of solutions.
(a) Number of solutions in the consistent data set.  
(b) Number of solutions in the inconsistent data set. 

(c) Number of solutions in the partially-consistent data set.

Figure 5.13: Number of solutions for all data sets.
5.8 Discussion

The experimental results substantiated the effectiveness of the proposed algorithms over other PSO variants for the task matching problem. Generally, the proposed algorithms provided better statistical results, reliability and consistency, and a fast and accurate convergence rate. However, although the MGLPSO algorithm produced better results for inconsistent and partially-consistent data sets, it did not do so well for the consistent data set. Therefore, we proposed several algorithms to overcome the issues pertinent to this algorithm for certain data sets.

The three task matching scenarios (data sets) considered in the thesis had conflicting characteristics, thereby incorporating different landscape solutions spaces. For the consistent data set, MGLPSO resulted in a load imbalance issue. Following our observations on MGPLSO for this data set and its inefficiency in producing good results, we proposed HGLPSO and $\mu$-MGLPSO. These two proposed techniques incorporate different methods and ideas to improve the performance on the consistent data set. One of the critical issue we considered in designing the algorithms was balancing between exploration and exploitation to improve the search performance.

We conducted various experiments and studied the statistical results. Based on our observation of these results, we found that GLPSO tends to be superior to MGLPSO for the consistent data set. This is due to the different depths in search behaviour between GLPSO and MGLPSO–GLPSO is an exploration technique while MGLPSO is an exploitation technique. By combining these two techniques, we developed the HGLPSO algorithm. One of the main issue with the original MGPLSO was that the algorithm stagnated at local optima. We observed through experimen-
tation the particles learning process in GLPSO and incorporated this into MGPLSO. By doing so, made MGLPSO more comparatively immune to getting stuck in a local optimum for the consistent data set. We also observed that we had to balance between exploration and exploitation throughout the running of the algorithm. We realized the mutation rate affected this balance. Therefore, we assigned different mutation rates to all particles and developed the $\mu$-MGLPSO algorithm. The mutation process is therefore used to manage the balance between exploration and exploitation search. For the consistent data set, having a high mutation rate implies shifting many tasks from the fastest resource to other resources. Hence, we balanced the resource utilization effectively.

Next, the proposed genetic learning algorithms for the multi-objective task matching problem were better compared to other algorithms. Creating the exemplars not only helped to obtain high-quality solutions but also improved the efficiency of MO-MGLPSO for the consistent data set, explicitly. In detail, incorporating the information of the cumulative makespan helped the MO-MGLPSO to be more guided by distributing more tasks to other resources; this consequently reduced the load on the fastest resource and the load balancing became more evenly distributed among the resources. We then incorporated GLPSO’s learning strategy into MO-MGLPSO and developed MO-HGLPSO. This algorithm worked better in low resource heterogeneity environment.

Finally, multi-objective optimization algorithms, MO-HGLPSO and MO-$\mu$-MGLPSO algorithms, were comparative to MO-MGLPSO in inconsistent and partially-consistent data sets. The proposed algorithms performed well on heterogeneous environments.
The proposed algorithms dramatically outperformed their competitors in terms of HV metric and the convergence rate. In addition, the algorithms were particularly superior for partially-consistent data sets. Overall, the statistical results indicated that proposed genetic learning algorithms for multi-objective task matching problem performed well by converging to high-quality non-dominated solutions with less number of iterations.

5.9 Summary

This chapter presented various genetic learning PSO algorithms for single and multi-objective task matching problems. The chapter began with introducing two improved algorithms upon MGLPSO, which are hybrid genetic learning PSO (HGLPSO) and MGLPSO with adaptive mutation ($\mu$-MGLPSO) algorithms. These two algorithms, along with MGLPSO, were tested extensively against state-of-the-art algorithms for single and multi-objective task matching problems. Generally, the results of the single objective demonstrated that the improved algorithms have overall better performance compared with other PSO variant algorithms. In cases where the MGLPSO seemed to struggle, which are consistent and partially-consistent data sets, the improved algorithms were significantly performed better, demonstrating their applicability to different grid environments. Moreover, the improved algorithms were able to yield better results with less number of iterations as the convergence rate test revealed.

For the multi-objective problem, the performance of the genetic learning PSO algorithms was competitive. The resultant non-dominated solutions from the three
genetic learning PSO algorithm were the best in comparison with other algorithms in terms of convergence, solutions distribution, and the number of non-dominated solutions in all data sets. In the next chapter, we study if the proposed techniques for independent tasks can be extended to workflow models. Please note that part of the work done in this chapter has been published in [15].
Chapter 6

Multi-objective Task Matching for Workflow Applications

In many complex applications, the workflow model is adopted to explicitly express the execution order. The predefined QoS in executing the workflow is a challenge due to the incurred data dependencies that may exist in the model that needs to be preserved during execution. So far, we have not addressed such applications. In this chapter, we show the challenges in studying multi-objective optimization workflow problems and our ideas in studying these problems. To this effect, we propose genetic learning PSO algorithms for multi-objective workflow problems.

The proposed algorithms aim to minimize two conflict objectives: makespan and cost of the execution. Section 6.1 presents an introduction to the workflow model in scientific applications. Then, Section 6.2 describes the workflow model and task matching problem for workflow applications. The proposed algorithms for the multi-objective workflow task matching are discussed in Section 6.3. In Section 6.4, the
setup and experimental results are presented, followed by a summary of the chapter in Section 6.6.

6.1 Introduction

Many scientific applications are complex. The data size is large and is both memory and compute intensive. They require large scale machines such as a Grid. Grid requires resources from various and diverse locations [232]. Using data resources on the grid for such applications is complex due to the data dependencies inherent in these problems. These applications follow a workflow model. The workflow model retains two intrinsic properties of the application: specific application requirements and the flow of execution [177]. In general, the workflow model is used in many scientific areas, such as business, computational science, and biology, in which a series of different composite computations are expressed and analyzed. Formally, the workflow model can be defined as a model such that a sequence of tasks can be expressed through the flow of data based on some specific rules to achieve an overall objective [117].

The workflow model originated from business applications. Commercial enterprises adopted the model as a tool to express the business process, coining the term business workflow. The business workflow model was used to automate a series of activities or tasks of an organization [165]. The rapid research advancement in developing workflows for business management has led to establishing the workflow management coalition (WfMC)\(^1\) in 1993. The concept of workflow has been dis-

\(^1\)http://www.wfmc.org/
persed in the scientific communities to represent large-scale data. Scientific workflow has become even more important in the era of big data. According to Barker and Van Hemert [26], the scientific workflow model produces a sequence of analytical steps to represent the procedure of computational experiments. This has aided in analysis, simulation and visualization in process discovery.

Although business and scientific workflows use the same concepts, they have different specific requirements, and hence need separate consideration. The thesis focuses on the scientific workflow or just referred to as workflow.

There are several advantages of using the workflow concept in developing applications on Grid [261]:

- Ability to coordinate the use of distributed resources for dynamic applications.

- Using various resources for different application domains can efficiently reduce the execution time.

- Integrating multiple collaborators involved in different parts of the workflow, fostering distributed collaborations on diverse domains.

The advancement of today’s Grid technologies has enabled various scientific applications with different services to support the workflow. Several scientific applications have adopted the workflow model to develop and implement large-scale experiments on the Grid platform, such as EMAN2 [229] for electron micrograph analysis, WIEN2K [32] for quantum chemistry, Montage [6] for astronomy, GriPhyN [22] for physics, and LEAD [58] for meteorological data analysis and weather forecasting. A workflow management system is used to define, manage, and execute workflows on
the Grid system. There are several workflow management projects developed, e.g.,
ASKALON [89], DAGMan/HTCondor [4], GrADS [5], GridFlow [43], Taverna [187],
Kepler [170], and Pegasus [76]. Through the workflow management system, the fol-
lowing issues are resolved and handled:

- Discover and represent the tasks of a given workflow so that the tasks can be
  executed on the Grid resources.

- Provide information such as availability and capability of the Grid resources.

- Allocate suitable resources (processors, memory, I/O, etc.) to execute the tasks
  efficiently.

The workflow management system plays a crucial role in executing workflows. The
scheduling algorithm considered in the workflow management system is an essential
component, and it can significantly affect the throughput of the Grid system. Gener-
ally, the scheduling algorithms here are guided via a set of QoS requirements provided
by the users. In order to fulfill the QoS requirements, the scheduling algorithms should
make use of the abundance of resources provided in the Grid environment effectively.
Figure 6.1 depicts a high-level overview of the scheduling process of workflows in the
Grid environment. As stated in Chapter 1, the task matching problem is the first
stage of the scheduling process.

6.2 Workflow Model and Preliminaries

The workflow task matching problem considered in this thesis is to find a solution
to match workflow tasks to heterogeneous resources. Also, several constraints may be
involved during the matching process to meet one or more objectives. This section explains in detail the workflow model and formally presents the problem definition.

### 6.2.1 Workflow Model

The workflow model has proven beneficial in simplifying the process of scientific applications with large data set [262]. Workflows are typically expressed as a set...
of computational tasks with predefined dependencies to represent the interaction between the tasks. The model determines the order of execution. Since the dependencies define existing priority relationships between tasks, a task cannot be executed before its predecessors (parents) have completed execution. In the context of scientific workflow, the dependencies represent the data flow between tasks. That is to say, one or more tasks require the output data generated from a parent task to begin processing of the current task. Based on the inherent complexity of the application domain, workflows may need computation or data resources for execution.

The workflow could be CPU-intensive, memory intensive, I/O intensive, or a combination of these [205]. The CPU-intensive workflows tend to have tasks that require high CPU usage, whereas memory intensive workflows require physically high memory usage. The tasks in the I/O intensive workflow need and produce a high amount of data, creating a high number of I/O operations.

Usually, the workflow model is represented using a directed graph. Directed graphs can be categorized as either a directed acyclic graph (DAG) and a directed cyclic graph (DCG) [75]. The former category is frequently used for the workflow. The extensible markup language (XML) is the formal language adopted in the workflow management system to represent the actual workflow. The scope of this thesis is limited to workflows modeled through DAG.

### 6.2.2 Directed Acyclic Graph (DAG)

The DAG representation is regularly adopted in scientific workflows, although there are other representation models [193]. Workflow management systems, such
as Pegasus, DAGMan, and ASKALON, execute workflows represented as DAGs. A DAG is a directed acyclic graph with no cycles. It consists of a set of vertices (or nodes) and a set of directed edges. Formally, a DAG graph can be expressed as $G = \{V, E\}$ where $V$ is the number of the vertices\(^2\), and $E$ is the number of directed edges. In the workflow context, vertices, $V$, represent tasks and edges, $E$, represent dependencies between tasks. Therefore, $G = \{V, E\}$ where $V = \{T_1, T_2, ..., T_n\}$ is a set of tasks and a set of dependencies between tasks $E$ for a given workflow. Data dependency between task $T_i$ and task $T_j$ is represented by as $E_{i,j}$. Also, we say task $T_i$ is parent of task $T_j$ and $T_j$ is a child of task $T_i$. This dependency indicates that task $T_i$ should be executed before task $T_j$. A task without parents is called an *entry* task, and a task without children is called an *exit* task. In this thesis, for the purpose of standardization, all DAGs are assumed to have one entry and one exit task. In case there are many entry or exit tasks, one pseudo (dummy) entry and exit tasks are added with zero execution and communication time.

Figure 6.2 illustrates a sample workflow with ten tasks represented by a DAG. In

\(^2\)Note that the terms vertex and tasks are used interchangeably in this thesis.
this example, task $T_1$ generates three intermediate output data that are used as input for tasks $T_2$, $T_3$, and $T_4$. Additionally, tasks $T_2$, $T_3$, and $T_4$ can not be executed until task $T_1$ is processed and its output data is received.

6.2.3 Resource Model

This thesis focuses on a Grid system consisting of a set of heterogeneous computational resources $R = \{r_1, r_2, r_3, ..., r_m\}$. These resources are fully connected, and communication between them is without contention. Furthermore, resources in $R$ have different specifications, such as computational power, memory, and communication bandwidth, varying the price of these resources per-use. We assume that this information is known prior to the matching process. A performance model is used on resources which estimates the execution time for each task.

6.2.4 Problem Definition: Workflow Task Matching on Grid

Previously in Chapter 4 and Chapter 5, an independent set of tasks were considered for the task matching problem. We assumed there was no dependencies between the tasks. However, the workflow task matching problem considered in this chapter is still the process of matching a set of tasks to a set of resources and orchestrating the matching between tasks and resources such that the order and dependencies between the tasks are preserved. Usually, there is a set of problem variables that defines the task matching problem, such as the characteristics of the workflow (e.g., the dependencies between tasks), the underlying system (e.g., number of resources), objective(s) of the matching (e.g., minimizing makespan), and so on. In general, the
task matching problem is NP-hard [236], even in a simple case, such as matching a set of tasks with uniform weights to an arbitrary number of processors. The problem of finding the matching of a workflow to a set of resources is even more complicated.

Several assumptions are made for the workflow task matching problem [277]:

1. All information needed is available before the matching starts.

2. Tasks cannot be preempted.

3. There is a one to one mapping on resources to tasks.

4. Precedence constraint is considered. A task (except entry task) can not be executed before its parents.

5. The time for setting up the communications is negligible, so there is no communication delay.

6. All resources are fully connected, which means there is a link between any two resources.

The generated matching (solution) for a given DAG is static since we require all necessary information before the matching process. The start time of each task is given and the problem is to assign the task to a resource. Several essential attributes are defined for the workflow task matching problem. Table 6.1 summarizes the notations used in this section.

The Grid framework considered here assumes that there a DAG is represented by \( n \) tasks and \( m \) resources. To represent the bandwidth linkage of the Grid system, each resource has a bandwidth capacity (transfer rate); the bandwidth rate of resource \( s \)
Table 6.1: Terms and notations used in workflow task matching problem.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>EST&lt;sub&gt;i,s&lt;/sub&gt;</td>
<td>Earliest start time of the i&lt;sup&gt;th&lt;/sup&gt; task on the s&lt;sup&gt;th&lt;/sup&gt; resource.</td>
</tr>
<tr>
<td>EFT&lt;sub&gt;i,s&lt;/sub&gt;</td>
<td>Earliest finish time of the i&lt;sup&gt;th&lt;/sup&gt; task on the s&lt;sup&gt;th&lt;/sup&gt; resource.</td>
</tr>
<tr>
<td>AFT&lt;sub&gt;i&lt;/sub&gt;</td>
<td>Actual finish time of the i&lt;sup&gt;th&lt;/sup&gt; task.</td>
</tr>
<tr>
<td>TT&lt;sub&gt;i,j&lt;/sub&gt;</td>
<td>Data transmission time between the i&lt;sup&gt;th&lt;/sup&gt; task and j&lt;sup&gt;th&lt;/sup&gt; task.</td>
</tr>
<tr>
<td>T&lt;sub&gt;i&lt;/sub&gt;</td>
<td>The i&lt;sup&gt;th&lt;/sup&gt; task.</td>
</tr>
<tr>
<td>r&lt;sub&gt;s&lt;/sub&gt;</td>
<td>The s&lt;sup&gt;th&lt;/sup&gt; resource.</td>
</tr>
<tr>
<td>ET&lt;sub&gt;i,s&lt;/sub&gt;</td>
<td>Execution time of the i&lt;sup&gt;th&lt;/sup&gt; task on the s&lt;sup&gt;th&lt;/sup&gt; resource.</td>
</tr>
<tr>
<td>band&lt;sub&gt;s&lt;/sub&gt;</td>
<td>Bandwidth rate of the s&lt;sup&gt;th&lt;/sup&gt; resource.</td>
</tr>
<tr>
<td>data&lt;sub&gt;i,j&lt;/sub&gt;</td>
<td>Size of data transferred from the i&lt;sup&gt;th&lt;/sup&gt; task and j&lt;sup&gt;th&lt;/sup&gt; task.</td>
</tr>
<tr>
<td>ω&lt;sub&gt;i&lt;/sub&gt;</td>
<td>Size of the i&lt;sup&gt;th&lt;/sup&gt; task.</td>
</tr>
<tr>
<td>l&lt;sub&gt;s&lt;/sub&gt;</td>
<td>Computation power of the s&lt;sup&gt;th&lt;/sup&gt; resource.</td>
</tr>
<tr>
<td>avail[s]</td>
<td>The time when the s&lt;sup&gt;th&lt;/sup&gt; resource is available for processing.</td>
</tr>
<tr>
<td>EC&lt;sub&gt;s&lt;/sub&gt;</td>
<td>The usage cost of the s&lt;sup&gt;th&lt;/sup&gt; resource.</td>
</tr>
<tr>
<td>C&lt;sub&gt;exe&lt;/sub&gt;(r&lt;sub&gt;s&lt;/sub&gt;)</td>
<td>The execution cost of using the s&lt;sup&gt;th&lt;/sup&gt; resource.</td>
</tr>
<tr>
<td>Cost(R)</td>
<td>The total cost of using all resources in the Grid system.</td>
</tr>
</tbody>
</table>

is denoted as band<sub>s</sub>. Each edge E<sub>i,j</sub> in the DAG is associated with a weight data<sub>i,j</sub>. The weight indicates the amount of data that needs to be transferred from task T<sub>i</sub> to task T<sub>j</sub>. The transfer time TT<sub>i,j</sub> is defined in Equation 6.1.

\[
TT_{i,j} = \frac{data_{i,j}}{band_s}
\]  

(6.1)

As shown in Equation 6.2, when both tasks T<sub>i</sub> and T<sub>j</sub> are assigned to the same resource
$r_s$, the transfer time is zero. In this case, we assume that the cost of intra-processor data communication is negligible.

$$TT_{i,j} = \begin{cases} 
0 & \text{if } r_s = r_p \\
TT_{i,j} & \text{otherwise}
\end{cases} \quad (6.2)$$

The execution time $ET_{i,s}$ of the $i^{th}$ task on $s^{th}$ resource is calculated as:

$$ET_{i,s} = \frac{\omega_i}{l_s} \quad (6.3)$$

where $\omega_i$ is the size of $i^{th}$ task and $l_s$ is the computation power of $s^{th}$ resource.

The Earliest Start Time $EST_{i,s}$ is the earliest time for $T_i$ on resource $r_s$ to be executed, and the Earliest Finish Time $EFT_{i,s}$ is earliest finish time of $T_i$ on resource $r_s$. $EST$ and $EFT$ are defined as follows [234]:

$$EST_{T_{entry},s} = \text{avail}[s] \quad (6.4)$$

$$EST_{i,s} = \max \left\{ \text{avail}[s], \max_{T_j \in \text{pred}(T_i)} (AFT_j + TT_{j,i}) \right\} \quad (6.5)$$

$$EFT_{i,s} = ET_{i,s} + EST_{i,s} \quad (6.6)$$

where $\text{pred}(T_i)$ is set of parents of task $T_i$, $\text{avail}[s]$ is the earliest time when resource $r_s$ is ready for task execution. That is, it is the earliest time for resource $s$ to complete all previous tasks before preparing to execute another task $T_i$. The Actual Finish Time, $AFT_j$, is the finish time of predecessors of task $T_i$. After the execution of task $T_i$ is completed, the value of $AFT_i$ is $EFT_{i,s}$. When all tasks of a given workflow have been executed, the makespan is the actual time of the last task, as shown in
Equation 6.7. If there are several $T_{exit}$, the makespan here is the highest completion time among them.

\[
\text{makespan} = AFT_{T_{exit}} \tag{6.7}
\]

The “pay-per-use” paradigm has been utilized in the utility Grid where users need to pay for the resource usage [103]. The incur cost, $C_{exe}(r_s)$, of using resource $r_s$ to execute a set of tasks $n$ is computed using Equation 6.8. Finally, the cost is defined in Equation 6.9 [227].

\[
C_{exe}(r_s) = \sum_{i=1}^{n} EC_s \times (ET_{i,s} + TT_{i,j}) \tag{6.8}
\]

\[
\text{Cost}(R) = \sum_{s=1}^{m} C_{exe}(r_s) \tag{6.9}
\]

### 6.2.5 Workflow Task Matching Example

An illustrative example is provided for a better understanding of the workflow task matching problem. The example is based on the DAG in Figure 6.2, which is composed of ten tasks. The Grid system incorporates three resources with different capabilities, where resource $r_1$ and $r_3$ are the slowest and the fastest, respectively. The bandwidth rate are same for all resources, which we assume is one time unit per second. Table 6.2 shows the estimated execution time of each task on all resources in seconds. Furthermore, Table 6.3 presents the communication time required between tasks based on Figure 6.2. For the sake of conciseness, the example considers only minimizing the makespan.

In this example, $T_2$, $T_4$, $T_8$, and $T_9$ are assigned to $r_1$ for execution. Resource $r_2$ is matched to execute $T_1$, $T_6$, $T_5$, and $T_{10}$. Tasks $T_3$ and $T_7$ are matched on to
Table 6.2: Sample execution time of all tasks on different resources based on Figure 6.2.

<table>
<thead>
<tr>
<th></th>
<th>$T_1$</th>
<th>$T_2$</th>
<th>$T_3$</th>
<th>$T_4$</th>
<th>$T_5$</th>
<th>$T_6$</th>
<th>$T_7$</th>
<th>$T_8$</th>
<th>$T_9$</th>
<th>$T_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1$</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>11</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>$r_2$</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>5</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>$r_3$</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 6.3: Sample data transfer time based on Figure 6.2.

<table>
<thead>
<tr>
<th></th>
<th>$T_1$</th>
<th>$T_2$</th>
<th>$T_3$</th>
<th>$T_4$</th>
<th>$T_5$</th>
<th>$T_6$</th>
<th>$T_7$</th>
<th>$T_8$</th>
<th>$T_9$</th>
<th>$T_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$T_2$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$T_3$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$T_4$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$T_5$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$T_6$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$T_7$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$T_8$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>$T_9$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>$T_{10}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

resource $r_3$. The execution process starts with task $T_1$ executed on resource $r_2$ since it is the only entry task. Based on the execution time from Table 6.2, resource $r_2$ takes 2 time unit to execute $T_1$. In other words, since task $T_1$ is the entry task, the
$EST$ is zero based on Equation 6.4. Then, the $EFT$ of task $T_1$ on resource $r_2$ is simply the execution time from Table 6.2. Resource $r_2$ transfers the output of task $T_1$ to tasks $T_2$, $T_3$, and $T_4$, which are assigned to different resources. The transfer time, $TT$, takes 7 time unit to complete. Tasks $T_2$, $T_3$, and $T_4$ use Equation 6.5 to estimate the $EST$ for execution. For task $T_2$, the $EST$ is the maximum value between the available time of resource $r_1$, which is zero since there is no task assigned yet, and the maximum completion time (execution and transfer) of its parents. Here task $T_2$ has only one parent and took 9 time units to be completed. Thus, the $EST$ of task $T_2$ is 9, and the same $EST$ for task $T_3$. For task $T_4$, the $EST$ here is the maximum value between the available time of resource $r_1$, which is 19 units time, and the completion time of its parent, which is 9 units of time. Thus, the $EST$ of task $T_4$ is set to 19. After completing the process of the three tasks, the $AFT$ of each task is updated to be equal to the $EFT$. For task $T_5$, the available time for resource $r_2$ is 9. Also, there are two parents for task $T_5$, of which the maximum completion time is selected to estimate the $EST$. The completion time of tasks $T_2$ and $T_3$ are 19 and 20, respectively. The $EST$ of task $T_5$ is now 20. The same process is applied for tasks $T_6$, $T_7$, and $T_8$. Tasks $T_8$ and $T_9$ are assigned to be executed on the same resource $r_1$. Since task $T_8$ is the only parent of task $T_9$, the $TT$ of task $T_8$ is zero, as both tasks are matched to the same resource. The $EST$ of task $T_{10}$ is selected as previously explained, and the $AFT$ of the task $T_{10}$ is the makespan. Figure 6.3 visualizes the matching process of workflow in Figure 6.2, in which the white grids indicate the execution time, whereas the grey grids represent the transfer time.
6.3 Workflow-based Multi-objective Genetic Learning PSO

Previously in Chapter 5, three multi-objective genetic learning algorithms, MO-MGLPSO, MO-HGLPSO, and MO-\(\mu\)-MGLPSO, were introduced for the independent task matching problem. The proposed algorithms in this chapter amalgamate the genetic operators to create exemplars by which particles are guided to find non-dominated solutions. We try to extend the algorithms to the workflow task matching problem. The extension is not as is. Some modifications were necessary. For example, the crossover operator is modified to work on the workflow model, whereas mutation and selection operators are kept the same as introduced in Chapter 5. The generated exemplars from the crossover operator are constructed iteratively by matching one task at a time based on the execution time on the matched resources. The execution and transfer time of the data transfer between tasks is considered in the crossover operator. In the DAG, there may be tasks with no parents (dependencies) and others that are dependent. Intuitively, tasks that have no parents are matched first, as there are no data required to the start processing. In this case, the crossover is based only...
on the execution time. Otherwise, the execution and transfer time are used for the crossover operation. The crossover operator procedure is explained below.

For each dimension of the exemplar $E_i$, a random particle $k$ is selected. As introduced earlier in Chapter 4, the direct representation is used for the solution encoding; the index indicates the task number, and the value in that dimension is the matched resource. If the $d^{th}$ dimension represents a task with no parents, the execution time of the given task (dimension) is estimated from both the current particle $i$ and random particle $k$. We also calculate the cost of using the matched resource to maximize profit. For each dimension, therefore, the time and cost are considered in the crossover operation. If the time and cost produced from matching resource of the current particle $i$ dominates that of the randomly selected particle $k$, the value of the corresponding $d^{th}$ dimension in the particle $i$ is copied into the same corresponding $d^{th}$ dimension of the exemplar $E_i$. However, in case that the time and cost resulted from matching the resource of the randomly selected particle $k$ dominates that of the current particle $i$, then the crossover is performed between the current particle $i$ and the randomly selected particle $k$ and the result is stored in the $d^{th}$ corresponding dimension of the exemplar $E_i$. Finally, if both results from the current particle $i$ and the randomly selected particle $k$ are non-dominated to each other, one of the two procedures is selected randomly\(^3\).

For tasks with parents, the data transfer time is calculated. We need to calculate the data transfer paths taken to reach the task node. The path with minimum cost is selected. But, when to start executing the task on the resource depends on the maximum waiting time for the task to be selected for execution. This is incorporated

\(^3\)Copying from particle $i$ or crossover between particles $i$ and $k$. 

in the crossover operation. For each dimension of the dependent tasks (with parents) a particle is selected randomly. The time and cost are calculated. To calculate the time, for each parent, the transfer time is estimated in addition to the execution time of the task on the matched resource. For the cost, we only consider the cost of resource usage (the matched resource). Hence, if the maximum time and cost resulting from particle \(i\) dominate that of randomly selected particle \(k\), the value of the corresponding dimension in particle \(i\) is saved in the exemplar \(E_i\); otherwise, the crossover is carried out if particle \(k\) dominates particle \(i\). When both particles are non-dominated to each other, we randomly select a particle for copying the values or perform crossover between particle \(i\) and particle \(k\). After each update of exemplar \(E_i\), the load of resources is updated to promote resource load balancing. Algorithm 11 shows the pseudocode of the modified crossover for the workflow task matching problem.

In the interests of clarity, an example is provided based on the workflow in Figure 6.2 to illustrate the modified crossover. In Algorithm 11, there are two cases that we consider: tasks with parents (line 17) and tasks without parents (line 4). We illustrate these cases using task 1 and task 7.

In Figure 6.2, task 1 has no parents. So the estimate time includes the execution time the cost of the usage of the matched resources. Let us assume that the matched resource from particle \(i\) is 2 and particle \(k\) is 1. According to Table 6.2, task 1 will run faster on resource 2, which in this case, particle \(i\) has better matching resource than particle \(k\). Also, the costs of using resource 2 and resource 1 are estimated. Based on the results of particle \(i\) and particle \(k\), the crossover may or may not take place, as shown in Algorithm 11. For task 7, there are two parents that supposedly match
Algorithm 11 Crossover operation for workflow task matching problem

1: /*Crossover operation*/
2: for $d = 1$ to $D$ do
3: randomly pick a particle $k \in \{1, 2, 3, ..., N\}$
4: if task $d$ is an entry task then // Task $d$ has no parents.
5: if $f(x_{k,d}) \prec f(x_{i,d})$ then // Particle $k$ dominates particle $i$.
6: $E_{i,d} = \alpha \cdot x_{i,d} + (1 - \alpha) \cdot x_{k,d}$
7: else if $f(x_{i,d}) \prec f(x_{k,d})$ then // Particle $i$ dominates particle $k$.
8: $E_{i,d} = x_{i,d}$
9: else // Particle $i$ and particle $k$ are non-dominated.
10: $r \sim U(0, 1)$ // Generate a random number.
11: if $r < 0.5$ then
12: $E_{i,d} = x_{i,d}$
13: else
14: $E_{i,d} = \alpha \cdot x_{i,d} + (1 - \alpha) \cdot x_{k,d}$
15: end if
16: end if
17: else // Task $d$ has parents.
18: find the path that maximize the execution time for $x_{i,d}$ and $x_{k,d}$
19: if $f(x_{k,d}) \prec f(x_{i,d})$ then // Particle $k$ dominates particle $i$.
20: $E_{i,d} = \alpha \cdot x_{i,d} + (1 - \alpha) \cdot x_{k,d}$
21: else if $f(x_{i,d}) \prec f(x_{k,d})$ then // Particle $i$ dominates particle $k$.
22: $E_{i,d} = x_{i,d}$
23: else // Particle $i$ and particle $k$ are non-dominated.
24: $r \sim U(0, 1)$ // Generate a random number.
25: if $r < 0.5$ then
26: $E_{i,d} = x_{i,d}$
27: else
28: $E_{i,d} = \alpha \cdot x_{i,d} + (1 - \alpha) \cdot x_{k,d}$
29: end if
30: end if
31: end if
32: update the load of the resource based on $E_{i,d}$
33: end for
to resources before starting to process task 8. There are two possible paths, one of which maximizes the execution time if selected. Let us suppose that particle $i$ and particle $k$ have matched resources 2 and 3 for task 7, respectively. From Table 6.3, the transfer time from task 3 to task 7 takes 9 sec and 2 sec to transfer data from task 4 to task 7. Certainly, the path from task 3 to task 7 is selected since the goal is to minimize the path that maximizes the time (waiting time). At this point, particle $i$ takes time 17 sec (9 sec transfer time and 8 sec execution time on resource 2), and particle $k$ needs 14 sec. The cost is calculated for both resources, and the results are compared the same shown in Algorithm 11. Every time a dimension in the exemplar $E_i$ is updated, the resource load is updated, i.e., we add the time taken by the resource when executing and sending data to restrain resource imbalance issue. Note that in the example, the bandwidth is assumed to be the same between resources, which is not necessarily true in a heterogeneous environment like that of the Grid system.

As already explained, the other two operators, which are the mutation and the selection operators, are kept as introduced for multi-objective task matching problem. Furthermore, the PSO has no change for the workflow task matching problem. So, MO-MGLPSO, MO-HGLPSO, and MO-$\mu$-MGLPSO adopt the crossover in Algorithm 11 with the same mutation, selection, and PSO introduced in Chapter 5.

### 6.3.1 Task Priority

All feasible complete matching is subject to precedence constraints in the workflow. That is, a task can not be processed before its predecessors. For instance, in the workflow shown in Figure 6.2, task 2, task 3, and task 4 can be processed after
task 1. In our proposed algorithms, exemplars are constructed by matching tasks individually. In the workflow depicted in Figure 6.2, tasks are in order, where this is not the case in some workflows. In other words, task priority is different, depending on the structure of the workflow. To circumvent this obstacle, a level-by-level graph partitioning is adopted. Before the proposed algorithms begin, the workflow is partitioned into \( l \) levels. Each level has a set of tasks that are independent of each other, i.e., there is no data transfer between tasks of the same level. Tasks at level 1 have no predecessors, and there are no successors for tasks at level \( l \). A level \( k \) is assigned to each task (excluding the entry task) using Equation 6.10. The level \( k \) for task \( i \) is the maximum level of its predecessors plus 1; thus, there is at least one parent at level \( k - 1 \) for each task.

\[
\text{level } k = \left\{ \max_{T_j \in \text{pred}(T_i)} (\text{level } k) \right\} + 1 \tag{6.10}
\]

After partitioning the workflow, the tasks are sorted in ascending order based on the level. Hence, the tasks can be processed one by one without violating priority constraints. An illustrative example of partitioning the workflow is shown in Figure 6.4. The workflow in Figure 6.2 is partitioned into several levels using Equation 6.10. At each level, the tasks are independent and can be processed in any order.

### 6.3.2 Match Generation

The pseudocode in Algorithm 12 is adopted to transform the positions in the proposed algorithms into a candidate matching to calculate the makespan and cost associated with it. Initially, the workflow is partitioned into levels, and the tasks are arranged in order based on their level. Note that this process is done once before the
algorithms start. We have included this in Algorithm 12 to show that the tasks are sorted by their level (priority). Makespan, cost, the available time for each resource, and the actual finish time for each task are set to 0. The algorithms then start by estimating the execution time matrix for each task on every resource and constructing the data size transfer matrix between tasks; both of these matrices are represented in an “adjacency matrix”. Table 6.2 is an example of exe_time matrix. The entry exe_time[s][i] represents the time task i requires resource s and is estimated using Equation 6.3. The entry trans_size[j][i] indicates the output data size from task j to task i. This entry is used later to estimate the transfer time. At this point, all necessary information is available to generate the matching and mapping between tasks and resources are identified.

The value of EST is based on two cases. If the task has no parents, then the task can start running as soon as the resource is available. The second case is when the task has one or many parents. In this case, the task can start as soon as the last
parent task completes the execution and sends all required output data. Nevertheless, when the matched resource is busy processing other tasks, the execution of the task is postponed until the resource becomes available. Therefore, the higher number of the processing time of the last parent task and the available time of the resource is selected as $EST$. The processing time $PT$ is calculated based on two values, which are the execution time of the task and the transfer time needed for the task to send the output data. The former is simply the entry $exe\_time[s][i]$ where $s$ and $i$ are the number of resources and tasks, respectively. The latter is estimated through converting the entry $trans\_size[j][i]$ using Equation 6.1 and cumulatively adding the transfer time to $transfer$ only when a child of the task is matched to a different resource, as shown in Equation 6.2. Finally, the value of $EFT$ is obtained by adding the values of $PT$ and $EST$, which is used to update the value of $avail[s]$ and $AFT_{task}$. Once the algorithm is done processing the tasks of the workflow, the makespan is obtained by Equation 6.7. Also, using Equations 6.8 and 6.9, the cost of the matching is determined.

The Algorithm 12 is used to calculate the fitness value of the exemplar $E_i$ after the mutation operator and particles in the PSO phase to find non-dominated solutions.

### 6.3.3 Objective function

The multi-objective workflow task matching problem takes two objectives to be optimized. This first objective is $makespan$. The makespan here is the completion time of the last task in the Grid system. The second objective is $cost$, which is the cost of executing all the tasks on the Grid system. The cost of using the resources is inversely proportional to the processing time. Hence, these two objectives are
Algorithm 12 Match generation

1: partition the workflow
2: set the priority of the tasks based on the level
3: sort the tasks based the priority
4: makespan = 0
5: cost = 0
6: avail = 0 // set the available time for all resources.
7: $AFT = 0$ // set the actual finish time for all tasks.
8: calculate exe_time matrix $[R] \times [T] $ // execution time of tasks on resources.
10: for $i = 0$ to $|T| - 1$ do // for every task in the workflow.
11:     task = $T[i]$
12:     if task has no parents then
13:         set $EST_{i,s} = avail[s]$
14:     else
15:         set $EST_{i,s} = \max\{avail[s], \max_{T_j \in \text{pred}(T_i)} AFT_j\}$
16:     end if
17:     set $exe = \text{exe}_\text{time} [s][\text{task}]$
18:     for each child of task do
19:         calculate the transfer time according to Equations 6.1 and 6.2
20:     end for
21:     $PT = exe + \text{transfer}$  // processing time.
22:     $EFT_{i,s} = PT + EST_{i,s}$
23:     set $avail[s]$ and $AFT_{task}$ to $EFT_{i,s}$
24: end for
25: calculate the makespan according to Equation 6.7
26: calculate the cost according to Equations 6.8 and 6.9
conflicting with each other. The objective function is formulated as:

$$\text{Minimize } f = (\text{makespan, cost})^T.$$  \hspace{1cm} (6.11)

### 6.4 Experimental Design

The experimental setup for the workflow matching problem is described in this section. The Grid environment for the experiments is composed of five resources. The resources follow the consistent model in which resource 1 is the slowest, and resource 5 is the fastest for all tasks. The usage cost of each resource was estimated based on the equipment cost and the cost of return within two years. For example, we assumed that resource 1 costs 1000 dollars. Also, we added another 50\% of the cost as other possible expenses such as electricity and maintenance. So, resource 1 would cost around 1500 dollars for the next two years. Now, the cost is converted into a cycle period of an hour that covers the cost in two years, and in this case, this resource should be leased 0.09 per hour to cover the cost of the resource in two years. Table 6.4 shows the cost of each resource. The network topology of the Grid system is a fully connected network, meaning that every resource is connected to all other resources in the system. This is an assumption that we have maintained in the experimental setup. In the simulation, we assume the network bandwidth between resources is heterogeneous. That is, the network bandwidth, for example, between resource 1 and resource 2 is not the same as that between resource 1 and resource 3. The network bandwidth here means the transfer rate; higher bandwidth means faster transfer rate. Table 6.5 presents the network bandwidth between resources. The network bandwidth is randomly generated. The bandwidth in Table 6.5 is in MB per
Table 6.4: Cost of the resources.

<table>
<thead>
<tr>
<th>Resource number</th>
<th>Cost in two years</th>
<th>Cost per hour</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1500</td>
<td>0.09</td>
</tr>
<tr>
<td>2</td>
<td>3000</td>
<td>0.19</td>
</tr>
<tr>
<td>3</td>
<td>6000</td>
<td>0.37</td>
</tr>
<tr>
<td>4</td>
<td>12000</td>
<td>0.74</td>
</tr>
<tr>
<td>5</td>
<td>15000</td>
<td>0.93</td>
</tr>
</tbody>
</table>

Table 6.5: Transfer bandwidth between resources.

<table>
<thead>
<tr>
<th>r1</th>
<th>r2</th>
<th>r3</th>
<th>r4</th>
<th>r5</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>0</td>
<td>6</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>r2</td>
<td>6</td>
<td>0</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>r3</td>
<td>8</td>
<td>5</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>r4</td>
<td>8</td>
<td>6</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>r5</td>
<td>10</td>
<td>8</td>
<td>9</td>
<td>1</td>
</tr>
</tbody>
</table>

second. For instance, the transfer speed between resource 1 and resource 2 is 6 MB per second.

The experiments were simulated on a PC featuring an AMD A8-3870 APU operating at 3.0 GHz with 8 GB of RAM using Python. All initial solutions were generated randomly. The population size for all algorithms was set to 20 particles. The maximum number of iterations was 500 for each optimization process. The data set considered in the experiments were Real-world Workflow Applications. The ex-
Experiments were carried out on extremely large workflows, 1000 tasks and repeated on average 30 times.

### 6.4.1 Real-world Workflow Applications

To evaluate our proposed algorithm, five different real-world workflows were selected as benchmarks. The workflows are *Inspiral*, *Montage*, *Epigenomics*, *CyberShake*, and *SIPHT*. These workflows have been used frequently used in scheduling optimization problems [50; 242]. The workflows are published by Pegasus project⁴, a workflow management system developed at University of Southern California [77]. Each workflow is generated in an XML file, describing the workflow properties and dependencies. For our experiments, we parsed the XML file of each workflow and extracted the tasks dependencies and other necessary information such as the execution time of each task, the size of input and output data, and the task’s ID. For the purpose of illustration, the real-world workflows are depicted in Figure 6.5. Depending on the domain for which they are designed, the aforementioned workflows have different topological structures. Additionally, these workflows have different computational characteristics in terms of CPU, memory, and data transfer. The inspiral benchmark workflow, as shown in Figure 6.5a, is a memory-intensive workflow used in the physics domain to identify gravitational waves. Montage benchmark workflow is an I/O intensive workflow and is used to generate mosaics of the sky by a set of images. Thus, for a vast region of the sky, astronomers can create a composite picture. Montage workflow is shown in Figure 6.5b. In the bioinformatics

⁴https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator
field, epigenomics and SIPHT workflows are adopted. Epigenomics is used for the automation of sequencing operations for many genomes, whilst SIPHT workflow is used by the national center for biotechnology information [2] to automate the search of untranslated RNAs (sRNA) of bacterial replicons. Epigenomics and SIPHT are CPU-intensive workflows, and are depicted in Figures 6.5c and 6.5e, respectively. Finally, CyberShake is a memory and data-intensive workflow application. CyberShake is used by the Southern California Earthquake Center (SCEC) [3] to create proba-
bilistic hazard curves using seismograms of a region of interest. A simple structure of the CyberShake workflow is illustrated in Figure 6.5d. More details about these workflows can be found in [29; 135].

6.4.2 Quality Indicator for MOP

There are various quality indicators used to evaluate the performance of MOEAs [285]. Among them, the hypervolume (HV), which has been introduced in Chapter 5, and inverted generational distance (IGD) [127] are commonly used indicators. The HV and IGD are popular indicators in evolutionary multi-objective optimization (EMO) community, as they measure the convergence to the Pareto front and diversity of solutions. A larger value of HV or a smaller value of IGD is preferable, indicating better performance of the algorithm.

The IGD of an obtained solution set is the average distance from each point in the reference set (Pareto front) to the nearest points of the obtained solution set. Let $A = \{a_1, a_2, ..., a_{|A|}\}$ be the obtained solution set and $Z = \{z_1, z_2, ..., z_{|Z|}\}$ be the reference set. The IGD of the solution set $A$ is estimated using the reference set $Z$ as

$$IGD_Z(A) = \frac{1}{|Z|} \sum_{j=1}^{|Z|} \min \{d(z_j, a_k) | a_k \in A\}$$

(6.12)

where $d(z_j, a_k)$ is the Euclidean distance between a reference point $z_j$ and a solution point $a_k$. For better visualization of IGD, Figure 6.6 provides a pictorial example. As can be seen, the IGD metric requires a reference set to be calculated. Unfortunately, the reference set of real-world optimization problems, including task matching problem, is unknown. One common practice of approximating the reference set is to combine all results from the tested algorithms and select the non-dominated so-
For our experiments, we ran each algorithm 10 times on each workflow and sampled over 800 solutions combined from all algorithms for each workflow. Then, we selected non-dominated points from the collected solutions to be the approximated reference set (Pareto front). Finally, the best 50 non-dominated solutions based on the density of crowding distance are chosen to be reference set.

The HV metric calculates the area enclosed between the obtained solution set and the reference set. To calculate the HV, we chose the nadir points from the approximated reference set. The nadir point defines the boundary of the Pareto front, i.e., the extreme point that maximizes the objectives. The nadir point was used to normalize the obtained results were in the range $[0, 1]$. After normalizing the results, we selected a slightly worse point, $[1.1, 1.1]$ as a reference point to calculate the HV, as recommended in $[123; 267; 276]$. The value of HV could be zero if the obtained solution is not close enough to the reference set $[156; 155]$. 

Figure 6.6: Illustration of IGD indicator.
Table 6.6: Statistical results of MOEA variants.

<table>
<thead>
<tr>
<th></th>
<th>Inspiral</th>
<th>Montage</th>
<th>Epigenomics</th>
<th>CyberShake</th>
<th>SIPHT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HV</td>
<td>IGD</td>
<td>HV</td>
<td>IGD</td>
<td>HV</td>
</tr>
<tr>
<td><strong>MO-MGLPSO</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>± 6.43E-01</td>
<td>± 2.13E-01</td>
<td>± 1.81E-01</td>
<td>± 3.44E-01</td>
<td>± 3.42E-01</td>
<td>± 0.00E+00</td>
</tr>
<tr>
<td>± 6.90E-02</td>
<td>± 5.71E-02</td>
<td>± 4.30E-02</td>
<td>± 8.68E-02</td>
<td>± 8.56E-02</td>
<td>± 0.00E+00</td>
</tr>
<tr>
<td>± 8.02E-01</td>
<td>± 1.46E-01</td>
<td>± 1.79E-01</td>
<td>± 7.23E-01</td>
<td>± 1.52E-01</td>
<td>± 2.85E-01</td>
</tr>
<tr>
<td>± 7.67E-02</td>
<td>± 3.15E-02</td>
<td>± 2.03E-02</td>
<td>± 5.78E-02</td>
<td>± 6.52E-02</td>
<td>± 2.68E-01</td>
</tr>
<tr>
<td>± 6.73E-01</td>
<td>± 2.14E-01</td>
<td>± 1.88E-01</td>
<td>± 3.98E-01</td>
<td>± 3.13E-01</td>
<td>± 0.00E+00</td>
</tr>
<tr>
<td>± 7.06E-02</td>
<td>± 4.85E-02</td>
<td>± 4.55E-02</td>
<td>± 8.94E-02</td>
<td>± 9.20E-02</td>
<td>± 2.89E-01</td>
</tr>
<tr>
<td>± 6.73E-01</td>
<td>± 2.14E-01</td>
<td>± 1.88E-01</td>
<td>± 3.98E-01</td>
<td>± 3.13E-01</td>
<td>± 0.00E+00</td>
</tr>
<tr>
<td>± 7.06E-02</td>
<td>± 4.85E-02</td>
<td>± 4.55E-02</td>
<td>± 8.94E-02</td>
<td>± 9.20E-02</td>
<td>± 2.89E-01</td>
</tr>
<tr>
<td>± 6.73E-01</td>
<td>± 2.14E-01</td>
<td>± 1.88E-01</td>
<td>± 3.98E-01</td>
<td>± 3.13E-01</td>
<td>± 0.00E+00</td>
</tr>
<tr>
<td>± 7.06E-02</td>
<td>± 4.85E-02</td>
<td>± 4.55E-02</td>
<td>± 8.94E-02</td>
<td>± 9.20E-02</td>
<td>± 2.89E-01</td>
</tr>
<tr>
<td>± 6.73E-01</td>
<td>± 2.14E-01</td>
<td>± 1.88E-01</td>
<td>± 3.98E-01</td>
<td>± 3.13E-01</td>
<td>± 0.00E+00</td>
</tr>
<tr>
<td>± 7.06E-02</td>
<td>± 4.85E-02</td>
<td>± 4.55E-02</td>
<td>± 8.94E-02</td>
<td>± 9.20E-02</td>
<td>± 2.89E-01</td>
</tr>
</tbody>
</table>

6.5 Results and Discussion

The proposed algorithms were compared with the same MOEAs in Chapter 5. In addition, the same parameter settings were used as shown in Table 5.6, as reported by their references.

6.5.1 Comparison with MOEAs Variants for Extremely Large-scale Workflows

To analyze the proposed algorithms, the value of HV and IGD are tabulated in Table 6.6, in which the mean (the top vale) and the standard deviation are reported. The best mean of the HV and IGD in each workflow are marked in boldface for clarity. It can be seen from Table 6.6 that MO-HGLPSO performed the best among the proposed algorithms and the other algorithms in general. More specifically, MO-
HGLPSO was the best for the inspiral workflow in terms of IGD value, and HV and IGD values for Montage and Cybershake workflows. For the inspiral workflow, MO-$\mu$-MGLPSO was better than MO-HGLPSO by 0.12% for HV. However, MO-HGLPSO was the best by 3.42% in IGD compared with MO-$\mu$-MGLPSO. A possible explanation is that the solution of MO-HGLPSO was more diverse, whereas the solution of MO-$\mu$-MGLPSO was slightly closer to the convex portion of the Pareto front. The MO-MGLPSO algorithm was ranked the last algorithm regarding the IGD value, indicating that the algorithm failed to converge properly to the Pareto front. NSGA-2 algorithm also showed poor performance for the inspiral workflow, scoring the worst algorithm for the HV metric. MOCLPSO and MOPSO showed competitive results for both HV and IGD metric. As for montage workflow, MO-HGLPSO was the best algorithm in terms of HV and IGD. Compared with NSGA-2, MO-HGLPSO achieved better performance by 51.30% for HV metric and 38.27% for IGD metric. Additionally, MO-HGLPSO outperformed MOPSO by 9.13% and 22.96% regarding IGD and HV, respectively. MO-MGLPSO ranked the second after MO-HGLPSO for IGD and the fourth for HV. Similarly, MO-$\mu$-MGLPSO was the second among the algorithms for the HV metric and the fifth for the IGD metric. The performance of MOCLPSO degraded in the Montage workflow and scored as the worse algorithm in both IGD and HV metrics. The superiority of MO-HGLPSO continued for the Cybershak workflow. In fact, for the Cybershak workflow application, only MO-HGLPSO and MO-$\mu$-MGLPSO were converged significantly better than the other algorithms. The other algorithms were not even close enough to calculate the HV metric, which can be considered as a failure run. It seems that getting a good convergence for the
Cybershak workflow application is more complicated compared with other workflows. For MO-HGLPSO, there were two out of 30 runs that were not close for HV calculation, and 13 out of 30 runs were far from the Pareto front for the MO-µ-MGLPSO algorithm. Thus, MO-HGLPSO was more reliable and stable with respect to other algorithms.

Despite the excellent performance of the MO-HGLPSO and MO-µ-MGLPSO, there were two workflows in which they were inferior to their competitors; these workflows are the epigenomics and SIPHT. Epigenomics and SIPHT are CPU-intensive workflows, as described before. In both workflows, MOPSO was the best with respect to HV and IGD metrics. For the epigenomics workflow, MOPSO outperformed MO-HGLPSO by 6.22% and 39.44% in terms of HV and IGD, respectively. The next algorithm that performed better was MO-HGLPSO, followed by MO-µ-MGLPSO with respect to the HV metric. MO-MGLPSO ranked last showing poor performance in HV and IGD metrics. The proposed algorithms performed poorly in the SIPHT workflow. The efficiency of MO-HGLPSO and MO-µ-MGLPSO were weaker than MOPSO, NSGA-2, and CLPSO algorithms.

To understand the performance of the algorithms, a visual representation of the HV and IGD values are shown in Figures 6.7 and 6.8, respectively. Obviously, from these figures, the MO-HGLPSO algorithm maintained an excellent performance in most workflows except the SIPHT workflow. For the inspiral workflow, as shown in Figures 6.7a and 6.8a, MO-HGLPSO, and MO-µ-MGLPSO showed competitive results. For the HV metric, the obtained results of MO-µ-MGLPSO were slightly better than that of MO-HGLPSO, as indicated by the box plot of each algorithm. On Con-
trary, MO-HGLPSO was clearly better as shown in Figure 6.8a for IGD metric. The produced box plot of MO-HGLPSO was more condensed with less median. Furthermore, from Figure 6.7a, MO-HGLPSO and MO-μ-MGLPSO were the best compared with other algorithms for HV metric where their results were competitive with that of CLPSO and MOPSO for IGD metric in Figure 6.8a. Figure 6.7b depicts the obtained HV for Montage workflow, in which MO-HGLPSO was significantly superior to all algorithms. Moreover, MO-MCLPSO was noted as the second best algorithm, followed by CLPSO and MO-MGLPSO algorithms. As for the IGD metric depicted in Figure 6.8b, MO-MGLPSO, MOGLPSO, and MO-HGLPSO were competitive, showing almost the same performance. NSGA-2 and MOCLPSO algorithms were substandard
for HV and IGD metrics. MO-HGLPSO outperformed dramatically the competitors in the cybershake workflow in terms HV and IGD metrics. From Figure 6.7d, MO-HGLPSO as well as MO-µ-MGLPSO were the only algorithms to converge to the Pareto front. Moreover, the produced box in Figure 6.8d for the MO-HGLPSO algorithm was narrow with low median value, indicating that the algorithm was more competent in getting high-quality solutions for such workflow.

For the epigenomic workflow, the results obtained by MOPSO were the best; the box plot of MOPSO in Figure 6.7c has a higher median and Q3 (the third quarter) in comparison to that of other algorithms. Also, MO-HGLPSO, MO-µ-MGLPSO, and MOCLPSO obtained medians almost at the same level. Similarly, as shown...
Table 6.7: Friedman rank test of the MOEAs for HV metric.

<table>
<thead>
<tr>
<th>Workflow</th>
<th>Friedman ranks</th>
<th>MO-MGLPSO</th>
<th>MO-HGLPSO</th>
<th>MO-μ-MGLPSO</th>
<th>MOGLPSO</th>
<th>MOCLPSO</th>
<th>MOPSO</th>
<th>NSGA-2</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inspiral</td>
<td>Friedman ranks</td>
<td>5.47</td>
<td>2.13</td>
<td>1.83</td>
<td>5.13</td>
<td>3.40</td>
<td>3.83</td>
<td>6.20</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td></td>
<td>Rank</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Montage</td>
<td>Friedman ranks</td>
<td>4.00</td>
<td>1.30</td>
<td>2.23</td>
<td>3.17</td>
<td>6.47</td>
<td>4.60</td>
<td>6.23</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td></td>
<td>Rank</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Epigenomics</td>
<td>Friedman ranks</td>
<td>6.30</td>
<td>2.83</td>
<td>3.13</td>
<td>5.80</td>
<td>3.00</td>
<td>1.93</td>
<td>5.00</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td></td>
<td>Rank</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>CyberShake</td>
<td>Friedman ranks</td>
<td>4.75</td>
<td>1.47</td>
<td>2.78</td>
<td>4.75</td>
<td>4.75</td>
<td>4.75</td>
<td>4.75</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td></td>
<td>Rank</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>SIPHT</td>
<td>Friedman ranks</td>
<td>6.13</td>
<td>4.13</td>
<td>4.23</td>
<td>5.43</td>
<td>2.13</td>
<td>1.60</td>
<td>4.33</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td></td>
<td>Rank</td>
<td>7</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Average rank</td>
<td>5.4</td>
<td>1.8</td>
<td>2.6</td>
<td>4.6</td>
<td>3.6</td>
<td>2.8</td>
<td>5.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Final rank</td>
<td>7</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

in Figure 6.8c, MOPSO was the best for IGD metric, whereas MO-HGLPSO, MO-μ-MGLPSO, and MOCLPSO had similar medians with smaller box plots for MO-HGLPSO and MOCLPSO. The MO-MGLPSO was the worst in both metrics for the epigenomic workflow. The proposed algorithms had abortive results, as illustrated in Figure 6.7e for SIPHT workflow. The proposed algorithms yielded wider box plots with less median than of CLPSO, MOPSO, and NSGA-2 algorithms. The same results were noticed for the IGD metric in Figure 6.8e.

The nonparametric Friedman test was applied to indicate whether there is a significant difference in the results. The test was conducted at 0.05 significance level. Tables 6.7 and 6.8 display the results for HV and IGD metrics, respectively. Among the algorithms, MO-HGLPSO was the best, followed by MO-μ-MGLPSO for the HV metric, as presented in Table 6.7. The MO-HGLPSO algorithm was among the top three algorithms for all workflows. Furthermore, in comparison with other algorithms,
Table 6.8: Friedman rank test of the MOEAs for IGD metric.

<table>
<thead>
<tr>
<th>Workflow</th>
<th>Friedman ranks</th>
<th>MO-MGLPSO</th>
<th>MO-HGLPSO</th>
<th>MO-µ-MGLPSO</th>
<th>MOGLPSO</th>
<th>MOCLPSO</th>
<th>MOPSO</th>
<th>NSGA-2</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inspiral</td>
<td>Friedman ranks</td>
<td>5.40</td>
<td>2.80</td>
<td>2.77</td>
<td>5.73</td>
<td>3.00</td>
<td>3.17</td>
<td>5.13</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td>Rank</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td>7</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Montage</td>
<td>Friedman ranks</td>
<td>2.40</td>
<td>2.63</td>
<td>4.17</td>
<td>2.93</td>
<td>6.57</td>
<td>3.23</td>
<td>6.07</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td></td>
<td>Rank</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>7</td>
<td>4</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Epigenomics</td>
<td>Friedman ranks</td>
<td>6.37</td>
<td>2.87</td>
<td>3.43</td>
<td>6.17</td>
<td>2.90</td>
<td>1.90</td>
<td>4.37</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td></td>
<td>Rank</td>
<td>7</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>CyberShake</td>
<td>Friedman ranks</td>
<td>2.97</td>
<td>1.30</td>
<td>1.80</td>
<td>4.10</td>
<td>6.87</td>
<td>5.03</td>
<td>5.93</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td></td>
<td>Rank</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>SIPHT</td>
<td>Friedman ranks</td>
<td>5.77</td>
<td>4.80</td>
<td>4.77</td>
<td>5.63</td>
<td>1.93</td>
<td>1.67</td>
<td>3.43</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td></td>
<td>Rank</td>
<td>7</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Average rank</td>
<td>Friedman ranks</td>
<td>4.8</td>
<td>2.4</td>
<td>3.2</td>
<td>5.2</td>
<td>4.4</td>
<td>3.0</td>
<td>5.0</td>
<td></td>
</tr>
<tr>
<td>Final rank</td>
<td>Friedman ranks</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>7</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

the performance of MO-HGLPSO was more stable. Additionally, Table 6.8 presents the test for the IGD metric, in which MO-HGLPSO was ranked first. However, MOPSO was ranked second, and MO-µ-MGLPSO third. MO-MGLPSO was only algorithm that performed best in Montage workflow. Thus, the statistical tests and results reveal the efficiency of MO-HGLPSO and MO-µ-MGLPSO overall for most of the real-world workflows.

6.5.2 Convergence Rate

An experiment to measure the convergence speed of each algorithm was conducted. We observed the mean of HV and IGD metrics for each 50 iterations. Figure 6.9 presents the mean of HV obtained by the MOEAs for all real-world workflows. MO-HGLPSO and MO-µ-MGLPSO showed an improvement of HV value in inspiral, Montage, and Cybershake workflows. In Figure 6.9a, MO-HGLPSO and MO-µ-MGLPSO
Chapter 6: Multi-objective Task Matching for Workflow Applications

(a) Inspiral workflow.  (b) Montage workflow.

(c) Epigenomics workflow.  (d) CyberShake workflow.  (e) SIPHT workflow.

Figure 6.9: Convergence rate of the HV for the real-world workflows.

(a) Inspiral workflow.  (b) Montage workflow.

(c) Epigenomics workflow.  (d) CyberShake workflow.  (e) SIPHT workflow.

Figure 6.10: Convergence rate of the IGD for the real-world workflows.
converged rapidly in 50 iterations and maintained the value of HV to be slightly more than 0.8. Other algorithms could not reach to more than 0.75, which indicates the efficiency of MO-HGLPSO and MO-$\mu$-MGLPSO algorithms. A noticeable decrease was seen in the HV value for MO-MGLPSO and MOGLPSO after 50 iterations. This may imply that the algorithms were stagnated at local optimum and could not find more diffuse solutions on the Pareto front. A similar observation was noticed in Figure 6.10a for the IGD metric. MO-HGLPSO and MO-$\mu$-MGLPSO converged remarkably better than other approaches. Further, we noticed that there was a slight increase in the IGD metric for MO-$\mu$-MGLPSO after 200 iterations. This can be interpreted as when MO-$\mu$-MGLPSO converges, the non-dominated solutions of MO-$\mu$-MGLPSO become less spread out. For MO-MGLPSO and MOGLPSO algorithms, similar behavior was exposed for the IGD metric. Figure 6.9b shows that the proposed algorithms achieved the best HV value for 50 iterations in Montage workflow. MO-HGLPSO and MO-$\mu$-MGLPSO reached to more than 0.7 for the HV metric. Also, MOGLPSO surpassed MO-MGLPSO after 200 iterations. For the IGD metric, as presented in Figure 6.10b, MO-$\mu$-MGLPSO was excelled by MO-MGLPSO, MO-HGLPSO, MOGLPSO, and MOPSO. Again, the obtained solutions of MO-$\mu$-MGLPSO loss diversity during the optimization process. MO-HGLPSO outperformed all algorithms, followed by MO-$\mu$-MGLPSO in Cybershake workflow, as depicted in Figures 6.9d and 6.10d for HV and IGD, respectively. Similar performance was observed in epigenomics and SIPHT workflows for the proposed algorithms. In Figure 6.9c and 6.9e, MO-HGLPSO and MO-$\mu$-MGLPSO were the best, reaching better value of HV than that of other algorithms. Then, after around 100 iterations, the performance was debasing gradually.
Figure 6.11: Solutions quality of the real-world workflows.

Likewise, for IGD metric, MO-HGLPSO and MO-μ-MGLPSO obtained low IGD value, and the performance became worsening after around 100 iterations. Seemingly, the proposed algorithms—and the genetic learning algorithms in general—were trapped into local optimum region and could not explore the Pareto front efficiently. On the other hand, MOPSO, NSGA-2, and MOCLPSO were more effective in such workflows. The results, therefore, show that the proposed algorithms are ineffective for CPU-intensive workflow applications.

6.5.3 Quality of Non-dominated Solutions

To observe the performance of each approach intuitively, we plotted in Figure 6.11 the non-dominated solutions that have the best of either HV or IGD. Furthermore, the
approximated Pareto font of each workflow was illustrated. Figure 6.11a shows the obtained solutions in the inspilral workflow. Clearly, the solutions obtained by MO-HGLPSO was more diverse than that of MO-MGLPSO and MO-µ-MGLPSO. The solutions of MO-MGLPSO were concentrating on the upper part of the Pareto front. The solutions of MO-µ-MGLPSO also converged to a small portion of the Pareto front. MCLPSO seemed to find most of the Pareto front, and yet, there was deficiency in the convergence. For the Montage workflow, as shown in Figure 6.11b, the solutions produced by MO-HGLPSO converged very close to the Pareto front, dominating most of the generated solutions from other algorithms. MO-µ-MGLPSO created more narrow solutions than that of the MO-HGLPSO algorithm. Other algorithms obtained solutions with less quality compared to MO-MGLPSO and MO-µ-MGLPSO algorithms. The performance of the proposed algorithms was notably significant to their competitors for the Cybershake workflow illustrated in Figure 6.11d. For epigenomics workflow, solutions of MOPSO were more converged and widespread over the Pareto front. Besides, MO-HGLPSO was able to obtained results close to that of MOPSO, as presented in Figure 6.11c. Finally, Figure 6.11e shows the obtained results for SIPHT workflow, in which MOPSO was able to discover better quality trade-off solutions than the solutions of the proposed algorithms. Furthermore, NSGA-2 worked more effectively in SIPHT workflow than other workflows.

6.5.4 Discussion

From the above-discussed experimental results, few observations can be drawn. First, the results of this chapter corroborate the results of the previous chapter, prov-
ing that the HGLPSO algorithm works well in general for the task matching problem (single or multi-objective) when the Grid environment is consistent. Furthermore, MO-$\mu$-MGLPSO manifested to be comparable with well-designed multi-objective algorithms such as MOPSO and NSGA-2 for some workflow applications. On the other hand, MO-MGLPSO was not effective in most of the workflow applications except for Montage workflow.

Second, incorporating the load balancing information seemed to help MO-HGLPSO and MO-$\mu$-MGLPSO more than MO-MGLPSO. The MO-MGLPSO algorithm discovered the upper part of the Pareto frontier which may be because it considered minimizing the makespan over the cost. Moreover, the exploration-exploitation trade-off in the MGLPSO algorithm generally was not effective in the consistent matching problem. However, the load balancing information helped, for example in MO-$\mu$-MGLPSO to converge and discover more solutions around the “knee” of the Pareto frontier in some workflows. The knee area represents the balance between the objective. Using some information about the non-dominated solutions may help the solution process. by incorporating such information in MO-HGLPSO during crossover operation may help to discover several areas of the Pareto frontier, as shown in some workflows.

Finally, in epigenomics and SIPHT workflows, there was a deficiency in the proposed algorithms. This might be a result of balancing between the resources. That is, some big tasks were assigned to slow resources due to assigning many tasks to fast resources. This, in turn, increased the load of slow resources and accordingly, increased the makespan and cost. This may explain the problem of the proposed algorithms with respect to CPU-intensive workflows.
6.6 Summary

Though many multi-objective algorithms can be used for workflow matching problem, there are no implementation details in using the genetic learning approach. Therefore, three novel genetic learning PSO algorithms for workflow matching problem were presented in this chapter. The three proposed algorithms were designed to cope with the heterogeneity of the Grid environment. Furthermore, each algorithm implied a different learning approach, affecting the exploration-exploitation trade-offs. The proposed algorithms constructed the exemplars through partitioning the graph into different levels and match each level separately. Moreover, the load information of each resource was embedded in the learning process to preclude load imbalance issues.

To verify the effectiveness of the proposed algorithms, five extremely large-scale real-world workflows were considered. Additionally, popular algorithms designed for multi-objective problems were used for comparison. The statistical results demonstrated that two of the proposed algorithms, which are MO-HGLPSO and MO-\(\mu\)-MGLPSO, outperformed in general most of the competitor algorithms in terms of HV and IGD metrics in many workflows. While solutions of MO-HGLPSO tended to be more widespread, the convergence of the solutions of MO-\(\mu\)-MGLPSO was slightly more in some workflow and condensed. MO-MGLPSO was mostly the worst algorithm in general for the workflow matching problem, except for one workflow, Montage. Nevertheless, there are workflows for which the proposed algorithms were ineffective; these workflows are epigenomics and SIPHT. The epigenomics and SIPHT are workflows used in genomics and are CPU-intensive workflows. In such workflows,
the proposed algorithms seemed to match many CPU-intensive tasks to slow resources to balance the load, hindering the convergence to better results effectively.
Chapter 7

Conclusions and Future Work

This thesis proposed three novel genetic learning PSO algorithms for the task matching problem in the Gird environment. The reason for proposing the genetic learning approach was to circumvent some of the existing problems in PSO, such as premature convergence, oscillation phenomenon, and two steps forward, one step back phenomenon. The proposed techniques were studied for independent tasks with no dependencies for both single and multi-objective optimization problems and dependent tasks represented as a DAG for multi-objective workflow optimization problems.

The first proposed technique, the MGLPSO algorithm, was designed to overcome the PSO shortcomings. Rather than using distance and velocity only to guide the particles, this technique had two main phases, genetic learning phase and PSO phase. In the genetic learning phase, the exemplar was constructed progressively. Each dimension of a given exemplar learns through a crossover operation of randomly picked particle or its current particle. The mutation and selection operations ensure that the exemplar is improved iteratively. Finally, the generated exemplar from the genetic
learning phase is used by the PSO phase. The MGLPSO algorithm was experimented on three different Grid environments, aiming to minimize the makespan (the maximum execution time). The statistical results showed that the MGLPSO algorithm outperformed the original genetic learning PSO (GLPSO) and PSO algorithms in most problem cases. The genetic learning technique embedded in the PSO algorithm showed promising results furthering our understanding of the genetic learning PSO. However, the MGLPSO algorithm suffered on a specific data set, the consistent data set. MGLPSO was not able to balance the load of the resources for the consistent data set.

Hybrid genetic learning PSO (HGLPSO) and $\mu$-MGLPSO were designed to overcome the weakness of MGLPSO. In HGLPSO algorithm, the swarm is divided into two sub-populations. Each sub-population utilizes different genetic learning approaches. The first sub-population uses the MGLPSO learning process, while the second sub-population follows the genetic learning approach of the originally proposed technique, GLPSO.

The $\mu$-MGLPSO algorithm employed an adaptive method for the mutation operation. Based on the average fitness of the swarm, particles away from the average point were assigned a higher mutation rate, while closer particles were assigned a lower mutation rate. This was designed so that the particles in the swarm have different exploration and exploitation tendencies. The algorithms performed well for consistent data set. A higher mutation rate shifted tasks from the fastest resource to other resources, thereby load balancing the resource utilization effectively.

The three proposed algorithms were extended for the multi-objective optimiza-
Chapter 7: Conclusions and Future Work

The objectives we considered were minimization of two parameters, makespan and flowtime. The resultant non-dominated solutions from the three genetic learning PSO algorithm were the best in comparison with other algorithms in the literature in terms of convergence, solution distribution, and the number of non-dominated solutions in all data sets. The promising results allowed us to consider a more complicated problem using the workflow model.

In comparison to task matching problem with independent tasks, the multi-objective optimization workflow model was more complicated. The workflow model was considered level by level. We had to make some modifications to the crossover operation for the three algorithms considered to accommodate task dependent workflow model. We simulated and experimented on real-world benchmarks. MO-HGLPSO and MO-µ-MGLPSO performed well for some benchmarks, none of the algorithms performed well for CPU-intensive applications. The proposed algorithms, only two, which are MO-HGLPSO and MO-µ-MGLPSO, performed better compared to other multi-objective algorithms in most of the real-world workflow applications.

7.1 Future Work

The research in this thesis can lead to promising unexplored avenues for future research in the genetic learning approaches and task matching problem. These are briefly discussed below.

- Fine-tuning of PSO parameters

  In Chapter 4, we investigated the impact of different parameters corresponding to the genetic learning phase. Other quintessential parameters may affect the
overall performance of the algorithm significantly. Such parameters are the inertia weight and acceleration coefficient [78]. In this thesis, the parameters of the genetic learning PSO algorithm were set according to Gong work [109], which follows the standard settings of the global PSO algorithm. It is worth investigating to appropriately tune these parameters for the task matching problem though this can be a laborious and tedious task. Another possible approach to examine is the effect of the linear decrease in these parameters on the genetic learning PSO algorithms.

- **Genetic learning PSO with different topology structures**

  The proposed genetic learning PSO algorithms in this thesis made use of the standard topology, the fully-connected topology. In our work [14], we examined fully-connected, ring, and pyramid topologies in the three Grid environments. The results showed that each topology tends to work better than the other two in a specific Grid environment. As future research, we will investigate the impact of each topology on the genetic learning PSO algorithms. For instance, we can apply the ring communication structure when generating exemplars. So, each particle only learns from its two immediate neighbors. However, this may degrade the convergence speed of the proposed algorithms in favor of increasing diversity.

- **Segment-based learning approach**

  Optimizing large-scale problems is difficult, as increasing the number of problem dimension implies as exponentially increasing in the search space [189; 52]. In this thesis, we evaluated the proposed algorithms on large-scale optimiza-
tion problems and showed that the proposed algorithms scaled very well as the number of dimensions increases. However, there was a definite increase in the number of function evaluations due to constructing each dimension of a given exemplar consecutively. One approach that can be used to reduce the number of function evaluations is the Segment-based learning approach [257]. In this approach, the dimensions are divided into segments randomly, and each segment learns from the best individuals without hindering the convergence speed. Therefore, the effect of using the Segment-based learning approach to generate the exemplars is an important research topic for future direction.

- **Adaptive sub-population**

In the HGLPSO algorithm, the two sub-populations have different abilities of exploration and exploitation search. It is preferable for an optimization algorithm to be more explorative at the beginning of the optimization problem to discover any potential region. Then gradually, the search becomes exploitative on promising vicinity. Thus, possible future research is to investigate the size impact of each sub-population on the HGLPSO algorithm. For example, the size of the exploration sub-population can be reasonably large compared to the other sub-population when starting the optimization process. Then, the other sub-population (exploitative sub-population) becomes more substantial, emphasizing the exploitative search. Another approach can be changing the size of each sub-population adaptively to maintain the exploration and exploitation search, instead of fixed sub-populations as of the HGLPSO algorithm.

- **Parallel genetic learning PSO**
Generally, population-based algorithms, including the PSO algorithm, are intrinsic parallel [13]. Parallel computing architectures like multi-core CPUs and graphics processing units (GPUs) are popular approaches. The GPU approach has been increasingly employed in many computationally-intensive areas – machine learning [81; 192], for example. Moreover, the work [240] adapted the GPU technology to parallelize the PSO algorithm. The proposed algorithms in this thesis are clearly “embarrassingly parallel”. That is to say, the proposed algorithms can be separated into multiple parallel tasks. Hence, parallelizing the proposed algorithms on the GPU architecture for single and multi-objective matching problems is another possible future work.

- **Matching problem with constrains**
  Optimization problems under some constraints could be more intricate, as for many real-world applications. In some cases, there might be QoS requirements (constraints) from users that need to be satisfied when matching the submitted tasks. That is, a user might send a set of tasks with QoS constraints such as a predefined deadline, budget, or both constraints. The matching algorithm, therefore, needs to assign the submitted tasks without violating these constraints. Applying the proposed algorithms to such a problem is intended as a future topic for performance evaluation.

- **Genetic learning PSO in Blockchain**
  The blockchain is a subject undergoing intense research recently. The blockchain is a distributed ledger composing of different transactions recorded in consecutive blocks. Blockchain provides a peer-to-peer network between parties in
which each party holds a complete record of the ledger. The ledger is updated with blocks and acts as a trusted source among the parties. This concept was introduced by Satoshi Nakamoto [184]. *Smart contracts* is used to provide a conflict-free method when parties are dealing with each other. Smart contracts can be utilized to model different decentralized real-world optimization problems [239]. Hyperledger Fabric platform [1] is an open source platform that can facilitate the blockchain technology in industries. Thus, the proposed algorithms can be implemented on the blockchain network by transforming the task matching problem into smart contracts.
Bibliography


[57] Sung Chune Choi and Hee Yong Youn. Task mapping algorithm for heterogeneous computing system allowing high throughput and load balancing. In


[63] Maurice Clerc. The swarm and the queen: towards a deterministic and adap-


[89] Thomas Fahringer, Radu Prodan, Rubing Duan, Francesco Nerieri, Stefan Podlipnig, Jun Qin, Mumtaz Siddiqui, Hong-Linh Truong, Alex Villazon, and Marek Wieczorek. Askalon: A grid application development and computing


[110] Andreia P Guerreiro and Carlos M Fonseca. Computing and updating hypervol-


[129] Hesam Izakian, Ajith Abraham, and Vaclav Snasel. Comparison of heuristics for scheduling independent tasks on heterogeneous distributed environments. In *Proceedings of International Joint Conference on Computational Sciences and
Optimization (CSO 2009), Hainan, Sanya, China, volume 1, pages 8–12, 24–26 April 2009.


[134] Yaochu Jin, Markus Olhofer, and Bernhard Sendhoff. Dynamic weighted aggregation for evolutionary multi-objective optimization: Why does it work and how? In Proceedings of the 3rd Annual Conference on Genetic and Evolu-


[149] Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran. A taxonomy
and survey of grid resource management systems for distributed computing.  


[156] Miqing Li, Shengxiang Yang, Jinhua Zheng, and Xiaohui Liu. ETEA: A


[162] Qiuzhen Lin, Songbai Liu, Qingling Zhu, Chaoyu Tang, Ruizhen Song, Jianyong Chen, Carlos A Coello Coello, Ka-Chun Wong, and Jun Zhang. Particle swarm


[168] Xiao-Fang Liu, Zhi-Hui Zhan, Ying Gao, Jie Zhang, Sam Kwong, and Jun Zhang. Coevolutionary particle swarm optimization with bottleneck objective


[180] Kaisa Miettinen, Francisco Ruiz, and Andrzej P Wierzbicki. Introduction to


[187] Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark
Greenwood, Tim Carver, Kevin Glover, Matthew R Pocock, Anil Wipat, et al.
Taverna: a tool for the composition and enactment of bioinformatics workflows.


[218] Behrooz Shirazi, Hsing-Bung Chen, and Jeff Marquis. Comparative study of task duplication static scheduling versus clustering and non-clustering tech-


[224] N. Srinivas and Kalyanmoy Deb. Multiobjective optimization using nondomi-


[231] M Fatih Tasgetiren, Yun-Chia Liang, Mehmet Sevkli, and Gunes Gencyilmaz. A particle swarm optimization algorithm for makespan and total flowtime min-


on Computational Intelligence (SSCI 2015), Cape Town, South Africa, pages 218–225, December 7–10 2015.


[263] Jia Yu and Rajkumar Buyya. Scheduling scientific workflow applications with


[276] Yu-Hui Zhang, Yue-Jiao Gong, Tian-Long Gu, and Jun Zhang. Ensemble mat-


