Cooperative Multi-agent Systems for Single and Multi-objective Optimization

Nasser Lotfi

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of philosophy
in
Computer Engineering

Eastern Mediterranean University
October 2015
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Serhan Çiftçioğlu
Acting Director

I certify that this thesis satisfies the requirements as a thesis for the degree of Doctor of Philosophy in Computer Engineering.

Prof. Dr. Işık Aybay
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Doctor of Philosophy in Computer Engineering.

Asst. Prof. Dr. Adnan Acan
Supervisor

Examining Committee

1. Prof. Dr. Tolga Çiloğlu
2. Prof. Dr. İbrahim Özkan
3. Asst. Prof. Dr. Adnan Acan
4. Asst. Prof. Dr. Mehmet Bodur
5. Asst. Prof. Dr. Ahmet Ünveren

ABSTRACT

Solving combinatorial and real-parameter optimization problems is an important challenge in all engineering applications. Researchers have been extensively solving these problems using evolutionary computations. In this thesis, three new multi-agent architectures are designed and utilized in order to solve combinatorial and real-parameter optimization problems.

First architecture introduces a novel learning-based multi-agent system (LBMAS) for solving combinatorial optimization problems in which all agents cooperate by acting on a common population and a two-stage archive containing promising fitness-based and positional-based solutions found so far. Metaheuristics as agents perform their own method individually and afterwards share their outcomes with others. In this system, solutions are modified by all running metaheuristics and the system learns gradually how promising metaheuristics are, in order to apply them based on their effectiveness.

In the second architecture, a novel multi-agent and agent interaction mechanism for the solution of single objective type real-parameter optimization problems is proposed. The proposed multi-agent system includes several metaheuristics as problem solving agents that act on a common population containing the frontiers of search process and a common archive keeping the promising solutions extracted so far. Each session of the proposed architecture includes two phases: a tournament among all agents to determine the currently best performing agent and a search

procedure conducted by the winner. The proposed multi-agent system is experimentally evaluated using the well-known CEC2005 benchmark problems set.

The third architecture presents a creative multi-agent and dynamic multi-deme architecture based on a novel collaboration mechanism for the solution of multi-objective real-parameter optimization problems. The proposed architecture comprises a number of multi-objective metaheuristic agents that act on subsets of a population based in a cyclic assignment order. This multi-agent architecture works iteratively in sessions including two consecutive phases: in the first phase, a population of solutions is divided into subpopulations based on the dominance ranks of its elements. In the second phase, each multi-objective metaheuristic is assigned to work on a subpopulation based on a cyclic or round-robin order. The proposed multi-agent system is experimentally evaluated using the well-known CEC2009 multi-objective optimization benchmark problems set.

Analysis of the experimental results showed that the proposed architectures achieve better performance compared to majority of their state-of-the-art competitors in almost all problem instances.

Keywords: Multi-agent systems, Metaheuristics, Combinatorial Optimization, Multiprocessor Scheduling, Agent Interactions, Multi-objective Optimization, Pareto Optimality

ÖZ

Bileşimsel ve gerçek parametreli en iyileme problemlerini çözmek tüm mühendislik uygulamalarında önemli bir sorundur. Araştırmacılar uzun süredir evrimsel algoritmaları kullanarak bu problemlerin çözümü üzerinde uğraşmaktadırlar. Bu tezde, bileşimsel ve gerçek parametreli en iyileme problemlerini çözmek için üç yeni çok ajanlı sistem mimarisi önerilip ve tasarlanmıştır.

İlk sistem mimarisi bileşimsel en iyileme problemlerini çözmek amacıyla öğrenebilen çok ajanlı sistemi (LBMAS) tanıtır. Bu sistemde tüm ajanlar ortak nüfus ve çift aşamalı arşiv üzerinden işbirliği yaparlar. Sistemdeki çift aşamalı arşiv içerisinde uygunluk ve konumsal bakımından iyi olan çözümler bulunmaktadır. Önerilen sistemde metaheuristic'ler ajan olarak kendi yöntemlerini yürütüp, daha sonrasında bulunan sonuçları başkalarıyla paylaşıyorlar. Bu sistemde, bulunan çözümler çalışan tüm Metaheuristic'ler tarafından değiştirilir ve sistem metaheuristic'lerin ne kadar etkili olduklarını sınayarak öğreniyor.

İkinci mimaride, tek amaçlı gerçek parametreli en iyileme problemlerini çözmek için çok ajanlı yeni bir sistem ve ajan etkileşim mekanizması öneriliyor. Önerilen çok ajanlı sistemde çeşitli Metaheuristic'ler ortak nufüs ve ortak arşiv üzeride çalışıyorlar. Ortak arşiv, şu ana kadar bulunan umut verici çözümleri içeriyor. Önerilen mimarideki her adım iki aşamayı içerir: birinci aşamada tüm ajanlar arasında en iyi performans gösteren ajanı bulmak için turnuva yapılıyor ve ikinci aşamada ise arama prosedürü, kazanan ajan tarafından devam ettiriliyor. Önerilen

çok ajanlı sistem tanınmış CEC2005 problem kümesindeki problemleri çözümü üzerinden değerlendirilmiştir.

Üçüncü çok ajanlı mimaride çok amaçlı gerçek parametreli en iyileme problemlerini çözmek için yeni bir işbirliği mekanizması sunulmuştur. Önerilen mimaride metaheuristic ajanlar döngüsel bir atama sırasına göre alt nüfuslar üzerinde çalışırlar. Bu çok ajanlı mimari ardışık iki faz üzerinden döngülenerek çalışır: ilk aşamada, çözüm nüfus unsurları baskınlık değerine göre alt nufüslara ayrılırlar, ikinci aşamada ise her çok amaçlı metaheuristic yuvarlak döngü usülüne göre bir alt nufüs üzerinde çalışmak için görevlendirilir. Önerilen çok ajanlı sistem tanınmış CEC2009 deneysel problemler kümesindeki çok amaçlı en iyileme problemleri kullanarak değerlendirilmiştir.

Deney sonuçlarının analizi, önerilen mimarilerin hemen tüm deneysel problemler üzerinde rakiplerinden daha iyi başarıma sahip olduklarını göstermiştir.

Anahtar Kelimeler: Çok ajanlı sistemler, Metaheuristic, Bileşimsel en iyileme, çok işlemcili planlama, Ajan etkileşimleri, Çok amaçlı en iyileme, Pareto en iyilik

DEDICATION

I would like to dedicate my thesis to my beloved parents, brothers and sisters who were very supportive and always encouraged me.

ACKNOWLEDGMENT

I would like to express my deepest gratitude to my supervisor Asst. Prof. Dr. Adnan Acan for his excellent guidance, caring, patience and providing me with an excellent atmosphere for doing this research.

I wish to thank my committee members Asst. Prof. Dr. Mehmet Bodur and Asst. Prof. Dr. Ahmet Ünveren who were more than generous with their expertise and precious time and always willing to help and give their best suggestions.

Finally, I would like to thank my parents and my brothers and sisters. They were always supporting and encouraging me with their best wishes.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF SYMBOLS / ABBREVIATIONS

MAS	Multi-Agent System
ω	Universe Set
MOP	Multi-objective Optimization Problem
AMF	Agent Metaheuristic Framework
RIO	Role Interaction Organization
CBM	Coalition–Based Metaheuristic
MAGMA	Multi-Agent Metaheuristic Architecture
JMS	Java Messaging Service
DA	Directory Agent
MAGA	Multi-Agent Genetic Algorithm
MacroAEM	Macro Agent Evolutionary Model
HMAGA	Hierarchical Multi-Agent Genetic Algorithm
COP	Constrained Optimization Problems
AMA	Agent-based Memetic Algorithm
AES	Agent-based Evolutionary Search
APAA	Agent-based Parallel Ant Algorithm
EMAS	Evolutionary Multi-Agent System
selEMAS	semi-elitist Evolutionary Multi-Agent System
LBMAS	Learning-Based Multi-Agent System
GA	Genetic Algorithm
SA	Simulated Annealing
DE	Differential Evolution
ACO	Ant Colony Optimization

GDA	Great Deluge Algorithm
TS	Tabu Search
CE	Cross Entropy
ES	Evolutionary Strategy
PSO	Particle Swarm Optimization
P_c	Crossover Probability
P_m	Mutation Probability
f_{obj}	Objective Function
P_{best}	Personal Best
G_{best}	Global Best
μ	Population Size
λ	Offspring Size
ABC	Artificial Bee Colony
PMA	Population Management Agent
MOO	Multi-Objective Optimization
NSGAII	Non-dominated Sorting Genetic Algorithm
MOGA	Multi-Objective Genetic Algorithm
SPEA2	Strength Pareto Evolutionary Algorithm
MODE	Multi-Objective Differential Evolution
AMOSA	Multi-Objective Simulated Annealing
MOPSO	Multi-Objective Particle Swarm Optimization
SPA	Solution Pool Agent
DAG	Directed Acyclic Graph
MSP	Multiprocessor Scheduling Problem
FFT	Fast Fourier Transformation

IRR	Internal Rate of Return
FAR	Friedman Aligned Ranks

Chapter 1

INTRODUCTION

1.1 Introduction

Solving combinatorial and real-parameter optimization problems is an important task in almost all engineering applications. The optimization problems which this thesis deals with are combinatorial- and real-parameter optimization problems. Researchers have been extensively solving these kinds of problems using evolutionary computations and metaheuristics. In this thesis, three new multi-agent architectures are designed and applied in order to solve combinatorial and real-parameter optimization problems. A multi-agent system (MAS) includes a set of agents and their environment in which the agents are designed to perform particular tasks. The rest of this chapter is organized as follows: Fundamental issues of multi-agent systems are presented in Section 1.2. Section 1.3 illustrates description of metaheuristics briefly. Single-objective optimization, combinatorial optimization and multi-objective optimization problems are explained in sections 1.4, 1.5 and 1.6 respectively.

1.2 Multi-agent Systems

Fundamentally, a multi-agent system (MAS) comprises a set of agents and their environment in which the agents are designed to perform particular tasks. In this respect, individual agents are computational procedures that perceive their environment, make inferences based on the received percepts and their learned experience and acts on their environment to reach predefined design goals [1]. A generic description of a MAS is shown in Figure 1.1.

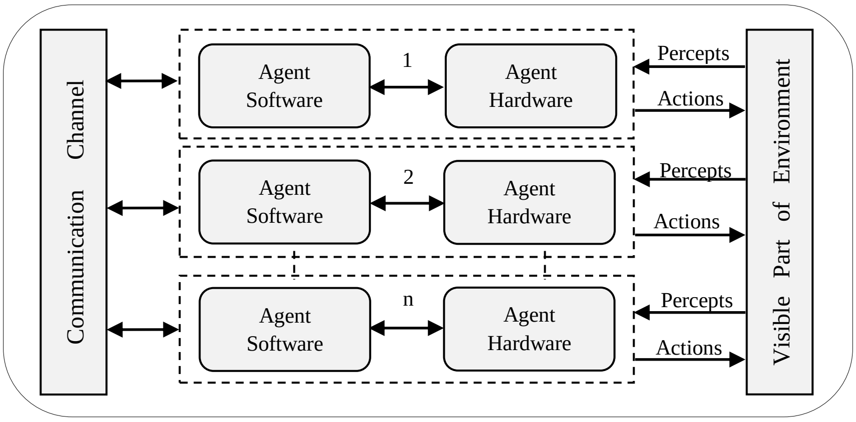


Figure 1.1. Generic description of a multi-agent system

In intelligent MASs, individual agents are required to be autonomous that means learning capability through interactions with the environment as well as adapting to changes in the environment caused by agents' actions internally and the environments' dynamic externally. Individual agents are also attributed to have other important properties that are outside the scope of our descriptions. The full list of intelligent agent's properties can be found in [2].

An agent in a MAS can be considered as an entity with an architecture comprising two fundamental components, namely the agents' hardware and the agents' software. While the agents' hardware is consisting of sensors and actuators to monitor and act on the environment, the software includes procedures for processing the percepts, making inferences on goal-based actions, updating knowledge base and maintaining records on changes in the environment. Based on their architectural characteristics and computational capabilities, agents are classified as reflexive, maintaining state, goal-based and utility-based agents. A detailed description of agents in each of these categories can be found in [3]. Agents within our proposed systems in this thesis can be described as utility-based agents with a particular goal of minimizing the objective functions where the utility of a particular action (operator) is measured in terms of the corresponding fitness value found through evaluation of the objective functions. The detailed block diagrams description of individual utility-based agents employed within the proposed frameworks are given in next chapters.

As indicated in Fig. 1, agents in a MAS are interacting and communicating with each other through a communication channel that can be implemented either as a centralized star model where each agent can communicate through a master agent or as distributed inter-agent dialogs any pair of agents can exchange messages using some protocols [4]. Obviously, the second method is general, multipurpose and flexible, however it requires agent communication languages and dedicated message passing protocols to be implemented on each individual agent. Star model is easier to implement for small-size MASs, including reasonably small number of agents, since one communication protocol needs to be implemented on all agents.

The third fundamental part of a MAS is the environment which is sensed and changed by its agents to reach their goals. As a place to live and manipulate by the agents, the environment is a shared common resource for all agents [2]. It takes the role of specifying positions, locality, and limitations on actions of agents. Agent environments can also be classified based on their spatial properties and accessibility of attributes. A general description of agent environments and their categorical properties can be found in [2].

The MASs proposed in this thesis implement adaptations of the above mentioned architectural elements under the consideration of individual agent models, their problem environment, goals and computational resources. Details of the proposed MASs implemented for combinatorial optimization problems, single-objective real-valued function optimization and multi-objective real-valued function optimization are presented in next chapters.

1.3 Metaheuristics

Solving optimization problems is a challenging issue in almost all engineering applications. Optimization algorithms are applied to solve these kinds of problems and among them the metaheuristics are becoming more popular [6]. Most of Metaheuristics are nature-inspired and they are divided to trajectory- and population based type in which the trajectory-based metaheuristics deal with a single solution and the population-based ones handle the population of solutions.

Metaheuristics implement some forms of stochastic optimization which comprises the set of algorithms that employ random methods to find the global or near-global optimal solutions. Metaheuristics are applied to solve wide range of optimization problems [5].

Some of the well-known trajectory based metaheuristics are Simulated Annealing [23], Great Deluge Algorithm [25], Cross Entropy [27] and Tabu Search [26]. Meanwhile the Genetic Algorithm [20], Ant Colony Optimization [24], Particle Swarm Optimization [32] and Differential Evolution [21, 22] are considered as population-based metaheuristics. Principles and basic descriptions of metaheuristic algorithms used in this thesis and in the proposed multi-agent systems are discussed later in chapter 3.

1.4 Combinatorial Optimization Problems

A combinatorial optimization problem is the particular kind of problems in which a solution of problem comprises a combination of unique components chosen from a finite and determinate set [5]. The objective of these kinds of problems is to find the optimal combination of components. Travelling Salesman Problem, Knapsack Problem and Set Covering Problem are the examples of combinatorial optimization problems. As an example, in travelling salesman problem, there are a number of cities and routes between the pairs of cities in which each route has a cost. The salesman is going to find a lowest cost tour starting from a city, visiting all other cities only once and come back to the same city. Therefore, in TSP problem, the components are cities and the aim is to find optimal combination of these components [5].

Combinatorial optimization problems can be solved by metaheuristics in order to find optimal or near-optimal solutions.

1.5 Single-Objective Optimization Problems

Optimization is a process or method to find something as optimal as possible in terms of objective functions. In single-objective optimization problems, there exist only one objective function to be optimized and the aim is to either minimize or maximize it using appropriate algorithms [7].

A general single-objective optimization problem is minimization or maximization of f(x) subject to $g_i(x) \le 0, i = \{1, 2, 3, \dots, m\}$ and $h_j(x) = 0, j = \{1, 2, 3, \dots, p\},\ x \in$

ω in which $g_i(x)$ and $h_j(x)$ indicate constraints that must be considered as f(x) is being optimized. A solution of problem minimizes or maximizes the f(x) where x is the n-dimensional decision variable vector and ω is the universe for x. The method and approach to find the global optimal is called as global optimization [7].

1.6 Multi-objective Optimization Problems

Multi-objective optimization problem aims to find a vector of decision variables which satisfies all constraints and optimizes all objective functions that are usually in conflict with each other. Optimization process tries to find the acceptable values of all objective functions to satisfy the decision maker.

A general multi-objective optimization problem is the minimization or maximization of $F(x) = (f_1(\mathrm{x}), \ldots, f_k(x))$ subject to $g_i(x) \leq 0, i = \{1, 2, 3, \ldots, m\}$ and $h_j(x) = 0, j = \{1, 2, 3, \ldots, p\}$, $x \in \omega$ in which $g_i(x)$ and $h_j(x)$ indicate constraints that must be considered as f(x) which it's being optimized and ω contains all possible x values [7].

The definition of "optimum" is changed when the problem deals with some objective functions. In multi-objective optimization problems, the goal is to find good "trade-offs" instead of a single solution in global optimization. The most commonly accepted term for "optimum" in MOPs is Pareto Optimum [7].

A solution $x \in \omega$ is Pareto optimal if and only if there is no $x' \in \omega$ in which $v = F(x') = (f_1(\mathrm{x}'), \ldots, f_k(x'))$ dominates $u = F(x) = (f_1(\mathrm{x}), \ldots, f_k(x))$. Pareto dominance is represented as $v \leq u$ in which v dominates u if and only if v is partially less than u:

$$(\forall_i \in \{1, \dots, k\}, v_i \leq u_i \; \wedge \exists i \in \{1, \dots, k\}: v_i < u_i). \quad (1.1)$$

Based on the aforementioned concepts, the Pareto Optimal Set, P^*, is defined as:

$$P^* := \{x \in \omega \mid \neg\exists\, x' \in \omega : F(x') \leq F(x)\} \qquad (1.2)$$

Meanwhile, for a given MOP, F(x), and P^*, the Pareto Front Pf^* is defined as:

$$Pf^* := \{u = F(x) \mid x \in P^*\} \qquad (1.3)$$

Also, the non-dominated solutions are called as Pareto Front as well. Main goal of multi-objective algorithms is to preserve the non-dominated points in objective space and correspondence solutions in decision space and move towards the Pareto Front set with maintaining the diversity at the same time [7].

Chapter 2

STATE-OF-THE-ART IN MULTI-AGENT SYSTEMS

2.1 Introduction

A multi-agent system includes a set of agents and their environment in which the agents are designed to perform particular tasks. In this respect, individual agents are computational procedures that perceive their environment, make inferences based on the received percepts and their learned experience and acts on their environment to reach predefined design goals [1]. A generic description of a MAS is shown in Figure 1.1.

The important features of an agent in a multi-agent system are as following; however, supporting all of them by an agent depends on tasks and environment [73].

- Autonomy: Agents are autonomous to decide about interactions.
- Reactivity: Agents observe the environment and interact against environment changes.
- Pro-activeness: Agents acts on environment are goal-oriented to lead the system into desired form.
- Social ability or communicative: Agents use communication languages to interact with other agents.
- Learning or Adaptive: Agents learn according to past experiences and they change their behavior accordingly.

- Local views: Agents don’t know whole system and they can see their scope only.
- Decentralization: There is no controlling agent in the system.

According to [74], agents are grouped into 5 classes in terms of intelligence and capabilities as following:

- Simple reflex agents: This kind of agent acts only based on current perception. If the environment is not fully observable, this agent is not able to be successful.
- Model-based reflex agents: This kind of agent chooses the action in the same way with reflex agents but it stores some information about un-observable environment to handle partially observable environments.
- Goal-based agents: This agent is kind of model-based agent and stores information about desired environment. This way, it chooses the acts to lead the system toward desired goals.
- Utility-based agents: This agent knows how to measure goodness of states and how to distinguish between goal- and non-goal states.
- Learning agents: This agent initially starts to operate in un-known environment and then learn gradually how to deal with the system.

Meanwhile, the agent architecture is divided into three groups as following [73]:

- Deliberative Architectures: This architecture represents the symbolic model of the world explicitly and decides via logical reasoning.

- Reactive Architectures: This architecture doesn't have any kind of central symbolic world model and also doesn't use any complex symbolic reasoning.
- Hybrid Architectures: Reactive agent is not so efficient, because it makes the decision quickly without a formal search. In contrast, deliberative agent uses much time to choose the best behavior. Therefore, an efficient and quick architecture can be made by combination of these two architectures.

In intelligent MASs, individual agents are required to be autonomous that means learning capability through interactions with the environment as well as adapting to changes in the environment caused by agents' actions internally and the environments' dynamic externally. An agent in a MAS can be considered as an entity with an architecture comprising two fundamental components, namely the agents' hardware and the agents' software. While the agents' hardware is consisting of sensors and actuators to monitor and act on the environment, the software includes procedures for processing the percepts, making inferences on goal-based actions, updating knowledge base and maintaining records on changes in the environment. Based on their architectural characteristics and computational capabilities, agents are classified as reflexive, maintaining state, goal-based and utility-based agents.

Agents in a MAS are interacting and communicating with each other through a communication channel that can be implemented either as a centralized star model where each agent can communicate through a master agent or as distributed inter-agent dialogs any pair of agents can exchange messages using some protocols [4].

The third fundamental part of a MAS is the environment which is sensed and changed by its agents to reach their goals. As a place to live and manipulate by the agents, the environment is a shared common resource for all agents [2]. It takes the role of specifying positions, locality, and limitations on actions of agents. Agent environments can also be classified based on their spatial properties and accessibility of attributes.

Multi-agent systems and evolutionary algorithms can be integrated for solving difficult problems; hence, such a system is called agent-based evolutionary algorithms. There are three types of frameworks as follows [73]:

1. Agents are responsible for their actions and the system behavior
2. Agents represents the solutions
3. Sequentially use of multi-agent system and evolutionary algorithm

First type agents guide the system to solve the problem by specifying the actions and system behavior. The agents in this framework can use evolutionary algorithms for learning and improving the system efficiency. In [75], authors proposed a multi-agent system which uses genetic algorithm to determine a set of functions for each agent. Meanwhile, in the [76, 77] authors use evolutionary algorithms as learning algorithms within the multi-agent systems.

In the second type, an agent represents a candidate solution; so, in evolutionary algorithm a population of solutions can be considered as a population of agents. However, an agent can contain other information as well such as learning techniques. In such a system, agents cooperate and compete with neighbors to increase their fitness. The number of neighbors an agent can cooperate with can be four [78], eight [79] or all agents in the entire environment [80];

In the third framework, multi-agent system and evolutionary algorithm are used iteratively or sequentially to solve a problem. As an example, in the [81] for solving dynamic job-shop scheduling problem, authors applied multi-agent system for initial task allocations and then used genetic algorithms for optimizing the scheduling.

The rest of this chapter is organized as follows: The state-of-the-art in multi-agent systems for single-objective optimization is presented in Section 2.2 and Section 2.3 illustrates the related works on multi-agent systems for multi-objective optimization.

2.2 Multi-agent systems for single-objective optimization

Multi-agent systems including metaheuristics as individual agents are widely used to provide cooperative/competitive frameworks for optimization. Many efforts have been done on this field and there exist some outstanding literatures in this context [4, 8]. It has already been shown through several implementations that multi-agent systems with metaheuristic agents provide effective strategies for solving difficult optimization problems. This section covers the state-of-the-art approaches of multi-agent systems for single-objective optimization.

2.2.1 An organizational view of metaheuristics

Meignan et al. proposed an organizational multi-agent framework to hybridize metaheuristics algorithms [8]. Their agent metaheuristic framework (AMF) is fundamentally developed for hybridization of metaheuristic based on an organizational model. In this model, each metaheuristic is given a role among the tasks of intensification, diversification, memory and adaption. This organization model is named as RIO (Role Interaction Organization) and an illustrative description of its architectural model is given in Figure 2.1.

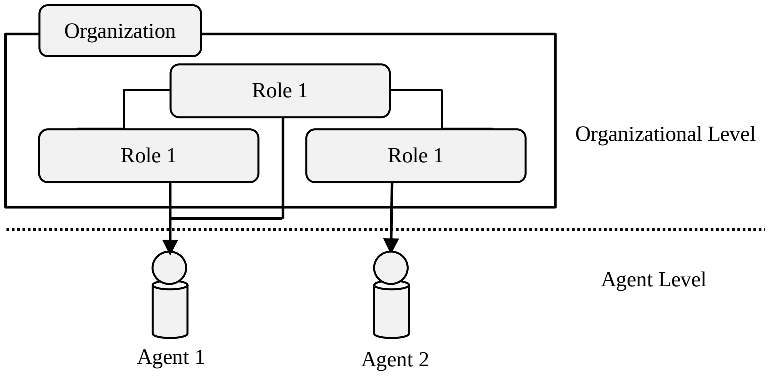


Figure 2.1. The RIO model of a multi-agent system of metaheuristics

The authors exploited the ideas and basic concepts of adaptive memory programming (AMP) which unifies several metaheuristics concepts considering their common characteristics [9]. The proposed multi-agent system based on this organizational framework is used to develop a hybrid algorithm called the coalition–based metaheuristic (CBM). CBM is used for the solution of vehicle routing problem and the obtained exhibited that even though CBM is not as good as its competitors in terms of solution quality, it provides close to optimal solutions in significantly small computation times.

2.2.2 Cooperative metaheuristic system based on Data-mining

Cadenas et al. introduced a multi-agent system of cooperative metaheuristics in which each metaheuristic is implemented as an agent and they try to solve a problem in cooperation with each other. A coordinating agent monitors and modifies the behavior of other agents based on their performance in improving the solution quality [10]. Individual agents communicate using a common blackboard part of which is controlled by each agent and they record their best solution found so far on the blackboard. The blackboard is monitored by the coordinator agent to decide on the performance of agents to derive conclusions on how to modify their behavior.

The coordinator agent uses a fuzzy rule from which inferences are derived based on the performance data of individual agents. A block diagram description of this multi-agent system is presented in Figure 2.2.

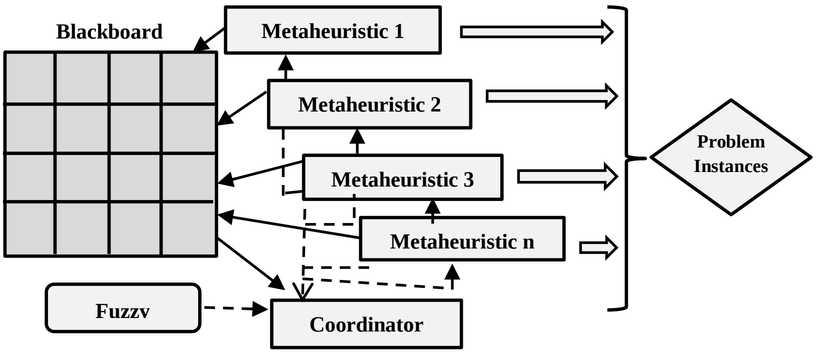


Figure 2.2. The multi-agent system architecture proposed in [10]

The authors applied the above-mentioned multi-agent system for the solution 0/1 knapsack problems and experimental results exhibited that the proposed cooperative system generates slightly better solutions compared to application of non-cooperative nature-inspired metaheuristics. It is also reported by the authors that the computational cost of extraction of fuzzy rules can be too large.

2.2.3 Coordinating metaheuristic agents with swarm intelligence

Another cooperative multi-agent system of metaheuristics is proposed by M.E. Aydin through creating a population of agents with search skills similar to those of simulated annealing (SA) algorithm [11]. SA agents carry out runs on their own individual solutions and their accepted solutions are collected into a pool which is further manipulated by a coordinating metaheuristic for the purpose of exchanging information among SA agents' solutions and preparing them new seeds for the next iteration. Architectural description this method is shown in Figure 2.3.

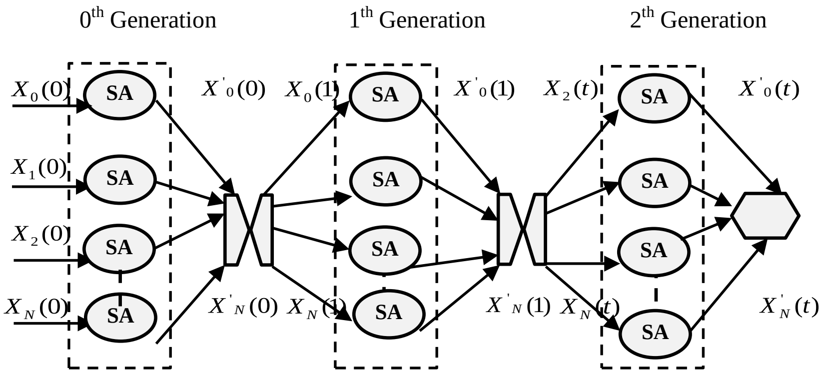


Figure 2.3. Multi-agent system based on coordination of population of SA agents

The coordinating metaheuristics considered in this approach are evolutionary simulated annealing, bee colony optimization, and particle swarm optimization. The authors used this multi-agent system for the solution of multidimensional knapsack problem. It has been observed that multiple SA agents coordinated by PSO resulted in the best solution quality. In addition to this, number of inner SA iterations has a significant effect on the performance of overall multi-agent system.

2.2.4 A multi-agent architecture for metaheuristics

The multi-agent metaheuristic architecture (MAGMA) proposed by Milano et al. is a multi-agent system containing four conceptual levels with one more agents at each level [12]. Agents at level-0 are solution constructors while agents at level-1 apply a particular metaheuristic for the improvement of solutions constructed at level-0. Basically, the search procedures of level-1 agents are iteratively applied until a termination condition is satisfied. Level-2 agents are global observers such that they decide on strategies to direct the agents towards promising regions of solution space and to get rid of locally optimal solutions. The authors have experimentally demonstrated that these three levels are enough to describe simple (non-hybrid) multi-agent systems of metaheuristics capable of solving difficult optimization problems. Block diagram description of MAGMA is given in Figure 2.4.

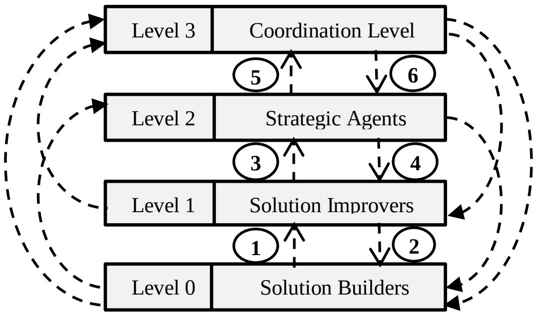


Figure 2.4. Conceptual description of levels in MAGMA [12]

The level-3 shown in Figure 2.4 represents the presence of coordinating agents that are responsible for communication and synchronization. Implementation of this level aims the development of high-level cooperative multi-agent systems in which hybridization of multiple metaheuristics is possible. Multilevel structure and the multi-agent system organization of MAGMA allow all direct communications between all levels, however only some of them are implemented in [12]. The authors used iterated local search (ILS) within MAGMA framework for the solution MAXSAT problems with 1000 variables and 10000 clauses and their results exhibited that the resulting system achieved the best solutions with higher frequency compared to random restart ILS method.

2.2.5 Multi-agent cooperation for solving global optimization problems

Another coordination- and cooperation based multi-agent system named MANGO [13] was proposed for solving global optimization problems. MANGO is a Java-based multi-agent framework implemented by APIs capable of running on different machines and share the results based on message passing mechanism. MANGO provides directory service, yellow pages service and message types, permitting agent developers to choose any coordination mechanism according to requirements. Each agent is a Java program performs specific tasks in parallel. In this framework, cooperation is carried out over the service oriented architecture. The search agents who provide the search mechanisms are service providers and who request services are service consumers. MANGO implements the communication in two levels: low-level is done over Java Messaging Service (JMS) dealing with network protocols and high-level exchanges the messages between agents which are provided by using mailboxes. This way, agents can check their own mailbox whenever they want. MANGO environment as a distributed system is illustrated in Figure 2.5.

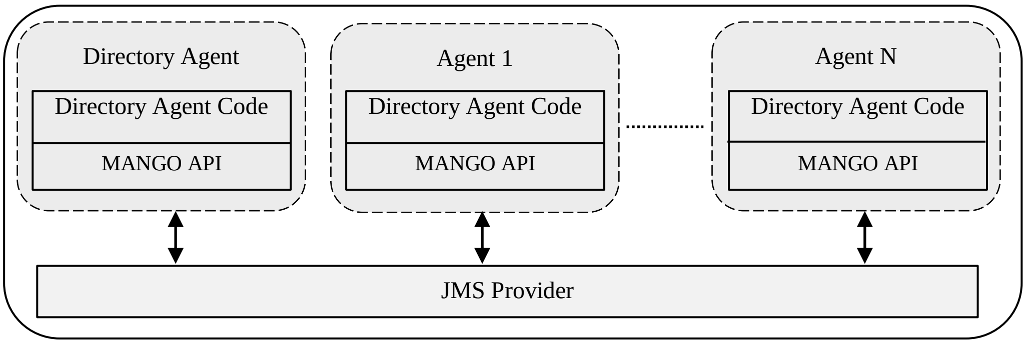


Figure. 2.5: MANGO Environment [13]

MANGO includes a special agent named by directory agent (DA) taking responsibility for managing communication resources and providing two types of services. First type manages JMS communication resources and the second type is the directory service. MANGO can use any of optimization algorithms for the agents and the agent designer decides which algorithm should be applied [13]. The authors of MANGO did not provide a detailed test of the system using hard numerical optimization benchmarks, hence its success for practical cases is not known.

2.2.6 Multi-Agent Evolutionary Model for Global Numerical Optimization

The Multi-Agent Genetic Algorithm (MAGA) proposed by Liu et al. is designed to solve the global numerical optimization problems [82]. An agent in MAGA is used to represent a candidate solution of the problem being solved and energy value of the agent is the negative value of the corresponding objective function. The aim of agent is to increase the energy value as much as possible. The agent lattice in MAGA is illustrated as Figure 2.6. All agents live in the lattice environment and they compete and cooperate with their neighbors in order to minimize the objective function value.

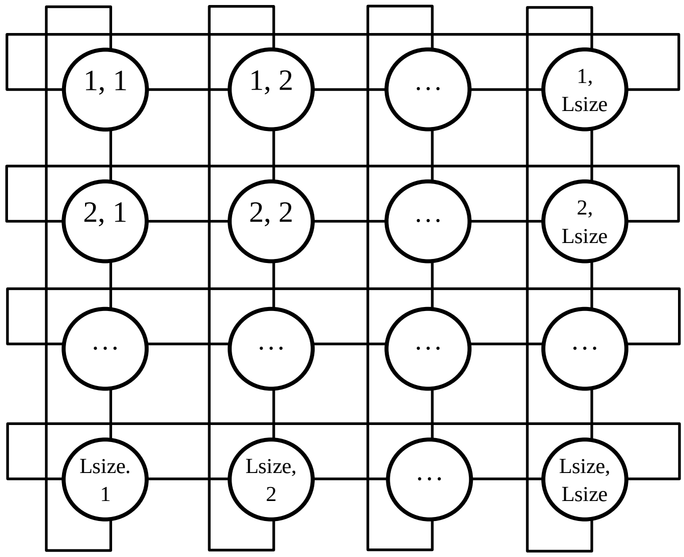


Figure 2.6. Agent lattice model [82]

Moreover, authors proposed the Macro Agent Evolutionary Model (MacroAEM) in which the sub-functions form macro agents with three new behaviors (competition, cooperation and selfishness) to optimize the objective functions. Consequently, the authors integrated the MacroAEM and MAGA in order to form a new algorithm named by Hierarchical Multi-Agent Genetic Algorithm (HMAGA). Theoretical analysis showed that the HMAGA is able to converge to global optima. Meanwhile, experimental evaluation of MAGA and HMAGA indicated good performance when the dimensions are increased from 20 to 10,000; so that, it can find good solutions for large scale optimization problems at a low computational cost [82].

2.2.7 An Agent Based Evolutionary Approach for Nonlinear Optimization with Equality Constraints

Barkat ullah et al. proposed an agent-based evolutionary algorithm for solving constrained optimization problems (COPs) [83]. In the proposed multi-agent system, the agents use a new learning method which has been designed to deal with equality constraints in the early generations. In the later generations, agents use other learning processes to improve their performance. Authors proposed an agent-based Memetic algorithm (AMA) for solving constrained non-linear optimization problems which integrated agent concept with memetic algorithms. An agent in this system represents a candidate solution and tries to improve its fitness using a self-learning method. The agents are considered in a lattice environment to communicate and exchange information with neighbors. Figure 2.7 shows AMA learning process.

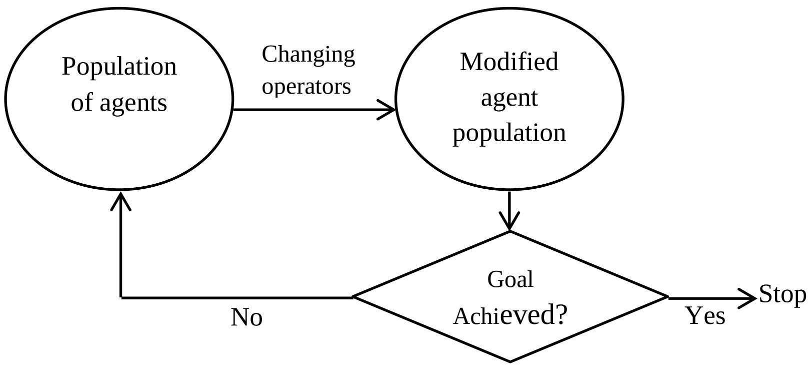


Figure 2.7. AMA model [83]

In this method, the constraints are handled without any penalty functions or additional parameters and the experimental results illustrated that the performance of proposed algorithm is promising [83].

2.2.8 Agent Based Evolutionary Dynamic Optimization

Yan et al. proposed an agent-based evolutionary search (AES) algorithm for solving dynamic 0-1 optimization problems [84]. The proposed approach inspired of living organisms updates the agents to track the dynamic optimum. In the proposed method, all agents in the environment compete with their neighbors and collect knowledge in order to learn and increase the energy function. In this algorithm, for maintaining the diversity, some immigrations and mapping schemes are used. In AES, each agent represents a candidate solution using a 0-1 array and the agent energy value is equal to objective function value [84]. Agents are placed on a lattice environment and interact with their neighbors as shown in Figure 2.8.

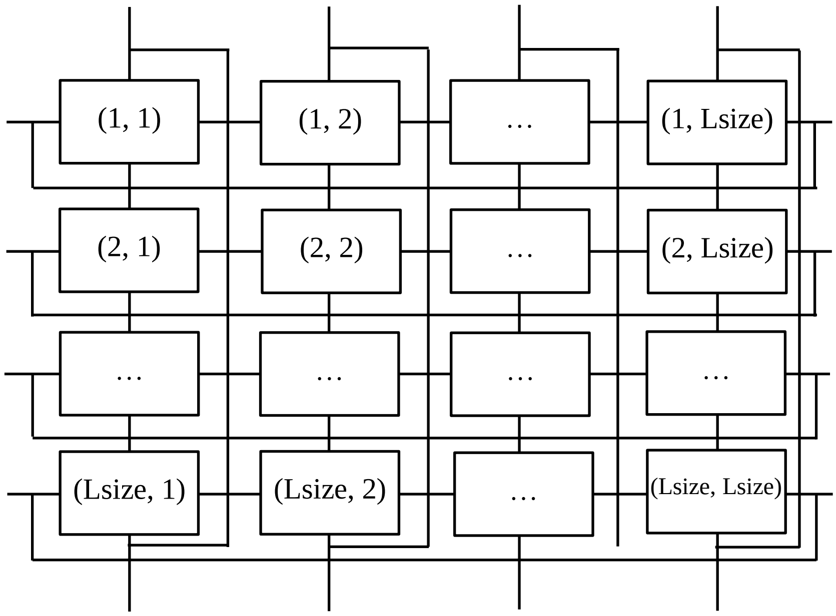


Figure 2.8. Agent lattice model [84]

Two agents can communicate if and only if there is a line between them. In the procedure of AES, all parameters are initialized and every agent in the lattice is evaluated. Afterwards, one behavior among competitive and learning is executed for each agent in the lattice repeatedly until some termination criteria are satisfied. For each agent, there are eight agents in its neighborhood to carry out the competitive behavior in terms of energy value. The aim of learning behavior is to improve energy value of each agent by applying the mutation and crossover operators [84].

Evaluation of this method shows good enough performance in solving dynamic optimization problems [84].

2.2.9 An Agent-Based Parallel Ant Algorithm with an Adaptive Migration Controller

Lin et al. in [85] proposed an agent-based parallel ant algorithm (APAA) for solving numerical optimization problems. In order to improve the algorithm's performance and enhance different parts of solution vector, the method uses two cooperating agents to reduce the scale of the problem handled by each of them. Each agent in APAA owns tunable and untenable vectors in which tunable vectors are optimized by an ant algorithm. Outstanding tunable vectors from an agent are moved to other agent as new untenable vectors in which the migration strategy is adjusted based on stagnation degree in optimization process. For solving the migration problem, a stagnation-based asynchronous migration controller was proposed by authors. APAA is convenient for solving large-scale problems and architectural framework is shown in Figure 2.9. The algorithm divides the solution vector *X* into two sub-vectors *X1* and *X2* in which the union of *X1* and *X2* is *X*. Meanwhile, each of *A1* and *A2* agents optimizes *X1* or *X2*. It means that if *X1* is tunable vector of *A1*, *X2* is untenable for it.

Evaluations of APAA showed better and faster results for benchmark functions in high dimensional spaces.

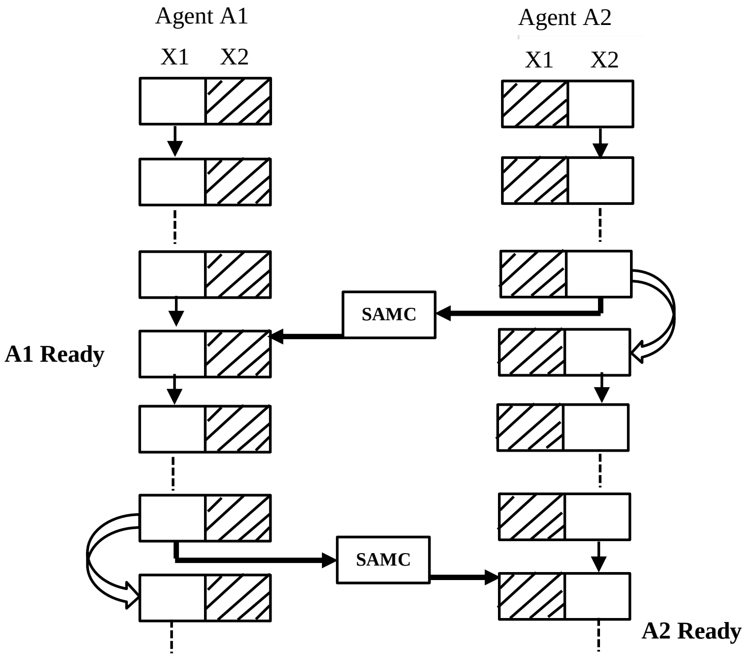


Figure 2.9. The APAA framework [85]

2.3 Multi-agent systems for multi-objective optimization

This section covers the state-of-the-art approaches of multi-agent systems for multi-objective optimization.

2.3.1 Multi-agent Evolutionary Framework based on Trust for Multi-objective Optimization

Jiang et al. proposed a novel multi-agent evolutionary framework based on the trust value for solving multi-objective optimization problems [14]. The authors considered individual solutions as intelligent agents in the proposed architecture. Also, the evolutionary operators and control parameters are represented as services, and intelligent agents choose services in each generation based on their trust values in order to produce new offspring agents. A trust value measures the suitability of the services for solving a particular problem. Once a new offspring is created, it starts to compete with other agents in its environment. A particularly selected service provides a positive outcome when the created offspring via that service can survive to the next generation; otherwise, the service affords a negative outcome. The trust value of services is calculated based on the count of positive and negative outcomes achieved so far. In order to balance between exploration and exploitation capabilities of the proposed approach, services are selected with probabilities that are proportional to the trust values. The authors implemented their methodology within state-of-the-art MOO metaheuristics NSGAII, SPEA2 and MOEA, and have shown that improvements are achieved with respect to the hypervolume measure.

2.3.2 Co-Evolutionary Multi-Agent System with Sexual Selection Mechanism for Multi-Objective Optimization

Drezewski et al. introduced a co-evolutionary multi-agent system (SCoEMAS) with sexual selection method based on Pareto domination [15]. In this system, the Pareto front includes a population of agents which are created from co-evolutionary interactions between sexes. Each sex has particular criteria and the agents belonging to a sex are evaluated based on the associated criteria. The system has one resource that is shared by the agents and environment. SCoEMAS includes a set of sexes, set of actions and a set of relations. The set of actions comprises operators for killing agents, searching for domination, distribution of resources, searching for partners, recombination, and migration. Meanwhile, the relation set models a competition between species to get the available resources. SCoEMAS realizes the sexual selection mechanism in which each agent has a vector of weights that are used for the selection of a recombination partner. This proposal has a comprehensive description of an evolutionary MAS, however its initial implementation exhibited poorer performance compared to NSGAII and SPEA2 algorithms. Drezewsky et al. introduced another work on MAS for MOO that is based on inspirations from host-parasite mechanisms and the corresponding method is named as HPSoEMAS [16]. Many components of this approach are similar to those of SCoEMAS and its performance compared to existing well-known metaheuristics is also close to that SCoEMAS.

2.3.3 Crowding Factor in Evolutionary Multi-Agent System for Multiobjective Optimization

Dorohinicky et al. proposed an evolutionary multi-agent system (EMAS) in which a new parameter called the crowding factor is introduced [17]. The main idea of EMAS is the integration of evolutionary algorithms to a MAS at population level such that the agents are able to generate new agents by using recombination and mutation operators or die and became eliminated from the system. The fitness of agents is expressed in terms of the amount of gained non-renewable resource called life energy. Therefore, the agents with high life energy have more chance to be selected for recombination and, in contrast, the low life energy increases the possibility of death. The crowding factor represents the degree of closeness of agents in terms of the similarity of solutions they represent. EMAS is implemented with a mechanism of reducing life energy of agents having solutions close to each other. The authors have studied the effects of crowding factor on the quality of Pareto fronts using simple test problems and they demonstrated the positive impact of lower crowding factors on extraction of better Pareto fronts. However, the obtained results are not compared to results of any state-of-the-art methods.

2.3.4 Genetic algorithms using multi-objectives in a multi-agent system

A multi-agent system consisting of several heuristics within the genetic algorithm framework is proposed by Cardon et al. for the optimization of Gantt diagrams in job-shop scheduling problem. The goal of the optimization task is the minimization of delays and completion of jobs according to deadlines given in problem description. The skeleton of the proposed model is based on the contract-net protocol that aims to discover a good scheduling through agent negotiations. Authors used appropriate methods for selection, crossover and mutation operators [18]. The MAS starts with a task distribution to individual agents and each agent of this system includes a genetic algorithm as its main search mechanism. The communications among agents using the contract-net protocol leads the system to optimize the scheduling according the above mentioned objective function. Experimental results have been reported over 5 instances of job shop scheduling problem and illustrations showed that the delay decreases quickly. No comparison to other methods or other multi-agent systems in literature is provided by the authors.

2.3.5 Elitist Evolutionary Multi-Agent System

Siwik et al. proposed a semi-elitist evolutionary multi-agent system (selEMAS) for the purpose of avoiding stagnation and preserving agents representing high-quality solutions [19]. Elitism ensures that non-dominated solutions will survive in the next generation. Also for maintaining diversity of solutions in selEMAS, self-adapting niching and distributed crowding methods are used. The goals of agents in selEMAS are to survive and create offspring. This way, agents collects non-renewable resources called life energy and as long as their life energy is upper than death threshold, they stay alive. Meanwhile, when the amount of life energy is more than reproduction threshold, they can compete with other agents to produce offspring. Experimental results using one particular test problem exhibited that significant improvements are achieved compared to non-elitist EMAS method.

The multi-agent system (MAS) proposed in chapter 3, 4 and 5 possesses novel properties compared to the above pioneering implementations. The multi-agent system in chapter 3 includes several metaheuristics as problem solving agents acting on a common population and it also maintains a two-stage common archive keeping the promising solutions in fitness value and in spatial distribution. The proposed MAS approach runs in consecutive sessions and each session includes two phases: in the first phase a particular metaheuristic is selected based on its fitness value in terms of its improvements achieved in objective function value and the second phase lets the selected metaheuristic conduct its particular search procedure until some termination criteria are satisfied. In all phases and iterations of the proposed framework, all agents use the same population and archive in conducting their search procedures. This way, agents cooperate by sharing their search experiences through accumulating them in a common population and common archive. The proposed MAS includes dedicated agents to initialize parameters, retrieve data from common population and archive, and control communication and coordination of agents' activities. The resulting MAS framework is used to solve a hard combinatorial optimization problem and analysis of the obtained results showed that the objectives on the design of the proposed MAS are almost all achieved.

The MAS proposed in chapter 4 includes several metaheuristics as problem solving agents acting on a common population and it also maintains a common archive keeping the promising solutions extracted so far. The proposed MAS approach runs in consecutive sessions and each session comprises two phases: the first phase sets up a tournament among all agents to determine the currently best performing agent and the second phase lets the winner to conduct its particular search procedure until termination criteria are satisfied. In all phases and iterations of the proposed framework, all agents use the same population and archive in conducting their search procedures. This way, agents compete with each other in terms of their fitness improvements achieved over a fixed number of fitness evaluations in tournaments, and they cooperate by sharing their search experiences through accumulating them in a common population and a common archive. The proposed MAS includes one supervisory agent that controls communication and coordination of agents' activities through monitoring the common population and the common archive. The resulting MAS framework is used to solve real-valued optimization problems within the well-known CEC2005 benchmarks set. Analysis of the obtained results showed that the objectives on the design of the proposed MAS are almost all achieved.

The MAS proposed in chapter 5 encompasses novel characteristics compared to the above mentioned MO MAS frameworks. The proposed method comprises some MOO metaheuristic agents acting on subsets of a common population. In addition to an assigned subset of population elements, agents also maintain their local archives keeping the non-dominated solutions extracted during a particular session. The proposed method runs in consecutive sessions and each session includes two phases as follows: First phase divides the common population into subpopulations according to dominance ranks of its elements, so that, first subpopulation contains the solutions with rank 1, elements of the second subpopulation have rank 2, and so on. In the second phase, each MOO metaheuristic agent is assigned to one particular subpopulation and starts improving its elements for the purpose of lowering their ranks and making them closer to the best Pareto front found so far. Due to the round-robin type assignment strategy, each metaheuristic operates on a different-rank subpopulation in subsequent sessions. A session starts with a new assignment of metaheuristics and ends when termination criteria are satisfied. In each session, extracted non-dominated solutions are kept in local archives and all non-dominated solutions found so far are combined into a global archive at the end of the session. Upon completion of a session, updated subpopulations in each MOO metaheuristic are combined together to update the common population and to recalculate the ranks of individual solutions before starting the next session. This way, metaheuristic agents share their experiences through improved solutions when collecting them in a common population and a common global archive. The proposed MAS includes one supervisory agent that controls communication and coordination of agents' activities through monitoring individual sessions, common population and the common archive. The resulting MAS architecture is used to solve real-valued multi-objective optimization problems within the well-known CEC2009 benchmarks set. Analysis of the obtained results showed that the resulting MAS is in fact a powerful alternative for the solution of hard numerical MOO problems.

Chapter 3

DESCRIPTION OF METAHEURISTICS USED WITHIN THE PROPOSED MULTI-AGENT SYSTEMS

3.1 Single-objective metaheuristics used within the proposed multi-agent systems

3.1.1 Genetic Algorithms (GA)

Genetic algorithms (GAs) are search and optimization algorithms developed based on inspirations from principles of natural evolution. Their algorithmic and computational descriptions are first developed by John Holland in 1975 [20, 35, 36]. Basically, GAs operate on a population potential solutions and representations of individual solutions in the solution space are called chromosomes. Content of chromosome is named as genotype of the corresponding individual, whereas the evaluation of the underlying objective function for a chromosome is called the fitness or phenotype. Starting from a randomly initialized population of solutions, GAs run over consecutive generations and modify individual chromosomes through three genetic operators, namely natural selection, crossover and mutation. Natural selection operator works on the current population and selects individual to be used by the crossover operator. Natural selection is a stochastic operator that favors higher-fitness individuals to pass their genetic characters to future generations. Crossover operator takes more than individual and mixes their genetic characters (or allelic values) to generate a number of offspring, with the objective that offspring will have better fitness values than their parents. Crossover is a kind of intensification operator that does not introduce new genetic information into the population. In fact, this task is performed by the mutation operator that assigns random domain-specific allelic values to genetic location. Mutation is a diversification operator and it is usually applied with a small probability. When a new population of offspring is generated, it replaces the old population and a new generation starts with the same sequential application of genetic operators. Generations terminate when predefined termination criteria are satisfied. An algorithmic description of GAs is given in Algorithm 3.1. Details of implementation and problem specific representational issues of GAs can be found in [29].

Algorithm 3.1. Genetic Algorithms(*Pop*,P_C,P_m),

1. *Iteration* = 1;
2. *Pop* = Initial population;
3. *Fitness*=f_{obj}*(Pop);*
4. *Best_Solution* = Best-fitness chromosome within the *Pop*;
5. *Termination_Cond=FALSE;*
6. While not(*Termination_Cond),*
    i. *Mating_Pool*=Selection(*Pop*);
    ii. *Offspring*=Crossover(P_C,*Mating_Pool*);
    iii. *New_Pop*=Mutation(P_m,*Offspring*);
    iv. New_Fitness= f_{obj}*(New_Pop);*
    v. Update the *Best_Solution*;
    vi. *Pop=New_Pop*;
    vii. *Fitness=New_*Fitness;
    viii. *Iteration=Iteration+1*;
    ix. *Check(Termination_Cond)*;
7. End While.
8. Return *Best_Solution* found so far.

3.1.2 Artificial Bee Colony Optimization (ABC)

Bee colony optimization is a general-purpose population-based metaheuristic inspired from the foraging behavior of honey bees [31]. Based on the natural analogy, this method maintains a bee swarm of three different types of individuals, namely workers (or employed bees), onlookers and scouts. Even though there are a couple different implementations of ABC method, the basic principles are as follows:

all individuals have the same representation and each artificial bee, with its strategically associated move operators, is a potential solution to the underlying problem. The algorithm runs in two (or three) consecutive phases. In the first phase, all bees of the swarm (named as employed bees) construct a new solution using one or more of the available moves. The second phase, named as backward pass or onlookers' phase, solutions build in the first phase are sorted in non-increasing order of their fitness values and some bees are further allowed to continue exploring the search space. These onlooker bees are simply selected by roulette wheel selection and onlooker bees may use specialized move operators to apply a kind of local search around the potentially promising solutions. Finally, depending on the type of implementation, a third phase of diversification moves may be performed by those bees that either could not achieve sufficient performance over a specified life-time or have significantly poor fitness compared to the best solution found so far. These bees use mutation type move operators and are named as scouts. Algorithm 3.2 presents the pseudocode of a generic ABC algorithm.

Algorithm 3.2. Artificial Bee Colony Algorithm(Input Problem),

1. *B = Size of Bee Swarm;*
2. *Max_Iter = Max. number of iterations;*
3. *BS=Best Solution;* *% It is initially empty*
4. *BF=Best Fitness;* *% It is initially +Inf*
5. *DONE=False;*
6. while not(DONE),
7. % First phase: Employed Bee phase or Recruitment Phase
   For j=1 to B Do
      Let each bee $b_i \varepsilon B$ build a solution using one or more of the strategic operators from the set of available moves. Evaluate the built solutions and get their fitness values;
8. % Second phase: Onlookers phase or Backward phase
      Bees exchange the information about their constructed solutions and decide about which of them can be used for further exploration for the purpose of refinement around potentially promising solutions. This is implemented through sorting the constructed solutions in non-decreasing order of their fitness values and allows a number of them to continue the search. Onlooker bees are generally determined by roulette wheel selection;
9. % Third phase: Scout bees phase
      Those artificial bees that do not satisfy predefined performance criteria in their life time are re-initialized randomly (or modified by mutation operators) to allow them to get rid of their locally optimal valley;
10. *Update BS=Best Solution;*
11. *DONE=Check_Termination(Iter,Max_Iter,BS);*
12. end while.

Performance of ABC algorithms has already been tested using real-valued and combinatorial optimization problems and they are found competitive to well-known metaheuristics for hard problem instances.

3.1.3 Particle Swarm Optimization (PSO)

Particle swarm optimization (PSO) is another well-known swam based optimization algorithm developed based on observations on bird flocks' behavior while they travel over long distances. It is first introduced as a computational procedure by Eberhart and Kennedy in 1995 [32, 33, 37]. PSO is a simple to implement algorithm that can be described fundamentally as follows: in swarm of particles (birds), each individual has two spatial components, namely the velocity, V, and the position, X. An individual plans its trajectory by iteratively updating its velocity and position using two kinds of information. The first one is on the accumulated personal-experience (*Pbest*) that a single particle gains throughout its search history while the second (*Gbest*) reflects the collective experience of all individuals within the swarm. The firs kind of information is named as personal best whereas the second one represents the global best solutions extracted so far. PSO is algorithmically presented in Algorithm 3.3.

Algorithm 3.3. Particle Swarm Optimization Algorithm(Input Problem)

1. *Swarm_Size= Size of particle swar;*
2. *Swarm_Pos=Randomly initialize positions of particles considering the ranges of variables;*
3. *Swarm_Vel=Randomly initialize velocities of particles between* V_{min} *and* V_{max};
4. *Fitness=Evaluate* F_{obj}*(Swarm_Pos);*
5. *Set* P_{best} *solutions of particles and* G_{best} *solution of swarm.*
6. *Term_Cond=False;*
7. while not(*Termination_Cond)*,
   i. For each particle, calculate $V_{i+1}= \omega *V_i + C_1*R_1*(X_i-P_{best})+C_2*R_2*(X_i-G_{best}),\ 0<\omega<1$ is the momentum coefficient, C_1 and C_2 are two real constants and $R_1,R_2\varepsilon[0,1]$ are two uniformly selected random numbers;
   ii. For each particle, set $X_{i+1}=X_i+V_{i+1;}$
8. $Fitness_{i+1}$*=Evaluate* $F_{obj}(Swarm\_Pos(X_{i+1}))$*;*
9. *Update* P_{best} *of each particle and* G_{best} *of swarm;*
10. *Check(Term_Cond);*
11. *end while*

3.1.4 Differential Evolution (DE)

Differential evolution, developed by Storn and Price in 1996 [21, 22], is also a population based metaheuristic that is quite powerful for real-valued optimization problems. Like GAs, DE generates new solutions using its algorithm specific selection, crossover and mutation operators and it has numerous variations depending on how arguments of these operators are selected and used. Starting with a randomly built initial population of N individual solutions *Xi, i=1,…,N*, the conventional implementation of DE starts by generating N mutant solutions Vi, one for solution *Xi*, by adding the weighted difference of two randomly chosen solution *Xr1* and *Xr2* to a third randomly chosen solution *Xr3* such that $r1 \neq r2 \neq r3 \in \{1,2,\ldots,N\}$ and $i \notin \{r1,r2,r3\}$. Then, each individual Xi is crossed over with its mutant vector *Vi* to generate a trial vector *Ui* as follows (3. 1):

$$U_i(j) = \begin{cases} V_i(j), if\ rand([0,1]) \leq p_c\ or\ j = \delta \\ X_i(j), if\ rand([0,1]) > p_c\ and\ j \neq \delta \end{cases} \qquad (3.\ 1)$$

where $pc \in [0,1]$ is the crossover probability, rand([0,1]) is a uniformly selected random number in [0,1] and δ is a randomly selected index in $\{1,2,\ldots,N\}$. Finally, fitness of *Ui* is evaluated and it replaces *Xi* only if its fitness is better than that of *Xi*, otherwise *Xi* keeps its presence within the population. A pseudo code of DE algorithm is given Algorithm 3.4.

Algorithm 3.4. Differential Evolution Algorithm(*Input Problem*)

1. *N=Population Size*;
2. *Pop= Randomly initialize* $X_i \in Pop$ *considering the ranges of variables;*
3. Fitness=F_{obj}(Pop);
4. Done=False;
5. while not(*Done)*,
   i. for i=1 to N,
   ii. Randomly determine X_{r1}, X_{r2}, $X_{r3} \in Pop$ *such that* $i \neq r_1 \neq r_2 \neq r_3$;
   iii. *RNum=rand();* // *A uniformly distributed random number in [0,1]*
   iv. *if RNum < CR,* // *0<CR<1*
   v. *Compute the mutant vector* $V_i=X_{r1}+F*(X_{r2}-X_{r3})$, *where F is the scaling factor;*
   vi. *else*
   vii. $V_i=X_i$;
   viii. *Compute the trial vector* U_i *by crossing over* X_i *and* V_i *as described above;*
   ix. *Evaluate the fitness of* U_i;
   x. *if* $F_{obj}(U_i)$ *is better than* $F_{obj}(X_i)$, *then* U_i *replaces* X_i;
   xi. *end for*
   xii. *Check(Done);*
6. end while;

3.1.5 Evolution Strategies (ES)

Evolution Strategies (ES) is the oldest evolutionary algorithm developed in 1960s for the purpose of continuous numerical optimization [30]. It is one of the well-known metaheuristics for which a mathematical analysis of convergence exists and it is among the known powerful algorithms for numerical optimization problems. ES works on a population P of individual solutions represented as N-dimensional vectors in *RN*, where *N* is the number variables under consideration. ES has unique recombination, mutation and selection operators with parameters that are adaptively changed depending on the varying fitness landscape topology throughout the execution of algorithm. In principle, ES works as follow: first one or more individuals are selected from the current population and their offspring are generated using duplication and recombination operators. Then, offspring are mutated to obtain a population of new solutions. This way, starting with a population of μ individuals, a new population containing λ offspring is generated and, in general $\lambda>\mu$. The selection procedure, called as environmental selection, selects μ individuals from a combined population $\mu+\lambda$ solutions to build the population of next generation. An algorithmic description of the ES algorithm is presented in Algorithm 3.5.

Algorithm 3.5. (μ^+,λ)-Evolution Strategies Algorithm(Input Problem)
1. Set values of μ and λ;
2. Set ρ=Number of solutions to be used recombination;
3. Initialize $P=(X_i,S_i,F_{obj}(X_i))$, $1 \le i \le \mu$, where S_i is the set of strategy parameters (e.g., step sizes for variables) for the i-th solution X_i;
4. $Fit_i=F_{obj}(X_i)$, $1 \le i \le \mu$;
4. Done=False;
5. while not(Done),
6. for i=1 to λ,
7. (S',X')=Select_Mate(ρ,P);
8. S_i''=s_recombine(S',X');
9. X_i''=x_recombine(S',X');
10. S_i'''=s_mutate(S_i'');
11. X_i'''=x_mutate(X_i'');
12. F_i =F_{obj}(X_i'');
13. end for;
14. Set $P_U=P\cap\{(X_i''',S_i'''), 1\le i\le\lambda$;
15. P=Environmental_Select(P_U);
16. Check(Done);
17. end while.

3.1.6 Simulated Annealing (SA)

Simulated annealing (SA) is a trajectory based optimization method developed by Kirkpatrick, Gelatt and Vecchi in 1980s [23]. This algorithm is inspired from thermal equilibrium of particle systems and procedures used in annealing of metals. It is known that, at sufficiently high temperatures metals pass to liquid state and particles freely move around within a container. Then, if the metal is allowed to cool down sufficiently slowly so that it is allowed to reach thermal equilibrium at each temperature step, particles reach to a specific arrangement at which the internal energy of the metal takes its minimum value. This arrangement of particles is called the ground state. Hence, physical annealing is actually a procedure of minimizing the internal energy of the particle system under consideration. This physical process is transformed into a computational procedure, the simulated annealing algorithm for which a proof of global convergence with probability one is also available [38]. SA is successfully used for the solution of hard numerical and combinatorial optimization problems. SA's main drawback is its slow convergence for hard problems due to large number of moves required at temperature step to get closer to thermal equilibrium. A pseudo code for SA is given in Algorithm 3.6.

Algorithm 3.6. Simulated Annealing Algorithm(Input Problem)

```
T=T_0 ; % Compute or estimate the initial temperature;
N_T = Maximum number of new solutions (moves) to be generated at each temperature step;
Set cooling coefficient α;
Set conditions for termination;
S=Initialize the starting solution randomly;
Done=False;
while not(Done),
    for i=1 to N_T,
        S'=Make_Move(S); % Generate S' from S using an available move.
        δ=F_obj(S)-F_obj(S');
        if δ<0,
            S=S';
        else
            R=exp(-δ/T);
                if R>rand(0,1),
                    S=S';
                endif
        endif
    end for
    T=α*T;
    Check(Done);
end while.
```

3.1.7 Great Deluge Algorithm (GDA)

Great Deluge algorithm is a trajectory-based optimization algorithm that is similar to simulated annealing (SA) except for its dynamically adjusted level-based acceptance mechanism [25, 39]. The algorithm starts with a randomly constructed initial solution and has three fundamental parameters to be set initially. These are the estimated value of fitness for a globally optimal solution, the maximum number of iterations and the initial value of the level parameter. Usually, the initial value of the level parameter is set equal to the fitness of the initial solution. Throughout the execution of GDA, the value of the level parameter is decayed linearly or nonlinearly, and the acceptance of new solutions depends on the level parameter. The basic GD algorithm is described in Algorithm 3.7.

Algorithm 3.7. Great Deluge Algorithms(Input Problem)
1. Iter = 0; % Initialize the iteration counter
2. S_{Iter} = Build an initial solution;
3. Compute $F_{obj}(S_{Iter})$;
4. S_{best}=S_{Iter}; F_{best}= $F_{obj}(S_{Iter})$; % This the best solution found so far;
4. Max_Iter= Set the maximum number of iterations;
5. FGbest = Estimate the fitness of a globally optimal solution;
6. Level(Iter)= $F_{obj}(S_{Iter})$;
7. Set the level decay parameter, ΔLevel = ($F_{obj}(S_{Iter})$ − FGbest)/Max_Iter;
8. NILength = Not improving length limit; /*This is one of termination conditions when the algorithm gets stuck at a locally optimal solution.*/
9. Set Not improving counter to zero, NICount = 0;
10. Done = False;
11. while(not(Done)),
12. Generate a new solution, S_{Iter+1}, starting from S_{Iter}, using the available operators;
13. Compute F_{New} = $F_{obj}(S_{Iter+1})$;
14. if F_{best}>F_{New};
15. S_{best}=S_{Iter+1};
16. F_{best} = F_{New};
17. NICount=0;
18. elseif $F_{New} \leq$ Level(Iter),
19. NICount = 0;
20. else
21. S_{Iter+1}=S_{Iter}; % Reject the move
22. NICount = NICount + 1;
23. If NICount > NILength,
24. DONE = 1;
25. end if;
26. end if;
27. Level(Iter+1) ← Level(Iter) − ΔLevel;
28. Iter ← Iter + 1;
29. If Iter == Max_Iter,
30. DONE ← 1;
31. end if;
32. end while.

3.2 Multi-objective metaheuristics used within the proposed multi-agent systems

3.2.1 Non-dominated Sorting Genetic Algorithm (NSGA II)

Non-dominated sorting genetic algorithm (NSGAII) is a well-known evolutionary multi-objective optimization algorithm developed in 2002 by K. Deb et al. [42]. NSGAII applies elitism and crowding operators to preserve high-quality solutions and increase spread along the Pareto front. NSGAII starts with a randomly initialized population and computes the ranks of solutions such that the rank of a solution is the number of other population elements dominating this particular individual. In fact, each rank represents a particular Pareto front in objective space. Accordingly, all solutions are sorted in increasing order of their ranks and they are assigned a rank-fitness proportional to their levels or fronts. Then, the algorithm uses the computed fitness-ranks and applies selection, crossover and mutation operators to create the offspring population. At the end of each generational step, parent and offspring populations are combined, ranks of solutions are computed and the new population is filled from ranked-sets in increasing order of rank values. If the number of elements of the latest rank exceeds the remaining space to be filled, the some of its elements are eliminated based on crowding distance criterion. The above described procedural steps are repeated until predefined termination criteria are satisfied. For the problems having strong parameter interactions, NSGAII is effective in extracting Pareto fronts closer to the optimal one. A detailed description of the NSGAII algorithm can be found in [42].

3.2.2 Multi-objective Genetic Algorithm (MOGA)

MOGA is another evolutionary multi-objective optimization algorithm for approximating the Pareto-Front based on ranking niche formation strategies [43]. The rank of the j-th individual is computed as the number of other individuals dominating it plus 1; hence all non-dominated solutions are assigned rank 1. Consequently, fitness values of individuals are determined using interpolation of rank values. The second distinctive feature of MOGA is its implementation of niche size, δshare, in objective space. δshare represents a measure on the distance between two individuals so that they may decrease each other's fitness values. Given a solution set S in objective space, MOGA first computes minimum (mq) and maximum (*Mq*) values along each objective axis q. Then, the number hypercubes of size δshare that can be placed within the hyperparallelogram with corners *(m1,…,mk)* and *(M1,…,Mk)* is computed, which facilitated the computation of δshare from a simple comparison of volumetric equality idea. Experimental work on a real-world engineering problem showed that MOGA is successful on gradual improvement of Pareto fronts.

3.2.3 Multi-objective Differential Evolution (MODE)

In general, multi-objective implementations of differential evolution are based on extension of the single-objective differential evolution (DE) algorithm. MODE, proposed by Xue et al. [45], has similarities with the DE variant DE/best/1/bin. The proposed method implements a Pareto based approach for the selection of the best individual as follows: if the trial solution is dominated, then the best is randomly chosen from subset of non-dominated solutions. If the trial solution is non-dominated, then it is chosen as the best individual. For the purpose of population management, the authors used a $(\mu+\lambda)$-selection strategy, Pareto ranking and crowding distance mechanisms are used to get solutions that have a well spread along the computed Pareto Front. MODE is used to solve unconstrained problems of high dimensionality and it is shown to generate improved solutions compared to SPEA algorithm.

3.2.4 Multi-objective Particle Swarm Optimization (MOPSO)

Coello et al. proposed the multi-objective particle swarm optimization (MOPSO) method that extends the standard PSO algorithm to deal with multi-objective optimization problems [47]. This method maintains an external global repository to store the non-dominated solutions extracted within the algorithm. MOPSO also uses the concept of Pareto dominance to determine the flight direction. An important issue in MOPSO algorithm is the generation of hypercubes in which coordinates of a particle is defined with respect to its objective function values. These hypercubes are then used to determine a repository element that acts as the global best solution in velocity computation of the particle under consideration. For this purpose, fitness values of hypercubes are first scaled inversely proportional to their cardinality and the one from which the global best will be taken is determined through roulette wheel selection. Detailed description of the MOPSO is presented in [47].

3.2.5 Archived Multi-objective Simulated Annealing (AMOSA)

Archived multi-objective simulated annealing (AMOSA), proposed by Bandyopadhyay et al. [46], is a multi-objective optimization method based on the standard simulated annealing algorithm (SA). Like the SA algorithm, AMOSA is also a trajectory based method that maintains an archive to store the non-dominated solutions found through algorithm execution. Archive size changes between a hard limit HL that is the maximum size of the Archive on termination and a soft limit SL that is the maximum size to which the Archive may increase before it is reduced size HL with clustering. Acceptance of a new solution is based on three main criteria according to the dominance relation among the new solution and its parent. Accordingly, the acceptance probability of a new solution is computed using the domination counts of new solution, its parent and the archive elements. A detailed description of the AMOSA algorithm is presented in [46].

3.2.6 Strength Pareto Evolutionary Algorithm (SPEA2)

Strength Pareto Evolutionary Algorithm is an evolutionary multi-objective optimization method proposed by Zitzler et al. [44]. The algorithm uses a regular population and maintains an external archive for storage of non-dominated solutions. Each archive element $A(i)$ is assigned a strength value $S(i)$ which is equal to the number of population elements that are dominated by or equal to *A(i).* For archive elements, *S(i)* also represents the fitness value *FA(i)* of *A(i).* For a population element *P(j),* its fitness *FP(j)* is calculated from the sum of *S(i)* values of archive members that dominate or equal to *P(j).* A one is added to this sum to avoid zero fitness values. These fitness values, *FA (i)* and *FP (j)*, are called the raw fitness and they may cause ranking difficulties when most individuals do not dominate each other. To solve this problem, SPEA2 introduces density information to differentiate between individuals having identical raw fitness values and actual fitness of an individual is taken as the sum of its raw fitness and the density information. Following the actual fitness computation, external archive is updated by extracting non-dominated solutions from union of population and old archive members. Finally, a mating pool is formed using updated archive elements through binary tournament selection and offspring individuals are generated with crossover and mutation operators. Experimental evaluations over sets of well-known test problems demonstrated that SPEA2 achieved a success similar to that of NSGAII.

Chapter 4

LEARNING-BASED MULTI-AGENT SYSTEM FOR SOLVING COMBINATORIAL OPTIMIZATION PROBLEMS

4.1 Introduction

Due to NP-complete computational complexity, solving hard combinatorial optimization problems using exhaustive search methods is not computationally feasible. Hence, metaheuristics like evolutionary algorithms are applied to reach a near optimal solution within reasonable running times. Like evolutionary algorithms, other nature- and bio-inspired metaheuristics have been developed and their success for the solution of difficult combinatorial optimization problems is demonstrated through experimental evaluations.

This chapter presents a novel multi-agent system in which a number of metaheuristic agents act cooperatively through sharing their individual experiences gained individually and the overall multi-agent system favors those agents based on their performance in search for good solutions. The proposed learning-based multi-agent system (LBMAS) is supported by a two-stage external memory archive. The first stage stores promising solutions based on their fitness values. The second stage keeps promising solutions that are apart from each other based on a defined dissimilarity measure. Individual metaheuristics act one at a time and average improvement achieved by each individual agent in fitness function is recorded. Then, to decide which metaheuristic is the best to employ for the next turn, individual average improvement of each agent is taken as its fitness and the agent selection is carried out using roulette-wheel selection method. The proposed multi-agent system also contains dedicated coordination agents for data and message transfer among agents, retrieval of common population and the common archive elements, and initialization of algorithm parameters. A detailed architectural description the proposed multi-agent system is presented in Section 4.2.

The seven metaheuristics implemented within the framework of the proposed approach are Genetic Algorithms (GAs) [20], Differential Evolution (DE) [21, 22], Simulated Annealing (SA) [23], Ant Colony Optimization (ACO) [24], Great Deluge Algorithm (GDA) [25], Tabu Search (TS) [26], and the Cross Entropy (CE) [27] method. Detailed descriptions of these metaheuristics can be found in the associated references. There are additional four agents implemented within the proposed system. They are as follows, Problem Agent initializing the parameters of the input problem, Solution Pool Agent handling all transactions with the common population, Archive agent handling retrieval and update operations associated with the common two-stage archive and the Manager Agent that handles coordination and performance based employment of individual agents.

Next Section introduces a detailed description of the proposed multi-agent system of metaheuristic agents. Experimental results in solving a well-known combinatorial optimization problem are given in chapter 7.

4.2 The proposed multi-agent system for solving combinatorial optimization problems

This section introduces the proposed learning-based multi-agent system (LBMAS) and agent interaction mechanism for solving a single objective combinatorial optimization problem, namely the multiprocessor scheduling problem. The proposed multi-agent system allows collaboration of metaheuristic agents over a common population and a two-stage common archive in such a way that promising solutions are searched over different regions of the search space using the currently most effective agent. In order to achieve the objectives of the proposed multi-agent system, agents responsible for initialization, data retrieval, archive management and agent coordination are also maintained within the system. Figure 4.1 illustrates the architectural components and functional interactions within the proposed system.

This multi-agent framework includes 7 metaheuristic agents and 4 system agents. When selected, each metaheuristic agent applies its own search strategy and returns its discovered solutions to the manager agent. Then, the manager agent distributes this data to solution pool and archive agent which they update the common population and common archive respectively. Selection of metaheuristics is carried out using roulette-wheel selection principle where fitness values of metaheuristics are taken as their level of improvements in objective function. Initially all metaheuristics have the same rate of being selected and these improvement rates are increased or decreased based on performance of individual agents. In this respect, when the average fitness improvement achieved by particular agent is positive, its improvement rate is increased proportional to the improvement. On the other hand, if the agent’s average improvement in fitness is not above a predefined percentage threshold, then its improvement rate is decreased by a constant amount. However, the improvement rates are not reduced below a lower limit. A second important component of the proposed system, that is very effective on the overall performance of the proposed system, is the two-stage external memory architecture which is first proposed in [28].

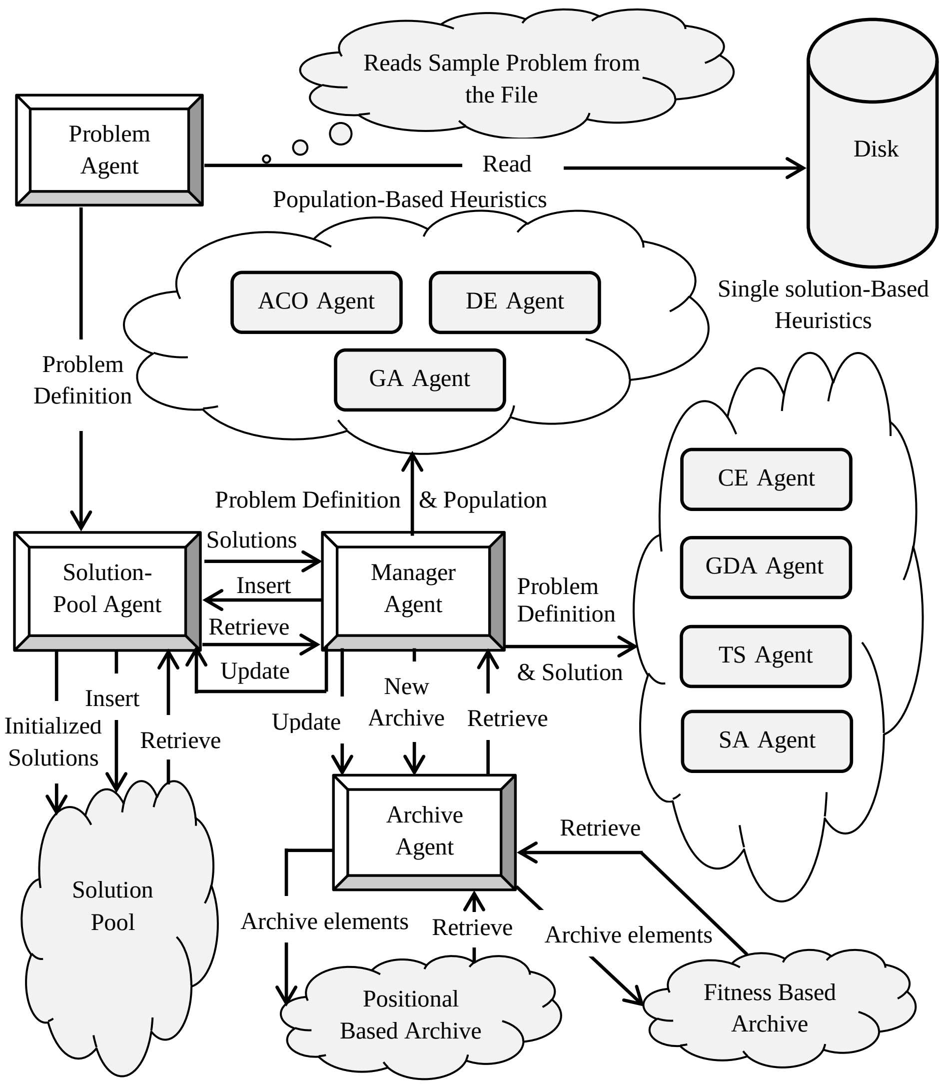


Figure 4.1. Architectural description of LBMAS concerning its metaheuristic agents and the four functional agents

In this architecture the first level acts as a short term memory keeping the promising solutions considering their fitness values. Hence, elements of the first stage are frequently updated each time a solution better than the worst element is extracted. The second stage archive acts as a long term memory that is updated only after the first stage archive is updated for a predefined number of times. Furthermore, elements of the second stage archive are selected so that they are mutually dissimilar based on a similarity measures. In the proposed system, hamming distance is used for similarity measure and elements of second stage archive are required to be dissimilar in at least half of their elements. This way, exploitation of promising solutions from different regions of the solution space is achieved, that is a very important issue for multimodal optimization problems.

As illustrated in Figure 4.1, the seven metaheuristic agents implemented within the framework of the proposed approach are GAs, DE, SA, ACO, GDA, TS, and CE. The other four agents implemented within the proposed system are as follows: Problem Agent that handles all initialization procedures including the parameter settings, update rules, and variable ranges. Solution Pool Agent handles all transactions with the common population and associated communications with other agents. Archive agent performs retrieval and update operations associated with the common two-stage archive and the Manager Agent coordinates activities of agents and carries out the performance based selection of individual agents. Most of the critical operations for stable running of the proposed system are performed by the Manager Agent.

The resulting MAS framework is used to solve a hard combinatorial optimization problem and analysis of the obtained results showed that the objectives on the design of the proposed MAS are almost all achieved. Evaluation and experimental results is presented in chapter 7.

Chapter 5

A TOURNAMENT-BASED COMPETITIVE-COOPERATIVE MULTI-AGENT ARCHITECTURE FOR REAL PARAMETER OPTIMIZATION

5.1 Introduction

Real parameter optimization problems are of great importance in engineering applications and they are widely solved by metaheuristics due to their computationally simple search mechanisms, applicability in diverse range of problems, and power to extract near optimal solutions using feasible computational resources. In real parameter optimization, the objective is to find the global minimum/maximum real value for a function of the form, where $X \in R^n$ and $x_i \in X$ represent a solution vector and its component along the ith axis, respectively. Usually, each x_i is restricted to take its values from a specified domain D_i. In most of the practical cases, the size of search space and fitness-landscape complexity makes the problem intractable for extracting the globally optimal solution. Consequently, near-optimal solutions satisfying predefined acceptability criteria become the targets of the optimization task.

Several metaheuristics that are particularly suited for the solution of real-valued optimization problems have been designed and their successes are demonstrated through solving difficult benchmark problem instances. Description of all available metaheuristics developed for real-valued optimization is not an objective of this study that is also impractical due to rapid developments in this hot research field. Instead, we want to illustrate the effectiveness of a multi-agent architecture of metaheuristic agents for single-objective real-valued optimization problems. In this respect, we considered metaheuristics that are mostly cited in literature due to their pioneering statue and published success for problems under consideration. Particularly, metaheuristic agents with search mechanisms of evolution strategies (ES) [30], simulated annealing (SA) [23], genetic algorithms (GA) [20, 29], artificial bee colony optimization (ABC) [31], particle swarm optimization (PSO) [32], differential evolution (DE) [21, 22] and great Deluge algorithm (GDA) [25] are taken into account within the proposed multi-agent framework. Basic descriptions and application principles of these metaheuristics will be presented in the following sections.

A multi-agent system (MAS) is a social environment for a population of agents each of which performs a goal-oriented task on the environment using their own operators. Basically, each agent gets a set of percepts from their environment, processes the percepts under light of their accumulated knowledge, and act on the environment through their available operators to achieve a predefined design goal. MASs can be categorized as homogeneous where all agents are identical in architecture and capabilities or heterogeneous where each agent may have its own architectural components and computational procedures [2]. In either of the two categories, a multi-agent system is designed to carry out a particular task through social interaction of its agents. This social interaction is also usually of two types, namely cooperation or competition. Both of these social interactions require agents to use communication mechanisms through which they can share or exchange information.

Details of popular multi-agent systems and their design approaches are given in the next section. State-of-the-art literature on MASs designed for real-valued function optimization is also presented in detail in sections below.

This chapter presents a heterogeneous MAS for the solution of real-valued single-objective optimization problems. In the proposed framework, each agent performs the function of conventional implementation of a particular metaheuristic. Agent architectures are made of the data structures included in their associated metaheuristic algorithm while the search operators provided by each metaheuristic set up the action sets of corresponding agents. All agents work on the same population and have a common memory. This way, agents exchange information through using and sharing elements of the same population while they are sharing their accumulated experience within the maintained common archive keeping the most promising solutions found so far. Both cooperative and competitive interactions take place within the proposed system: the system works in consecutive sessions and at the beginning of each session agents compete with each other to get the task of searching for new solutions, whereas each agent cooperates with the others by sharing its extracted solutions within the common population and its accumulated experience within the common archive. The proposed MAS is experimentally evaluated using the well-known IEEE CEC2005 benchmark problems set that includes 25 benchmark functions with different modal and fitness landscape complexities [33, 34]. Comparative analysis of the obtained results showed that the proposed framework performs significantly better than its state-of-the-art competitors in almost all problem instances. Architectural, computational and inferential implementation details of the proposed heterogeneous competitive-cooperative multi-agent system are given in Section 5.2. Also, Description of experimental suit, test problems, algorithm parameters and comparative analyses in terms of quantitative and statistical computations is presented in chapter 7.

5.2 The proposed heterogeneous competitive-cooperative multi-agent system for real-valued optimization

This section introduces the proposed multi-agent architecture and agent interaction mechanism for the solution of single objective type of real-parameter optimization problems. As briefly mentioned above, the proposed multi-agent system contains a number of population- and trajectory-based metaheuristics that both compete and cooperate in consecutive sessions to optimize a given objective function. Architectural description of the proposed MAS is illustrated in Figure 5.1. Due to different descriptions of benchmark problems, a problem agent is prepared to read the problem files and initialize the problem parameters, such as number and ranges of variables, as described in their associated definitions. The problem agent sends the formalized description of the input problem to the Population management agent (PMA) that is responsible from all management tasks over the shared solution pool or common population. In this respect, the first task of the PMA is the initialization of the solution pool. From then on, it handles all transactions related to the solution pool and it is the only agent that can manipulate the shared population. For the purpose of population management, PMA exchanges data and messages between the strategy agent that controls and synchronizes the overall execution the proposed multi-agent system. Communication between PMA and the strategy agent include retrieval of population elements by the strategy agent when it is needed to relocate them into an activated metaheuristic procedure, whereas the strategy agent sends the results of individual metaheuristics to PMA to update the common population.

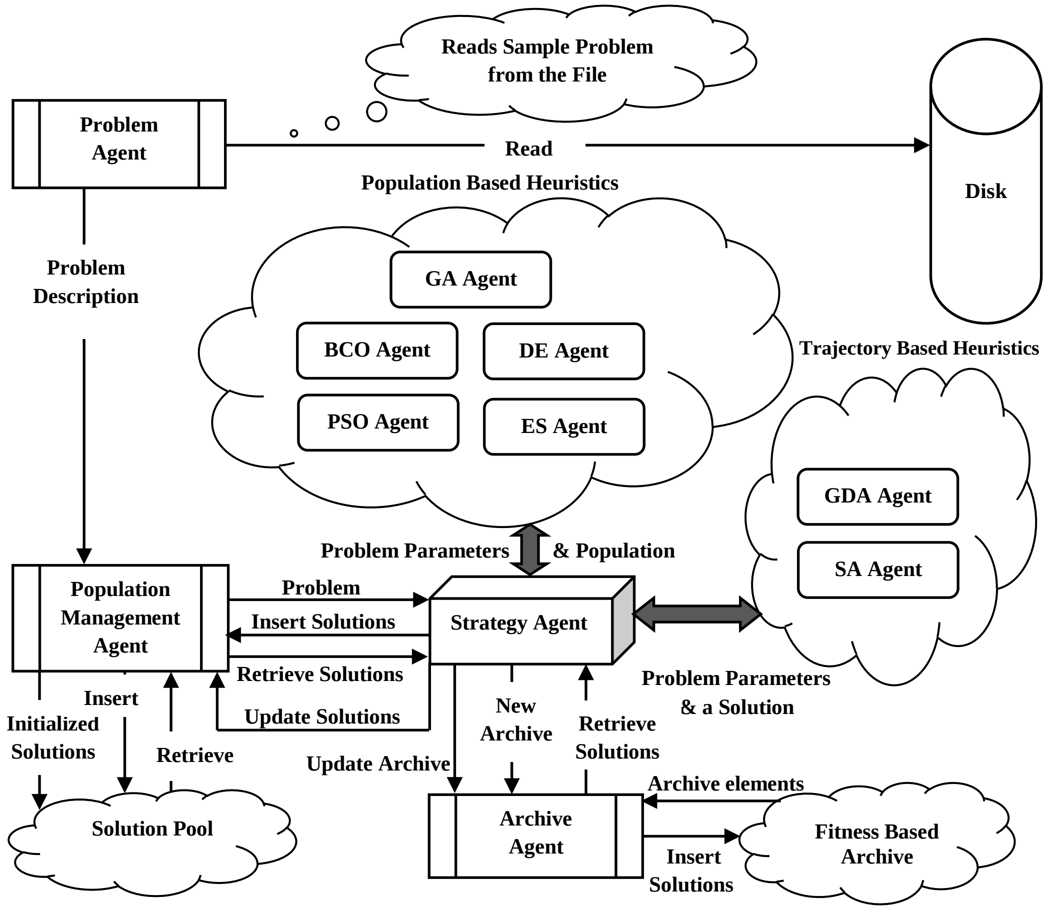


Figure 5.1. Architectural description of the proposed multi-agent system

The strategy agent is the one that interacts with all other agents in the system to organize tournament sessions, enable the winning agents and update the common population and the archive. Depending on the needs of a metaheuristic agent in action, it retrieves whole or part of the common population and sends the received solutions to the corresponding metaheuristic agent. It also communicates with the archive agent to get whole or subset of archive elements asked by a particular metaheuristic agent and to return improved solutions to archive agent for the purpose of updating the common archive. The strategy agent acts as the toolbox of the proposed multi-agent system and an algorithmic description of its functions is given in Algorithm 5.1.

Algorithm 5.1. Strategy Agent (*Problem, Fitness_evaluation_count*),

1. Initialization(*Metaheuristic*),
   i. Initialize GA (*Pop_size,Gen_size,*P_C,P_m);
   ii. Initialize DE (*Pop_size,Gen_size,*P_C,P_m);
   iii. Initialize BCO(*Bee Swarm,iteration*);
   iv. Initialize SA (*Temperature value ,Coolingate ,Terminate_condition*);
   v. Initialize GDA (*Estimated_quality,iteration*);
   vi. Initialize PSO (*Pop_size,Gen_size*);
2. Parameter initialization,
   i. *Tournament_count*=10;
   ii. *Tournament_size* = *Fitness_evaluation_count* / *Tournament_count*;
3. For *i*=1 to *Tournament_count,*
   i. *fitness-evaluation*=0;
   ii. For each metaheuristic agents,
      i. *Pop*=Retrieve solution/solutions from *Population_Management* and *archive* agent;
         i. If (metaheuristic is *PSO*),
            Global-best=*Best_Solution* in Archive;
      ii. *Previous_Fitness*= f_{obj}*(Pop);*
      iii. Run metaheuristic(*Pop*) and increase *fitness-evaluation* counter accordingly;
      iv. *Population_Management*.Update (*obtained solutions*);
      v. Calculate *improvement_rate* for metaheuristic:
         New_Fitness=f_{obj}*(Obtained_Pop);*
         Previous-best= Max(*Previous_Fitness*)*;*
         New-best=Max(*New_Fitness*);
         Improvement_rate= ((*Previous-best* - *New-best*) / *Previous-best*)*100;
      vi. *Archive_Agent*.Update(*New_Archive*)
   iii. *Winner_metaheuristic*= One with Max(*improvement_rate*);
   iv. Continue the running of *winner_metaheuristic* until *fitness-evaluation*=*Tournament_size*;
4. Return *Best_Solution* found so far;

Figure 5.2 presents the workflow and process of the algorithm used by strategy agent in CMH-MAS.

The task of the archive agent is to answer calls from the strategy agent and update the archive contents based on the results returned by the strategy agent. Archive size is fixed and its initial contents are determined upon a dialog between the PMA and the archive agent after PMA initializes the common population. Considering of individual metaheuristic agents, as shown within two clouds in Figure 5.1, there are two fundamental types, namely population- and trajectory-based, implemented in this proposed system. While trajectory-based metaheuristics work on a single solution at a time, population based metaheuristics manipulate the whole or a subset of the population at every search step.

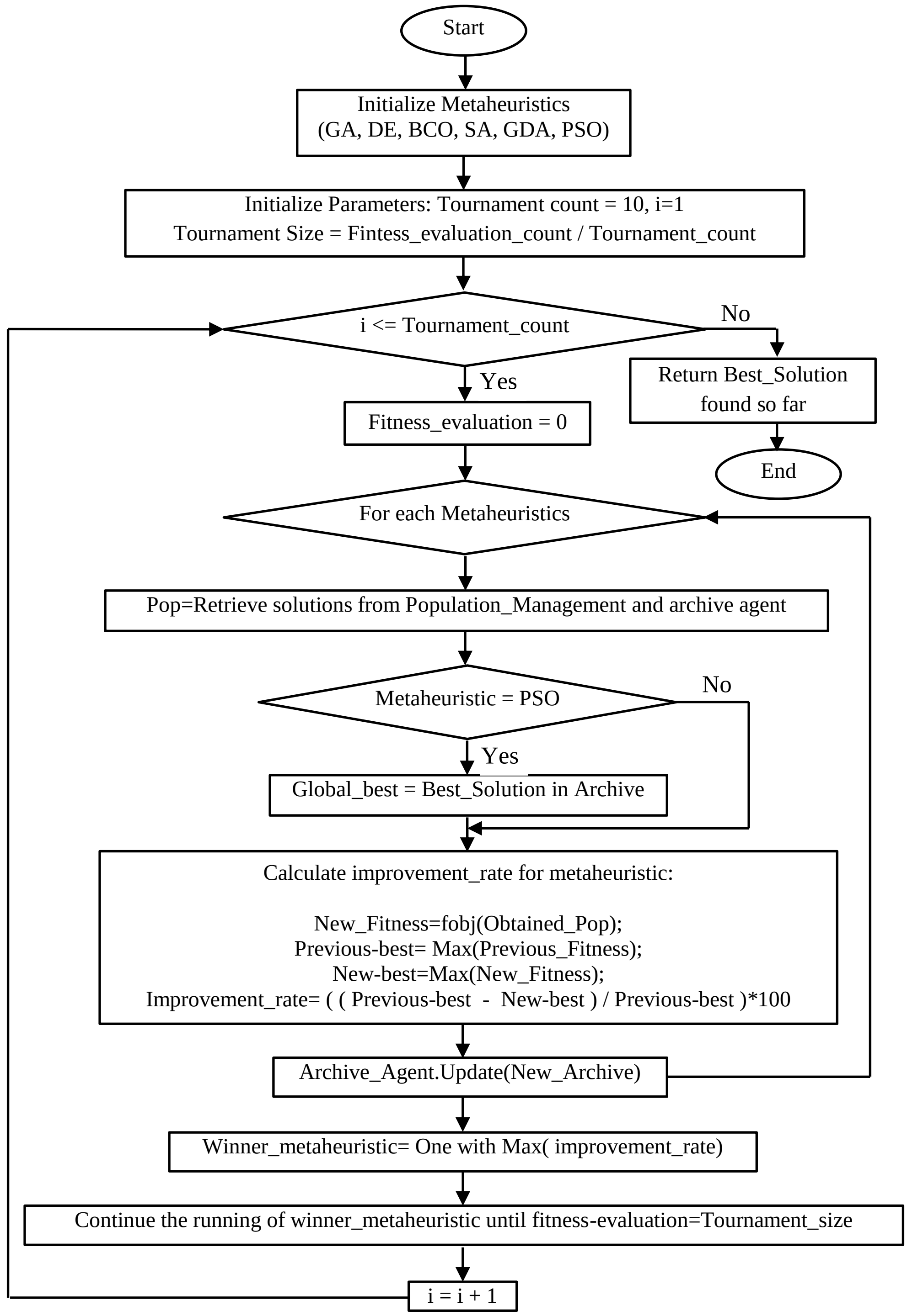


Figure 5.2. Strategy Agent for CMH-MAS

Depending on the metaheuristic agent in action, the strategy agent takes the appropriate algorithm and problem parameters, archive elements and population individuals and insert them into the skeleton of the corresponding computational procedure. It is important to note at this point that, all metaheuristic methods implemented within this proposal are in their most basic form. In fact, one of our objectives is to demonstrate that the proposed competitive and cooperative multi-agent system, composed of basic implementations of metaheuristics, outperforms almost all of the advanced state-of-the-art algorithms without bringing any significant computational complexity.

Currently, the proposed multi-agent system includes 7 metaheuristic agents, namely GA, DE, ABC, PSO, ES, GDA and SA, however the system is fully scalable since the addition or deletion of a new metaheuristic agent requires simple modifications in the strategy agent only. As mentioned in the description of metaheuristic methods, the two trajectory-based algorithms used are GDA and SA while all the others belong to the class of population based metaheuristics.

The proposed multi-agent system approach runs in consecutive sessions and each session comprises two phases: the first phase sets up a tournament among all agents to determine the currently best performing agent and the second phase lets the winner to conduct its particular search procedure until termination criteria are satisfied. In all phases and iterations of the proposed framework, all agents use the same population and archive in conducting their search procedures. This way, agents compete with each other in terms of their fitness improvements achieved over a fixed number of fitness evaluations in tournaments, and they cooperate by sharing their search experiences through accumulating them in a common population and a common archive. The strategy agent controls communication and coordination of agents' activities through monitoring the common population and the common archive. In the tournament phase, each agent performs a fixed number of iterations over the common population and gets a success score in terms the fitness improvements it achieved by itself. The agent with the best score is the winner of the tournament. Then, the winner agent is allowed to conduct its search algorithm using the common population until either its procedure gets stuck at a locally optimal or the number of generation is terminated. In both the tournament and the following computational steps, a particular metaheuristic agent sends an inquiry to the strategy agent for the delivery of algorithm parameters and population elements and it returns evolved versions of solutions back to the strategy agent. New solutions are then sent to PMA and archive agent to update the common population and archive. In both update operations, elements of common population and archive that are worse than the new solutions are replaced by the better ones. All metaheuristic agents in the proposed system use the same real-valued vector representation of solutions; therefore there is no need to convert solutions when they are exchanged between different agents of the system.

Effectiveness of the resulting multi-agent framework in solving hard real-valued optimization problems is investigated in both qualitative and statistical evaluations. Results presented in chapter 7 include both self and comparative analysis of experimental trials and, they clearly demonstrate that the objectives on the design of the proposed MAS are almost all achieved.

Chapter 6

A MULTI-AGENT, DYNAMIC RANK-DRIVEN MULTI-DEME ARCHITECTURE FOR REAL-VALUED MULTI-OBJECTIVE OPTIMIZATION

6.1 Introduction

Real-world problems often are defined over a number of objectives which usually contradict with each other [40]. In this respect, multi-objective optimization (MOO) that provides a set of solutions, presenting a number of tradeoff alternatives among the problem objectives, is of a great algorithm design challenge, particularly for engineering applications. In real parameter multi-objective optimization, an unconstraint minimization problem including m objective functions over R_n is defined as follows:

$$\begin{aligned} & \min_{X \in D} F(X) \\ & s.t.\ F(X) = (f_1(X), \ldots, f_m(X)) \\ & \quad X = (x_1, \ldots, x_n) \in R^n \\ & \quad x_i \in D_i \ and\ D = D_1 \times \ldots \times D_n \end{aligned} \qquad (6.1)$$

For a multi-objective optimization problem, a set solutions representing all the discovered tradeoffs among the problem objectives is commonly known as a Pareto-optimal set in which Pareto-optimality is defined in terms of a dominance relation between two solutions as follows: given two solutions u and v, u is said to dominate

v, $u \leq v$, if u is not worse than v in all objectives and u is better than v for at least one objective. For example, for a minimization problem, solution vector u is better than solution vector v with respect to objective i, if $fi(u) \leq fi(v)$, and u dominates v if $fi(u) \leq fi(v)$, $\forall i$ and $\exists j$ for which $fj(u) < fj(v)$. When neither u dominates v nor vice versa, we say that the two objective vectors are non-dominated. The main goal, in the solution of a MOO problem, is to obtain a Pareto global optimum set of feasible solutions such that all solutions within this set are pairwise non-dominated. As often done in literature, any set of non-dominated objective vectors is called a Pareto Front [41].

Solution approaches for multi-objective optimization problems can be put into two categories: exact solution methods and approximation algorithms. Exact methods aim to compute the complete optimal Pareto front whereas approximate solution algorithms try to extract good solutions but with no guarantee of their Pareto optimality. Exact methods such as branch-and-bound, branch-and-cut, and dynamic programming have so far been successfully applied for the solution small size problems with two objectives; however infeasible computational time requirements of exact methods are the main cause of the popularity of state-of-the approximation algorithms. Among several approximation algorithms for MOO, metaheuristic-based approaches gain high popularity due to their low computational complexity and success in extracting optimal or very close-to-optimal Pareto fronts for high-dimensional difficult problems that are either specially designed experimental benchmarks or originating from practical applications. Among numerous proposals of MOO metaheuristics, some of those that are well-known by their success are non-dominated sorting genetic algorithm (NSGA II) [42], multi-objective genetic algorithm (MOGA) [43], strength Pareto evolutionary algorithm (SPEA 2) [44], multi-objective differential evolution (MODE) [45], multi-objective simulated annealing (AMOSA) [46] and multi-objective particle swarm optimization (MOPSO) [47]. Brief descriptions of these algorithms are given in the following sections. The method proposed in this chapter implements the above mentioned MOO metaheuristics as individual agents of a multi-agent system (MAS) in which each agent acts on a subset of the common population based on the dominance ranks its elements.

A multi-agent system (MAS) is a social environment for a population of agents each of which performs a goal-oriented task on the environment using their own operators. Basically, each agent gets a set of percepts from their environment, processes the percepts under light of their accumulated knowledge, and act on the environment through their available operators to achieve a predefined design goal. MASs can be categorized as homogeneous where all agents are identical in architecture and capabilities or heterogeneous where each agent may have its own architectural components and computational procedures [2]. In either of the two categories, a multi-agent system is designed to carry out a particular task through social interaction of its agents. This social interaction is also usually of two types, namely cooperation or competition. Both of these social interactions require agents to use communication mechanisms through which they can share or exchange information. State-of-the-art literature on MASs designed for real-valued function optimization is presented in detail in sections below.

This chapter presents a novel heterogeneous multi-agent and rank-driven dynamic multi-deme architecture for the solution of multi-objective optimization. Proposed architecture contains implementations of a number MO metaheuristics as individual agents that cooperatively work on different-rank Pareto fronts for the purpose of finding the optimal comprises of the objective functions. Agent architectures are made of the data structures included in their associated metaheuristic algorithm and the search operators provided by each metaheuristic constitute the action sets of corresponding agents. There is one population which is divided into disjoint subsets based on dominance ranks of its elements. The number of subsets and their cardinality depends on level of iterations. It is clear that, the number of subsets is larger on initial iterations and cardinality of lower-rank subsets is smaller than those of higher rank subsets, and these conditions change in reverse direction as the number of iterations increase. The proposed multi-agent architecture works iteratively in sessions including two consecutive phases: in the first phase, a population of solutions is divided into subpopulations based on dominance ranks of individual solutions. In the second phase, each multi-objective metaheuristic is assigned to work on a subpopulation based on a cyclic or round-robin order. Hence, each metaheuristic operates on a different-rank subpopulation in subsequent sessions, where a session starts with a new assignment of metaheuristics and ends when termination criteria are satisfied. Individual agents have their local archives of non-dominated solutions extracted in a session, while there is a global archive keeping all non-dominated solutions found so far. At the end of each session, all subpopulations are combined into one global population to be used for the initialization of the next session. Similarly, all local archives are merged with the global archive to get the set of all non-dominated solutions found by all metaheuristics through working on subsets of different rank-levels. This way, the metaheuristics cooperate with each other by sharing their search experiences through collecting them in a common population and a common global archive. The proposed MAS is experimentally evaluated using the well-known IEEE CEC2009 benchmark problems set that includes 20 benchmark functions [48]. Comparative analysis of the experimental results demonstrated that the proposed architecture achieves better performance than majority of its state-of-the-art competitors in almost all problem instances. Description of the proposed MAS in details is given in the next Section.

In the rest of this chapter, the proposed heterogeneous, dynamic rank-driven multi deme MAS for real valued multi-objective optimization is described in detail in Section 6.2. Also, in chapter 7 description of experimental suit, test problems, algorithm parameters, results and comparative analyses in terms of quantitative and statistical computations are presented.

6.2 The Proposed Rank-Driven, Dynamic Multi-Deme and Multi-agent Architecture

This section describes the proposed multi-agent multi-deme architecture based on a novel collaboration mechanism for the solution of multi-objective real-valued optimization problems. As briefly mentioned above, the proposed system includes a number of multi-objective optimization algorithms which operate as individual agents. The multi-agent system works in consecutive sessions where each session is composed of task distribution to agents, execution of the assigned tasks and delivering the results to the associated agents to initialize and start the next session. Architectural description of the proposed method is presented in Figure 6.1. In this architecture, there is a problem agent to read formulation of the multi-objective optimization problem and to initialize the related parameters such as number of variables, variable domains and number of objectives. The problem agent sends the problem description and its parameter values to the Solution Pool Agent (SPA) which manages all the transactions associated with the shared global population. As a first task, SPA initializes the solution pool with randomly built solutions and computes their objective function values. The next operation carried out by SPA is the computation of dominance ranks of solutions in the global population. In this respect, the dominance rank of a solution s is the number of other population elements dominating s. SPA also initializes the initial order of agents by simply generating a random permutation of integers from 1 to N, where N is the number of agents in the system. In subsequent sessions, agent order is changed by rotating this permutation right by one step. Based on the retrieve message of the strategy agent, SPA sends the global population, corresponding objective function values, rank information and initial order of agents to be used for the task assignment purpose. The archive agent deals with all transactions associated with the global archive and it communicates with the strategy agent for initialization, retrieval and update operations. Upon receiving the current sets of non-dominated solutions from MOO agents through the strategy agent, archive agent unites its current contents with the received sets and eliminates those dominated solutions from this combination. The updated global archive is sent back to the strategy agent to be used as a shared resource for all agents during their executions. The heart of the proposed system is the strategy agent (SA) that communicates with all other agents and carries out task assignments, data collection, data transfer and control of all agent activities. An algorithmic description of SA's functions is presented in Algorithm 6.1.

At the beginning of each session, SA receives the global population and dominance ranks of solutions from SPA and then divides the solutions into subpopulations based on their dominance rank orders. That is, solutions having a rank of *1* form the first subpopulation and solutions having a rank of *k* form the *k-th* subpopulation.

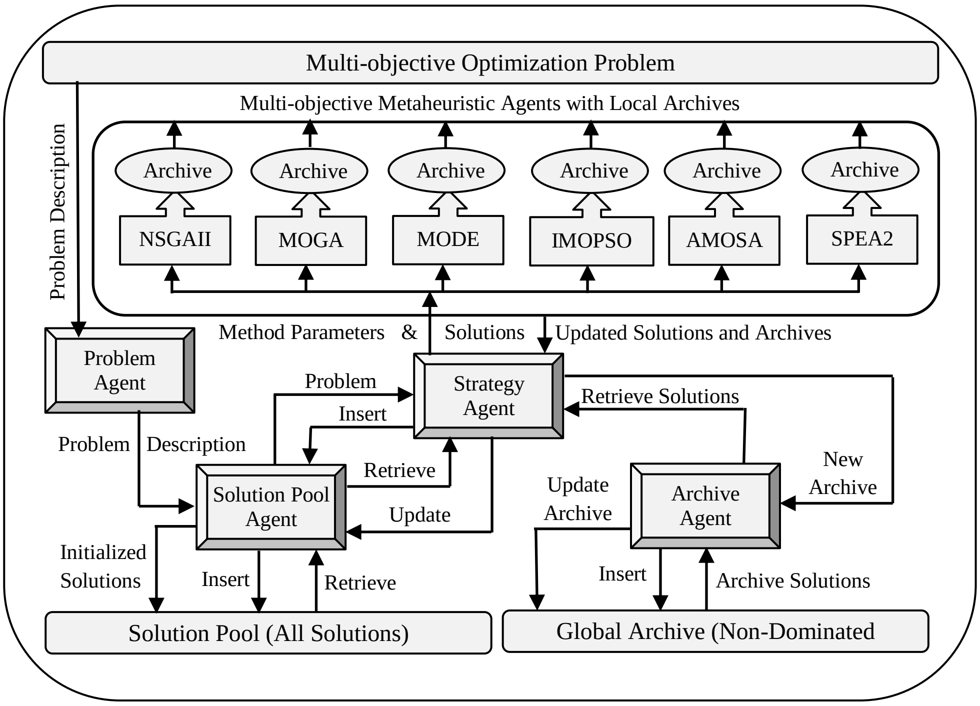


Figure 6.1. Architectural description of the proposed multi-agent system

Obviously, cardinalities of subpopulations changes from session to session and hence number of solutions within a subpopulation may be a few while many solutions may be grouped into another subpopulation. In order to balance the load of individual agents and distribute global population elements evenly over subpopulations, two variables, *Min_AgPopSize* and *Max_AgPopSize*, describing the minimum and the maximum number of solutions in each subpopulation is defined. In this respect, if the number of solutions in a subpopulation is less than *Min_PopSize*, then randomly selected solutions from higher-rank subpopulations are copied to get a cardinality of *Min_PopSize*. Similarly, if the number of elements in a subpopulation is larger than *Max_PopSize*, then randomly selected elements are removed from this subpopulation to reduce cardinality to *Max_PopSize*. Then after, SA sends each subpopulation to a

MOO agent in the order that is determined initially by SPA and updated by SA at the beginning of each session.

Algorithm 6.1. Strategy Agent(*MOO_Problem, Global_Pop, Rank_List, Agent_Set, Agent_Order*),

```
1.  If Global_Archive = ϕ,
    i.   Global_Archive = Global_Pop(Rank_List(0));        // Initialization of global archive
2.  For i=0 to Num_Agents-1,                               // Rank-driven task assignment
    i.    Agent_Pop(Agent_Order(i+1))=Global_Pop(Rank_List(i,:));
    ii.   if |Agent_Pop(Agent_Order(i+1))| < Min_Pop_Size,
    iii.      Insert randomly selected elements from higher rank subpopulations, in order, until
                  the cardinality of Agent_Pop(Agent_Order(i+1)) is more than Min_Pop_Size;
    iv.   else if |Agent_Pop(Agent_Order(i+1))| > Max_Pop_Size,
    v.        Remove randomly selected (|Agent_Pop(Agent_Order(i+1))|- Max_Pop_Size) agents
              from Agent_Pop(Agent_Order(i+1)).
3.  For i=1 to Num_Agents,                                 // Initialize individual agents
    i.   Initialize_Agents(Agent_Set(i), Agent_Parameters(Agent_Set(i)),Agent_Pop(Agent_Order(i)));
4.  For i=1 to Num_Agents,                                 // Start  running each individual agent
    i.    [Local_Archive(Agent_Set(i)),Local_Pop(Agent_Set(i))] =
                      Agent_Set(i, Agent__Pop(Agent_Order(i),Session_Fevals);
5.  Return Union(Local_Archive) to Archive Agent and Union(Local_Pop) to Solution Pool Agent;
```

Figure 6.2 presents the workflow and process of the algorithm used by strategy agent in RdMD-MAS.

Each individual MOO agent in the system is implementation of a particular metaheuristic. Agents have their own local populations and archives keeping the solutions extracted with their own search strategy and the extracted non-dominated solutions, respectively. It should be noted at this point that, all multi-objective metaheuristic methods implemented in this thesis are in their most basic form. Indeed, composing of basic implementations of metaheuristics without any additional computational complexity and showing that resulting composition is competitive to state-of-the-art modern algorithms is one of our objectives. After receiving the task assignment message from SA, each MOO agent runs its underlying search mechanism and sends the improved population elements and the local archive contents back to SA when the termination criteria are satisfied.

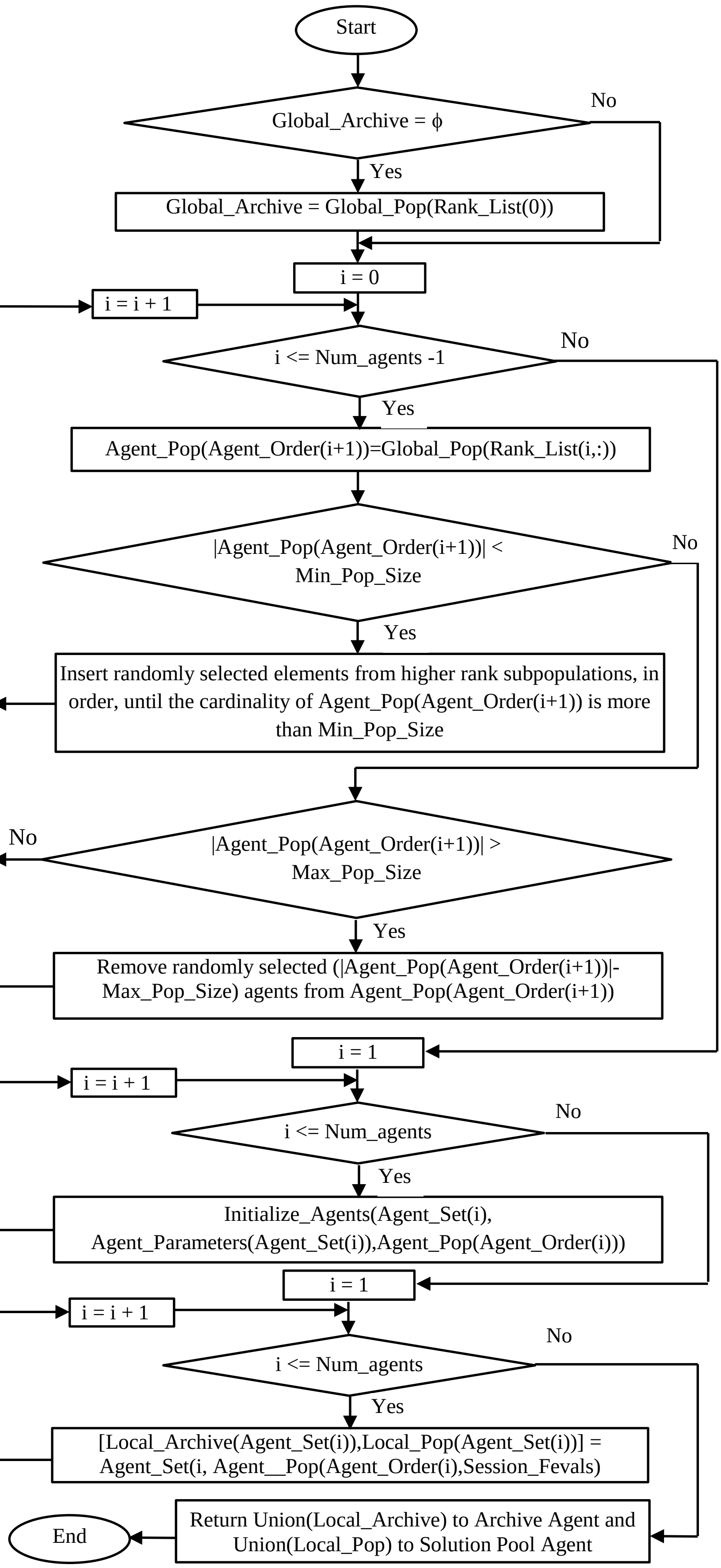


Figure 6.2. Strategy Agent in RdMD/MAS

SA sends union of local populations to SPA and union of local archives to archive agent to update global population and global archive. After then, a new session is started as explained above.

Currently, the proposed multi-agent system comprises six metaheuristic agents, namely MOGA, NSGAII, SPEA2, MODE, MOPSO and AMOSA. However the system is fully scalable to add a new multi-objective metaheuristic or delete an existing one. All metaheuristic agents in the proposed system use the same real-valued vector representation of solutions; therefore there is no need to convert solutions when they are exchanged between different agents of the system. Effectiveness of the resulting multi-agent framework in solving hard real-valued optimization problems is investigated in the next section. Results presented in chapter 7 clearly demonstrate that the objectives on the design of the proposed multi-agent system are almost all achieved.

Chapter 7

EXPERIMENTAL RESULTS AND EVALUATIONS

7.1 Evaluation of learning-based multi-agent system for solving combinatorial optimization problems

This section presents experimental evaluation of the proposed method for the solution of multiprocessor scheduling problem that is a hard combinatorial optimization problem (MSP) [49, 50].

MSP is represented as a directed acyclic graph (DAG) consisting of a set of vertices and a set of directed edges between the vertices. Vertices demonstrate the parallel code partitions as tasks in which each task has its own execution time. Meanwhile, each directed edge indicates the execution order and the required time to make communication between tasks. This problem is aiming to schedule a DAG to a set of homogeneous fully connected processors. The objective is to find an optimal scheduling with minimum total completion time to run the task graph on multiprocessors [49, 50]. Figure 7.1 represents a sample task graph representing a particular MSP [49]. Entry and finish points of this task graph are *t0* and *t18* respectively.

Solutions for MSP problem can be represented using simple data structure like Arrays. Figure 7.2 illustrates a sample solution for the task graph in Figure 7.1 [51]. In this representation, processors are assigned to tasks in which is assigned to.

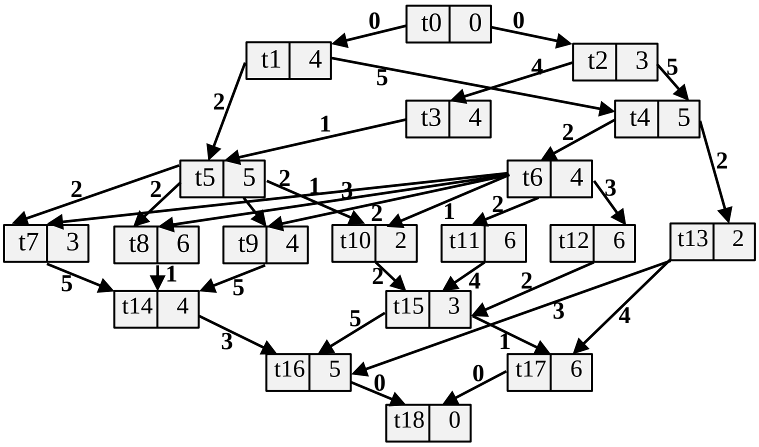


Figure 7.1. A sample task graph representing a particular MSP [16]

Meanwhile, the tasks order should be feasible in the sense that for generating feasible solutions, they are chosen randomly among the tasks which are ready to be executed. Once a task is finished, its successors which don't have other unfinished predecessors can be added to the ready list.

t_0	t_2	t_3	t_1	t_4	t_5	t_6	t_{11}	t_{13}	t_{12}	t_9	t_{10}	t_8	t_7	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}
p_0	p_1	p_2	p_0	p_1	p_2	p_1	p_1	p_1	p_1	p_0	p_0	p_0	p_2	p_0	p_1	p_0	p_1	p_1

Figure 7.2. Solution representation for task graph in Figure 7.1

Algorithmic parameters for metaheuristics used within the proposed multi-agent system are given in Table 7.1 It should be noticed that, even though the metaheuristics within the multi-agent system are executed several times, they are run in small population size to reduce total computation time.

Table 7.2 presents completion time of the MSP mentioned in figure 7.1 using LBMAS and 6 existent deterministic methods. The computed result is compared to well-known deterministic methods including MCP [52], LAST [53], HLFET [54], ETF [55], EZ [56] and LC [57].

Table 7.1. Algorithmic parameters for metaheuristics

Metaheuristic Agent	Algorithm Parameters
GA	|Pop|= 50, PC= 0.7, Pm= 0.1, Selection_method: Tournament Selection
ACO	|Pop|=50, Decay-Factor= 0.1, Heuristic-Coefficient= 2.5, Local-Pheromone-Factor= 0.1, Greediness-Factor= 0.9
DE	|Pop|=50, PC= 0.8, Pm =0.2, CR=0.7, F= 1.0
CE	Learning-Rate= 0.7
TS	Stopping-criteria= 200 iterations without solutioan change.
SA	T0=150, α=0.2, Tmin=0.1
GDA	Level= Fitness(Initial-sol), No-Improvement-Length limit= Level-Decay= (Fitnesss(Initial-sol) - Estimated_Best) / Max Iterations

Table 7.2. Completion time of task graph shown in Figure. 6 for all algorithms

Algorithm	LC	EZ	HLFET	ETF	LAST	MCP	LBMAS
Completion Time	39	40	41	41	43	40	39

Figure 7.3 presents the visual comparison of LBMAS to its competitors to provide a better quantitative evaluation.

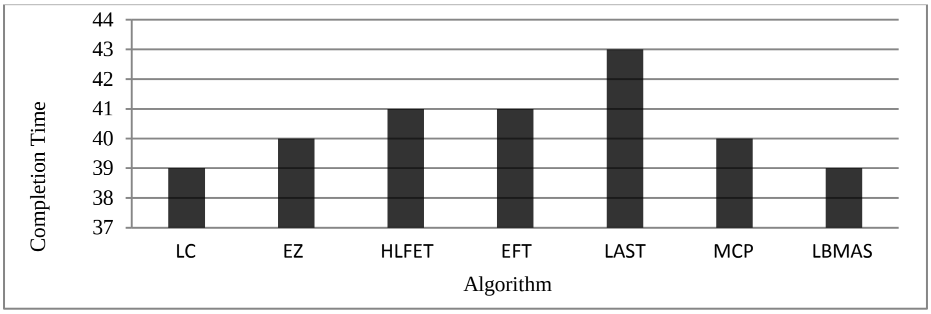
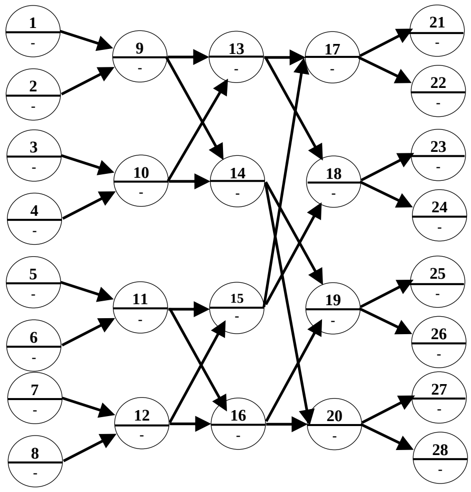


Figure 7.3. Comparison of LBMAS to other deterministic algorithms

It can be seen that, the completion time found by LBMS is 39. That means that the total running time of task graph shown in Figure 7.1 over 3 multiprocessors system is 39 units. It is clearly seen that LBMAS produced better scheduling than most of its competing algorithms. In particular, identical results are obtained with LC, however, LC assumes that the number of processors is unlimited, whereas LBMAS assumes only 3 processors.

We continue the evaluation of LBMAS over two other useful benchmarks of MSP, namely Fast Fourier Transformation (FFT) and Internal Rate of Return (IRR) [58]. FFT graph has three types of edge weights, so we deal with three problems FFT1, FFT2 and FFT4. Figure 7.4 presents the FFT and IRR task graphs [58].



Vertex#	FFT-1	FFT-2	FFT-4
1 – 8	1	60	20
9 – 12	20	50	20
13 – 16	30	5	30
17 – 20	20	5	20
21 - 28	1	5	5

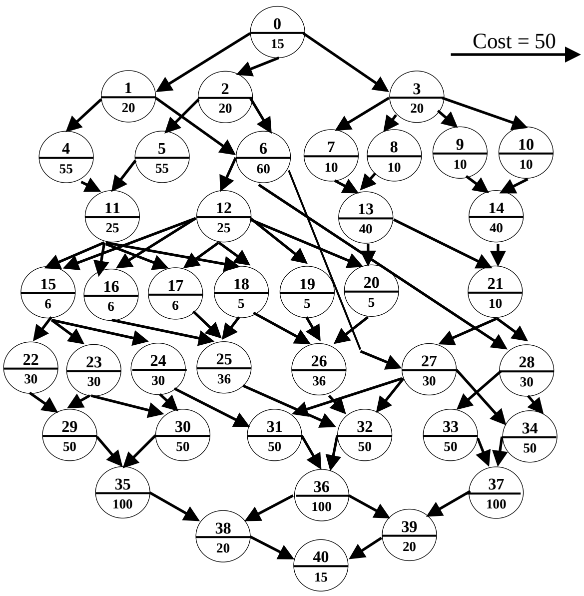


Figure 7.4. FFT (Up) and IRR (Down) task graphs [58]

Table 7.3 shows the experimental results of LBMAS compared to two existing remarkable evolutionary methods, namely BCGA [59], CGL [60], and the MCP algorithm. LBMAS generated better scheduling for FFT and IRR graphs. In this experiment the number of processors is assumed to be four.

Table 7.3. Completion time of applying MCP,CGL, BSGA and LBMAS on FFT and IRR graphs

Graph	Serial Time	Nodes #	Edges #	MCP	CGL	BCGA	LBMAS
FFT1	296	28	32	148	152	124	124
FFT2	760	28	32	205	270	240	193
FFT4	480	28	32	710	260	255	195
IRR	1330	41	69	600	600	580	475

Also, below in Table 7.4 we compare LBMAS to three more competing algorithms, namely DLS [61], MH [62] and SES [63].

Table 7.4. Completion time of applying DLS, MH, SES and LBMAS on FFT and IRR graphs

Graph	Serial Time	Nodes #	Edges #	DLS	MH	SES	LBMAS
FFT1	296	28	32	175	175	173	124
FFT2	760	28	32	275	280	255	193
IRR	1330	41	69	600	710	650	475

In Table 7.3 and 7.4, LBMAS is evaluated upon four task graphs with certain number of nodes (Tasks) and edges. Also, the serial running time of these graphs on a single processor are given in the tables. It can be seen that, the completion time of scheduling discovered by LBMAS for FFT1 is *124* which is equal to BCGA and better than others. Also, LBMAS achieves better completion time for FFT2, FFT4 and IRR in comparison to all competitors. In other words, LBMAS is able to find a scheduling of IRR on a set of four processors with completion time of *475*, while the total completion times found by MCP, CGL, BCGA, DLS, MH and SES are *600, 600, 580*, *600*, *710* and *650* respectively.

Figure 7.5 shows the improvement rate values for the problems FFT4 (up) and IRR (down) adjusted by LBMAS during the execution. According to the Figure 7.5, TS and GA metaheuristics have larger values as improvement rates for FFT4 and IRR respectively, means that their chance to be selected is more than others. Five metaheuristics are applied for solving multiprocessor Scheduling Problem. Improvement rate cannot be lower than 10, in order to give small chance to worst metaheuristics to be selected. This way, Roulette Wheel Selection mechanism will not cause damaging convergence.

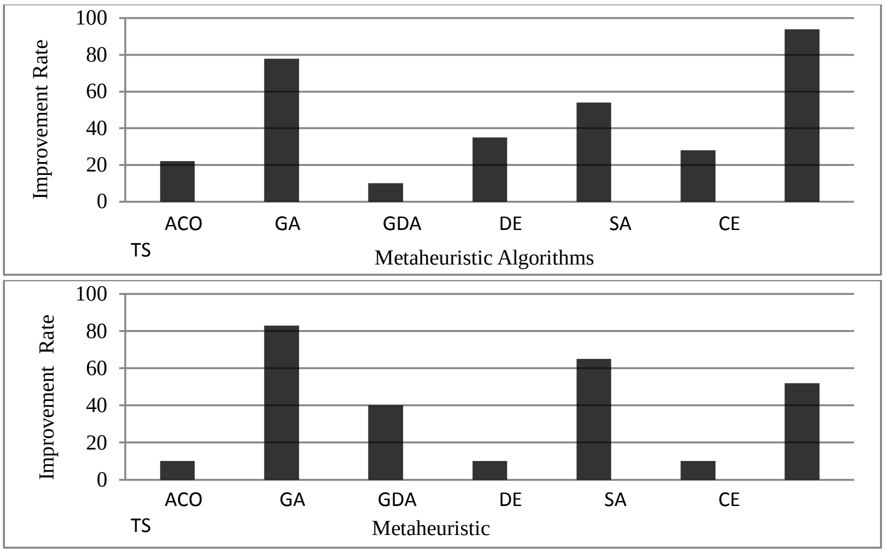


Figure 7.5. Improvement rate values for FFT4 (Up) and IRR (Down)

As mentioned in previous sections, in LBMAS, metaheuristics are run several times according to their improvement rates. Metaheuristics are executed in small sizes, because they are supposed to run more times. This way, multi-agent system will be quick without any time complexity problems. Figure 7.6 below shows the reliability of LBMAS and demonstrates that our LBMAS obtains almost same results in 20 different runs for FFT4 graph. In this figure, the vertical axis values shows the completion time of FFT4 graph and the horizontal values indicates the run number which is totally 20 independent runs. It can be seen that in 12 runs out of 20 different runs, the system reaches to *195* and in other 8 runs the obtained value is very close to *195*. Therefore, the system is reliable without any outstanding fluctuation.

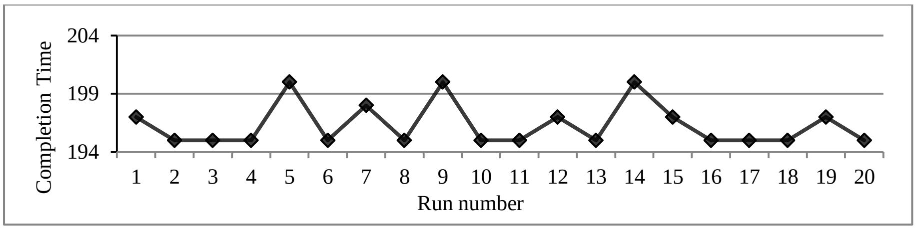


Figure 7.6. Reliability of LBMAS in 20 different runs

Finally, below in Figure 7.7, the evolution of solutions during applying metaheuristics on IRR graph is illustrated. It can be seen that the completion time of the best solution is reducing until the *475* is reached. In this figure, the vertical axis values present the completion time of IRR and the horizontal axis values show the sample number in which the total number of samples is 80. This figure shows that in the early samples the speed of evolution is outstanding and then it is gradually converged to 475.

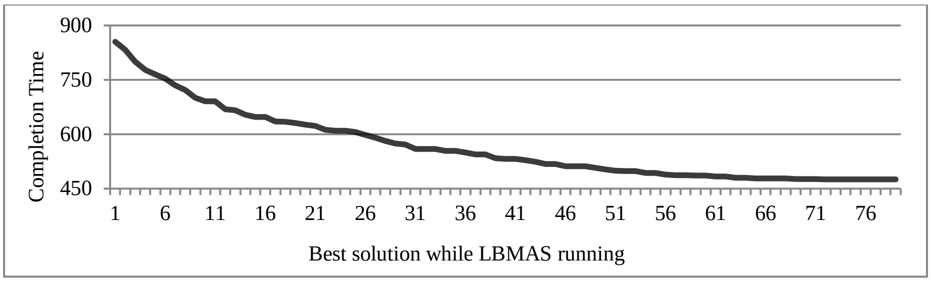


Figure 7.7. Evolution of solutions

7.2 Evaluation of Tournament-Based Competitive-Cooperative Multi-agent Architecture for Real Parameter Optimization

Performance evaluation of the proposed algorithm and exhibition of its comparative success against state-of-the art metaheuristics are carried out over the difficult problems in CEC2005 benchmarks [34]. Details of these benchmark functions could not be given here due to space limitations, but definitions, categorization and fitness landscape characteristics of all these functions are described clearly the reference given above. For each of the test functions, the number of independent runs and the termination criterion, in terms of the number of fitness evaluations, are set the same as the ones used in the corresponding reference so that fairness is guaranteed for comparative evaluations. Algorithmic parameters of the proposed method are kept the same for all the test functions and no interactive intervention is made throughout the program executions. Additionally, number of variables for the test functions is also taken the same as the ones specified in the corresponding references.

Algorithmic parameters of the metaheuristic methods used within the proposed multi-agent system are given in Table 7.5. For the five population-based methods, population size is set as 100, whereas the two trajectory based algorithms start from a single solution when each time they are activated. All of the parameters in Table 1 are collected from well-known conventional implementations of the corresponding metaheuristic algorithms. Implementation of the proposed system is carried out using Matlab® programming language environment and a personal PC with 8 GB main memory and 2.1 GHz clock speed. The precision for the floating-point operations is set to 15 fractional digits.

Table 7.5. Algorithmic parameters of the metaheuristic methods used within the proposed system

Metaheuristic Agent	Algorithm Parameters
GA	|Pop| =100, P_C =0.7, P_m=0.1, Selection method: Roulette wheel
PSO	|Swarm| =100, ω=0.8, C_1=2.0, C2=2.0
DE	|Pop| =100, P_C =0.8, P_m =0.2, CR=0.7, F= 1.0
ABC	|B| =100, Num. Scouts=|B|, Trials Limit=10.
ES	|Pop| (μ) = λ /2, λ = 4+[3ln n], ρ=0.3
SA	T_0=300, α=0.2, T_{min}=0.1
GDA	Level= f_{obj} (Initial sol.), β=(f_{obj}(Initial sol.) - Estimated_Best) / Max. Iterations NILength=100

As mentioned above, the set of benchmark problems on which performance of the proposed multi-agent system, named as CMH-MAS from this point on, is comparatively evaluated is the CEC2005 problem set for numerical global optimization. There are 25 problems within set, they are mostly generated from the set of classical benchmarks through random shifting, random shifting and rotation, and hybrid composition operations. Accordingly, there are 5 unimodal and 20 multimodal benchmark problems in this set and the set of multimodal problems include 2 expanded and 11 hybrid composition problems. As illustrated in [34], some of these functions have multimodal fluctuating landscapes that are hard for many well-known metaheuristics. Detailed description of the problems within this set and the experimental conditions under which the runs are performed are explained in [34]. Accordingly, all results are averages over 25 runs and maximum number of fitness evaluations is set to *Problem_Size*1.0e+4*. Problem sizes (number of variables) for each benchmark problem instance is set as 10, 30 and 50, that means 1875 runs (625 runs for each problem size) performed totally for the 75 problem instances. Results of other algorithms that are already used to solve these problems are downloaded from [64] and average results of CMH-MAS are compared to these average fitness values over 25 runs for all problems.

Table 7.6 illustrates the results of 11 algorithms which attended the CEC2005 contest and CMH-MAS for problem size of 10 variables. It can be seen that CMH-MAS is the best performing algorithm for 11 of the 25 problems and it shares the first position with its competitors for other 4 problems. Particularly, for some of the benchmark problems the average fitness score of the proposed method is much better than its competitors. Considering the performance of CMH-MAS for unimodal and multimodal problem instances, one can see that CMH-MAS has achieved the first position for problems of both types. This shows that the proposed system is capable locating good solutions over both single- or multi-modal fitness landscapes. Over a number of problem instances, CMH-MAS does not take the first place among its competitors, however for these 10 instances the proposed approach took the second or the third position for 4 *(F13, F14, F15, and F21),* the fifth position for position 2 (*F16* and *F22*), the sixth position for 2 (*F10* and *F17*) and the seventh position 2 (*F9* and *F19*) problems. In summary, among 12 competitors CMH-MAS took the either fifth, seventh, or the seventh position for 6 problems from a set of 25 benchmarks.

Table 7.6. Average fitness values of all algorithms used to solve CEC2005 benchmarks for D = 10

Algorithm	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F9**	**F10**	**F11**	**F12**	**F13**
BLX-GL50 [30]	1.00E-009	1.00E-009	5.71E+002	1.00E-009	1.00E-009	1.00E-009	1.17E-002	2.04E+001	1.15E+000	4.97E+000	2.33E+000	4.07E+002	7.50E-001
BLX-MA [31]	1.00E-009	1.00E-009	4.77E+004	2.00E-008	2.12E-002	1.49E+000	1.97E-001	2.02E+001	4.38E-001	5.64E+000	4.56E+000	7.43E+001	7.74E-001
COEVO [32]	1.00E-009	1.00E-009	1.00E-009	1.00E-009	2.13E+000	1.25E+001	3.71E-002	2.03E+001	1.92E+001	2.68E+001	9.03E+000	6.05E+002	1.14E+000
DE [33]	1.00E-009	1.00E-009	1.94E-006	1.00E-009	1.00E-009	1.59E-001	1.46E-001	2.04E+001	9.55E-001	1.25E+001	8.47E-001	3.17E+001	9.77E-001
DMS-L-PSO [34]	1.00E-009	1.00E-009	1.00E-009	1.89E-003	1.14E-006	6.89E-008	4.52E-002	2.00E+001	**1.00E-009**	3.62E+000	4.62E+000	2.40E+000	3.69E-001
EDA [35]	1.00E-009	1.00E-009	2.12E+001	1.00E-009	1.00E-009	4.18E-002	4.20E-001	2.03E+001	5.42E+000	5.29E+000	3.94E+000	4.42E+002	1.84E+000
G-CMA-ES [36]	1.00E-009	1.00E-009	1.00E-009	1.00E-009	1.00E-009	1.00E-009	1.00E-009	2.00E+001	2.39E-001	7.96E-002	9.34E-001	2.93E+001	6.96E-001
K-PCX [37]	1.00E-009	1.00E-009	4.15E-001	7.94E-007	4.85E+001	4.78E-001	2.31E-001	2.00E+001	1.19E-001	2.39E-001	6.65E+000	1.49E+002	6.53E-001
L-CMA-ES [38]	1.00E-009	1.00E-009	1.00E-009	1.76E+006	1.00E-009	1.00E-009	1.00E-009	2.00E+001	4.49E+001	4.08E+001	3.65E+000	2.09E+002	4.94E-001
L-SADE [39]	1.00E-009	1.00E-009	1.67E-005	1.42E-005	1.23E-002	1.20E-008	1.99E-002	2.00E+001	**1.00E-009**	**4.97E+000**	4.89E+000	4.50E-007	**2.20E-001**
SPC-PNX [40]	1.00E-009	1.00E-009	1.08E+005	1.00E-009	1.00E-009	1.89E+001	8.26E-002	2.10E+001	4.02E+000	7.30E+000	**1.91E+000**	2.60E+002	8.38E-001
CMH-MAS	**3.78E-010**	**1.33E-010**	**2.61E-010**	**4.55E-010**	**1.08E-010**	**6.60E-011**	**1.01E-010**	**2.00E+001**	1.28E+000	9.94E+000	7.54E-001	**3.16E-009**	2.40E-001

F14	F15	F16	F17	F18	F19	F20
2.17E+000	4.00E+002	9.35E+001	1.09E+002	4.20E+002	4.49E+002	4.46E+002
2.03E+000	2.70E+002	1.02E+002	1.27E+002	8.03E+002	7.63E+002	8.00E+002
3.71E+000	2.94E+002	1.77E+002	2.12E+002	9.02E+002	8.45E+002	8.63E+002
3.45E+000	2.59E+002	1.13E+002	1.15E+002	4.00E+002	4.20E+002	4.60E+002
2.36E+000	4.85E+000	9.48E+001	1.10E+002	7.61E+002	7.14E+002	8.22E+002
2.63E+000	3.65E+002	1.44E+002	1.57E+002	4.83E+002	5.64E+002	6.52E+002
3.01E+000	2.28E+002	**9.13E+001**	1.23E+002	3.32E+002	**3.26E+002**	**3.00E+002**
2.35E+000	5.10E+002	9.59E+001	**9.73E+001**	7.52E+002	7.51E+002	8.13E+002
4.01E+000	2.11E+002	1.05E+002	5.49E+002	4.97E+002	5.16E+002	4.42E+002
2.92E+000	**3.20E+001**	1.01E+002	1.14E+002	7.19E+002	7.05E+002	7.13E+002
3.05E+000	2.54E+002	1.10E+002	1.19E+002	4.40E+002	3.80E+002	4.40E+002
2.36E+000	1.35E+002	1.01E+002	1.18E+002	**3.00E+002**	7.26E+002	4.18E+002

F21	F22	F23	F24	F25
6.89E+002	7.59E+002	6.39E+002	2.00E+002	4.04E+002
7.22E+002	6.71E+002	9.27E+002	2.24E+002	3.96E+002
6.35E+002	7.79E+002	8.35E+002	3.14E+002	2.57E+002
4.92E+002	7.18E+002	5.72E+002	2.00E+002	9.23E+002
5.36E+002	6.92E+002	7.30E+002	2.24E+002	3.66E+002
4.84E+002	7.71E+002	6.41E+002	2.00E+002	3.73E+002
5.00E+002	7.29E+002	**5.59E+002**	2.00E+002	3.74E+002
1.05E+003	6.59E+002	1.06E+003	4.06E+002	4.06E+002
4.04E+002	7.40E+002	7.91E+002	8.65E+002	4.42E+002
4.64E+002	7.35E+002	6.64E+002	2.00E+002	3.76E+002
6.80E+002	7.49E+002	5.76E+002	2.00E+002	4.06E+002
4.80E+002	**6.62E+002**	5.59E+002	**2.00E+002**	**2.00E+002**

For problem instances of size 30, there are 8 algorithms attended CEC2005 real-valued optimization contest and Table 7.7 illustrates the average fitness values these 8 algorithms and CMH-MAS. For this set larger size problems CMH-MAS extracted the best solutions and took the first place for 16 of 25 problems. The proposed system performed significantly better than its competitors in terms of solution quality also. For the remaining 9 instances CMH-MAS took the second position for 2 (*F14* and *F21*), the *F16*) problems. It can be seen that, even for these 9 problems, CMH-MAS is better than majority of its competitors for more than half of the instances. Furthermore, it is still the case that the proposed system performed equivalently well for both unimodal and multimodal problems.

Experimental results associated with problem size D=50 are presented in Table 7.7. Two powerful competitors for CMH-MAS for this problem size are implementations based on the evolution strategies with covariance matrix adaptation. Against these two powerful competitors, CMH-MSA achieved the best average fitness scores for 17 of the 25 benchmark problems. The proposed systems took the third position for only three problems, namely *F5, F11,* and *F15*. A comparison of results on Tables 7.6, 7.7, and 7.8 clearly indicates that the proposed system exhibits almost the same level of success for problem sizes 10, 30, and 50. Hence, it can be claimed that the proposed system is scalable.

Table 7.7. Average fitness values of all algorithms used to solve CEC2005 benchmarks for D = 30.

Algorithm	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**
BLX-GL50	1.00E-009	1.00E-009	3.11E+003	1.68E+001	3.33E+002	2.60E-007
BLX-MA	1.00E-009	8.72E-006	8.77E+005	3.97E+001	2.18E+003	4.95E+001
COEVO	7.97E-001	4.40E-001	3.67E+002	4.80E+003	8.34E+003	1.21E+003
DE	1.00E-009	3.33E-002	6.92E+005	1.52E+001	1.70E+002	2.51E+001
G-CMA-ES	1.00E-009	1.00E-009	1.00E-009	1.11E+004	**1.00E-009**	1.00E-009
K-PCX	1.00E-009	1.00E-009	5.79E+001	1.11E+003	2.04E+003	1.75E+000
L-CMA-ES	1.00E-009	1.00E-009	1.00E-009	9.26E+007	1.00E-009	1.00E-009
SPC-PNX	1.00E-009	6.95E-007	1.10E+006	**8.13E-007**	4.24E+003	1.52E+001
CMH-MAS	**9.93E-011**	**1.85E-010**	**2.50E-010**	1.85E+003	3.60E+001	**1.29E-010**

Algorithm	**F7**	**F8**	**F9**	**F10**	**F11**	**F12**	**F13**
BLX-GL50	1.00E-009	2.09E+001	1.51E+001	3.52E+001	2.47E+001	9.52E+003	5.15E+000
BLX-MA	1.33E-002	2.07E+001	6.81E-001	9.06E+001	3.11E+001	4.39E+003	3.96E+000
COEVO	1.41E-001	2.09E+001	1.31E+002	2.32E+002	3.77E+001	1.01E+005	9.02E+000
DE	2.96E-003	2.10E+001	1.85E+001	9.69E+001	3.42E+001	2.75E+003	3.23E+000
G-CMA-ES	1.00E-009	2.01E+001	9.38E-001	1.65E+000	5.48E+000	4.43E+004	2.49E+000
K-PCX	1.50E-002	**2.00E+001**	**2.79E-001**	**5.17E-001**	2.95E+001	1.68E+003	1.19E+001
L-CMA-ES	1.00E-009	**2.00E+001**	2.91E+002	5.63E+002	1.52E+001	1.32E+004	2.32E+000
SPC-PNX	1.46E-002	2.09E+001	2.39E+001	6.03E+001	1.13E+001	1.31E+004	3.59E+000
CMH-MAS	**1.58E-010**	2.09E+001	1.12E+001	4.63E+001	**3.76E+000**	**3.85E+002**	**1.83E+000**

Algorithm	**F14**	**F15**	**F16**	**F17**	**F18**	**F19**	**F20**
BLX-GL50	**1.21E+001**	3.04E+002	8.87E+001	1.35E+002	9.04E+002	9.04E+002	9.03E+002
BLX-MA	1.26E+001	3.56E+002	3.26E+002	2.79E+002	8.78E+002	8.80E+002	8.79E+002
COEVO	1.32E+001	4.11E+002	3.81E+002	4.54E+002	1.06E+003	1.05E+003	1.06E+003
DE	1.34E+001	3.60E+002	2.12E+002	2.37E+002	9.04E+002	9.04E+002	9.04E+002
G-CMA-ES	1.29E+001	2.08E+002	**3.50E+001**	2.91E+002	9.04E+002	9.04E+002	9.04E+002
K-PCX	1.38E+001	8.76E+002	7.15E+001	1.56E+002	8.30E+002	8.31E+002	8.31E+002
L-CMA-ES	1.40E+001	2.16E+002	5.84E+001	1.07E+003	8.90E+002	9.03E+002	8.89E+002
SPC-PNX	1.31E+001	3.68E+002	7.47E+001	8.54E+001	9.05E+002	9.05E+002	9.05E+002
CMH-MAS	1.25E+001	**2.21E+001**	1.48E+002	**8.37E+001**	**8.15E+002**	**8.26E+002**	**8.16E+002**

F21	F22	F23	F24	F25
5.00E+002	8.74E+002	5.87E+002	8.77E+002	2.11E+002
5.00E+002	9.08E+002	5.59E+002	**2.00E+002**	2.11E+002
6.04E+002	1.16E+003	9.22E+002	1.10E+003	1.03E+003
5.00E+002	8.97E+002	5.34E+002	2.00E+002	7.30E+002
5.00E+002	8.03E+002	5.34E+002	9.10E+002	2.11E+002
8.59E+002	1.56E+003	8.66E+002	2.13E+002	2.13E+002
4.85E+002	8.71E+002	5.35E+002	1.41E+003	6.91E+002
5.00E+002	8.81E+002	5.34E+002	2.00E+002	2.13E+002
5.00E+002	**5.06E+002**	**5.34E+002**	2.11E+002	**2.10E+002**

Table 7.8: Average fitness values of all algorithms used to solve CEC2005 benchmarks for D = 50

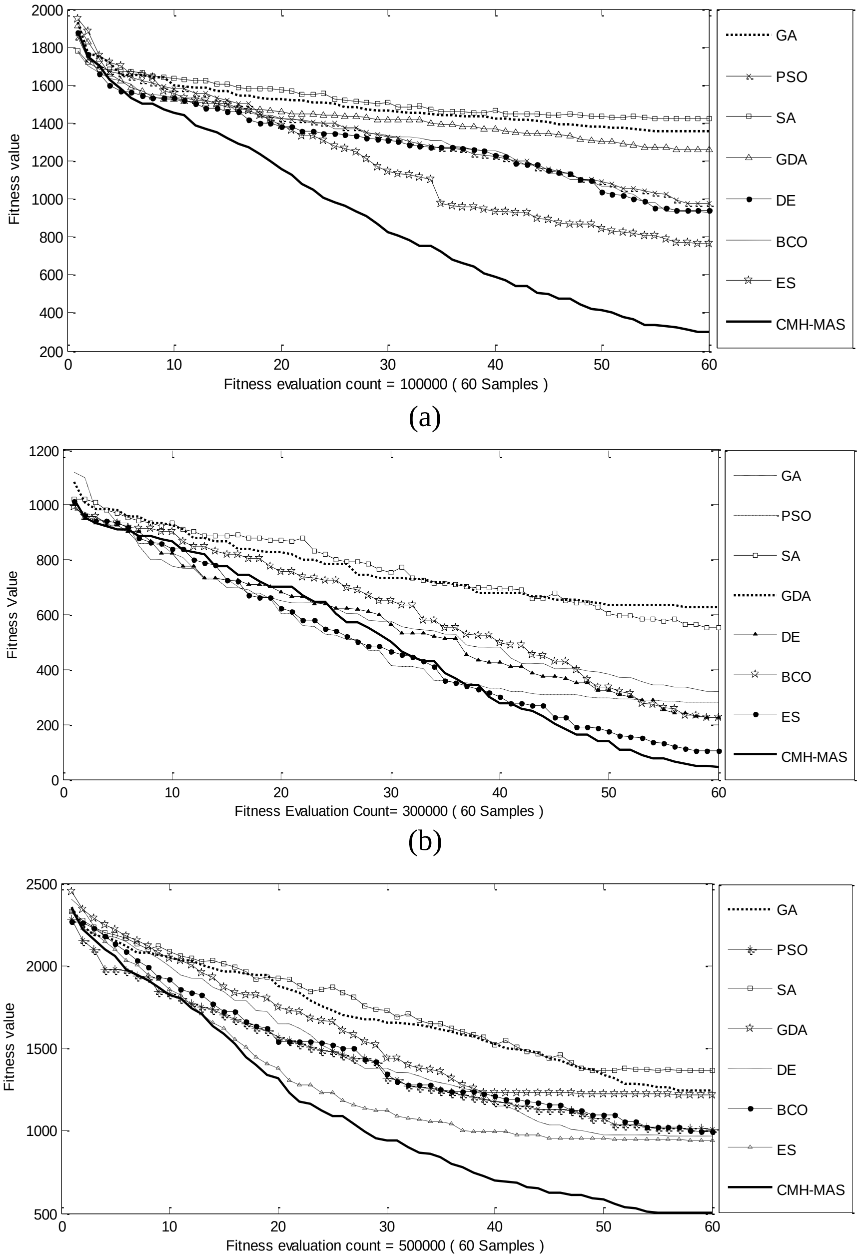
Algorithm	F1	F2	F3	F4	F5	F6
G-CMA-ES	1.00E-009	1.00E-009	1.00E-009	4.68E+005	**2.85E+000**	1.00E-009
L-CMA-ES	1.00E-009	1.00E-009	1.00E-009	4.46E+008	3.27E+000	1.00E-009
CMH-MAS	**1.28E-010**	**1.48E-010**	**2.03E-010**	**8.91E+004**	5.12E+002	**3.03E-010**

F7	F8	F9	F10	F11	F12	F13
1.00E-009	2.01E+001	**1.39E+000**	**1.72E+000**	**1.17E+001**	2.27E+005	4.59E+000
1.00E-009	**2.00E+001**	5.67E+002	1.48E+003	3.41E+001	8.93E+004	4.70E+000
1.77E-010	2.08E+001	1.35E+002	8.85E+001	5.32E+001	**9.85E+002**	**3.85E+000**

F14	F15	F16	F17	F18	F19	F20
2.29E+001	**2.04E+002**	**3.09E+001**	2.34E+002	9.13E+002	9.12E+002	9.12E+002
2.39E+001	2.50E+002	7.09E+001	**1.05E+003**	9.06E+002	9.11E+002	9.01E+002
2.14E+001	5.35E+002	6.53E+001	1.16E+002	**8.38E+002**	**8.36E+002**	**8.37E+002**

F21	F22	F23	F24	F25
1.00E+003	8.05E+002	1.01E+003	9.55E+002	2.15E+002
5.00E+002	9.10E+002	**6.37E+002**	8.43E+002	4.77E+002
7.20E+002	**5.01E+002**	7.23E+002	**2.16E+002**	**2.15E+002**

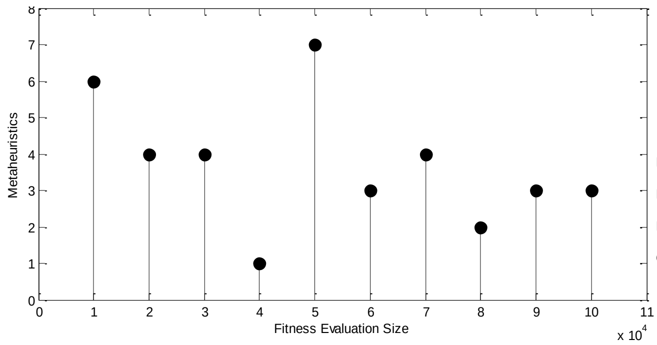
To demonstrate the convergence of CMH-MAS compared to its component agents, convergence graphs for three randomly selected functions of size 10, 30 and 50 are plotted in Figure 7.8.a-c. It can be seen that convergence speed of CMH-MAS is faster than those its component agents, particularly towards the end of iterations. A fundamental observation is that while convergence plots of metaheuristic algorithms get flat quickly and continues with small improvements in fitness values, convergence plots of the proposed multi-agent system continues to decrease with a sufficient degree of slope. This shows the capability of the proposed method in escaping from locally optimal solutions.



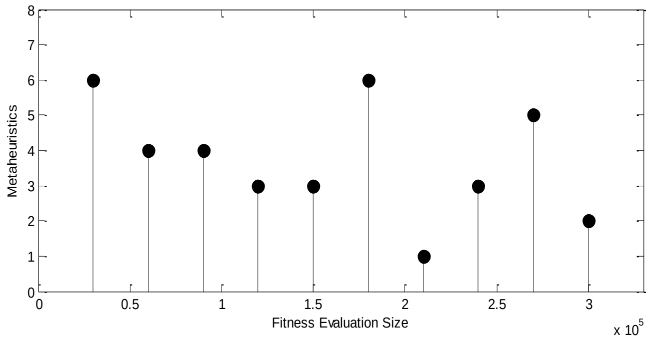
(c)

Figure 7.8. Convergence speed plots of CMH-MAS and its components agents for three randomly selected problems: F18 of size 10 (a), F10 of size 30 (b) and F22 of size 50 (c)

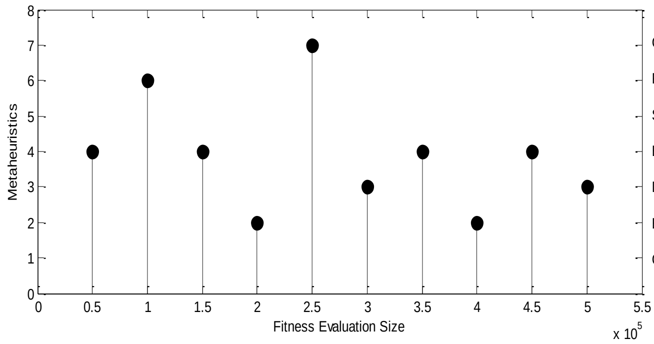
In order to demonstrate how winners of tournament-based competitions are changed along different sessions of the proposed system and how frequently a particular metaheuristic wins the tournaments, steps when each metaheuristics wins are plotted in Figure 7.9 for three randomly selected problems of sizes 10, 30 and 50. A basic observation on this figure shows that a particular metaheuristics exhibits better performance, in terms of the amount of improvement in fitness function, and wins the tournament. Also, the same metaheuristic may win the tournaments multiple times at different sessions of the proposed system. This is indeed an important observation since changes in fitness landscape from beginning towards the end search process requires different strategies to be implemented. This is in fact why CMS-MAS is more effective than its component metaheuristic agents.



(a)



(b)



(c)

Figure 7.9. Metaheuristics that won the tournament competitions at different stages of CMH-MAS for problem F10 of size 10 (a), F18 of size 30 (b), and F8 of size 50 (c).

To present the convergence of CMH-MAS compared to same CMH-MAS without our proposed strategy, convergence graphs for one randomly selected function of size 30 is plotted in Figure 7.10. CMH-MAS is compared to CMH-MAS with randomly selection of metaheuristics and the figure shows that convergence speed of CMH-MAS is faster than it. This shows the good effect of applying proposed strategy in our proposed multi-agent system.

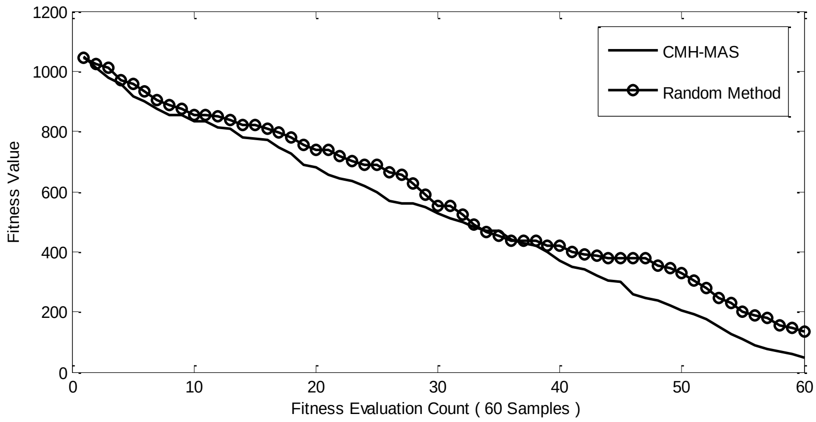


Figure 7.10. Convergence speed plots of CMH-MAS and same CMH-MAS with random method strategy for F18 with size 30.

For pairwise statistical tests of CMH-MAS and its competitors, nonparametric Wilcoxon signed ranks test is selected to show that the phenotypic population extracted by CMH-MAS is different from those of other metaheuristics under consideration. In fact, both parametric and nonparametric statistical tests can be used for this purpose. As it is clearly explained in the comprehensive tutorial of Derrac et al [65], parametric tests are applied based on assumptions like normality, independence, and homoscedasticity. Since these assumptions are hard to be guaranteed for any stochastic search procedure, non-parametric tests that do not require any of the above mentioned assumptions, are practically more preferable in the statistical analysis of experiments. To compute the Wilcoxon signed test scores of

CMH-MAS against its competitors, the procedure presented in [65] is used and the corresponding results are illustrated in Table 7.9 for the three problem dimensions. In this table, each row shows the pairwise scores between the metaheuristic labeling the row and CMH-MAS, where *R+* is the sum of ranks corresponding to problem instances for which CMH-MAS is better than the corresponding metaheuristic and *R-* represents the sum of ranks for problems for which CMH-MAS is worse than its competitor under consideration. The significance level (α) and the p-values are the computed parameters that are used for handling the null hypothesis stating that "CMH-MAS and its competitor are statistically similar". Basically, if p-value is less than α then the null hypothesis is rejected and smaller significance levels indicates higher confidence on the rejection of null hypothesis.

Table 7.9. Wilcoxon signed test results for pairwise statistical analysis of CMH-MAS against its competitors for problem all problem instances of size 10, 30 and 50

D = 10				
Method	R^+	R^-	**α**	**p-value**
BLX-GL50	224	76	0.05	0.034491
BLX-MA	296	29	0.01	0.000328
COEVO	325	0	0.01	0.000012
DE	254	46	0.01	0.002964
DMS-L-PSO	194	82	0.1	0.088524
EDA	268	32	0.01	0.000748
G-CMA-ES	173	80	0.2	0.131132
K-PCX	239	61	0.05	0.010995
L-CMA-ES	266.5	33.5	0.01	0.000873
L-SADE	158	95	0.5	0.306465
SPC-PNX	266	34	0.01	0.000919
D = 30				
Method	R^+	R^-	**α**	**p-value**
BLX-GL50	228	48	0.01	0.006194
BLX-MA	257	43	0.01	0.002235
COEVO	300	0	0.01	0.000018
DE	247	29	0.01	0.000916
G-CMA-ES	215	61	0.05	0.019183
K-PCX	256	69	0.05	0.011876
L-CMA-ES	279	46	0.01	0.001721
SPC-PNX	212	41	0.01	0.005506
D = 50				
Method	R^+	R^-	**α**	**p-value**
G-CMA-ES	204	96	0.15	0.122865
L-CMA-ES	243	82	0.05	0.030311

Investigation of scores in Table 7.9 clearly indicates that null hypothesis is rejected with strong confidences against all competitors of CMH-MAS for the three experimental sets associated with dimensions 10, 30 and 50. There are three exceptional cases for which α values are above 0.1 while the null hypothesis is still rejected. These three cases correspond to comparisons with G-CMA-ES for D=10 and D=50, and comparison to L-SADE for D=10. It can be seen from Table 7.9 that average fitness values for these algorithms and CMH-MAS are different, however rank order of these algorithms and CMH-MAS are close to each other that causes the α values become above 0.1.

In order to determine the order of CMH-MAS compared to its competitors, one-to-all (or 1×N) Friedman Aligned Ranks Test is implemented for all experimental results obtained from the three sets of benchmark problems. The computational procedure followed for the implementation of this test is exactly the same as the one described in [25]. Tables 7.10, 7.11 and 7.12 illustrate the Friedman aligned ranks of all algorithms for all problem instances of sizes 10, 30 and 50, respectively. A table entry (e.g. an aligned rank) the location of a (problem, algorithm) pair when all such pairs are ranked from 1 to k.N, where k is the number of algorithms and N corresponds to the number of problem instances. As explained in [65], the Friedman aligned ranks test statistics, *FAR*, is computed and is compared for significance with a χ^2 distribution with (*k-1*) degrees of freedom. The p-values computed using FAR statistics are indicators of significant differences among algorithms under consideration. Accordingly, Table 7.13 presents the FAR and the corresponding p-values for all algorithms used to solve the CEC2005 problems of sizes D=10, 20 and 30. It can easily be seen that, compared to its competitors, the average values associated with CMH-MAS is the smallest in all of the three cases indicating that CMH-MAS is the best performing algorithm and it is significantly better than its competitors for all problems of sizes 10, 30, 50. This is a clear indication on the scalability and highly improved search capability of the proposed multi-agent system. In addition to this, the p-values computed from the *FAR* statistics are very close to zero that further indicates that there is significant difference among all algorithms under consideration, which also implies that CMH-MAS is statistically different than its competitors. This result is fully compatible with the conclusion derived from Wilcoxon signed ranks test.

Table 7.10. Friedman aligned ranks for all (problem, algorithm) pairs for D=10

	Friedman Aligned Ranks											
D = 10 Problem	BLX-GL50	BLX-MA	COEVO	DE	DMS-L-PCO	EDA	G-CMA-ES	K-PCX	L-CMA-ES	L-SADE	SPC-PNX	CMH-MAS
F1	185	186	187	188	189	190	191	192	193	194	195	184
F2	196	197	198	199	200	201	202	203	204	205	206	183
F3	21	298	13	17	14	20	15	19	16	18	299	12
F4	2	8	3	4	11	5	6	9	300	10	7	1
F5	124	132	148	125	130	126	127	256	128	131	129	123
F6	138	151	238	145	142	143	139	147	140	141	242	137
F7	175	214	177	209	178	222	172	215	173	176	182	171
F8	217	181	211	218	164	212	165	166	167	168	225	169
F9	115	111	240	114	106	152	110	108	254	107	146	118
F10	116	120	243	233	105	119	100	101	250	117	135	163
F11	150	227	235	136	228	221	144	234	180	231	149	134
F12	284	55	294	43	33	288	39	90	247	32	263	31
F13	207	208	223	220	161	229	179	174	162	156	213	157
F14	155	153	226	224	159	170	216	158	230	210	219	160
F15	276	246	257	239	25	270	95	289	88	28	237	53
F16	96	104	262	232	97	252	94	98	112	102	154	103
F17	76	89	259	81	77	113	85	70	293	80	84	82
F18	46	286	291	36	280	65	26	279	69	275	51	24
F19	45	278	287	35	269	92	23	277	66	267	27	274
F20	44	281	290	48	285	260	22	283	38	268	37	34
F21	266	273	255	56	75	54	57	295	30	50	265	52
F22	253	78	261	133	93	258	236	71	245	241	248	74
F23	67	282	272	47	244	68	40	292	264	79	49	41
F24	58	72	249	59	73	60	61	271	297	62	63	64
F25	109	99	42	296	83	86	87	121	251	91	122	29
Sum	3421	4329	5361	3337	3421	3876	2631	4418	4397	3439	3947	2573
Average	**136.84**	**173.16**	**214.44**	**133.48**	**136.84**	**155.04**	**105.24**	**176.72**	**175.88**	**137.56**	**157.88**	**102.92**

Table 7.11. Friedman aligned ranks for D=30

D = 30 Instance	Friedman Aligned Ranks BLX-GL50	BLX-MA	COEVO	DE	G-CMA-ES	K-PCX	L-CMA-ES	SPC-PNX	CMH -MAS
F1	126	127	161	128	129	130	131	132	125
F2	134	139	159	144	135	136	137	138	133
F3	14	223	13	222	10	12	11	224	9
F4	3	4	7	2	8	5	225	1	6
F5	26	203	219	25	22	192	23	218	24
F6	48	60	217	57	46	49	47	53	45
F7	141	146	151	145	142	148	143	147	140
F8	153	150	154	160	122	119	120	155	156
F9	94	83	190	95	84	82	201	100	91
F10	62	97	191	101	55	54	210	78	70
F11	168	182	185	183	107	180	111	108	106
F12	19	18	221	17	220	16	21	20	15
F13	157	118	170	115	114	179	113	116	112
F14	117	123	152	158	124	162	163	149	121
F15	93	181	189	184	50	212	52	186	33
F16	61	194	198	187	44	58	56	59	199
F17	42	99	193	72	105	43	215	35	34
F18	174	102	197	175	176	75	110	178	68
F19	165	103	195	166	167	73	164	171	71
F20	169	104	196	172	173	76	109	177	69
F21	85	86	188	87	88	207	79	89	90
F22	77	98	200	92	51	214	74	81	27
F23	96	80	206	63	64	202	67	65	66
F24	205	28	211	29	209	32	216	30	31
F25	37	38	213	208	39	40	204	41	36
Sum	2466	2786	4376	2987	2484	2696	3002	2751	1877
Average	**98.64**	**111.44**	**175.04**	**119.48**	**99.36**	**107.84**	**120.08**	**110.04**	**75.08**

Table 7.12. Friedman aligned ranks for all (problem, algorithm) pairs for D=50

Problem	Friedman Aligned Ranks G-CMA-ES	L-CMA-ES	CMH-MAS
F1	45	46	32
F2	43	44	33
F3	39	40	35
F4	2	75	1
F5	13	14	71
F6	37	38	36
F7	41	42	34
F8	31	30	50
F9	11	70	17
F10	5	73	7
F11	26	51	56
F12	74	4	3
F13	48	49	29
F14	47	52	28
F15	16	20	66
F16	25	54	53
F17	12	72	8
F18	60	57	23
F19	59	58	22
F20	61	55	24
F21	68	9	27
F22	62	63	10
F23	67	15	21
F24	69	64	6
F25	18	65	19
Sum	979	1160	711
Average	**39.16**	**46.4**	**28.44**

Table 7.13. Friedman Aligned Ranks statistics and the corresponding p-values over all algorithms used to solve problem instances of sizes D=10, 30, and 50

D = 10	
Algorithms	Average values of Friedman Aligned Ranks over all problem instances
BLX-GL50	136.84
BLX-MA	173.16
COEVO	214.44
DE	133.48
DMS-L-PSO	136.84
EDA	155.04
G-CMA-ES	105.24
K-PCX	176.72
L-CMA-ES	175.88
L-SADE	137.56
SPC-PNX	157.88
CMH-MAS	102.92
F_{AR}	**83.159**
p-value	**0.0**
D = 30	
Algorithms	Average values of Friedman Aligned Ranks over all problem instances
BLX-GL50	98.64
BLX-MA	111.44
COEVO	175.04
DE	119.48
G-CMA-ES	99.36
K-PCX	107.84
L-CMA-ES	120.08
SPC-PNX	110.04
CMH-MAS	75.08
F_{AR}	**453.51**
p-value	**0**
D = 50	
Algorithms	Average values of Friedman Aligned Ranks over all problem instances
G-CMA-ES	39.16
L-CMA-ES	46.4
CMH-MAS	28.44
F_{AR}	**165.166**
p-value	**0**

In order to evaluate the time complexity of CMH-MAS, the procedure described in [41] is exactly followed. These rules aim to express time complexity of an algorithm independent of the computing platform. According to this procedure, published in CEC2005 competition framework, all competitors should provide three run-time measurements, *T0, T1* and *T2. T0* is the time required to execute a given fixed code by one million times repetition, *T1* is the computational time to compute value of CEC2005 benchmark *F3*, 200,000 times and *T2* is the average running time of a particular algorithm for the optimization of benchmark F3 over 5 times with 200,000 fitness evaluations. Consequently, the time complexity of the algorithm under consideration is computed as *(T2 - T1) / T0*.

Tables 7.14, 7.15 and 7.16 illustrate time complexities of all algorithms participated to CEC2005 competition and CMH-MAS for problem sets D=10, 30 and 50, respectively.

Remembering from previous discussions that CMH-MAS's time complexity is better than G-CMA-ES, that was the winner of CEC2005 competition, for problem sizes D=10 and D=30. Another observation on time complexity of our proposal is that it increases gradually with increasing problem size. Hence, it is interesting that G-CMA-ES has an smaller time complexity for D=50; however, time complexity of CMH-MAS is in the order of L-CMA-ES for D=50. This can be seen as a clear evidence that the time complexity of our proposal is closer to that of CMA-ES algorithms in the worst-case.

Table 7.14. Time complexity of algorithms with D=10

Algorithm	T0	T1	T2	Time Complexity
BLX-GL50	90919 ns	22316 ns	11950 ns	10.689
BLX-MA	420 ms	1414 ms	4440 ms	7.20
CoEVO	1.3 s	2.0 s	22.1 s	15.5
DE	0.29 s	1.2 s	1.502 s	1.041
DMS-L-PSO	36.445 s	30.64 s	77.038 s	1.273
EDA	6.93 s	0.753	2.328 s	0.227
G-CMA-ES	0.4 s	17 s	32 s	37.5
K-PCX	0.24 s	1.25 s	34.37 s	138 s
L-CMA-ES	0.4 s	32 s	51 s	47.5
L-SaDE	40.071	31.68	68.80 s	0.826
SPC-PNX	0.610 s	26.797 s	136.04 s	179.102
CMH-MAS	**0.2543 s**	**22.8 s**	**30.42 s**	**29.9599**

Table 7.15. Time complexity of algorithms with D=30

Algorithm	T0	T1	T2	Time Complexity
BLX-GL50	9091 ns	82981 ns	20646 ns	13.5811
BLX-MA	420 ms	8630 ms	1345 ms	11.48
CoEVO	1.3 s	1.6 s	16.8 s	11.7
DE	0.29 s	7.64 s	8.492 s	2.9379
G-CMA-ES	0.4 s	24 s	41	42.5
K-PCX	0.24 s	24.60 s	105.75 s	338.12
L-CMA-ES	0.4 s	41 s	45 s	10
SPC-PNX	0.407 s	32.218 s	135.55 s	253.894
CMH-MAS	**0.2543s**	**27.931 s**	**36.12 s**	**32.5902**

Table 7.16. Time complexity of algorithms with D=50

Algorithm	T0	T1	T2	Time Complexity
G-CMA-ES	0.4 s	49 s	56 s	17.5
L-CMA-ES	0.4 s	49 s	68 s	47.5
CMH-MAS	**0.2543 s**	**36.43 s**	**48.54 s**	**47.6134**

7.3 Evaluation of Multi-Agent Architecture for Real-Valued Multi-Objective Optimization

Performance evaluation of the proposed algorithm and exhibition of its comparative success against state-of-the art metaheuristics are carried out over the difficult problems in CEC2009 benchmarks [66]. Details of these benchmark functions could not be given here due to space limitations, but definitions, categorization and fitness landscape characteristics of all these functions are described clearly in the reference given above. For each of the test functions, the number of independent runs and the termination criterion, in terms of the number of fitness evaluations, are set the same as the ones used in the corresponding reference so that fairness is guaranteed for comparative evaluations. Algorithmic parameters of the proposed method are kept the same for all the test functions and no interactive intervention is made throughout the program executions. Additionally, number of variables for the test functions is also taken the same as the ones specified in the corresponding references.

Algorithmic parameters of the metaheuristic methods used within the proposed multi-agent system are given in Table 7.17. All of the parameters in Table 7.17 are collected from well-known conventional implementations of the corresponding metaheuristic algorithms. Implementation of the proposed system is carried out using Matlab® programming language environment and a personal PC with 8 GB main memory and 2.1 GHz clock speed.

Table 7.17. Algorithmic parameters of the metaheuristic methods used within the proposed system

Metaheuristic Agent	Algorithm Parameters
MOGA	|Pop| = 40, P_C =0.7, P_m=0.2, Gaussian_Sigma_Pm=20
MOPSO	|Pop| = 40, C1=2.0, C2=2.0, ω_{max}=0.9, ω_{min}=0.4
MODE	|Pop| = 40, Scaling_Factor=0.5, P_C =0.7,
SPEA2	|Pop| = 40, P_C =0.9, P_m=1.0/Num_Vars, Distribution_Index=20
AMOSA	Archive_H_{limit}=20, Archive_S_{limit}=50, Max_Temp=200, Coolin_Rate=0.95, Min_Temp=0.00025, Gamma=2.0, Hill_Climbing_Num=20,
NSGAII	|Pop| = 40, P_C =0.9, P_m=1.0/ Num_Vars, Distribution_Index=20,

As mentioned above, the set of benchmark problems on which performance of the proposed multi-agent system, named as RdMD/MAS from this point on, is comparatively evaluated is the CEC2009 numerical MOO competition benchmarks. There are 10 multi-objective unconstrained problems within this set; they are mostly generated from the set of classical benchmarks through random shifting, random shifting and rotation, and hybrid composition operations. Among these problems *UF1* to *UF7* are two-objective and *UF8*, *UF9* and *UF10* are three-objective problems. Detailed description of problems within this set and the experimental conditions under which the runs are performed are presented in [66, 67]. Accordingly, all results are averages over 30 runs and maximum number of fitness evaluations is set to 300,000. Based on the CEC2009 competition rules, problem size (number of variables) for each benchmark problem instance is set as 30 and the IGD (Inverted Generational Distance) values are used to compare performance of algorithms. For this purpose, results of algorithms that participated in CEC2009 MOO competition are taken from [67] and average results of RdMD/MAS over 30 runs are compared to these published values for all problems. As stated in [67], the maximum number of final Pareto-front solutions to be used for the computation of IGD scores is 100 for two-objective problems and 150 for three-objective problems.

Table 7.18 illustrates the Min, Max and Average IGD values associated with the proposed RdMD/MAS algorithm for the 10 benchmark problems over 30 independent runs for each problem.

Table 7.18. Min, Max and Average IGD values of RdMD/MAS in 30 runs

Function	Average	Min	Max	Std
UF1	0.00531	0.00519	0.00601	0.00028
UF2	0.00669	0.00652	0.00723	0.00026
UF3	0.03283	0.03156	0.03933	0.00318
UF4	0.02347	0.02305	0.02649	0.00138
UF5	0.08422	0.07195	0.09329	0.00641
UF6	0.03931	0.02915	0.04893	0.00648
UF7	0.00912	0.00784	0.01078	0.02852
UF8	0.11232	0.11168	0.12447	0.01542
UF9	0.06875	0.06375	0.07763	0.00435
UF10	0.23856	0.17757	0.36231	0.07635

Results in Table 7.18 show that RdMD/MAS is a successful and robust algorithm illustrated with small IGD values and their standard deviations. The largest IGD values belong to test problems *UF8* and *UF10* problems, however the performance order of RdMD/MAS are 6th and 2nd, among 14 algorithms, for *UF8* and *UF10*, respectively.

Tables 7.19, 7.20, 7.21 and 7.22 illustrate the ranking of all algorithms that took part in CEC2009 MOO contest and RdMD/MAS with respect to the average IGD scores. Based on the published results in [67], the best performing five algorithms in the competition are MOEAD [68], MTS [69], DMOEADD [70], LiuLi [71] and GDE3 [72] in order. Hence, the winner of the competition was MOEAD. It can be seen over the three tables that RdMD/MAS performed better than MOEAD in 5 of the 10 test problems. The proposed MAS takes the first position for one test problem (*UF4*) and takes the first, second or third positions in 80% of the ten benchmark problems. Among 14 rank positions, the worst rank of RdMD/MAS is the sixth position that is taken for test problem *UF8*. The proposed method took the second rank position for the difficult three-objective test problem *UF10* for which the achieved average IGD score is significantly better than the competitors in lower ranks. Except the test problem *UF8*, it can be seen that in those 5 problems for which MOEAD is better than RdMD/MAS, ranks of the proposed algorithm are quite close to those of MOEAD; whereas for the other test problems for which RdMD/MAS is better than MOEAD, ranks of MOEAD are far from those of the proposed algorithm.

Table 7.19. Average IGD values obtained by RdMD/MAS and its 13 competitors for UF1, UF2 and UF3

Rank	UF1	IGD	UF2	IGD	UF3	IGD
1	MOEAD	0.00435	MTS	0.00615	MOEAD	0.00742
2	RdMD/MAS	**0.00531**	MOEADGM	0.00640	LiuLiAlgorithm	0.01497
3	GDE3	0.00534	RdMD/MAS	**0.00669**	RdMD/MAS	**0.03283**
4	MOEADGM	0.00620	DMOEADD	0.00679	DMOEADD	0.03337
5	MTS	0.00646	MOEAD	0.00679	MOEADGM	0.04900
6	LiuLiAlgorithm	0.00785	OWMOSaDE	0.00810	MTS	0.05310
7	DMOEADD	0.01038	GDE3	0.01195	ClusteringMOEA	0.05490
8	NSGAIILS	0.01153	LiuLiAlgorithm	0.01230	AMGA	0.06998
9	OWMOSaDE	0.01220	NSGAIILS	0.01237	DECMOSA-SQP	0.09350
10	ClusteringMOEA	0.02990	AMGA	0.01623	MOEP	0.09900
11	AMGA	0.03588	MOEP	0.01890	OWMOSaDE	0.10300
12	MOEP	0.05960	ClusteringMOEA	0.02280	NSGAIILS	0.10603
13	DECMOSA-SQP	0.07702	DECMOSA-SQP	0.02834	GDE3	0.10639
14	OMOEAII	0.08564	OMOEAII	0.03057	OMOEAII	0.27141

Table 7.20. Average IGD values obtained by RdMD/MAS and its 13 competitors for UF4, UF5 and UF6

Rank	UF4	IGD	UF5	IGD	UF6	IGD
1	RdMD/MAS	**0.02347**	MTS	0.01489	MOEAD	0.00587
2	MTS	0.02356	GDE3	0.03928	RdMD/MAS	0.03931
3	GDE3	0.02650	RdMD/MAS	**0.08422**	**MTS**	**0.05917**
4	DECMOSA-SQP	0.03392	AMGA	0.09405	DMOEADD	0.06673
5	AMGA	0.04062	LiuLiAlgorithm	0.16186	OMOEAII	0.07338
6	DMOEADD	0.04268	DECMOSA-SQP	0.16713	ClusteringMOEA	0.08710
7	MOEP	0.04270	OMOEAII	0.16920	MOEP	0.10310
8	LiuLiAlgorithm	0.04350	MOEAD	0.18071	DECMOSA-SQP	0.12604
9	OMOEAII	0.04624	MOEP	0.22450	AMGA	0.12942
10	MOEADGM	0.04760	ClusteringMOEA	0.24730	LiuLiAlgorithm	0.17555
11	OWMOSaDE	0.05130	DMOEADD	0.31454	OWMOSaDE	0.19180
12	NSGAIILS	0.05840	OWMOSaDE	0.43030	GDE3	0.25091
13	ClusteringMOEA	0.05850	NSGAIILS	0.56570	NSGAIILS	0.31032
14	MOEAD	0.06385	MOEADGM	1.79190	MOEADGM	0.55630

Table 7.21. Average IGD values obtained by RdMD/MAS and its 13 competitors for UF7 and UF8

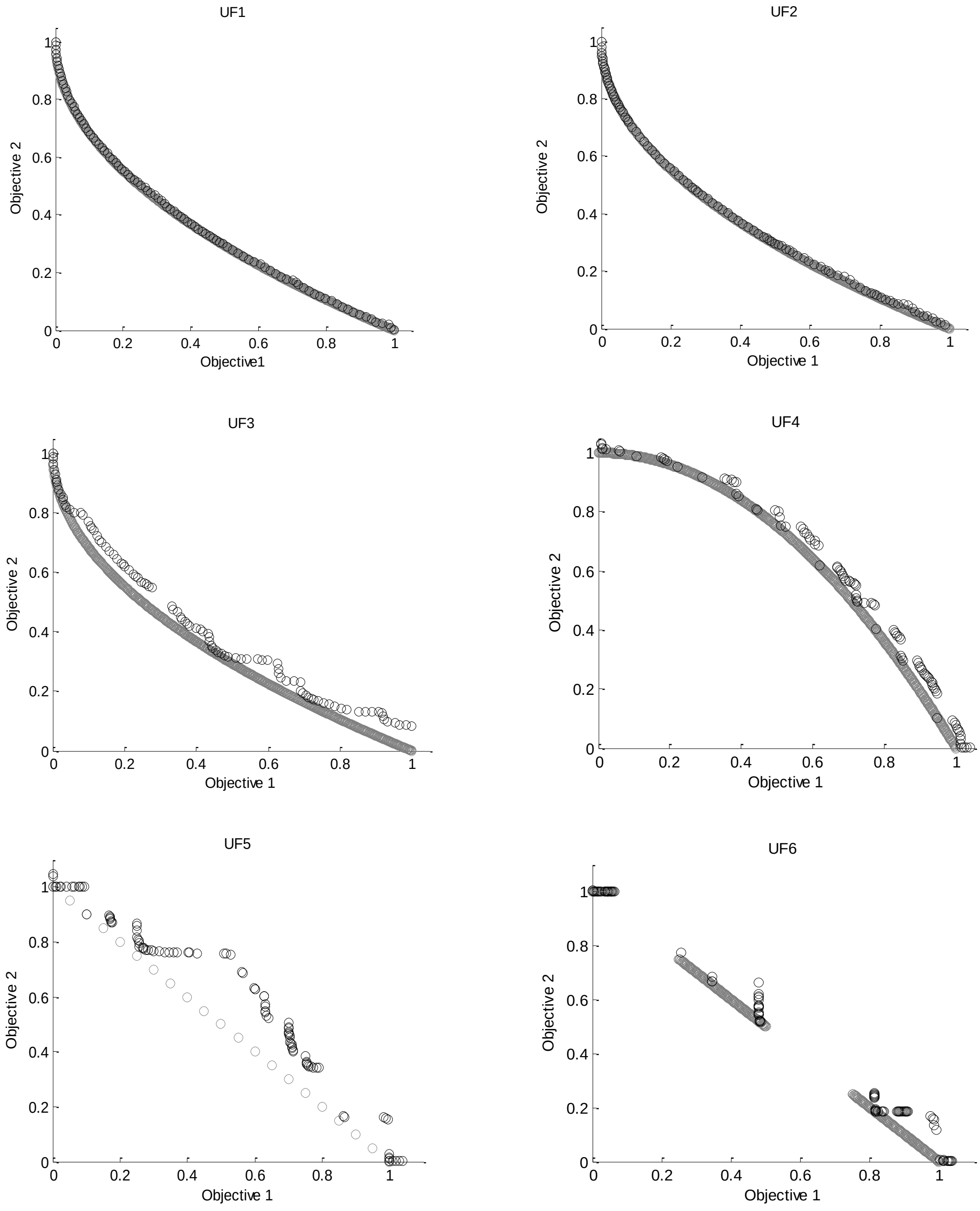
Rank	UF7	IGD	UF8	IGD
1	MOEAD	0.00444	MOEAD	0.05840
2	LiuLiAlgorithm	0.00730	DMOEADD	0.06841
3	MOEADGM	0.00760	LiuLiAlgorithm	0.08235
4	RdMD/MAS	**0.00912**	NSGAIILS	0.08630
5	DMOEADD	0.01032	OWMOSaDE	0.09450
6	MOEP	0.01970	RdMD/MAS	**0.11232**
7	NSGAIILS	0.02132	MTS	0.11251
8	ClusteringMOEA	0.02230	AMGA	0.17125
9	DECMOSA-SQP	0.02416	OMOEAII	0.19200
10	GDE3	0.02522	DECMOSA-SQP	0.21583
11	OMOEAII	0.03354	ClusteringMOEA	0.23830
12	MTS	0.04079	MOEADGM	0.24460
13	AMGA	0.05707	GDE3	0.24855
14	OWMOSaDE	0.05850	MOEP	0.42300

Table 7.22. Average IGD values obtained by RdMD/MAS and its 13 competitors for UF9 and UF10

Rank	UF9	IGD	UF10	IGD
1	DMOEADD	0.04896	MTS	0.15306
2	RdMD/MAS	**0.06875**	RdMD/MAS	**0.23856**
3	NSGAIILS	0.07190	DMOEADD	0.32211
4	MOEAD	0.07896	AMGA	0.32418
5	GDE3	0.08248	MOEP	0.36210
6	LiuLiAlgorithm	0.09391	DECMOSA-SQP	0.36985
7	OWMOSaDE	0.09830	ClusteringMOEA	0.41110
8	MTS	0.11442	GDE3	0.43326
9	DECMOSA-SQP	0.14111	LiuLiAlgorithm	0.44691
10	MOEADGM	0.18780	MOEAD	0.47415
11	AMGA	0.18861	MOEADGM	0.56460
12	OMOEAII	0.23179	OMOEAII	0.62754
13	ClusteringMOEA	0.29340	OWMOSaDE	0.74300
14	MOEP	0.34200	NSGAIILS	0.84468

Comparisons between RdMD/MAS and MTS, which is the second best performing algorithm in the contest, show that the proposed algorithm is better than MTS in 7 of the 10 test problems. MTS's performance is better for the test problems *UF2, UF5* and *UF10* only. Similar, evaluations compared to the third, fourth, and the fifth rank algorithms of CEC2009 MOO competition exhibit that RdMD/MAS achieved significantly better ranks than DMOEADD, LiuLi and GDE3 algorithms in 8, 7, and 6 test problems, respectively.

Figure 7.11 illustrates the plots of best computed Pareto-Fronts obtained by RdMD/MAS against the optimal one (PF-True) published as a result of the competition. The plots of computed Pareto fronts for two-objective test problems include 100 non-dominated solutions whereas those for three-objective problems cover 150 solutions. These solutions are selected based on the descriptions in [68] as follows: Solutions on the computed Pareto front are clustered into 100 (150 for three-objective case) classes and a member that is nearest to the PF-True from each class is selected.



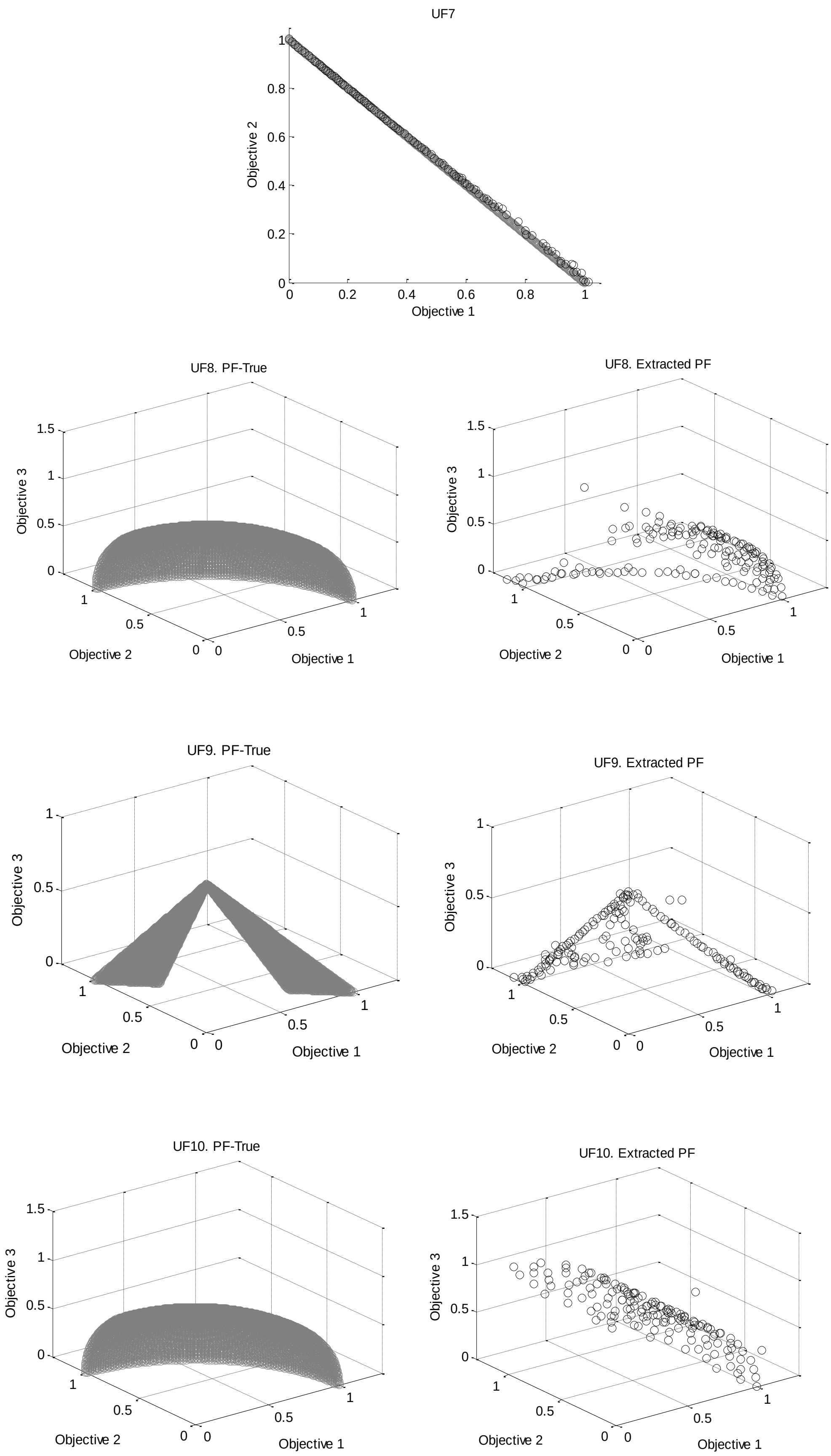


Figure 7.11. Pareto-Front found by RdMD/MAS for problems UF1 to UF10

Plots of computed Pareto fronts against the optimal ones demonstrate that the set of non-dominated solutions found by RdMD/MAS has a good spread and the computed PFs are quite close to PF-trues. Considering the test problems *UF5* and *UF6* for which there are local jumps out of the associated PF-trues, however these jumps also occurred on all plots given in [68, 69, 70, 71, 72] and the magnitude of these jumps for the computed PF of RdMD/MAS are significantly smaller than many of those found by the competitors.

To demonstrate the convergence of RdMD/MAS compared to its component agents, convergence graphs for UF5 are plotted in Figure 7.12. It can be seen that convergence speed of RdMD/MAS is faster than those its component agents, particularly at all level of iterations. A fundamental observation is that while convergence plots of metaheuristic algorithms get flat quickly and continues with small improvements in fitness values, convergence plots of the proposed multi-agent system continues to decrease with a sufficient degree of slope. This shows the capability of the proposed method in escaping from locally optimal solutions.

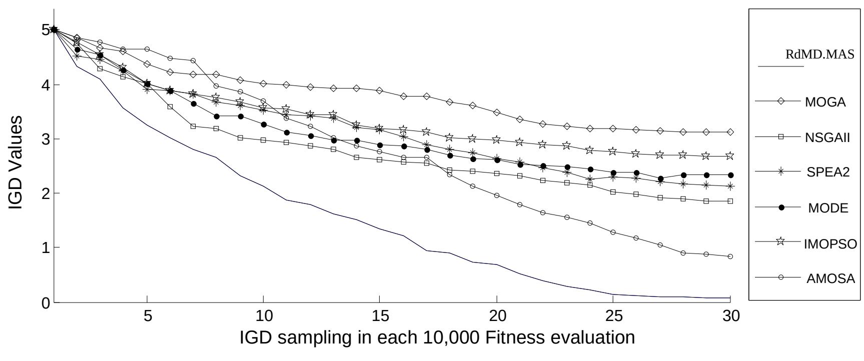


Figure 7.12. Convergence speed plots of RdMD/MAS and its components for UF5

To present the convergence of RdMD/MAS compared to same RdMD/MAS without our proposed strategy, convergence graph for one randomly selected function of UF5 is plotted in Figure 7.13. RdMD/MAS is compared to RdMA/MAS with randomly selection of metaheuristics and the figure shows that convergence speed of RdMA/MAS is faster than it. This shows the good effect of applying proposed strategy in our proposed multi-agent system.

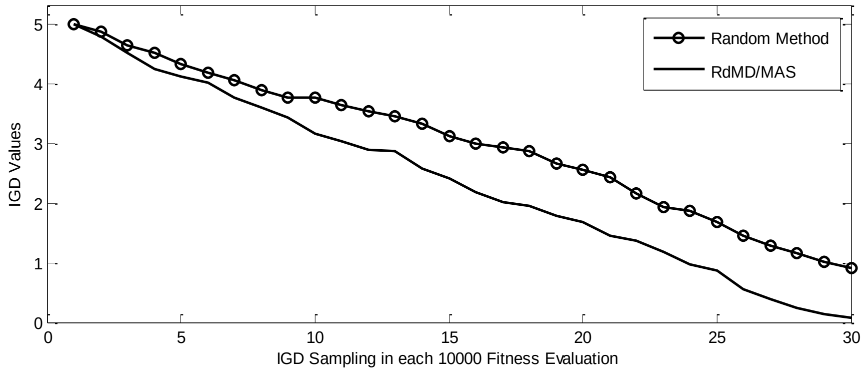


Figure 7.13. Convergence speed plots of RdMD/MAS and same RdMD/MAS with random method strategy for UF5

The last step of experimental evaluations is the Friedman Aligned Ranks Test that is implemented over all average IGD scores achieved by the 13 algorithms in CEC2009 MOO contest and the proposed RdMD/MAS algorithm. The objective of this test is of twofold: checking the statistical similarity of our results to those of others and determining the order of RdMD/MAS compared to its all competitors. This test is carried out according to the computational procedure described in [34]. Table 7.23 presents the Friedman aligned ranks of all algorithms for all problems. In this table, all (problem, algorithm) pairs are ranked from 1 to K.N; where K and N are the number of algorithms and problems respectively. As described in [65], the Friedman aligned ranks test statistics, FAR, is calculated by the corresponding formula and is compared for statistical significance with a χ^2 distribution with (K-1) degrees of freedom. The computed p-value with the χ^2 distribution is used to indicate the significant differences among the all algorithms under consideration. Consequently,

Table 7.24 shows the average rank values, FAR and the corresponding p-values. It is clearly seen that, the average value with regard to RdMD/MAS is the smallest one indicating that RdMD/MAS is the best performing algorithm. Meanwhile, the p-value is very close to zero that implies that there is significant statistical difference among results of all algorithms, which also means that RdMD/MAS is statistically different than its competitors.

Table 7.23. Friedman aligned ranks for all (problem, algorithm) pairs

Function	**MOEAD**	**GDE3**	**MOEADGM**	**MTS**	**LiuLiAlgorithm**	**DMOEADD**	**NSGAIILS**	**OWMOSaDE**	**ClusteringMOEA**	**AMGA**	**MOEP**	**DECMOSA-SQP**	**OMOEAII**	**RdMD/MAS**
UF1	49	51	55	56	58	63	65	66	93	99	115	121	122	50
UF2	72	82	70	69	83	73	84	75	97	90	95	100	105	71
UF3	30	113	45	47	34	41	112	111	48	68	108	101	133	40
UF4	107	61	94	54	88	85	102	96	103	80	86	67	91	53
UF5	11	3	140	1	7	74	135	128	29	5	19	8	9	4
UF6	10	127	139	18	106	21	131	117	31	46	37	44	23	14
UF7	52	89	60	104	59	64	79	116	81	114	77	87	98	62
UF8	15	125	124	36	22	16	24	28	123	92	136	120	110	35
UF9	32	33	118	43	38	17	27	39	130	119	134	76	126	26
UF10	109	57	129	2	78	12	138	137	42	13	20	25	132	6
SUM	487	741	974	430	573	466	897	913	777	726	827	749	949	361
AVG	**48.7**	**74.1**	**97.4**	**43**	**57.3**	**46.6**	**89.7**	**91.3**	**77.7**	**72.6**	**82.7**	**74.9**	**94.9**	**36.1**

Table 7.24. Friedman Aligned Ranks statistic and the corresponding p-value over all algorithms

Algorithms	**Average values of Friedman Aligned Ranks over all problem instances**
MOEAD	48.7
GDE3	74.1
MOEADGM	97.4
MTS	43
LiuLiAlgorithm	57.3
DMOEADD	46.6
NSGAIILS	89.7
OWMOSaDE	91.3
ClusteringMOEA	77.7
AMGA	72.6
MOEP	82.7
DECMOSA-SQP	74.9
OMOEAII	94.9
RdMD/MAS	**36.1**
F_{AR}	**34.5146**
p-value	**0.0010**

Chapter 8

CONCLUSIONS AND FUTURE WORKS

Chapter 4 in this thesis presents a learning-based multi-agent system (LBMAS) of metaheuristics for solving combinatorial optimization problems. Its effectiveness is tested using the well-known multiprocessor scheduling problem (MSP) in comparison to existing famous algorithms. Experimental results obtained using the proposed method exhibit good improvements and showed that the search capability achieved is better than most the competitors and is at least as good as a few of the others.

Chapter 5 is about the design of a competitive-cooperative multi-agent system of metaheuristics for the solution of real-valued single-objective optimization problems. Its effectiveness is tested using a well-known set of benchmark problems and its performance is comparatively evaluated against well-known modern optimization algorithms.

Experimental results exhibited that significant improvements have been obtained using the proposed algorithm and both the quantitative and statistical analyses put CMH-MAS to the first position against its competitors.

Chapter 6 presents a new approach for the design of a cooperative multi-agent system of metaheuristic agents for the solution of real-valued multi-objective optimization problems. Basic descriptions of a number of metaheuristics for MOO are implemented as individual agents. The global population is divided into subpopulations using dominance ranks and each subpopulation is optimized by an assigned agent where assignments change in a round-robin order. The effectiveness of the proposed MAS is tested using a well-known set of benchmark problems and its performance is comparatively evaluated against well-known modern MOO algorithms. Experimental results exhibited that significant improvements have been obtained using the proposed algorithm in comparison to well-known methods. Both the quantitative and statistical analysis put the proposed approach, RdMD/MAS to the first position against its competitors.

Further works are planned to use the proposed LBMAS with enhanced learning algorithms and use the resulting systems to solve other types of optimization problems. In addition to this, LBMAS is appropriate to be implemented on parallel processor environments. Also, further research is planned to extend the proposed MAS with additional MOO agents and consider its use for practical real-valued and combinatorial optimization problems. Meanwhile, CMH-MAS and RdMD/MAS are quite suitable to be implemented on parallel or graphics processors.

REFERENCES

[1] Stone, P., & Veloso, M. (2000). Multiagent systems: A survey from a machine learning perspective. *Autonomous robotics*. 8, 345-383.

[2] Panait, L., & Luke, S. (2005). Cooperative multi-agent learning, The State of the Art. *Autonomous Agents and Multi-agent Systems*. 387-434.

[3] Stuart, R. & Norvig, P. (2003). Artificial Intelligence: A Modern Approach (2nd ed.). Prentice Hall, ISBN. 0-13-790395-2. chpt. 2.

[4] Sycara, K. P. (2015). Multi-agent systems: American association for artificial intelligence. *AI magazine*. 19. no. 2.

[5] Luke, S. (2014). Essentials of Metaheuristics. second edition. available at http://cs.gmu.edu/~sean/book/metaheuristics

[6] Yang, X. (2011). Metaheuristic optimization: algorithm analysis and open problems. Experimental Algorithms. LNCS. Springer, 21-32.

[7] Cello Coello, C. A., Lamont, G. B., & Van Veldhuizen, D. A. (2007). Evolutionary Algorithms for Solving Multi-objective Problems. Second edition. Springer.

[8] Meignan, D., Creput, J. C., & Koukam, A. (2008). An organizational view of metaheuristics. *Proceedings of frist international workshop on optimization on multi-agent systems*. 77-85.

[9] Storn, R., & Price, K. (1997). Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization*. 341–359.

[10] Cadenas, J. M., Garrido, M. C., & Munoz, E. (2008). Construction of a Cooperative Metaheuristic system based on Data Mining and Soft-Computing. *Methodological issues: Proceedings of IPMU'08*. 1246-1253.

[11] Aydin, M. E. (2013). Coordinating metaheuristic agents with swarm intelligence. *Journal of Intelligent Manufacturing*. 991-999.

[12] Milano, M., & Roli, A. (2004). MAGMA: A Multi-agent Architecture for Metaheuristics. *IEEE Transactions on systems, man, and cybernetics*. 33, 925-941.

[13] Aydemir, F. B., Gunay, A., Oztoprak, F., Birbil, S. E., & Yolum, P. (2013). Multiagent cooperation for solving global optimization problems: an extendible framework with example cooperation strategies. *Journal of Global Optimization*. Springer. 57, 499-519.

[14] Jiang, S., Zhang, S. J., & Ong, Y. S. (2012). A Multiagent Evolutionary

Framework based on Trust for Multiobjective Optimization. *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*. Spain. 299-306.

[15] Drezewski, R., & Siwik, L. (2006). Co-Evolutionary Multi-Agent System with Sexual Selection Mechanism for Multi-Objective Optimization. *IEEE Congress on Evolutionary Computation*. Canada. 769-776.

[16] Drezewski, R., & Siwik, L. (2006). Multi-Objective Optimization Using Co-Evolutionary Multi-Agent System with Host-ParasiteMechanism. *Lecture notes in computer science, 6th international conference on Computational Science*. Springer. UK. 871-878.

[17] Kisiel-Dorohinicki, M., & Socha, K. (2001). Crowding Factor in Evolutionary Multi-Agent System for Multiobjective Optimization. *Proceeding of the international conference on artificial intelligence*.

[18] Cardon, A., Galinho, T., and Vacher, J. (2000). Genetic algorithms using multi-objectives in a multi-agent system. *Robotics and autonomous systems*. Elsevier. 179-190.

[19] Siwik, L., & Natanek, S. (2008). Solving Constrained Multi-Criteria Optimization Tasks Using Elitist Evolutionary Multi-Agent System. *World congress on computational intelligence*. *IEEE CEC 2008*. 3358-3365.

[20] Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization and Machine Learning. *Addison-Wesley Longman Publishing Co.* Boston. USA.

[21] Price, K. V. (1999). An Introduction to Differential Evolution. *New Ideas in Optimization*. McGraw-Hill. London.

[22] Storn, R., & Price, K. (1997). Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization*. 341–359.

[23] Bertsimas, D., & Tsitsiklis, J. (1993). Simulated Annealing. 10-15.

[24] Dorigo, M., & Caro, G. D. (1999). The ant colony optimizationmeta-heuristic. *New Ideas in Optimization*. New York. McGraw-Hill. 11–32.

[25] Dueck, G. (). New Optimization Heuristics, the Great Deluge Algorithm and the Record-to-Record Travel. *Journal of Computational Physics*. 86-92.

[26] Chelouah, R., & Siarry, P. (2000). Tabu search applied to global optimization", European Journal of Operational Research. 256-270.

[27] Naeem, M., Xue, S., & Lee, D. C. (2009). Cross-entropy optimization for sensor selection problems. *communications and information technology*. ISCIT 2009. 396-401.

[28] Acan, A., & Unveren, A. (2014). A two-stage memory powered Great Deluge algorithm for global optimization. *Journal of Soft Computing*. Springer.

[29] Caruana, R., Eshelman, L. J., & Schaffer, J. D. (1989). Representation and Hidden Bias II: Eliminating Defining Length Bias in Genetic Search via Shuffle Crossover. *IJCAI*. 750-755.

[30] Kruisselbrink & Willem, J. (2012). Evolution strategies for robust optimization. *Leiden Institute of Advanced Computer Science (LIACS)*, Leiden university.

[31] Teodorovic, D., Lucic, P., Markovic, G. & Orco, M. D. (2006). Bee colony optimizations: principles and applications. *Neural network applications in electrical engineering*. IEEE. Serbia. 151-156.

[32] Kennedy, I., & Eberhart, R. ()1995. Particle Swarm optimization. *IEEE international conference on neural networks*. 1942-1948.

[33] Ballester, P. J., Jonathan, J. S., & and Gallagher, N. C. K. (2005). Real-Parameter Optimization Performance Study on the CEC-2005 benchmark with SPC-PNX. *IEEE conference publications*. 498-505.

[34] Suganthan, P. N., Hansen, N., Liang, J. J., Deb, K., Chen, Y. P., Auger, A., & Tiwari, S. (2005). Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization. Nanyang Technological

University. Singapore.

[35] Sivanandam, S. N., & Deepa, S. N. (2008). Introduction to genetic algorithms. *Springer verlog berlin Heidelberg*. Germany.

[36] Haupt, R. L., & Haupt, S. E. (2004). Practical genetic algorithms. *John wiley and sons*. New Jersey.

[37] Eberhart, R. C., & Kennedy, J. (1995). A new optimizer using particle swarm theory. *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*. Japan.

[38] Dekkers, A., & Aarts, E. (1991). Global optimization and simulated annealing. *Mathematical Programming*. 50, 367-393.

[39] Abuhamdah, A. (2012). Modified Great Deluge for Medical Clustering Problems. *International Journal of Emerging Sciences*. 2, 345-360.

[40] Abraham, A., Lakhmi, J., & and Goldberg, R. (2005). Evolutionary Multi-objective Optimization. *Springer-Verlag London*.

[41] Bosman, P. A. N. (2014). On Gradients and Hybrid Evolutionary Algorithms for Real-valued Multi-objective Optimization. *IEEE Transactions on Evolutionary Computation*. 16, 51-69.

[42] Deb, K., Agrawal, S., Pratap, A., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation*. IEEE Trans. 6, 182–197.

[43] Fonseca, C. M., & Fleming, P. J. (1993). Genetic algorithm for multiobjective optimization, formulation, discussion and generalization. *Genetic Algorithms, Proceeding of the Fifth International Conference*. 416-423.

[44] Zitzler, E., Laumanns, M., & Thiele, L. (2001). SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. *Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems*. 95–100.

[45] Xue, F., Sanderson, A. C., & Graves, R. J. (2003). Pareto-based Multi-objective Differential Evolution. *The proceeding of the 2003 congress on Evolutionary Computation (CEC'2003)*. Australia. 862-869.

[46] Bandyopadhyay, S., Saha, S., Maulik, U., & Deb, K. (2008). A Simulated Annealing Based Multi-objective Optimization Algorithm: AMOSA. *IEEE Transactions on Evolutionary Computation*. 12, 269-283.

[47] Coello, C. A., & Lechuga, M. S. (2002). MOPSO: A proposal for multiple objective particle swarm optimization. *The proceeding of congress on Evolutionary Computation (CEC'2002)*. US. 1051-1056.

[48] Q. Zhang, A. Zhou, S. Zhao, P. Suganthan, W. Liu and S. Santosh Tiwari, "Multiobjective optimization Test Instances for the CEC 2009 Special Session and Competition", IEEE, CEC2009 Competition, 2009.

[49] M. A. Al-Mouhamed, "Lower Bound on the Number of Processors and Time for Scheduling Precedence Graphs with Communication Costs", IEEE Transaction Software Engineering, Vol. 16, No. 12, pp. 1390-1401, 1990.

[50] Wu, A. S., Yu, H., Jin, Sh., & Lin, K. Ch. (2004). Schiavone, G.: An Incremental Genetic Algorithm Approach to Multiprocessor Scheduling. *IEEE Transaction on Parallel and Distributed Systems*. 15, 824-834.

[51] Parsa, S., Lotfi, S., & Lotfi, N. (2007). An Evolutionary Approach to Task Graph Scheduling. *International Journal of Lecture Notes Computer Science*. ICANNGA 2007. Springer. 110-119.

[52] Wu, M. Y. (2000). MCP Revisited. Department of Electrical and Computer Engineering. University of New Mexico.

[53] Baxter, J., & and Patel, J. H. (1989). The LAST Algorithm: A Heuristic-Based Static Task Allocation Algorithm. *Proceeding of International Conference on Parallel Processing*. 2, 217-222.

[54] Coffman, E. G. (1976). Computer and Job-Shop Scheduling Theory. *John-*

Wiley.

[55] Hwang, J. J., Chow, Y. C., Anger, F. D., & Lee, C. Y. (1989). Scheduling Precedence Graphs in Systems with Inter-processor Communication Times. *SIAM Journal on Computer*. 18. 244-257.

[56] Kim, S. J., & Browne, J. C. (1988). A General Approach to Mapping of Parallel Computation upon Multiprocessor Architectures. *Proceeding Of International Conference on Parallel Processing*. 2, 1-8.

[57] Sarkar, V. (1989). Partitioning and Scheduling Parallel Programs for Multiprocessors. *MIT Press*. Cambridge.

[58] McCreary, C. L., Khan, A. A., Thompson, J. J., & McArdle, M. E. (1994). A Comparison of Heuristics for Scheduling DAGS on Multiprocessors. *Proceedings of the 8th International Parallel Processing Symposium*. 446-451.

[59] Rinehart, M., Kianzad, V., & Bhattacharyya, Sh. S. (2003). A Modular Genetic Algorithm for Scheduling Task Graphs. *Department of Electrical and Computer Engineering, and Institute for Advanced Computer Studies*. University of Maryland.

[60] Correa, R., Ferreira, A., & Rebreyend, F. P. (1999). Scheduling Multiprocessor Tasks with Genetic Algorithms. *IEEE Transaction on Parallel and Distributed*

Systems. 10, 825-837.

[61] Sih. G. C., & Lee, E. A. (1990). Scheduling to Account for Inter-processor Communication Within Interconnection-Constrained Processor Network. *International Conference on Parallel Processing.* 9-17.

[62] El-Rewini, H., & and Lewis, T. G. (1990). Scheduling Parallel Program Tasks onto Arbitrary Target Machines. *Journal of Parallel and Distributed Computing*. 9, 138-153.

[63] Ahmad, E., Dhodhi, M. K., Ahmad, I. (2010). Multiprocessor Scheduling by Simulated Evolution. *Journal of Software*. 5.

[64] Benchmark functions (CEC'2005). *Special Session on Real-Parameter Optimization*. IEEE. UK. 2-5. http://sci2s.ugr.es/eamhco/cec2005_values.xls

[65] Derrac, J., García, S., Molina, D., & and Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*. 1, 3-18.

[66] Zhang, Q., Zhou, A., Zhao, S., Suganthan, P., Liu, W., & Santosh Tiwari, S. (2009). Multiobjective optimization Test Instances for the CEC 2009 Special

Session and Competition. *IEEE*. CEC2009 Competition.

[67] Zhang, Q., & Suganthan, P. N. (2008). Final Report on CEC’09 MOEA Competition, Working Report. *CES-887, School of Computer Science and Electrical Engineering*. University of Essex.

[68] Zhang, Q., Liu, W., & Li, H. (2009). The Performance of a New Version of MOEA/D onCEC09 Unconstrained MOP Test Instances. CEC 2009. *Proceedings of the eleventh conference on congress on Evolutionary Computation*. IEEE. Norway. 203-208.

[69] Tseng, L. Y., & Chen, C. (2009). Multiple Trajectory search for Unconstrained/Constrained Multi-objective Optimization, CEC 2009. *Proceedings of the eleventh conference on congress on Evolutionary Computation*. IEEE. Norway. 1951-1958.

[70] Liu, M., Zou, X., Chen. Y., & Wu, Z. (2009). Performance Assessment of DMOEA-DD with CEC 2009 MOEA Competition Test Instances. CEC 2009. *Proceedings of the eleventh conference on congress on Evolutionary Computation*. IEEE. Norway. 1951-1958.

[71] Liu, H., & Li, X. (2009). The multiobjective evolutionary algorithm based on determined weight and sub-regional search. *IEEE*. Normay. 1928-1934.

[72] Kukkonen, S., & Lampinen, J. (2009). Performance Assessment of Generalized

Differential Evolution 3 with a Given Set of Constrained Multi-Objective Test Problems. *IEEE*. Normay. 2913-2918.

[73] Sarker, R. A., & Ray, T. (2010). Agent-Based Evolutionary Search. 1–11.

[74] Russell, S., & Norvig, P. (2003). Artificial intelligence: a modern approach. Prentice Hall. *Upper Saddle River*.

[75] Vacher, J. P., Galinho, T., Lesage, F., & and Cardon, A. (1998). Genetic algorithms in a multi-agent system. *IEEE International Joint Symposia on Intelligence and Systems*. 17–26.

[76] Nunes, L., & Oliveira, E. (2004). Learning from multiple sources. *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*. 3, 1106–1113.

[77] Iantovics, B., & Enăchescu, C. (2008). Intelligent complex evolutionary agent-based systems. *Proceedings of the 1st International Conference on Bio-Inspired Computational Methods used for Difficult Problems Solving*. 116–124.

[78] Liu, J., Zhong, W., & Jiao, L. (2010). A multiagent evolutionary algorithm for combinatorial optimization problems. *IEEE Transactions on Systems, Man and Cybernetics*. 40(1), 229–240.

[79] Barkat Ullah, A. S. S. M., Sarker, R., Cornforth, D., & Lokan, C. (2009). AMA:

A new approach for solving constrained real-valued optimization problems. *Soft Computing*. 741–762.

[80] Hippolyte, J. L., Bloch, C., Chatonnay, P., Espanet, C., & Chamagne, D. (2007). A self-adaptive multiagent evolutionary algorithm for electrical machine design. *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. 1250–1255.

[81] Li, Q., & Du, L. (2009). Research on hybrid-genetic algorithm for mas based job-shop dynamic scheduling. *Second International Conference on Intelligent Computation Technology and Automation*. IEEE Press . 404–407.

[82] Liu, J., Zhong, W., & Jiao, L. (2010). Multi-agent Evolutionary Model for Global Numerical optimization. *Agent-Based Evolutionary Search*. 5, 13-48.

[83] Barkat Ullah, A. S. S. M., Sarker, R., & Lokan, Ch. (2010). An agent based evolutionary approach for nonlinear optimization with equality constraints. *Agent-Based Evolutionary Search*. 5, 49-76.

[84] Yan, Y., Yang, Sh., Wang, D., & Wang, D. (2010). Agent based evolutionary dynamic optimization. *Agent-Based Evolutionary Search*. 5, 97-116.

[85] Lin, Y., & Zhang, J. (2010). An agent-based parallel ant algorithm with an adaptive migration controller. *Agent-Based Evolutionary Search*. 5, 161-177.