Except as otherwise permitted under the Copyright, Designs and Patents Act 1988, this thesis may only be produced, stored or transmitted in any form or by any means with the prior permission in writing of the author. The author asserts his/her right to be identified as such in accordance with the terms of the Copyright, Designs and Patents Act 1988

A Hybrid Multi-Agent Architecture and Heuristics Generation for Solving Meeting Scheduling Problem

PhD Thesis

Serein Abdelmonam Alratrout

This thesis is submitted in partial fulfilment of the requirements for the Doctor of Philosophy, awarded by

Software Technology Research Laboratory

Faculty of Technology

De Montfort University

United Kingdom, England

May, 2009

Abstract

Agent-based computing has attracted much attention as a promising technique for application domains that are distributed, complex and heterogeneous. Current research on multi-agent systems (MAS) has become mature enough to be applied as a technology for solving problems in an increasingly wide range of complex applications. The main formal architectures used to describe the relationships between agents in MAS are centralised and distributed architectures.

In computational complexity theory, researchers have classified the problems into the followings categories: (i) P problems, (ii) NP problems, (iii) NP-complete problems, and (iv) NP-hard problems. A method for computing the solution to NP-hard problems, using the algorithms and computational power available nowadays in reasonable time frame remains undiscovered. And unfortunately, many practical problems belong to this very class. On the other hand, it is essential that these problems are solved, and the only possibility of doing this is to use approximation techniques.

Heuristic solution techniques are an alternative. A heuristic is a strategy that is powerful in general, but not absolutely guaranteed to provide the best (i.e. optimal) solutions or even find a solution. This demands adopting some optimisation techniques such as Evolutionary Algorithms (EA).

This research has been undertaken to investigate the feasibility of running computationally intensive algorithms on multi-agent architectures while preserving the ability of small agents to run on small devices, including mobile devices. To achieve this, the present work proposes a *new Hybrid Multi-Agent Architecture (HMAA)* that

generates new heuristics for solving NP-hard problems. This architecture is hybrid because it is "*semi-distributed/semi-centralised*" architecture where variables and constraints are distributed among small agents exactly as in distributed architectures, but when the small agents become stuck, a centralised control becomes active where the variables are transferred to a super agent, that has a central view of the whole system, and possesses much more computational power and intensive algorithms to generate new heuristics for the small agents, which find optimal solution for the specified problem.

This research comes up with the followings: (1) Hybrid Multi-Agent Architecture (HMAA) that generates new heuristic for solving many NP-hard problems. (2) Two frameworks of HMAA have been implemented; search and optimisation frameworks. (3) New SMA meeting scheduling heuristic. (4) New SMA repair strategy for the scheduling process. (5) Small Agent (SMA) that is responsible for meeting scheduling has been developed. (6) “Local Search Programming” (LSP), a new concept for evolutionary approaches, has been introduced. (7) Two types of super-agent (LGP_SUA and LSP_SUA) have been implemented in the HMAA, and two SUAs (local and global optima) have been implemented for each type. (8) A prototype for HMAA has been implemented: this prototype employs the proposed meeting scheduling heuristic with the repair strategy on SMAs, and the four extensive algorithms on SUAs.

The results reveal that this architecture is applicable to many different application domains because of its simplicity and efficiency. Its performance was better than many existing meeting scheduling architectures. HMAA can be modified and altered to other types of evolutionary approaches.

Declaration

The thesis presented here is mine and original. It is submitted for the degree of Doctor of Philosophy at De Montfort University. The work was undertaken between December 2004 and May 2009.

Acknowledgment

I have accumulated many debts of gratitude during the journey of researching and writing this thesis.

First and foremost, I owe my utmost thanks to the almighty **God,** the most Merciful and the most Gracious; for all his grace and blessing, without whom none of this work would have been done.

I am deeply indebted to my supervisor Dr. Francois Siewe, for his support, ideas, guidance and encouragements throughout my research study. I am also very grateful to Professor Hussain Zedan and Dr. Amelia Platt for their help and support.

I warmly express my deep appreciation to my mother for her endless love, support and sacrifices all the way through, which are too precious to forget. I also gratefully acknowledge father and mother in law, special thanks and sincere gratitude for their prayers and everlasting sympathy.

My big and warmest thanks go to my lovely and patient husband, ***Ashraf*** who has always been beside me, a tower of strength to me during difficult moments, for his love, understanding, full support, encouragement and cooperation without which I was not going to be able to do this research.

I cannot conclude this acknowledgement without conveying my considerable love and apologies to my beloved kids ***Betool*** and ***Bahaa Aldeen***; I would like to dedicate this work to them.

I would like also to express my special thanks to my colleagues in the STRL; Mai Alfawair and Omar Aldabbas for their support.

Lastly, for everybody who was not mentioned here, but had contributed direct or indirectly to perform this work, my sincerely acknowledgements.

Table of Contents

ABSTRACT I

DECLARATION III

ACKNOWLEDGMENT IV

TABLE OF CONTENTS VI

LIST OF FIGURES X

LIST OF TABLES XIII

CHAPTER 1 1

INTRODUCTION 1

1.1 Research Motivation 1
1.2. Research Question 5
1.3. Major Contributions 7
1.4 Thesis Outlines 9

CHAPTER 2 12

MULTI-AGENT SYSTEMS 12

2.1 Introduction 12
2.2. What is an Agent? 13
2.2. Intelligent Agents 14
2.3. Mobile Agent 16
2.4. Agents and Objects 17
2.5. Agents and Expert Systems 19
2.6. What is an MAS? 19
2.7. Why MAS 20
2.8. Multi-Agent Architectures 22
2.9 Summary 24

CHAPTER 3 25

SCHEDULING PROBLEMS 25

3.1. Introduction 25
3.2. Formalisation of Scheduling Problem 26
3.2.1. Constraint Satisfaction Problem (CSP) 27

3.2.2. Constraint Optimisation Problem (COP) *27*

3.2.3 Distributed Constraint Reasoning (DCR) *28*

3.3. TIMETABLING 31

3.3.1 The HuSSH System *31*

3.4. MEETING SCHEDULING PROBLEM 34

3.4.1 Definition *34*

3.4.2. Frameworks for Solving MSP *36*

3.5. Commercial Products for MSP *41*

3.6. SUMMARY 42

CHAPTER 4 .... **44**

EVOLUTIONARY ALGORITHMS **44**

4.1. INTRODUCTION 44

4.2. ADVANTAGES OF EVOLUTIONARY ALGORITHMS 45

4.3. GENETIC ALGORITHMS (GA) 46

4.4. GENETIC PROGRAMMING (GP) 48

4.5. LINEAR GENETIC PROGRAMMING (LGP) 51

4.6. HOW DOES GP DIFFER FROM HEURISTIC APPROACH? 52

4.7. LOCAL SEARCH PROGRAMMING (LSP) 53

4.8. SUMMARY 55

CHAPTER 5 .... **57**

HYBRID MULTI-AGENT ARCHITECTURE FOR MEETING SCHEDULING (HMAA) **57**

5.1. INTRODUCTION 57

5.2. MEETING SCHEDULING FRAMEWORKS 59

5.3. SOLUTION APPROACH FOR MEETING SCHEDULING WITHIN HMAA 60

5.4 HYBRID MULTI-AGENT ARCHITECTURE PROPOSED 62

5.5. SCENARIO: HYBRID MULTI-AGENT ARCHITECTURE (HMAA) NEGOTIATIONS 65

5.6. SCENARIO: MSP WITHIN HMAA 67

5.7. FULL FUNCTIONS SPECIFICATIONS OF THE HMAA 69

5.7.1 Interface Agent *69*

5.7.2 Small Agent *70*

5.7.3 Facilitator Agent *72*

5.7.4 Super Agent *72*

5.8 SUMMARY 73

CHAPTER 6 .... **75**

SMALL AGENT HEURISTIC 75

6.1. INTRODUCTION 75
6.2. A PRIORITISED/RANKED-MEETINGS SCHEDULING HEURISTIC 76
6.3. A SIMPLE LOCAL SEARCH REPAIR STRATEGY 85
6.4. IMPLEMENTATION PLATFORM...88
6.5 SUMMARY 90

CHAPTER 7 91

SUPER AGENTS CONSTRUCTION..91

7.1. INTRODUCTION 91
7.2. LGP_SUAS (SUPERAGENTLGP/SUPERAGENTLGP_SP) 92
7.2.1. Parents' Heuristics 94
7.2.2. Crossover Operations..96
7.2.3. Mutation Operation 101
7.2.4. Termination Criterion 105
7.3 LSP_SUAS (SUPERAGENTLSP/ SUPERAGENTLSP_SP) 107
7.3.1. Solution's Heuristic..109
7.3.2. Crossover Operations..111
7.3.3. Mutations Operation 114
7.3.4. Termination Criterion 123
7.4. SUMMARY 125

CHAPTER 8 126

EXPERIMENTAL RESULTS AND EVALUATION...126

8.1 INTRODUCTION..126
8.2. STAGE ONE: SMA EXPERIMENTS..128
8.3. STAGE TWO: HMAA EXPERIMENTS...138
EXPERIMENTS GROUP 1..138
EXPERIMENTS GROUP 2..157
EXPERIMENTS GROUP 3..170
8.6. SUMMARY 173

CHAPTER 9 175

CONCLUSION AND FUTURE WORK...175

9.1 SUMMARY 175
9.2 CONTRIBUTIONS 176

9.3. FUTURE WORK 180

REFERENCES **182**

APPENDIX A **198**

AN HMAA SCREEN SHOTS **198**

APPENDIX B **209**

JAVA CODE OF HMAA IMPLEMENTATION **209**

List of Figures

Fig. 1: Example of DCOP graph 30
Fig. 2: Weightings for period selection heuristics in the HuSSH system 33
Fig. 3: HMAA Architecture 64
Fig. 4: HMAA scenario 65
Fig. 5: HMAA scenario 66
Fig. 6: Sequence diagram for Meeting Scheduling Problem within HMAA 69
Fig. 7: prioritised scheduling 79
Fig. 8: Scheduling Pseudo Code for Optimisation problems 81
Fig. 9: Scheduling Pseudo Code for Search problems 84
Fig. 10: Local Search Pseudo Code 86
Fig. 11: Neighbourhood Function 87
Fig. 12: Pseudo Code for LGP 94
Fig. 13: LGP_parent1 heuristic 95
Fig. 14: LGP_parent2 heuristic 96
Fig. 15: LGP Crossover - the Parents 98
Fig. 16: LGP crossover - the Children 99
Fig. 17: LGP crossover - Child1 100
Fig. 18: LGP crossover - Child2 100
Fig. 19: LGP before mutation - Child2 102
Fig. 20: LGP after mutation-Child2 103
Fig. 21: LGP mutation - Child1 104
Fig. 22: LGP mutation - Child2 104
Fig. 23: Pseudo code for LSP 109
Fig. 24: SMA heuristic 110
Fig. 25: Solution Heuristic 111
Fig. 26: LSP crossover 112
Fig. 27: LSP crossover_neighbour$_1$ 113
Fig. 28: LSP crossover_neighbour$_2$ 113
Fig. 29: LSP mutation$_1$_neighbour$_1$ 115

Fig. 30: LSP $mutation_1$_$neighbour_1$ 116
Fig. 31: LSP $mutation_1$_$neighbour_2$ 116
Fig. 32: LSP $mutation_2$_$neighbour_1$ 118
Fig. 33: LSP $mutation_2$_$neighbour_1$ 119
Fig. 34: LSP $mutation_2$_$neighbour_2$ 119
Fig. 35: LSP $mutation_3$_$neighbour_1$ 121
Fig. 36: LSP $mutation_3$_$neighbour_1$ 122
Fig. 37: LSP $mutation_3$_$neighbour_2$ 122
Fig. 38: The feasibility of the ranking (comparing Stages 1 and 3) 130
Fig. 39: The feasibility of the ranking (comparing Stages 2 and 4) 130
Fig. 40: the feasibility of local search in (2) 131
Fig. 41: the feasibility of local search in (4) 132
Fig. 42: The performance of the FMSH 133
Fig. 43: The performance of Prioritised/ranked heuristic search problem compared with the local consistency approach 136
Fig. 44: Comparing the performance of Prioritised/Ranked-Meetings Scheduling heuristic for Search problem solving with Prioritised/Ranked-Meetings Scheduling heuristic for optimisation problem solving and local search repair strategy 136
Fig. 45: Comparing the performance of Prioritised/ranked heuristic optimisation problem and local search with local consistency approach 137
Fig. 46: Violation reduction 171
Fig. 47: Measurement of rounds/time 173
Fig. 48: First User Interface for FMAF 198
Fig. 49: a list of registered users 199
Fig. 50: the available SUAs. 199
Fig. 51: SMA Main User Interface 200
Fig. 52: The meeting menu 201
Fig. 53: Add Constraint Interface 202
Fig. 54: Add Meeting Interface 202
Fig. 55: Add Attendees Interface 203
Fig. 56: Add Domain Interface 204
Fig. 57: Add Meeting Menu Item 204

Fig. 58: Transcript Text Box 206
Fig. 59: View Meetings Text Box 206
Fig. 60: Local Search Menu Item 207
Fig. 61: SuperagentLGP Interface 207
Fig. 62: SuperagentLGP_SP Interface 208
Fig. 63: SuperagentLSP Inteface 208
Fig. 64: SuperagentLSP_SP Inteface 208

List of Tables

Table 1: Case 1 data table 140
Table 2: Case 1 results Table 143
Table 3: Case 2 data table 143
Table 4: Case 2 results table........ 145
Table 5: Case 3 data table 146
Table 6: Case 3 results table........ 148
Table 7: Case 4 data table 150
Table 8: Case 4 results table........ 152
Table 9: Case 5 data table 153
Table 10: Case 5 results table 156
Table 11: Case 6 data table 160
Table 12: Case 7 data table 162
Table 13: Case 8 data table 164
Table 14: Case 9 data table 166
Table 15: Case 6 results table 166
Table 16: Case 7 results table 167
Table 17: Case 8 results table 167
Table 18: Case 9 results table 168
Table 19: Results table........ 168

Abbreviations

ABT	Asynchronous Backtracking
AI	Artificial Intelligence
ANN	Artificial Neural Network
BDI	Believe Desired Intention
C_H	Hard Constraint
C_S	Soft Constraint
CSP	Constraint Satisfaction Problem
DAI	Distributed Artificial Intelligence
DB	Data Base
DCOP	Distributed Constraint Optimisation Problem
DCR	Distributed Constraint Reasoning
DisCSP	Distributed Constraint Satisfaction
EA	Evolutionary Algorithm
EAP	Evolutionary Automatic Programming
EC	Evolutionary Computation
FA	Facilitator Agent
HMAA	Hybrid Multi-Agent Architecture
GA	Genetic Algorithm
GP	Genetic Programming

HuSSH	Human Selection of Scheduling Heuristics
IA	Interface Agent
ID	IDentification number
IL-MAP	Incremental Limited information exchange Multi-agent Assignment Problem
LGP	Linear Genetic Programming
LS	Local Search
LSP	Local Search Programming
MAS	Multi-Agent Systems
MS	Meeting Scheduling
MSP	Meeting Scheduling Problem
NN	Neural Network
NP	Non Polynomial
P	Polynomial
SMA	SMall Agent
SUA	Super Agent
VCSP	Valued Constraint Satisfaction Problem

Chapter 1

Introduction

1.1 Research Motivation

Recent developments in the area of systems design and software engineering show that a new paradigm "agent systems" is emerging, especially as a solution for more demanding applications. Agent-based computing attracted much attention as a promising technique for distributed, complex and heterogeneous application domains [27, 34, 36, 57, 60, 61]. It has been hailed as "the next significant breakthrough in software development" [13], and "the new revolution in software technology" [73]. Current research on multi-agent systems (MAS) has become mature enough to be applied as a technology for solving problems, in an increasingly wide range of complex applications [27, 93].

Several questions arise regarding the best way to control agents' activities and applications performances: several formal models were proposed to describe the relationships between agents within MAS in the problem-solving process. The main formal representations are *centralised* and *distributed architectures*. In the former, the central agent is responsible for performing the task. Some of the difficulties raised in this type of architecture are because of the high risk of dependency on the controlling element. The number of computations required by the agent in order to store the global knowledge and to compute the tasks determines the size of the agent, which means that centralised agents tend to be large while the costs of sending large agents across the

network may be greater than the benefits they confer [40]. Centralisation needs all the agents to reveal potentially private information to the central server agent, which would bring down the level of the privacy and security in many large systems. Finally, centralised architectures are difficult to integrate into naturally distributed systems.

In decentralised or distributed architectures, the number of distributed agents can be configured into mechanisms of self-organisation without imposing external, centralised controls. The main difficulty is that decision-making is based on limited information about the environment, and does not refer to explicit deliberation. Because of their simplicity and mobility, individuals do not have an explicit representation of the collective task to be achieved [14]. Each of these types of architecture has its advantages and disadvantages that make each of them suitable for some domains and not for others. This situation impels the search for a solution.

In computational complexity theory, NP is the class of decision problems that can be solved by a non-deterministic Turing machine in polynomial time. NP-complete: a decision problem P is said to be NP-complete if (1) P is in NP-class and (2) all problems in the NP-class are reducible to P i.e. other problems in NP can be transformed into a problem P. And finally NP-hard are those problems that satisfy only condition (2) that all problems in the NP-class are reducible [50]. A method of computing solutions to NP-complete/NP-hard problems using the algorithms and computational power available nowadays in reasonable time frame has yet to be discovered [13]. Unfortunately, many practical problems such as route planning, scheduling and the creation of timetables belong to this very class. It is essential that these problems are

solved, and the only possibility of doing this is to use approximation techniques, since no straightforward solution technique is known.

A heuristic solution technique ***is*** an alternative [15]. Derek Partridge [20] has defined the *heuristic* by *"a rule of thumb: a procedure that achieves a certain goal on an acceptable proportion of occasion*". He also maintains that "*in fact, there is more likely some assurance that it won't always work*". It is therefore evident that heuristic strategies are generally powerful, but are not absolutely guaranteed to provide the best (i.e. optimal) solutions, or even to find a solution at all. This demands adopting some optimisation techniques such as Evolutionary Algorithms (EA) or Evolutionary Computation (EC).

Evolutionary computations have received increased interest and have been successfully applied to numerous problems from different domains [8, 11, 18, 32]. This because they offer benefits to researchers of optimisation problems, some of which are the *simplicity* of the approach, the *robust* response to changing circumstances, *flexibility*, and the possibility of application to problems where heuristic solutions are not available or which generally lead to unsatisfactory results. Nowadays, EA is considered to be an adaptable means of problem solution, especially for complex optimisation problems [25].

Researchers have recently adopted another form of problem solving research, "Evolutionary Automatic Programming" (EAP) [66]. Automatic Programming (AP) is best described by Archtur Samuel (Samuel, 1959): "*Tell the computer what to do, not how to do it*". Hence, the ultimate goal is that the programmer only needs to state what an algorithm must do, not how it does it, and the automatic programming algorithm

works out the implementation. EAP is used to refer to those systems that adopt evolutionary computations to automatically generate computer programs, and as such includes Genetic Programming (GP).

Several multi-agent architectures were recently intensively studied and applied to many NP-problems [45], and have consequently become widely accepted. This has motivated the researcher to propose "A Hybrid Multi-Agent Architecture (HMAA) where small agents can be distributed on small devices. The architecture is capable of solving many practical NP-hard problems while preserving the ability of small distributed agents to run on small devices". This research has been undertaken in order to investigate the feasibility of running computationally intensive algorithms (i.e. evolutionary algorithms) that generates new heuristics on multi-agent architectures, while preserving their ability to run on small devices (including mobile devices).

This architecture is hybrid, since it is "*semi-distributed/semi-centralised*" architecture where variables and constraints are distributed among small agents exactly as in distributed architectures. When the small agents become stuck, a centralised control becomes active in which the variables are transferred to a super agent that has a central view of the whole system and possesses much greater computational power and more intensive algorithms that enable it to generate new heuristics that find optimal solutions. The Meeting Scheduling Problem (MSP) has been adopted and investigated in order to examine and validate the idea.

In an MSP each scheduling agent manages the calendar for its user, while the basic objective is to find a common free time slot for all participants in a particular meeting. Where MSP is a NP-hard problem, it is not possible to optimally solve every instance of

MSP in an acceptable time using the algorithms and computing power available nowadays [7]. It has been solved within the MAS environment using heuristics, a solution that holds the promise of finding feasible solutions within a reasonable time [15, 72]. There are several heuristics for MSP, but most of them are of limited success because they use predefined deterministic heuristics which are domain specific, meaning that they work well in some environments and not in others.

MSPs and others have been solved within MAS and suffer from this limitation. Mobile agents within distributed MAS have limited capabilities and cannot perform complicated computations that would generate new solutions. "Overcoming this limitation of having restricted capabilities/heuristics that work well in some environments and not in others; by implementing computationally intensive algorithms on super/central agents, propose computationally intensive algorithms in order to generate new heuristics to be executed by the small agents, while preserving their simplicity and their ability to run on small devices" is another motivating factor for this study.

1.2. Research Question

The main question to be answered by the present research is:

"What is the feasibility of running computationally intensive algorithms for generating new heuristics, such as Genetic Programming, on multi-agent architectures, in order to solve many NP-hard problems while preserving the simplicity and the ability of agents to run on small devices?".

To answer this question, the following sub-questions must be addressed:

(1) Investigate and analyse one of the well-known NP-hard problems, such as the MSP. What are the existing solutions' techniques (heuristics) used to solve this problem? And what are the advantages and disadvantages of these techniques?

(2) Analyse some existing heuristic algorithms and repair strategies for MSPs in order to be able to propose new heuristics.

(3) Propose a new heuristic for MSP that performs scheduling by considering specific parameters and priorities.

(4) Investigate some local search strategies that can be used as repair strategies, and propose a neighbourhood structure (i.e. type of move) which could be implemented on small agents without affecting their mobility.

(5) Use evolutionary approaches to solve such problem as MSPs, an investigation must be carried out into what the evolutionary approach means and how it differs from other problem-solving approaches.

(6) Study the efficiency of such approaches, a hybrid multi-agent architecture must be proposed in which the central agent runs an evolutionary approach and proposes the sequences of moves to be executed by the small/mobile agents.

(7) Implement a prototype for the proposed architecture to evaluate the performance of the proposed heuristic algorithm, and examine the idea of the possibility of implementing computationally intensive algorithms on MAS while preserving the ability to run on small devices such as mobile phones. The extensive algorithm would give the agent the capability of generating new and sometimes better solutions while preserving the ability to run on small agents.

1.3. Major Contributions

This research results in the following:

1. Hybrid Multi-Agent Architecture (HMAA) for solving many NP-hard problems: in the proposed HMAA, variables and constraints are distributed among small agents, and when the small agents become stuck, a centralised control is activated, in which the variables are transferred to a super agent that has larger computational power and implements evolutionary algorithms in order to be able to find an optimal solution.

2. Two types of HMAA have been implemented: (i) DCOP that deals with the NP-problems as optimisation problem, in which the goal is to find an optimal solution that minimises the violation; (ii) DisCSP deals with NP-hard problems as search problem, where the goal is to find a feasible solution.

3. New SMA meeting scheduling heuristic takes into account two parameters: a set of domains and a set of ranked attendees. These parameters are necessary in order to measure the difficulty of a meetings' scheduling. The heuristic starts by ranking the meetings, in order to schedule the most difficult ones respectively.

4. New SMA local search repair strategy for the scheduling process which is activated when the scheduling ends with some violations. This repair strategy applies just to the DCOP framework. In the DisCSP framework the agents do not have to deal with violations of meetings, but only with unscheduled meetings.

5. Small Agent (SMA) for meeting scheduling has been developed. This SMA uses the proposed prioritised/ranked meeting scheduling heuristic and local search repair strategy in order to accomplish the scheduling process of behalf of its user. This SMA is small size and limited capabilities agent, and it could be implemented and run on small devices such as phone mobile device or PDAs. Hence, the users could manage meetings and know their calendar through small mobile devices that implement this proposed SMA.

6. A new concept, “Local Search Programming” (LSP), has been introduced to the evolutionary approach; this concept generates a new heuristic based on the existing one. This method was inspired by both Genetic Programming and local search techniques. In LSP, the programmer seeks to generate new heuristics/programs using local search strategies instead of Genetic Algorithm techniques.
7. Two types of super-agent (LGP_SUA and LSP_SUA) have been implemented in the HMAA, and two SUAs (local and global optima) have been implemented for each type. The first type is LGP_SUA (superagentLGP and superagentLGP_SP), it uses the LGP approach to generate new heuristics. The second is LSP_SUA (superagentLSP and superagentLSP_SP), it uses the LSP approach for the same purpose.
8. Finally, a prototype for the proposed Hybrid Multi-Agent Architecture (HMAA) has been implemented. The architecture employs the proposed meeting scheduling heuristic with the repair strategy on smaller agents, and the four extensive algorithms on super-agents.

The results reveal that this architecture is applicable to many different application domains because of its simplicity and efficiency. Its performance was better than many existing meeting scheduling architectures. It preserves agents' mobility, (i.e. the ability to run on small devices), while implementing evolutionary algorithms. HMAA is very robust in that it can implement more than one optimisation technique without affecting the size of the small agents. Moreover, the proposed evolutionary approach LSP has proved its success in generating new heuristics as LGP, indicating that the proposed LSP is good enough to be applied as a new evolutionary approach using local searching instead of genetic algorithms when there is one parent instead of two.

1.4 Thesis Outlines

This thesis is organised into 10 chapters as follows:

Chapter 2 presents a survey of multi-agent systems, definition of some terminologies and basic concepts, such as agents, intelligent agents, mobile agents, multi-agent systems. The differences between MAS and object and expert system are reviewed, as is the MAS advantages and existing architectures.

Chapter 3 introduces the scheduling problems, and investigates the most used formalisations to define many scheduling problems. A certain amount of investigation for timetabling problem (specifically the Human Selection of Scheduling Heuristics (HuSSH) system) has been documented. An illustration for meeting scheduling problems (MSP) has been done; MSP is a term that has been adopted in order to facilitate the investigation and validation of the concept behind new architecture.

Chapter 4 is a literature review about the evolutionary approaches (EA), the advantages of EA, and the most known disciplines of EA which are: genetic algorithms, genetic programming, and linear genetic programming. An illustration of how each one works and how it differs from other approaches. And finally a new EA concept “local Search Programming (LSP)” is introduced and discussed. Local Search Programming is a new method for generating or modifying the existing heuristics. This method is inspired from GP and local search techniques together by which the system is looking to generate new heuristics/programs using local search techniques instead of GA techniques.

Chapter 5 discusses the proposed HMAA; a clarification for the motivations to this new architecture is stated. The adopted formalisations for solving MSP within HMAA are discussed. The architecture of HMAA and Scenarios of HMAA negotiations and MSP solution approach within HMAA are illustrated. And finally full functions specifications of the HMAA are situated.

Chapter 6 illustrates the small agent proposed heuristic used to solve the meeting scheduling problem. It is a prioritised/ranked heuristic that gives initial solutions for the systems. Moreover, it discusses the proposed local search repair strategy used to optimise the violated solutions. All the algorithms used for scheduling and repair strategy are defined, some examples are illustrated.

Chapter 7 demonstrates super agents who use evolutionary approaches in order to generate new heuristic for small agent. Two types of super-agents have been defined in HMAA, and two super-agents have been implemented for each type. The first type uses the linear genetic programming approach. The second one uses local search

programming approach in generating new heuristic. The algorithms used by each super-agent are also presented.

Chapter 9 illustrates some experiments, done on the implemented (HMAA). Three main groups of experiments have been done; the first group states simple cases with different number of attendees, and different situations or combinations of meetings. The aim of these experiments is to show the feasibility of running the hybrid multi-agent architecture for a large number of attendees. The second group is more complicated cases and susceptible situations, to measure the feasibility of the system in very complicated and limited domain range data. And the final group is randomly selected cases to measure the feasibility of the architecture on different situations. An analysis for these experiments has been done.

Chapter 10 concludes the thesis and outlines future work.

Chapter 2

Multi-Agent Systems

Objectives

- To present a brief introduction to MAS.
- To define basic concepts in MAS.
- To illuminate some benefits for MAS.
- To present well known MAS architectures and their limitations.

2.1 Introduction

John McCarthy is known as the Father of Artificial Intelligence (AI) [24, 48, 85]. In 1956 he became the first person to coin the phrase “Artificial Intelligence”. His belief that computers can reason like humans and his attempts to make this happen has done much to further the development of AI; the modern approach to AI is centred around the concept of a rational agent. AI can be regarded as the study of the principles and design of artificial rational agents.

After about fifteen years, in the mid to late 1970s, Distributed Artificial Intelligence (DAI) evolved and diversified rapidly, since agents are seldom stand-alone systems. In many situations they coexist and interact with other agents in several different ways. Today DAI is an established and promising field of research and practical application bringing together and drawing on results, concepts, and ideas from many disciplines.

Gerhard Weiss [36] defined DAI as: *"the study, construction, and application of multi-agent systems (MAS), that is, systems in which several interacting, intelligent agents pursue some set of goals or perform some set of tasks"*.

The rest of this chapter presents an overview of MAS. Some of the terminology and basic concepts, such as agents, intelligent agents, mobile agents, multi-agent systems and the differences between MAS and object and expert system are reviewed, as is the reasons of utilising MAS. The chapter finally presents the most known MAS architectures and the current problems in these architectures.

2.2. What is an Agent?

Researchers agree that there is no universally accepted definition of the term "agent"; indeed, there is much debate and controversy on this subject [16, 36, 46]. Some researchers define an agent in terms of mental states such as beliefs, capabilities, choices and commitments [90], while others stress the ability of an agent to act autonomously in a dynamic environment [36]. This is due to the fact that different domains vary regarding the importance of the agent's attributes. For example, some agents are designed to undertake the whole task themselves while others must work together; some are mobile, some static; several communicate with each other via messages; some learn and adapt, others do not. However, there is general agreement that autonomy is central to the notion of agency.

Ferber [34] defines an agent as:

"A physical or virtual entity

1) Which is capable of acting in an environment,

2) Which can communicate directly with other agents,
3) Which is driven by a set of tendencies or goals (in the form of individual objectives or of a satisfaction/survival function which it tries to optimise),
4) Which possesses resources of its own,
5) Which is capable of perceiving its environment (but to a limited extent),
6) Which has only a partial representation of this environment (and perhaps none at all),
7) Which possesses skills and can offer services,
8) Which may be able to reproduce itself,
9) Whose behaviour tends towards satisfying its objectives, taking account of the resources and skills available to it and depending on its perception, its representation, and the communication".

2.2. Intelligent Agents

Before clarifying the term "intelligent agent", the question "what is intelligence?" must be answered. This question has been asked for thousands of years, by philosophy as well as by science. Gregory [81] says "*Innumerable tests are available for measuring intelligence, yet no one is quite certain of what intelligence is, or even just what it is that the available tests are measuring".* He also believes that "*Viewed narrowly, there seem to be almost as many definitions of intelligence as there were experts asked to define it*." Eike F Anderson [29] writes that "*the Greek philosopher Aristotle tried to identify intelligence as the rules of "right thinking", logical reasoning, by establishing patterns by which a true precondition would always lead to a true goal state*". He gives

the dictionary definition for intelligence as *"the capacity for understanding; ability to perceive and comprehend meaning*".

Shane Legg [89] identified some of the features involved in intelligence:

- A property of an individual who is interacting with an external environment.
- Ability to succeed or "profit".
- The individual is able to carefully choose their actions in a way that leads to them accomplish their goals.
- Learning, adaptation and experience.

He brings all these key features together and gives what he believes to be the essence of intelligence: "*intelligence measures an agent's ability to achieve goals in a wide range of environments"*.

The definition of *"intelligent agent"* that has become increasingly widely adopted is that in [68]: "*intelligent agent is the computer system that is capable of flexible autonomous actions in some environment in order to meet its design objectives*". Here, "autonomy" means a system that has control over its actions and internal state without direct intervention from humans or other systems.

Flexibility falls into three categories: reactivity, proactiveness and social ability [68].

1) Reactivity: the ability of the intelligent agent to perceive its environment, and *respond in a timely fashion* to changes that occur.
2) Proactiveness: the ability of the intelligent agent to show *goal-directed behaviour* by taking initiatives in order to satisfy the design objectives.

3) Social ability: the ability *to communicate* with the user, system resources and other agents as required in order performing their task(s).

2.3. Mobile Agent

A mobile agent consists of a self-contained piece of software or a program that can *migrate* and *execute* on different machines in a dynamic networked environment, and can sense and act autonomously and proactively in this environment in order to realise a set of goals or tasks [43, 83].

Mobile agents have several strengths. The following is a brief discussion of reasons for using mobile agents [53]:

1) They reduce the network load. Mobile agents allow one to package a conversation and dispatch it to a destination host where the interactions can take place locally.
2) They overcome network latency. Mobile agents offer a solution, since they can be dispatched from a central controller to act locally and directly execute the controller's directions.
3) They encapsulate protocols. When data are exchanged in a distributed system, each host owns the code that implements the protocols needed to properly code outgoing data and interpret incoming data. However, as protocols evolve to accommodate new efficiencies or security requirements, it is a cumbersome if not impossible task to upgrade protocol code properly. The result is often that protocols become a legacy problem. Mobile agents, on the other hand, are able to move to remote hosts in order to establish “channels” based on proprietary protocols.

4) They execute asynchronously and autonomously. Due to fragile and expensive wireless network connections, a continuous open connection between a mobile device and a fixed network will not be always feasible. In this case the task of the mobile user can be embedded in mobile agents, which can then be dispatched into the fixed network and can operate asynchronously and autonomously to accomplish the task. At a later stage the mobile user can reconnect and collect the agent with the results.
5) They adapt dynamically. Mobile agents have the ability to sense their execution environment and react autonomously to changes.
6) They are naturally heterogeneous. Mobile agents are generally independent of the computer and the transport layer and depend only on their execution environment. Hence they can perform efficiently in any type of heterogeneous network.
7) They are robust and fault-tolerant. The dynamic reactivity of mobile agents to unfavourable situations makes it easier to build robust and fault-tolerant distributed systems. If a host is being shut down, all agents executing on that machine will be warned and given time to dispatch and continue their operation on another host in the network.

2.4. Agents and Objects

The key advances in program design and development over the past three decades are in the field of abstract data types such as object-oriented programming, a powerful example of such abstractions. Wooldridge [60] said that "*probably the single most compelling argument in favour of agents for software engineering is that they represent*

yet another such abstraction. Just as many systems may naturally be understood and modelled as a collection of interacting but passive objects, so many other systems may be naturally understood and modelled as a collection of interacting autonomous agents".

It is not easy to recognise the differences between agents and objects. Silva et al. [91] defines an object as *"a passive or reactive element that has state and behaviour and can be related to other elements*". Objects, then, are entities that encapsulate some state, are able to perform actions or methods in this state, and communicate by message passing. And an agent is *"an autonomous, adaptive and interactive element that has a mental state".*

The differences between objects and agents according to [16, 61, 91] are:

- A different degree of autonomy. The object shows autonomy over its state, but does not exhibit control over its behaviour (it does not decide when to execute the method), object invokes methods from another object. Agents, on the other hand, request actions to be performed (the agent itself decides to initiate specific actions). However, agents can be implemented using object-oriented techniques by building some kinds of decision-making about whether to execute a method or initiate a specific action in the method of the agent itself [16, 91].
- An agent fixes the (mental) state of the element to consist of components such as beliefs, capabilities and decisions. The standard object model has nothing to say about how to build systems that integrate the notion of flexible (reactive, pro-activeness, social) autonomous behaviour. But again, object-oriented programs that integrate these types of behaviours could be built.

On the one hand, agents and objects do compete in the sense that agent technology is more appropriate than object technology for applications [60]. On the other hand it must be said that the agent-oriented view is complementary to the object-oriented one due to the fact that developers typically implement agents using object-oriented techniques.

2.5. Agents and Expert Systems

An agent is capable of deciding independently what to do in order to solve a problem but it cannot be considered as an expert system capable of solving problems or giving advice in some knowledge-rich domain [74]. The followings are the main differences between expert systems and agents [59]:

- Expert systems are inherently disembodied, which means they do not interact directly with any environment, but rather obtain their information through a user acting and giving feedback or advice to a third party.
- Unlike agent systems, expert systems are not required to be capable of acting or co-operating with other agents.
- Expert systems are not usually required to operate in anything like real-time.

2.6. What is an MAS?

An MAS is constructed as a society of agents with capabilities of communication and collaboration that interact in order to solve a common problem. Jennings defines MAS as "*a loosely-coupled network of problem solvers that work together to solve problems that are beyond their individual capabilities*" [46].

Omicini [1] has defined the MAS as "*Ensembles of autonomous agents acting and working independently from each other, each representing an independent locus of*

control of the all system. Each agent tries to accomplish its own task(s) and will typically need to interact with other agents and its surrounding environment in order to obtain access to information/ services that it does not need process to coordinate its activities to ensure its goal can be met".

To enable MAS to solve problems coherently, the agents must *communicate* amongst themselves, *coordinate* their activities and *negotiate* once they find themselves in conflict. Conflicts may result from simple competition for limited resources or from more complex situations in which agents disagree because of discrepancies between their domains of expertise. Coordination is required to determine organisational structure among a group of agents and for task and resource allocation, while negotiation is necessary for the detection and resolution of conflicts.

2.7. Why MAS

Developers have discovered that distributed computations are easer to understand and to develop, especially when the problem being solved is itself distributed. Sometimes distribution can lead to computational algorithms that might not have been discovered using centralised approaches. Moreover, the centralised approach is sometimes impossible, since systems and data are distributed, huge in extent and comprise many components.

Traditional Artificial intelligence (AI) has been concerned with how an agent can be constructed to function intelligently with a single locus of internal reasoning. But intelligent systems do not work in isolation, they work in social terms.

The factors driving the increasing interest in MAS research are as follows:

1) Some problems are too large to be solved by a centralised single agent, due to resource limitations or the sheer risk of having one centralised system that could fail at critical times.
2) Real-life problems are usually physically or functionally distributed.
3) MAS's allow for the interconnection and interoperation of multiple existing legacy systems (e.g., expert systems, decision support systems, etc.).
4) MAS's provide solutions in situations where expertise is distributed, for example in health care provisioning and manufacturing.
5) MAS's enhance speed, reliability (the capacity to recover from the failure of individual components, with graceful degradation in performance), extensibility (the capacity to alter the number of processors applied to a problem), and the ability to tolerate uncertain data and knowledge.
6) MAS's provide solutions to problems that can naturally be regarded as a society of autonomous interacting components-agents. For example, in meeting scheduling, a scheduling agent that manages the calendar of its user can be regarded as autonomous and as interacting with other similar agents that manage calendars of different users.
7) MAS's enhance performance in the areas of computational efficiency, reliability (graceful recovery of component failures, because agents with redundant capabilities or appropriate inter-agent coordination are found dynamically, as when they take up the responsibilities of agents that fail), extensibility (because the number and the capabilities of agents working on a problem can be altered), robustness (the system’s ability to tolerate uncertainty, because suitable information is exchanged among agents), maintainability (a system composed of

multiple components-agents is easier to maintain because of its modularity), flexibility (because agents with different abilities can adaptively organise to solve the current problem) and reuse (because functionally specific agents can be reused in different agent teams to solve different problems).

2.8. Multi-Agent Architectures

As multi-agent systems become more complex, questions arise about the best way to control agents' activities, and thus application performance. Any multi-agent process can be performed using centralised or distributed MAS systems' architectures [14]. Each one of them has its own advantages and disadvantages. The following discusses how each works to accomplish a process such as scheduling, and what the strengths and weaknesses of each architecture are.

In the centralised approach the tasks are performed by a single agent that has a global view of the system. This agent must accomplish the task to solve the given problem and distribute the results to all the agents. To do that, it must have global knowledge of the environment, as well as all the agents' private information. This architecture is preferable in many applications, since its stability, simplicity, as well as its provision of up to minute information, could easily optimise solutions. It would also improve resource utilisation.

However there are also problems inherent in centralisation. One concerns the significant impact on the amount of computation required by an agent to store global knowledge and to compute and perform the scheduling process. This results in the consumption of a huge amount of computing power and time, which affects the agent’s size. It also

needs all agents to reveal potentially private information to the central server agent, which would crash the level of the privacy and security in many large systems. All of these factors result in difficulties when such agents are used in naturally distributed systems [33].

The agent-oriented paradigm for software engineering overcomes these difficulties provides a basis for the construction of extremely large, complex systems in which components can be naturally distributed across a network of heterogeneous computers without requiring a complete analysis of their interactions [52]. In distributed architectures the tasks are not the responsibility of one agent but of many. Hence, all agents accomplish the scheduling of their own tasks according to what they know about the environment. This could involve more privacy than the centralised architecture. Allocating tasks to several small agents would be more applicable for distributed or very large applications, and would moreover reduce the computational power needs for each small agent, which would in turn preserve their mobility and their ability to run on small devices. Furthermore, distribution can lead to computational algorithms that might not have been discovered using centralised approaches.

Despite their advantages, distributed architectures have several difficulties. Neither up to date information nor the complete range of resources is available to all agents. Consequently information and computation is localised and communication limited; diversified goals also present significant challenges to the design of systems capable of achieving high levels of global utility since they make independent decisions based on local objectives which may result in conflicts. It is consequently sometimes very difficult to find a global optimal scheduling.

2.9 Summary

This chapter presents an overview of MAS that can be defined as a group of agents work together to solve problems beyond their individual capabilities. Although agents represent abstract software like objects, they cannot be implemented using object-oriented techniques. Agent implementation can be performed by building some kind of decision-making methods and integrating different autonomous behaviours. This, however, does not mean that agents can be considered as Expert systems because (1) agents can interact directly with environments, (2) communicate and cooperate with other agents and (3) respond in a timely fashion.

In MAS, Two architectures are well known and widely used; centralised and distributed architectures. Both architectures have advantages and disadvantages that make each of them suitable for certain situations and unfeasible for others. This motivates us to propose new MAS architecture that can cope with different wide situation.

Next chapter will investigate some problems that were tackled by MAS and will inspect some scheduling problems focusing on meeting scheduling problems. This problem will be adopted to validate and examine the proposed architecture in this research study.

Chapter 3

Scheduling Problems

Objectives

- To introduce the scheduling problem.
- To illustrate some formalisations used for scheduling problem.
- To present the HuSSH system.
- To illuminate the meeting scheduling problem.
- To present frameworks for solving meeting scheduling problems.

3.1. Introduction

In complexity theory, a distinction is made between *optimisation* problems and *decision* problems. Decision problems are often referred to as yes-no problems, while the intention of optimisation problems is to find objects that minimise or maximise the value of objective functions under constraints [95]. Michael remarks [67] that "*many scheduling problems do not have a polynomial time algorithm; these problems are the so-called NP-hard problems*". Numerous models and solutions to the various scheduling problems have been developed, ranging from exact methods such as branch-and-bound to heuristic and meta-heuristic techniques.

Solutions that satisfy the problem constraints are called *feasible*. Constraints are relationships among the entities and can be classified as hard or soft. On the one hand

hard constraints must not be violated under any circumstances. On the other, it is desirable that soft constraints are satisfied as much as possible, but if any of them are violated, a penalty will be applied and the solution will still be considered as being feasible [97].

In practice, the scheduling activity can be regarded as a *search problem* [65] for which it is necessary to find any feasible schedule, or as an *optimisation problem* for which the best feasible schedule is sought. The best solution is often defined to be the one with the lowest penalty (due to violation of the soft constraints).

The class of scheduling problems includes a wide variety of problems such as machine scheduling, events scheduling, timetabling, and meeting scheduling.

3.2. Formalisation of Scheduling Problem

The scheduling problem can be solved within one of the two main multi-agent systems architectures, *centralised* and *distributed* architectures. Each of these has its own advantages and disadvantages, as discussed in Section 2.8. A wide variety of centralised architecture solutions defined the scheduling problem as a *Constraint Satisfaction Problem (CSP)*, the solution to which involves finding an assignment of values to all variables such that all constraints are satisfied [54]. While the distributed architectures solutions defined the scheduling problem as *Distributed Constraint Reasoning (DCR)*. This is a CSP in which the variables and constraints are distributed among a network of automated agents [63]. DCR has been proposed as a way to model and reason about the interactions between agents' local decisions.

This section investigates the formalisations used to define many scheduling problems: CSP, COP, DCR, Distributed Constraint Satisfaction Problem (DisCSP) and the Distributed Constraint Optimisation Problem (DCOP).

3.2.1. Constraint Satisfaction Problem (CSP)

CSP [54, 63, 96] consists of n variables V= $\{x_1,x_2 ..., x_n\}$ whose values are taken from the finite, discrete domains D=$\{D_1, D_2 ..., D_n\}$ and a set of constraints on their values R=$\{R_1, R_2,..., R_m,\}$ where each $R_i(x_1,..., x_k)$ is a predicate on the cartesian product $(D_{i_1} \times \cdots \times D_{i_k})$ that returns that returns true if the value assignments of the variables satisfies the constraint. Solving a CSP is equivalent to find an assignment of values to all variables such that all constraints are satisfied.

3.2.2. Constraint Optimisation Problem (COP)

A COP [9, 51] is 4-tuples $< V, D, R, O >$ where V is a finite set of variables V=$\{x_1,x_2,...,x_m\}$. D is a set of domains D1, D2, ... , Dm. Each domain Di contains the finite set of values which can be assigned to variable x_i. R is a set of constraints. Each constraint involves some variables and defines a non-negative cost for every possible value combination of these variables. O is the objective function assigning a numerical quality value to a solution. An assignment is a pair including a variable, and a value from that variable's domain. A partial assignment is a set of assignments in which each variable appears at most once.

The cost of a partial assignment is computed over all constraints that involve only variables that appear in the partial assignment. Each such constraint defines some cost for the value assignments detailed in the partial assignment. All these costs are

accumulated, and the sum is denoted as the cost of the partial assignment. A partial assignment that includes all the variables is a full assignment and a full assignment with minimal cost is a solution. Intuitively, the optimisation problem is harder than the satisfaction problems, but both are NP-complete.

3.2.3 Distributed Constraint Reasoning (DCR)

This includes two main families of problems: *DisCSPs* and *DCOP*s.

3.2.3.1 Distributed Constraint Satisfaction Problems (DisCSPs)

DisCSP [63] is a CSP in which the variables and constraints are distributed amongst a network of automated agents. DisCSP has been proposed as a way to model and discuss the interactions between agents' local decisions. DisCSPs are composed of agents A={A_1, $A_2 ..., A_k$}, where each agent *Ai* has its own local variables $x_{i1}, x_{i2} ... x_{in}$, whose values are taken from the finite, discrete domains D_1, D_2 ..., D_n, and connected by constraints among variables of different agents.

Each agent controls or assigns the value of its variables, while agents must coordinate their choice of values so that a global objective function is satisfied. The global objective function is modelled as a set of constraints in which each agent is only assumed to have knowledge of the constraints in which its variable is involved. Every constraint is required to be true or false. In this limited representation, an assignment of values to variables must satisfy all constraints in order to be considered a solution.

This representation is inadequate for many real-world problems in which it is impossible to satisfy all constraints. For these types of problems we may wish to obtain

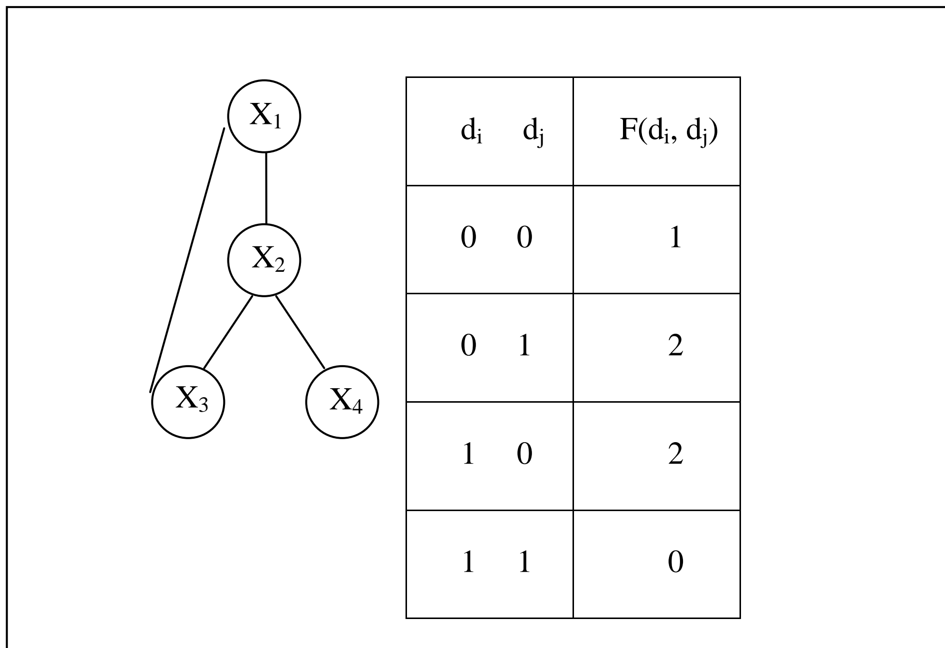
solutions that minimise the number of unsatisfied constraints. The next section presents a model that is able to deal with this type of optimisation problem.

3.2.3.2 Distributed Constraint Optimisation Problems (DCOPs)

DCOP [62, 70] is a distributed version of the COP (optimise the constraints), which is in turn derived from the CSP (satisfy all the constraints). A DCOP consists of n variables V={x_1, x_2..., x_n,}, each assigned to an agent, where the values of the variables are taken from a discrete domain D={D_1, D_2 ..., D_n}, respectively. The goal of the agents is to choose values for the variables to *optimise* (i.e. maximise or minimise) a global objective function. This function is described as the sum of a set of valued constraints related to pairs of variables. Thus, for a pair of variables x_i, x_j there is a cost function, defined as:

$$Fij : Di \times Dj \rightarrow N \qquad (3.1)$$

DCOP generalises the DisCSP, which has a limited power of representation since every constraint is required to be Boolean (i.e. satisfied or not satisfied). On the other hand, the cost functions in DCOP are the analogue of constraints from DisCSP. They take values of variables as input and, instead of returning "satisfied or unsatisfied", they return a valuation as a non-negative number (i.e. how much it does not satisfy).



d_i d_j	$F(d_i, d_j)$
0 0	1
0 1	2
1 0	2
1 1	0

Fig. 1: Example of DCOP graph

Fig. 1 shows an example of DCOP with four agents where each has a single variable with domain {0, 1}. The objective is to find an assignment A^* of values to variables such that the aggregate cost F (equation 3.2) is minimised.

$$F(A) = \sum_{xi,xj \in V} fij(di,dj), \text{ where } x_i \leftarrow d_i \text{ , } x_j \leftarrow d_j \text{ in } A \qquad (3.2)$$

For example, if all variables are assigned to the value 0 in A, then *F(A)*=4. If all variables are assigned to the value 1 in A, then *F(A)=0*, which is the optimal solution.

In DisCSP and DCOP some of the constraints may be public, and some are secrets of different participating agents. A simple example of such problem is meeting scheduling with secret constraints, meeting scheduling solving takes into account the fact that human agents often do not want to share full calendar information with other participants.

3.3. Timetabling

Roman Barták, and Hana Rudová [84] state that timetabling is a special case of scheduling. They have defined timetabling as: "*the allocation of given resources to objects being placed in space-time, in such a way as to satisfy as nearly as possible a set of desirable objectives*". A timetable shows at what time particular events are to take place. It does not necessarily imply an allocation of resources. In comparison with scheduling, in timetabling the importance of the resource allocation is restrained, although it is part of the scheduling process.

A certain amount of investigation for Human Selection of Scheduling Heuristics (HuSSH) timetabling system has been done, and illustrated in the following subsection.

3.3.1 The HuSSH System

Examination timetabling is concerned with putting exams into a limited number of timeslots (periods) subject to a set of constraints. The generally accepted hard constraints are:

1. No student can sit two exams simultaneously.
2. The scheduled exams must not exceed the room capacity.
3. Order constraints.
4. Room or period requirements.

The HuSSH system has been designed as a toolbox for designing heuristics for examination timetabling by users [17, 86]. Ahmadi et al have defined the HuSSH

system as a multidisciplinary research involving computer science and psychology for solving scheduling problems (examination timetabling) by combining the intelligence and flexibility of human schedulers with the power of automated scheduling systems [17, 86]. The aims of this project were:

1. To provide schedulers with a toolbox of intuitive heuristics to enrich their set of simple moves and heuristics at the level of construction and improvement of the schedule.

2. To provide expert schedulers with a visual representation of the problem for the better understanding of the data and the constraints using design principles from cognitive science.

3. The selection of heuristics by expert humans based on the characteristics of different contexts.

4. Learning human strategies regarding the selection of heuristics.

HuSSH is based on partitioning sequential heuristics for examination timetabling problems to *exam selection*, *period selection* and *room selection heuristics*. The user intervenes in the construction of examination timetabling in the HuSSH system at the level of heuristics selection. Some of the heuristics in the HuSSH system have proved to be difficult for the user to set up due to the high number of parameters and the complexity of their interrelationships. This is the case, for example, with the *period selection* heuristic. The period selection penalty function represents the potential cost of scheduling an exam in a period as the weighted sum of the violations in that period.

The system enables the user to adjust the weightings as shown in Fig. 2 to reflect the importance of different constraints at different stages of the solution process. The extensive changes in the weightings make it very difficult for the user to analyse the effect of each change, and random-like changes in the behaviour of the heuristics may appear.

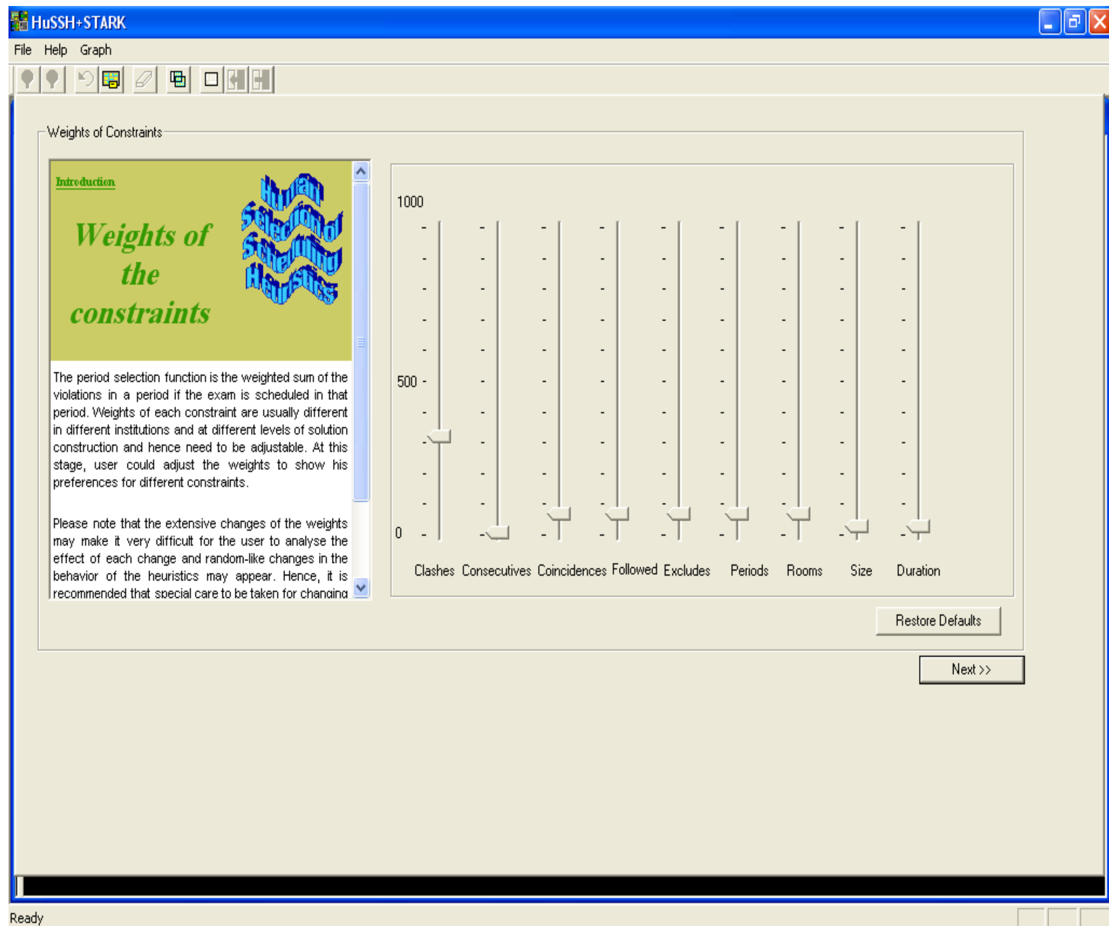


Fig. 2: Weightings for period selection heuristics in the HuSSH system

The present researcher contributions to this project include the design and implementation of an Artificial Neural Network (ANN). The results are published in [88]. This project has used artificial neural network (ANN) to find a relationship between the weightings and the final timetabling violations. The ANN provides

feedback to the HuSSH user in order to select a better set of weightings through a simple user interface.

The ANN has been built on the Enterprise miner/SAS platform. It is a Multi-Layer Perceptron Neural Network (MLP) NN with four hidden layers. The weightings of the network are adjusted using the back propagation (feedforward) algorithm. The NN has nine inputs which are the weightings of different parameters for the period heuristic, and nine target variables which are the violations in timetabling.

The SAS generates C code which can be integrated within the HuSSH system. The final version will use the neural network outcome as an animated chart that will show the changes in the violations based on the user's changes of the sliding bars of weightings.

A well-known distributed scheduling problem, closely related to timetabling, is meeting scheduling or calendar management. Meeting scheduling is a distributed scheduling problem in which each person wants to schedule their meetings with others who want to accomplish the same task. Researchers engaged in the present study have solved this problem using a multi-agent system. The next section explains meeting scheduling at greater length and outlines some techniques to solve it within the multi-agent system architecture.

3.4. Meeting Scheduling Problem

3.4.1 Definition

In the MSP each scheduler manages its calendar, while the basic objective is to find a common free time slot for all participants in a particular meeting. Meeting scheduling is

a time-consuming routine task that, when delegated to a personal assistant agent, promises to significantly reduce the daily cognitive load [78]. It is not possible to optimally solve every instance of MSP within an acceptable time using the algorithms and computing power presently available, since MSP is a NP-hard problem [6, 7, 38]. It has therefore been solved within the MAS environment using heuristics that hold the promise of finding feasible solutions within a reasonable time [15].

There are several solutions for meeting scheduling problems [30, 37, 38, 42, 65, 78, 79], but they have had only limited success because most existing meeting scheduling systems use predefined deterministic methods or heuristics that are domain specific, meaning that they work well in some environments and not in others. In practical environments, however, meeting scheduling is an ongoing reactive process, which means that the presence of real time information continually forces reconsideration. In other words it needs the agent to perceive its environment and response in a timely fashion to the changes that occur.

To manage the meeting scheduling process effectively, multiple agents must reason and communicate their local schedules and their individual calendar management in order to obtain good global performance. Many real-world problems can be represented as CSPs [54, 63] which are not distributed. On the other hand, multi-agent systems in real-world problems often present themselves in distributed form. Researchers have proposed DCR as a key paradigm and a theoretical foundation for problems in multi-agent systems [78]. In DCR a set of *variables* is distributed among a set of agents, and set of variables, *constraints*, requires agents to coordinate their value choices.

3.4.2. Frameworks for Solving MSP

Hassine et al. [6, 7, 37, 38] have devised a new approach based on distributed reinforcement of node and arc consistency (DRAC) to solve MSPs. Their work focuses mainly on satisfying meetings hosts' preferences as much as possible while taking into consideration all users' availability, minimising the number of messages exchanged and retaining as much of the privacy of the users as possible. A static and deterministic version of this approach was initially proposed [37], in which the authors deal with the problem as a distributed one.

They formalised the MS problem as a VCSP (Valued Constraint Satisfaction Problem) framework. This formalisation is a generalisation of CSP to the over-constrained problems by giving a weight or a valuation to each constraint that reflects the importance of satisfying that constraint. Their formalisation is defined by a quintuple (X, D, C, S, φ), where X is a set of meetings, D a set of possible dates (domain) for X, C is composed of two types of constraints (hard an soft constraints), S = (E, $\otimes$, ϕ) a valuation structure used to order the solutions obtained to the problem and φ: C→ E. E is the set of possible valuations, ϕ a total order on E; $\perp \in$ E corresponds to the maximal satisfaction and $\otimes$ is an aggregation operator used to aggregate valuations.

The local goal is to schedule meetings such that all the hard constraints C_H are satisfied, while trying to maximise global utility (the sum of the initiator preferences). The global goal is to schedule the maximum number of meetings satisfying all the inter-agent constraints (these are represented by a set of strong constraints, i.e. equality constraints). Their approach was that more than one initiator agent can be activated at the same time (dynamic MS). Each initiator starts by sending reduced timeslots (by reinforcement

node consistency) to the attendees and collecting the ranked time slots from them. The initiator then proposes timeslot that maximise utility, and collects responses (positive or negative).

Each time the initiator receives at least one negative answer it must change its proposal and decrease its degree of preferences. If it receives no negative responses, it will first update its hard constraints by adding this proposal, then update the dates of its as yet unscheduled meetings by eliminating the dates corresponding to that one (arc-consistency). The process continues until all meetings have been scheduled or proving that some of them could not be held of all agents in the system.

The initiator reduces the time slots of the corresponding meetings by removing the infeasible time slots based on the hard constraints. This process starts from the most constrained meetings according to their hard constraints (node consistency reinforcement). If the time slots become empty after the reduction process, then the meeting cannot take place, so the time slots must be changed. After the node consistency reinforcement phase, the initiator agent deletes all the dates that are already used for more important meetings; this is how the arc consistency reinforcement phase is done.

When the attendee receives the reduced time slots, he starts first by reinforcement node consistency, and then ranks the obtained slots according to its preference. When the attendees receive the proposal, they will send either positive answer to the initiator (if he does not accept the same proposal for another meeting) or negative answer.

Amnon Meisels and Oz Lavee in their work [65] have defined the Meeting scheduling problem as a distributed constraints satisfaction search problem (**DisCSP**) and they used

asynchronous backtracking (ABT) for solving MSP. Agents participate in multiple meetings, where each meeting is represented by a variable that needs to be assigned a time-slot. Additional information could be obtained in the form of Nogoods messages. During search for a consistent schedule for all meetings, agents can generate and send additional Nogoods to those sent by the ABT algorithm.

Their approach was that every agent of the meeting scheduling problem includes multiple variables, one for each meeting it attends. All agents are assumed to be ordered successively and variables of each agent are ordered successively too, so that the variables of agent A_{i+1} follow successively the variables of agent A_i.

The initiator assigns values to all local variables and sends proposals to another agent in the form of "ok?" message. The attendee updates the AgentView (which contains the most recent assignments received from agents with higher priority) with the received assignment and removes all eliminating explanations in all the local variables that contain the out of date assignment of the received variable.

When backtrack message (reply message from the attendees) is received with Nogood proposal. The initiator checks the consistency of the received Nogood with the AgentView. If it is consistent, then initiator updates the relevant assignments in the AgentView that it is Nogood assignment. It also removes the eliminated values (which are because of the relevant assignment) from the relevant local variables. Then it assigns values to all the local variables, checking that all the eliminators in all the variables are consistent with the AgentView.

If no consistent value is found, the eliminators of the current variable are resolved to form a Nogood, When the Nogood points to a local variable X_{ik} then a backjump to X_{ik}

will be performed. The backjump requires the removal of all eliminators from all the local variables $X_{i\ k+1..j}$, that were jumped over This procedure implements the backjumping algorithm for multi local variables.

If Nogood is not local variable, then the Nogood is sent in a backtrack message to the initiator of it, and the assignment of the Nogood is removed from the AgentView. Next, the local process for consistent assignments to all local variables starts from the beginning. When consistent assignments for all local variables have been found, all new assignments are sent by an (ok?) messages to all the attendees.

Modi et al. [78, 79] provided an approach to (*multi-agent meeting scheduling with rescheduling*) using (DCR); which is an extension to DisCSP (Distributed Constraint Satisfaction Problem). They formalised the MS problem as IL-MAP (Incremental Limited information exchange Multi-agent Assignment Problem) which is a special form of Distributed Constraint Reasoning (DCR). The major difference is that MAP allows a variable to be shared among a set of agents (participants) while DisCSP assigns each variable to a unique agent.

IL-MAP requires agents to assign values to variables where multiple agents must agree on the value assignments, and there is incremental scheduling of activities and there exist privacy restrictions on information exchange (initiator does not communicate information about meeting to any agent which is not a participant in that meeting). The basic distributed protocol is that the initiator proposes assignment to others who agree or refuse the proposed assignment based on their own existing assignment.

Their approach was that the initiator manages the negotiation of the meeting by proposing times (free timeslots ranked according to a complex set of user preferences)

and collecting responses (pending, impossible) from other attendees in a sequence of rounds and tries to find a mutually acceptable time. If time is found, the meeting is confirmed. Otherwise the process continues in rounds until the initiator runs out of times to propose in which case the process terminate with failure.

Attendees may tentatively bump a confirmed meeting in favour of a new meeting in order to decrease the possibility of scheduling failure. The attendees will bump the meeting if the scheduling difficulty for the new meeting is greater than the scheduling difficulty for the existing meeting. If the new meeting is confirmed in the timeslot, the bumped one will be rescheduled by the initiator. If the new meeting is confirmed in other slot of time or fails to be scheduled, the bumped one is re-instated into the original slot. There are many other researchers who have used one of the standard definitions to define MSP, such as Adrian Petcu and Melinda et al. [5, 58, 76].

Adrian Petcu has modelled the Meeting scheduling problem as Constraint Optimisation Problem (COP) where evaluation function maps each instantiation of variables of a constraint to a real number called utility, and simple aggregate function can sum up all of the utilities for all of the constraints (for a particular solution) which gives a way to measure the « goodness » of a solution. The task is to produce the « best » solution, that maximises the aggregated utility. They present a new complete method for distributed constraint optimisation, that extends the sum-product algorithm (which is true for tree-shaped constraint networks) to arbitrary topologies using a pseudotree arrangement of the problem graph.

Melinda et al. [58] in their work "*Active Preference Learning for Personalised Calendar Scheduling Assistance*" have modelled the meeting scheduling problem as a

standard constraint satisfaction problem (CSP), represented by a set of *variables* {day, start, dur}, for each variable a *domain* specifying its possible values (Dday = {mon, tue, wed, thu, fri}, Dstart =[12:00am,11:59pm], Ddur = [0,1440] min), a set of *constraints* on one or more variables, They have defined the calendar as a set of meetings. The scheduling problem (or meeting request) is a pair S = ⟨C, X⟩, where C is a calendar and X is a set of constraints over day, start, and dur.

All of the mentioned meeting scheduling solution techniques use predefined heuristics. Because a single heuristic cannot grantee to provide a solution in all domains, these predefined heuristics work well in specific domains and may not work in others. The proposed HMAA overcome this limitation by having a super agent that is able to generate a new heuristic dynamically when the predefined one failed. The new heuristic is constructed from the predefined one using an evolutional approach. In this way the newly generated heuristic is at least as good as the predefined one used to construct it.

3.5. Commercial Products for MSP

Commercially, there are several existing meeting scheduling software products but many of them have disadvantages which include: (1) they are considered as computational calendars solely with some special features, these products are not truly autonomous agent and is not capable of communicating and negotiating with other agents in order to schedule meetings in a distributed way taking into account the users preferences and calendar availability [35, 55]. (2) several need high computational power in order to accomplish the corresponding task; such example is Profit Scheduler for Meetings™ (PSforM) [80], (3) some of them require internet connection such as

TimeBridge [94], (4) in the case of dealing with mobile devices or PDAs, the available software can only send a text message or iCalender file format for PDAs. An example is Snap Schedule Employee Scheduling Software [92].

One of the well known MSP commercial product is Microsoft Outlook [55], there are number of problems in this product. Firstly, it ignores the negotiation step and issues of uncertainty about other users' calendars. Furthermore, the scheduling features in Microsoft Outlook rely largely on an open calendar systems; where users are required to make their calendars publicly viewable within the organisation. Finally, a major limitation of Microsoft Outlook is that it will not consider moving existing meetings on behalf of the user.

In turn, the proposed architecture overcomes most of the above disadvantages of the commercial meeting scheduling software. In the proposed architecture, the computational power needed for meeting scheduling is reduced; this is due to the fact that the size of the small agent, which is responsible to accomplish the scheduling process on behalf of the user, is very small. No internet connection is required to perform the scheduling, the small agent can be run on mobile devices, and users do not need to reveal potentially private information to the rest of the agents. Agents possibly run on mobile devices collaborate to schedule meetings on behalf of their users.

3.6. Summary

This chapter presents the scheduling problems; it starts with general definition for scheduling problems, and discusses the most accepted formalisations used to define the scheduling problems such as:

- CSP (consists of *variables* whose values are taken from the finite, discrete *domains* and a set of *constraints* on their values, and the goal is to find *feasible* values for variables that satisfy the constraints);
- COP (consists of *variables* whose values are taken from the finite, discrete *domains* and a set of *weighted constraints* on their values, COP can be defined as a regular CSP which constraints are weighted and the goal is to find a solution *maximising* the weight of satisfied constraints);
- DisCSP is a CSP in which the *variables* and *constraints* are *distributed* amongst a network of automated agents; the goal is to obtain solutions that *minimise the number of unsatisfied constraints*;
- DCOP is a distributed version of the COP (optimise the constraints), the goal of the agents is to choose values for the variables to *optimise (i.e. maximise or minimise) a global objective function*. This function is described as the sum of a set of valued constraints related to pairs of variables.

After that the chapter discusses two scheduling problems: timetabling and the meeting scheduling problem. It presents the author's work on timetabling, especially her contribution to the design and implementation of the HuSSH timetabling system. Then an overview of meeting scheduling problem (MSP) has been done. MSP has been adopted in order to facilitate the investigation and validation of the concept behind the new framework, HMAA, proposed in this research. Evolutionary algorithms that are discussed next chapter.

Chapter 4

Evolutionary Algorithms

Objectives

- To present the need for EAs.
- To present some EAs and illustrate the differences between them.
- To introduce a new concept in EA, called LSP.
- To illustrate the motivations and the characteristics of LSP.

4.1. Introduction

Evolutionary Algorithms (EA) and Evolutionary Computations (EC) have received increased interest, and are being successfully applied to numerous problems from different domains, including optimisation, automatic programming, machine learning, operations research, bioinformatics, and social systems [8]. There are several reasons for this: (1) it offers benefits for the optimisation of researchers' problems, (2) simplicity of the approach, (3) robust response to changing circumstances, (4) flexibility, and (5) the fact that it can be applied to problems where heuristic solutions are not available or generally lead to unsatisfactory results.

Many disciplines are grouped under EAs, all of which share a common conceptual base of *simulating the evolution of individual structures* via processes of selection, mutation and reproduction [3, 4]. These disciplines are: evolution strategies (Rechenberg 1964)

[25], evolutionary programming (Fogel, Owens and Walsh 1965), genetic algorithms (Holland 1975) [47] and genetic programming (Koza 1992) [49].

Section 4.2 discusses the advantages of EA, and the following ones (4.3, 4.4 and 7.5) illustrate some EAs and how each one does work, and differs from other approaches. Section 4.6 illustrates how GP differ from heuristic approach. Finally Section 4.7 introduces and discusses a new EA concept called "Local Search Programming" (LSP).

4.2. Advantages of Evolutionary Algorithms

EAs have many benefits for problem solving research, some of which are [8, 32]:

1. Traditional methods of optimisation do not respond well to dynamic changes in problem environments, and often require a complete restart in order to provide a solution. In contrast, EAs can be used to adapt solutions to changing circumstances by generating new solutions based on existing ones.

2. EAs offer a framework that makes it relatively easy to incorporate prior knowledge of the problem to produce a more efficient exploration of the state space of possible solutions.

3. EAs can be combined with other optimisation techniques, and can also be extended to multi-objective optimisation, which is of special interest in most financial applications.

4. The evaluation of each solution can be handled in parallel, and only selection (which requires at least pair wise competition) requires some serial processing.

5. EAs are able to address problems for which there are no human experts. However, human expertise should be used when it is available.

6. EA methods are inherently quantitative; therefore they are well suited for parameter optimisation.

7. They are simple and robust.

4.3. Genetic Algorithms (GA)

The GA originated in 1975 with John Holland work [39, 47]. It is one of the algorithms that *search a solution space for the optimal solution* to a problem, in cases where it is extremely difficult or impossible to find an exact solution [18, 19, 41, 47]. The key characteristic of the GA is the means by which it conducts searches. It mimics the operation of *evolution*, where a population of possible solutions is formed and new solutions are found by “breeding” or “cross-over”, which combines two solutions to produce two new individuals or solutions. GA takes a logically *centralised* view of problems, as it is possible to end up with a number of solutions, while a solution’s suitability is a measure of how “good” the solution is. The centralised view is taken by choosing the best solution found so far to form a new generation.

GA was first described by John Holland in the 1975 [47]. He and his associates Goldberg [19] and others were interested in artificial complex systems that would be able to adapt under changing environmental conditions. Their idea was that a population of individuals should behave like a natural system, where survival is supported by the elimination of useless or adverse properties. A GA is an iterative procedure which usually maintains a constant population size and basically works as follows:

1. An initial population of individuals is generated randomly or heuristically.

2. The population is evaluated and assigned a fitness value according to how well it solves the problem.

3. GAs use two operators (crossover and mutation) in order to generate new individuals:

    a. *Crossover* takes two individuals called parents and produces one or two new individuals called offspring by swapping parts of the parents. In its simplest form the operator works by exchanging substrings after a randomly selected crossover point.

    b. *Mutation* is essentially an arbitrary modification introduced to prevent premature convergence to local optima by randomly sampling new points in the search space.

4. During each generation iteration, the individuals in the current population are evaluated and given a fitness value.

5. The better solutions are repeatedly selected.

6. Steps 2, 3, 4 and 5 are repeated in an attempt to evolve a better solution.

Genetic Algorithms are by all means applicable to a wide range of problems as long as no problem specific knowledge is introduced [18, 19].

4.4. Genetic Programming (GP)

One of the most exciting uses of GAs is automatic program generation, pioneered by John Koza (1992) [49]. GP was originally formulated as an evolutionary method for *breeding programs* using expressions from the functional programming language LISP [11]. It addresses one of the central challenges of computer science, "*Automatic programming*", which is to evolve computer programs to do what needs to be done, without telling it how to do it [99], allowing computers to solve problems automatically. GPs were introduced many years ago and have been used to solve a wide range of practical problems, producing a number of human-competitive results and even patentable new inventions [82]. GP applies GAs to a population of programs that are typically encoded as tree-structures.

Trial programs are evaluated against a fitness function, and the best solutions selected for modification and re-evaluation. This modification-evaluation cycle is repeated until a correct program is produced. GP can be viewed as a branch of GA. The main difference between the two is the representation of the *solution*. GP creates *computer programs as the solution*, while GAs create a *string of numbers that represent the solution*. GP has been applied to a wide variety of problems with great success, equalling or exceeding the best human-created solutions to many difficult problems [21, 22, 23, 99].

GP, as mentioned before, provides a method for automatically creating a working computer program by genetically breeding a population of computer programs. It applies the paradigm of Darwin's theory of evolution (often characterised as "the survival of the fittest"), using the principles of Darwinian natural selection and

biologically inspired operations. It iteratively transforms a population of computer programs into new generations of programs by applying analogs of naturally occurring genetic operations. These operations also include *crossover* (sexual recombination), *mutation*, *reproduction*, *gene duplication* and *gene deletion*. The best individuals will survive and eventually evolve to do well in the given environment.

The following steps provide a summary of how GP solves problems:

1. Generate an initial population of random compositions of the function and terminals of the problem (computer programs).

2. Execute each program in the population and assign it a fitness value according to how well it solves the problem.

3. Create a new population of computer programs.

    a. Copy the best existing programs.

    b. Create new computer programs by mutation. *Mutation* is an important feature of GP. It creates a new child program by altering a randomly chosen part of a selected parent program. Two kinds of mutations are possible. In the first kind, a function can only replace a function, or a terminal a terminal. In the second kind, one entire sub-tree can replace another.

    c. Create new computer programs by crossover (sexual reproduction). The *Crossover Operation* is the most important primary operation for modifying structures in GP. In this kind of operation, two solutions

are sexually combined to form two new solutions or offspring. The parents are chosen from the population according to the fitness of the solution.

4. Repeatedly select the better populations.

5. Attempt to evolve better solutions by repeating Steps 2, 3 and 4.

Preparatory Steps for GP

A human user communicates the high-level statement of the problem to the genetic programming system by performing certain well-defined preparatory steps. The five major *preparatory steps* for the basic version of GP require the human user to specify the following [87]:

1) The set of terminals (e.g., the problem's independent variables, zero-argument functions and random constants) for each branch of the program.

2) The set of primitive functions for each branch of the program to be evolved.

3) The fitness measure (for measuring the fitness of individuals in the population). The most difficult and important concept in GP is the fitness function, the objective function GP aims to optimise. It determines how well a program is able to solve the problem.

4) Certain parameters for controlling the run (e.g. size of population).

5) The termination criterion and method for designating the result of the run. (This is simply a rule for stopping. Typically the rule is to stop either on

finding a program that solves the problem, or after a given number of generations).

Disadvantages of GP

The disadvantage of GP – and it is a massive one – is the *phenomenal amount of computing resources* required before any real-world problem can be tackled. Genetic programming is often impaired by *the huge size of the search space* and uses a *huge amount of processing time* even for apparently simple problem domains.

4.5. Linear Genetic Programming (LGP)

The linear structures are simply *flattened* representations of GP tree structures. LGP (Fing.30) is a GP variant that, instead of representing an individual as a tree, does so in the form of a “linear” list of instructions [56, 98]. LGP employs as genetic material a linear program structure whose primary characteristics are exploited to achieve acceleration of both execution time and evolutionary progress [64].

In linear GP programs are linear sequences of instructions, and the number of instructions can be *fixed*, meaning that every program in the population has the same length. Or in other cases the number of instructions can be *variable*, and consequently different individuals can be of different sizes.

The incentives to use linear GP [82] are that almost all computer architectures represent computer programs in a linear fashion, with neighbouring instructions normally being executed in consecutive time steps, and that computers do not naturally run tree-shaped programs, so interpreters or compilers have to be used in tree-based GPs. On the contrary, by evolving the binary bit patterns actually obeyed by the

computer, linear GP can avoid the use of this computationally expensive machinery and can consequently run several orders of magnitude faster. Moreover, simple linear structure lends itself to rapid analysis, and in some ways the search space of linear GP is also easier to analyze than that of trees.

Typical linear GP crossover works by exchanging continuous sequences of instructions between parents [69]. The two crossover points are the same in both parents, so the existing code does not change its position relative to the start of the program, and the child programs have the same lengths as their parents. Homologous crossover is often combined with a small amount of normal two-point crossover to introduce length changes into the GP population.

Two types of standard LGP mutation, micro- and macro-mutation, are usually employed. Micro-mutation involves an operand or an operator of an instruction being changed, while in macro-mutation a random instruction is inserted or deleted [69].

4.6. How does GP Differ from Heuristic Approach?

Heuristics and GP both use search processes in order to solve NP-hard problems. There are, however, some differences in their approaches:

- Genetic systems create possible new solutions, while heuristic systems tend to modify single solutions by addition and deletion, not by combination with other solutions.
- Heuristic systems represent their searches as changes in configurations of data, while GP searches and changes entire programs or routines.

- Heuristics focus on abstract knowledge, while genetic search methods use methods inspired by biological genetic operations.

- Heuristic artificial intelligence (AI) systems tend to be deterministic, while GP processes have a basic element of randomness.

- Genetic systems explore multiple paths simultaneously [8]: each individual in a population is a potential solution to the problems the environment poses. AI systems tend to focus on one path at a time, and to explore variations in rapid succession.

- GP tends to be any time algorithm; if there is a present need, the current best solution is used until something better evolves. In heuristic AI, on the other hand, it is more likely that the solution will not work [15, 20].

4.7. Local Search Programming (LSP)

The LSP is a new concept proposed in this research project for the Evolutionary Algorithms. It generates new heuristics or programs based on existing ones, using a method inspired by both GP [49] and local search [41] techniques. LSP seeks to generate new heuristics/programs using local search techniques instead of Genetic Algorithm techniques.

In LSA, the search for an approximate solution is conducted with respect to a neighbourhood structure defined on the set of feasible solutions F [44]; where LSA starts from an initial solution X and repeatedly replaces X with a better solution in its neighbourhood $N(x)$ until no better solution is found in $N(x)\cdot$

LSP seeks to apply LSAs to programs (i.e. sets of instructions) in order to search in their neighbourhood for new and better solutions. The LSA optimises the current heuristic by stepping from one heuristic/algorithm to one of its neighbours. The neighbourhood is composed of the heuristics that can be obtained by simple local changes from the current heuristic. Trial programs are evaluated against its parent/origin, and the best is selected to continue searching for optimisation. This search pattern is repeated until an optimal program is produced (i.e. local optimisation).

The main difference between LSA and LSP is that LSA optimises the *solution* while LSP optimises or modifies *computer programs as the solution*.

The difference between GP and LSP is that GP applies *GAs to a population of programs* while LSP applies *LSA to one program*, meaning that *GP needs two parents* to cross from one to the other, while *LSP works on one parent/solution and modifies it*.

Features of LSP

The LSP approach needs only one solution and searches its neighbourhood for better solutions. In many cases, there is often only one solution, so a two-parent heuristic or algorithm approach cannot be applied, and sometimes no parent is available, so that they are generated randomly.

LSP is a method for automatically creating a working computer program, which modifies one program to generate a neighbourhood, following local search approaches. It iteratively transforms one solution technique to another by stepping from one solution to one of its neighbours. LSP processes proposed to include *mutation*, *reproduction*, *duplication* and *deletion*.

Following are the proposed steps in LSP:

1. Generate an initial solution technique (heuristic). This may be a random composition of the function and terminals of the problem (computer programs).

2. Execute this initial program and assign it a fitness value according to how well it solves the problem.

3. Create a new neighbourhood computer program through one or more of the following steps.

   a) Duplicate part of the program.

   b) Create new computer programs by mutation.

   c) Delete part of the program.

4. The new neighbours (generations) are evaluated against the parent according to how well they solve the problem,

5. The best one is selected to continue in the LSP cycle by repeating Steps 3 and 4.

4.8. Summary

This chapter is a literature review of some Evolutionary Approaches (EA); all of which share a common conceptual base of *simulating the evolution of individual structures* via processes of selection, mutation and reproduction. The chapter discusses the needs for utilising EAs in problem solving; then illustrates some EAs which are: GA, GP, and LGP; it present the beginning and the motivations of each, demonstrates how each one does work, and does differ from other mentioned approaches.

Since LGP- that is a flattened representation of GP- has been adopted to be used in SUA to generate new heuristic for SMA; then the differences between GP and heuristics that lead to utilising GP have been discussed.

Finally a new EA concept called "Local Search Programming (LSP)" is introduced, the differences between LSP and GP and LSA have been illustrated. And the motivations for this new concept have been discussed.

The detailed structure and functional specifications of FMAF for solving MSP are discussed in the next chapter.

Chapter 5

Hybrid Multi-Agent Architecture for Meeting Scheduling (HMAA)

Objectives

-

- To clarify the motivation for HMAA.
- To present the adopted MSP frameworks.
- To demonstrate the solution approach for MSP with HMAA.
- To illustrate the architecture and the functional specifications of HMAA for solving MSP.

-

5.1. Introduction

This research proposes a “Hybrid Multi-Agent Architecture” for solving many NP-hard problems. The researcher believes that a method for computing solutions for NP-hard problems using the algorithms and computational power available nowadays within a reasonable time frame remains undiscovered. Unfortunately, many practical problems such as route planning, scheduling, calendar management/meeting scheduling and creation of timetables fall into this class. It is essential that these problems are solved, and the only possibility of doing so is to use approximation techniques, since no straightforward solution technique is known.

Researchers tend to use *Heuristic* techniques [15] because they are generally powerful. However, heuristics are not absolutely guaranteed to provide the best solutions, or even to find a solution at all. This demands adopting some optimisation techniques such as Evolutionary Algorithms (EA) or Evolutionary Computation (EC) [10, 28].

The present work proposes a *new Hybrid Multi-Agent Architecture (HMAA)* for solving NP-hard problems. This architecture is hybrid because it is a *semi-distributed/semi-centralised* architecture. In the proposed FMFA, variables and constraints are distributed among small agents exactly as in distributed architectures. But when these small agents become stuck, a centralised control becomes active where the variables are transferred to a super agent that has a central view of the whole system and possesses much more computational power and intensive algorithms such as EAs to find an optimal solution.

This can be done by defining different classes of agents including super agents_and small agents. Heuristics of small agents that are fixed and limited can be updated by the super agent that generates new skills/heuristics using evolutionary approaches.

Section 5.2 discusses the formalisation of the MSP adopted in this research; Section 5.3 illustrates the proposed solution approach for MSP with HMAA. The architecture of HMAA is clarified in Section 5.4, and Scenarios of HMAA negotiations and MSP within HMAA are illuminated in Section 5.5 and 5.6 respectively. Finally functional specifications for the architecture have been stated in Section 5.8.

5.2. Meeting Scheduling Frameworks

Each meeting x_i has a set of attendees which are one or more of the agents. This formal definition contains the following elements:

1. Agents: each agent represents its user.
2. n variables xi (i=1…n), each representing a meeting,
3. n domain sets Di (i=1…n), where each $D_i=\{t_1,t_2,\ldots,t_m\}$ is a set of timeslots which are the possible values of the corresponding variable x_i.
4. Constraints: define which domain values are valid assignments.

The HMAA implements the following two formalisations for the MSP:

1) **Distributed Constraint Satisfaction Problem (DisCSP)**:

DisCSP where MSP is considered as a search problem consisting of a set of distributed *agents*, each one having a set of variables represent *meetings* $\{x_1, x_2..., x_n,\}$, and each domain is a set of timeslots, and each agent has a *hard constraint* C_H, which stipulates that no two meetings are scheduled at the same time. The goal is to search for the value assignment that satisfies the agents' constraints.

2) **Distributed Constraint Optimisation Problem (DCOP)**:

DCOP where MSP is considered is an optimisation problem consisting of a set of distributed *agents*, each one with a set of variables represent *meetings* $\{x_1, x_2..., x_n,\}$, and each domain is a set of timeslots, and each agent has *a hard constraint* C_H which stipulates that no two meetings are scheduled at the same time. The goal of

the agents is to choose time slots from the domains for the meetings to *optimise* (i.e. minimise) the violation of constraints.

Two formalisations have been adopted in order to generalise HMAA more and enable it to encompass more specifications and needs. This is because in some cases meetings must be scheduled, leaving the choice of which meeting to attend to individual participants, while trying to minimise the overlapping meetings as much as possible (the optimisation problem). In other situations, not scheduling meetings leaves to the initiator the opportunity to enter new options or domains for these unscheduled meetings (the search problem). Hence, HMAA implements these two options, leaving the users the choice of which framework is more suitable to its situation.

5.3. Solution Approach for Meeting Scheduling within HMAA

Super Agents and SMall Agents in meeting scheduling architecture HMAA cooperate in such a way that the Super Agent is the centre of the whole system. It decides the moves that the SMall Agent should follow (the heuristic), in order to overcome a failure, or to optimise the current solution. SMall Agents obey the commands of this central agent. This central agent is called the *super agent*, and the other agents dominated by it are called *small agents.*

The small agents are responsible for autonomously managing the scheduling process on behalf the individuals they represent, through negotiation between each other. Agents negotiate by having one agent propose a meeting which the other agents accept or reject, based on whether or not the proposal fits their own schedules. Each agent knows its

user's calendar availability, and *the meetings to be scheduled with the attendees' ranks* in order to act on behalf of its user.

As mentioned in the previous section, the scheduling activity is regarded as an optimisation problem for which the best feasible schedule is sought, or a search problem for which a feasible schedule is sought.

Within the context of this research project the researcher has proposed a solution technique for meeting scheduling and repair strategy for the attained solution. The heuristic considers the different parameters (the available time domains for the meetings, and meeting and attendee rankings) in the scheduling process. If there is a violation, the repair strategy then starts from the initial violated solution and enters a loop that navigates the search space, stepping from one solution to one of its neighbours. The neighbourhood is composed of the solutions that can be obtained by a local change from the current one.

However, the capabilities of the small agents are restricted, since they adopt fixed negotiation skills that would sometimes fail to attain a feasible solution, so that it reaches a dead-end - i.e. there is no possible value for the current variable. To overcome these limitations, the small agents would pass on their situation to a super agent which has more sophisticated algorithms, based on more powerful functions that would generate new better solutions. This super agent is capable of generating new heuristics/negotiation skills using evolutionary approaches like Genetic Programming (GP).

5.4 Hybrid Multi-Agent Architecture Proposed

The proposed architecture is hierarchical as depicted in Fig. 3. Which are, from top to bottom: the Super Agent (SUA) layer, the Facilitator Agent (FA) layer, the Small Agent (SA) layer and the Interface Agent (IA) layer. Super Agent (SUA) flanked on one side by the Facilitator Agent (FA) and FA on the other side flanked by the Small Agent (SMA). The SMA adjoining the (FA) layer and the Interface Agent (IA); the latter is only connected to the SMA. The arrows are interactions between the architecture components. Each pair of adjacent layers can communicate with each other by exchanging data and messages.

The interface agent (IA) is reactive agent: it responds to changes in the environment; receives input from users then update the database with the input data, and exchanges data with the SMA (scheduling request). Each IA correlated on one SMA is the window of this SMA to the external environment, where SMA can perceive the environment, receive input from the environment, and present the attained results to the external environment through this IA.

The SMA is a Believe Desired Intention (BDI) small agent, cannot perform complex computations, but rather receives data from the IA or FA, incorporates small algorithms to accomplish specific tasks, and sends the results back to them. SMA can receive data from the meeting database and input updated data -the assignment field- to the same database. SMAs are interacting by sending and receiving messages through FA.

The FA is the central agent, like a server agent; any two or more agents who want to talk or exchange messages or data do so through the FA. The FA knows the agents' names,

IDs and addresses or locations, so it forwards the messages it receives to the corresponding agents. Any new agents added by the administrator or environment should be registered in the FA with their names, IDs and locations, which is why the FA is considered to be a reactive agent, FA receives data from environment (administrator), exchange data with SMAs (messages) and with SUA (heuristics).

Finally, the SUA is a BDI agent, performing the same tasks as the SMA, but it can have very large computational power and can implement computationally intensive algorithms like EAs that find more solutions and perform better for NP-hard problems. Meetings are variables to be assigned; they are the problems that the system HMAA would find a solution or value for. SMAs are the only agents responsible for assigning values or finding solutions for the problem; they can read data and update values. SUAs can read stored data from meetings in order to be able execute their algorithms, but cannot update values; they pass the result (heuristic) to the FA who forwards it to the corresponding SMA, which updates the values for the meetings according to advice received from the SUA in order to overcome the failure or to reduce the violation.

One SUA can be defined and large number of SMAs and IAs can be initialised, where each SMA correlated to one IA.

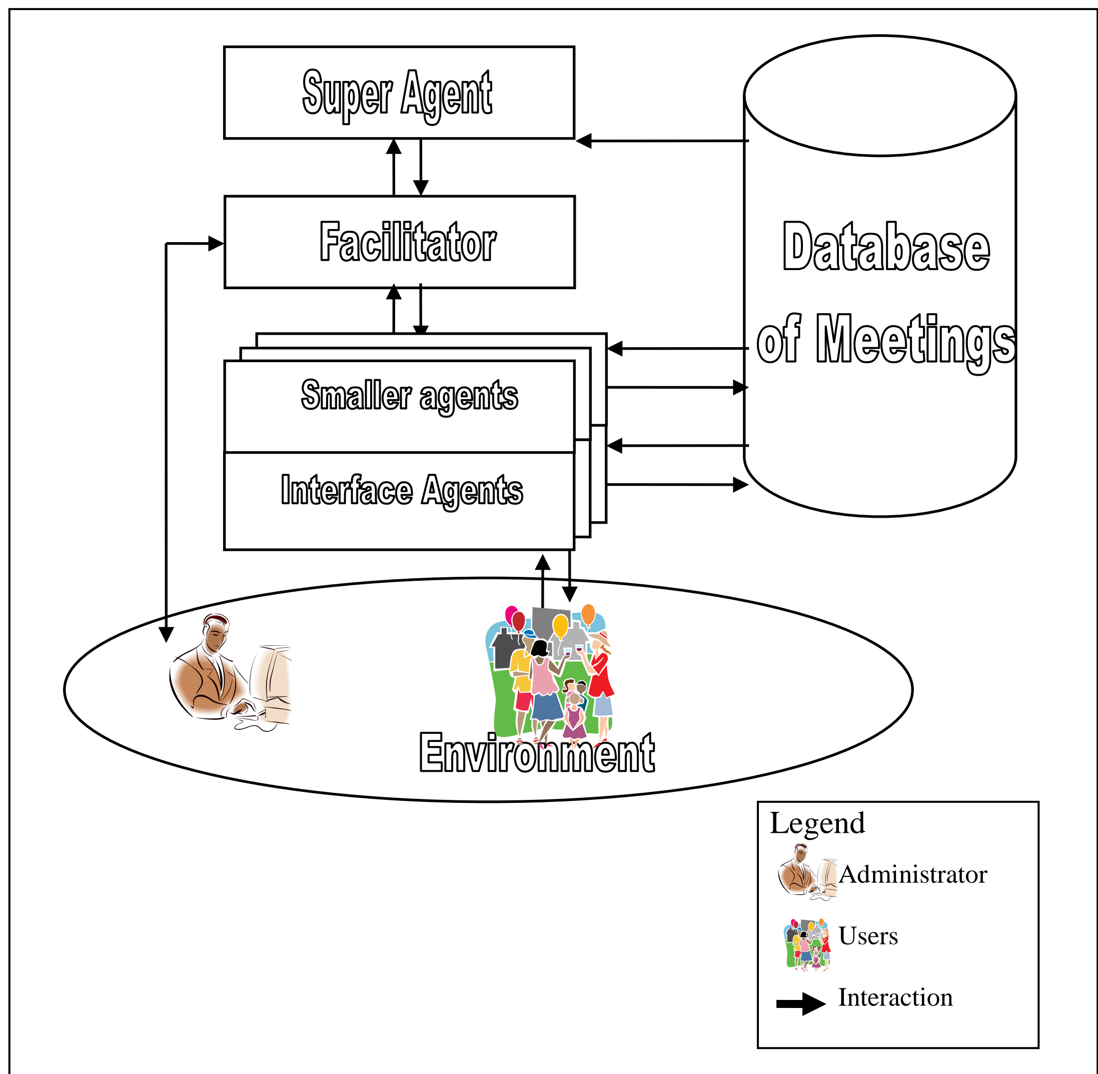


Fig. 3: HMAA Architecture

5.5. Scenario: Hybrid Multi-Agent Architecture (HMAA) Negotiations

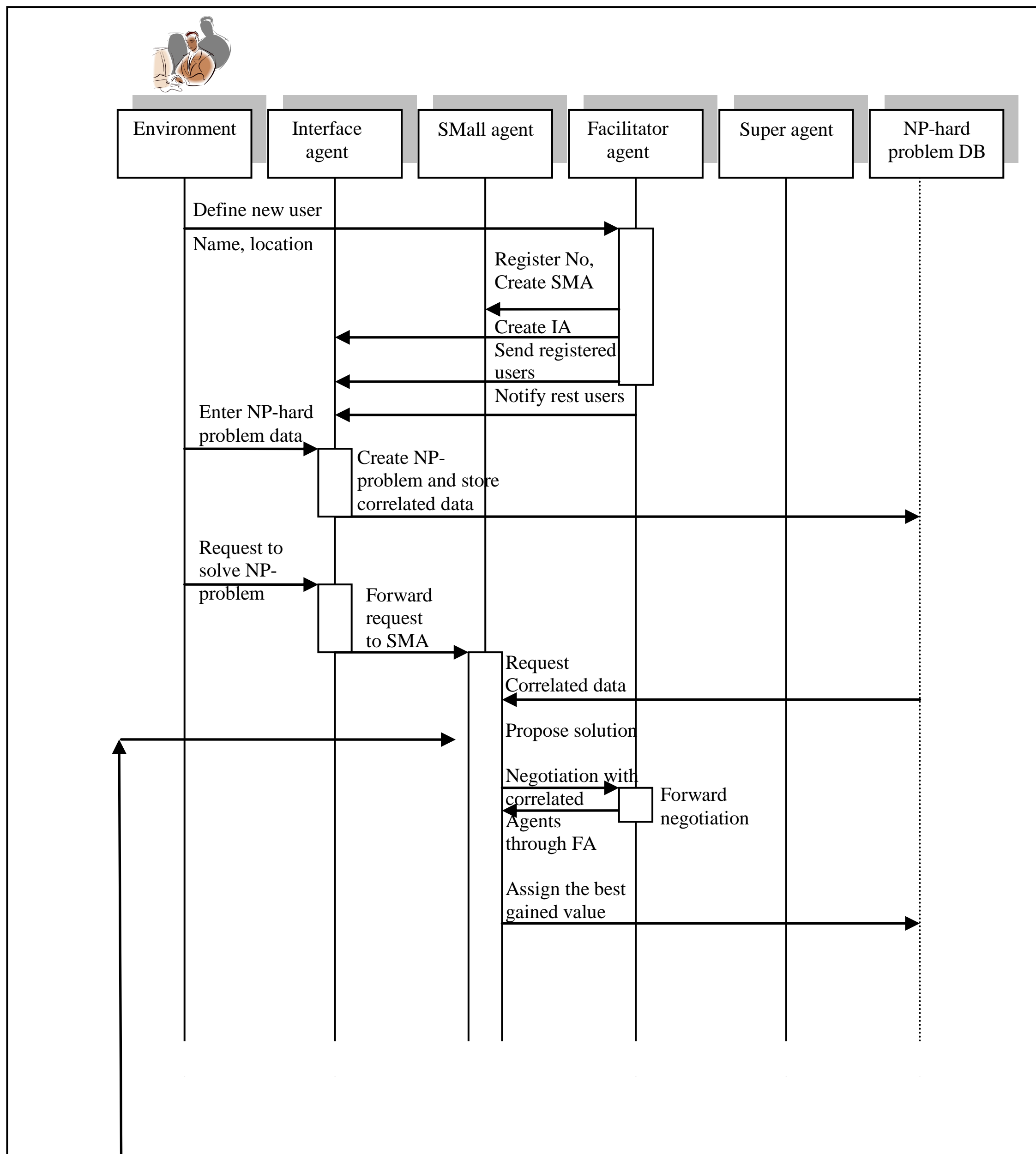


Fig. 4: HMAA scenario

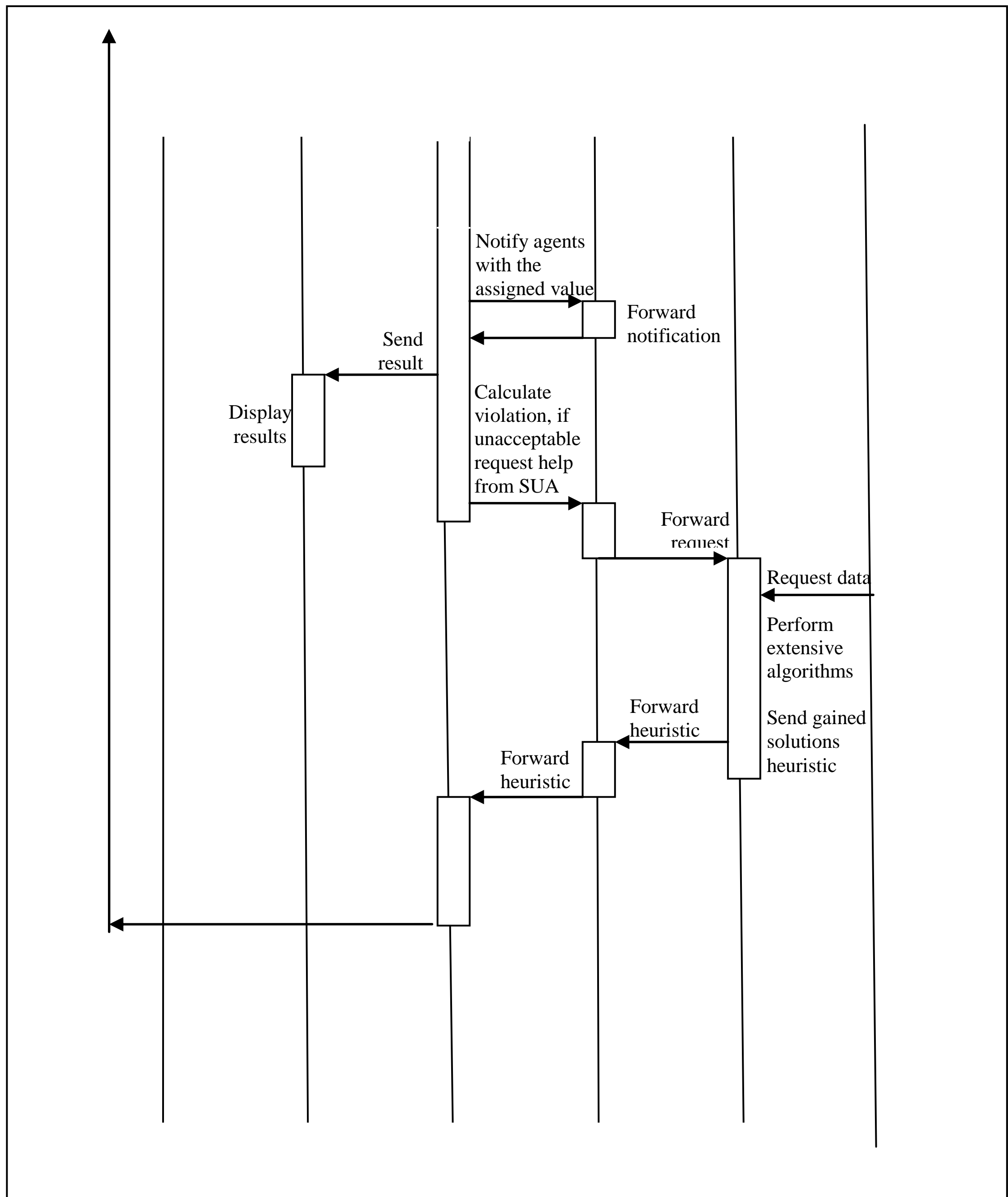


Fig. 5: HMAA scenario

Figs 4 and 5 illustrate how the proposed HMAA would work in tackling NP-hard problems. Each new user has to be registered with the FA by the administrator who

assigns the new user ID, creates the SMA and IA, sends details of the registered users to the new user and finally notifies the other users of the new user.

If logged users add NP-hard problems to be solved later, they have to enter the problem's data through IA, which would create the problem in the database and store the correlated data entered by the user. The user can ask for a solution or solutions for the problems entered through IA, which would at that point send the request to the SMA. Once the SMA receives the request, it inquiries the correlated stored data to be undertaken while solving the problem. The SMA then executes its heuristic and negotiates with the other correlated SMAs through the FA who forwards the messages between them with the aim of finding a solution for the specified problem. The best obtained solution is forwarded to all the correlated SMAs and stored in the problem database.

If this solution is unacceptable, the SMA asks for help from the SUA, which executes a much more powerful and extensive algorithm in order to find a better solution. Once the SUA receives a request for a new heuristic to solve the problem, it starts its evolutionary algorithms to generate a new heuristic for small agents. Once the SUA finds the solution it passes it to the related SMA through the FA. The SMA uses the newly received heuristic to reschedule the meetings, in order to overcome or minimise the violation.

5.6. Scenario: MSP within HMAA

The process starts when an employee decides to hold a meeting. As shown in Fig. 6; he registers in the HMAA by defining his *name* and *location* to the FA which stores this information, assigns an ID, create an SMA and an IA for him and send him a list of logged users. The user enters a meeting request, defines a list of *time domains* for this

meeting and a *list of attendees* from the list of logged users with *their ranks* to attend the corresponding meeting through the IA.

The IA accepts the data, creates a meeting as a variable to be assigned a value, and sends the request to the SMA. The SMA initiator proposes one time slot and sends the proposal to the FA for forwarding to the other SMA attendees who would check their calendars and send replies as to the proposal's acceptability or otherwise to the FA to send back to the SMA initiator. When the latter has received all the replies, it calculates the violation (i.e. how many conflicts replies there are). If the violation values greater than 0, it tries to find a better proposal, by putting forward the next time slot from the domain. If there is no better solution, he sends the best one available to the FA to forward to SMA attendees for confirmation (optimisation problem), and updates the meeting variable with the best gaining value.

If a violation remains after all the meetings have been scheduled, then the SMA contacts the SUA through the FA to try to find a better solution by performing extensive algorithms for the NP-hard problem. The SUA interrogates the database to obtain the relevant data and executes its evolutionary algorithms. Once the SUA finds a heuristic with better results it passes it to the corresponding SMA through FA, who will follow the recommendations of SUA, and update the meetings' values accordingly.

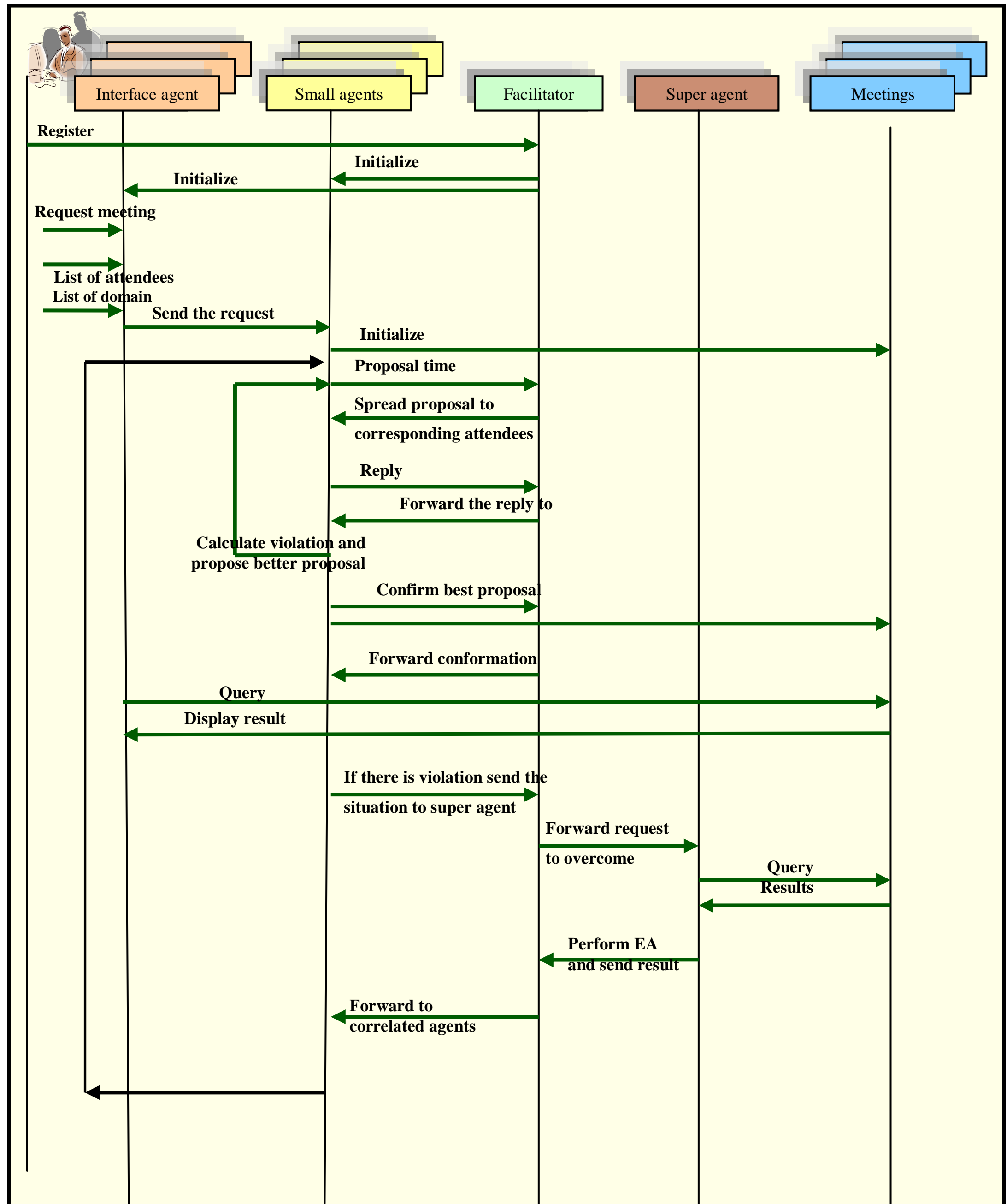


Fig. 6: Sequence diagram for Meeting Scheduling Problem within HMAA

5.7. Full Functions Specifications of the HMAA

5.7.1 Interface Agent

The functions of the Interface Agent are as follows:

1. Create a new meeting in the Data Base (DB): the user from the external environment can request a meeting to be scheduled through the Interface Agent (AI), who would response by asking for the meeting parameters.

2. Add meeting parameters (attendees with ranks, time domain): the user adds the meeting parameters (attendees with ranks, time domain) through interface agent. Who would accept the entrance, and accordingly create a new meeting with the entered parameters to be scheduled.

3. Fire the request of scheduling: once the user request scheduling for the unscheduled meetings through the IA, the IA contacts the corresponding small agent a forward the request to it.

4. Display results for the users: the user can inquiry his scheduled meetings from the IA, who would inquiry them from meeting database.

5.7.2 Small Agent

The functions of the Small Agents are as follows:

1. The scheduling for unscheduled meetings: once the small agent receives a request to schedule list of meetings he would:

   a) Inquiry the corresponding parameters (attendees and their ranks, time domains) for all the correlated meetings from the meeting database.

   b) Arrange the meetings according to their calculated priority, which based on these parameters.

c) Starts by the most important meeting-highest priority- and initiate a proposal for it from the corresponding domain.

d) Sends the proposal to the corresponding SMA attendees through the facilitator.

e) The facilitator forwards the proposal the corresponding SMA attendees.

f) Each SMA receives a proposal for a meeting finds the number of meetings he has in the same proposed timeslot (ex. 0 means no other meeting, 1 means having one meeting in the same proposed timeslot, 2 mean two meetings…).

g) Forward this number to the FA, who would pass it to the SMA initiator.

h) When the SMA initiator receives all reaction from all the SMA attendees, he would calculate the violation for this proposal, and keep it as the best proposal until he finds better proposal with less violation.

i) If the violation is more than 0, then the initiator sends the next proposal with the corresponding SMA attendees to the facilitator. And repeat steps (d-h) for all the time domains for the corresponding meeting.

j) Sends conformation to the corresponding SMA attendees through the FA with the solution, and update the meeting values in the database with this gained best proposal.

k) Accordingly all the SMA attendees would update their calendar with this confirmed meeting.

2. Repeat steps (c – k) for all the rest of the meetings.

3. Calculate the total violation: if the violation is unacceptable, the SMA would contact the SUA through the FA, and pass the existed situation in order to try to overcome the failure or reduce the violation.

4. Once the SMA's receive new solution approach/heuristic from the FA, they obey to the SUA recommendations and update their heuristic accordingly.

5.7.3 Facilitator Agent

The functions of the facilitator agent

1. Receive request for new user with name and location, give him ID, an register the new user in FA: in order to identifying a new user, the admin from the external environment should enter a user name and the location to Facilitator Agent, the FA would react and response to the entrance by giving him ID, and register the new user in FA.

2. Create SMA and IA for the new user and send him the logged users: the FA creates SMA and IA for the new user and sends him the logged users who are already registered to the FA.

3. Communication and cooperation between SMA and SUA through the messages between them.

5.7.4 Super Agent

The functions of the super agents

2. Generate new heuristics for SMA by executing EA: once the SUA receives the request through the FA, it would ask for information from the meeting database. The SUA execute evolutionally intensive algorithms like GP; this would hopefully find better new solutions and better schedules for the current situation.

3. The SUA passes the solution to the FA to forward it to the corresponding.

4. Calculate violation.

5.8 Summary

The chapter discusses the proposed HMAA; a clarification for the motivations to this new architecture has been stated. Then the adopted formalisations for solving MSP with HMAA have been discussed, these formalisations are: DisCSP and DCOP. Two formalisations have been adopted in order to generalise HMAA more and enable it to encompass more specifications and needs. This is because in some cases meetings must be scheduled, leaving the choice of which meeting to attend to individual participants, while trying to minimise the overlapping meetings as much as possible (DCOP). In other situations, not scheduling meetings leaves the initiator the opportunity to enter new options or domains for these unscheduled meetings (DisCSP). Hence, HMAA implements these two options, leaving the users the choice of which framework is more suitable to its situation.

In addition to all of the above, the chapter shows the architecture of HMAA which is hierarchical architecture composed of four adjacent layers: SUA, FA SMA, and IA. And the relationship between these layers has been clarified.

Scenarios of HMAA negotiations and MSP solution approach within HMAA have been illuminated. And finally full functions specifications of the HMAA have been situated.

The construction of SMA, which is the basic part of HMAA, is introduced in the following chapter.

Chapter 6

Small Agent Heuristic

Objectives

- To illustrate SMA main task in HMAA.
- To illustrate the proposed algorithm used by SMA to accomplish the task.
- To presents the repair strategy used by SMAs in order to improve their obtained violated solutions.

6.1. Introduction

This chapter is discussing fundamental part of HMAA for solving MSP which is SMA. SMAs are responsible for accomplishing the scheduling process on behalf of their users, and their main seek is to find timeslots from each domain to assign the corresponding meeting to, where all the attendees accept that assignment. The goal is to find feasible or optimal (depends on the chosen framework) solution. The proposed heuristic -used by SMA- is prioritised/ranked heuristic; this heuristic gives initial solutions for the systems. The heuristic starts by ranking the meetings –according to proposed equation -in order to schedule the most difficult ones –with highest rank-respectively.

After that the proposed local search repair strategy for the violated solutions is illustrated in Section 6.3. Section 6.4 discusses the platform used for developing a prototype, and some of the advantages of the adopted platform. Screenshots for the implemented prototype are in Appendix A.

6.2. A Prioritised/Ranked-Meetings Scheduling Heuristic

Within this research project, a new meetings scheduling heuristic is proposed. This prioritised/ranked heuristic gives initial solutions for the system. Each small agent is responsible for managing its local meetings. Each meeting has a set of domains and a set of ranked attendees; the rank of an attendee tells how important the attendee is to the meeting. The small agents are responsible for finding timeslots from each domain to assign the corresponding meeting to, where all the attendees accept that assignment. The goal is to find feasible or optimal (depends on the chosen framework) solution.

The heuristic starts by ranking the meetings in order to schedule the most difficult ones firstly. The rank of a meeting measures how difficult it is to schedule that meeting. The ranking for the meetings is calculated according to the following equation (5.1):

$$\textbf{Rank } (X_i) = \Sigma \text{ (rank (Attendee)* How_Much_Busy (Attendee, } D_i)) \quad (6.1)$$

The ranking for a meeting X_i is calculated by asking all the attendees of the meeting X_i how busy they are in the domain D_i for meeting X_i. Each attendee replies by sending "How_Much_Busy" message, "How_Much_Busy" message is a percentage value (between 0.0 and 1.0) indicates how busy the corresponding attendee is in D_i.

When How_Much_Busy value is received from a specific attendee, this value is multiplied by the stored attendee rank (entered by the user). The summation of this multiplication for all the attendees of the corresponding meeting (X_i) gives the rank of this meeting Rank (X_i).

By this; if the attendee is very busy in the domain (has many meetings already confirmed in one or more of the domain's timeslots) of a specific meeting, then How_Much_Busy message value for the corresponding meeting will be high. If the attendee is busy in the entire domain; then the How_Much_Busy is "1.0", and if he is free then How_Much_Busy is "0.0". Each time the attendee is busier, the How_Much_Busy message value is increased. On the other hand the higher ranked attendees (high attendee rank value); their busy messages values affect more the meeting's rank.

Example 1: in meeting X_1, attendee A_1 is busy in (9/10) of X_1 domain "0.9 busy" and his rank in that meeting "0.1"; attendee A_2 is free in X_1 domain "0.0 busy" and his rank in "0.9" then the meeting rank is:

Rank (X_1) = ((0.1*0.9) + (0.0*0.9))

Rank (X_1) = 0.09

While in another meeting X_2, A_1 is free "0.0 busy" and his rank "0.5", while A_2 is "0.5" busy and his rank "0.5" then the rank for X_2 is:

Rank (X_2) = ((0.5*0.0) + (0.5* 0.5))

Rank (X_2) = 0.25

From the results above it can be seen that X_2 is "more difficult" to schedule than X_1, although A_1 is busier (0.9) in X_1 than A_2 (0.5) in X_2; this is due to the fact that A_2 in X_2 is more ranked/important (0.5) than A_1 in X_1 (0.1). Which implies that the influence of the unavailability/ busyness of the more ranked attendees on the meeting rank is higher.

Example 2: if there are two agents A_1, A_2 and A_1 has a meeting (X_1) on 1st June. A_1 has initiated another two meetings (X_2, X_3) between (A_1, A_2) with the domains D_2= {2nd June, 3rd June}, and D_3= {1st June, 3rd June}) respectively. The ranks for the attendees (A_1, A_2) in X_2 are {0.9, 0.1}, and in X_3 are {0.4, 0.6} respectively.

Each attendee of A_1 and A_2 calculates "How-Much-Busy" with the following equation:

How_Much_Busy (A_1, D_2) = how busy A_1 is in D_2/ size of D_2= 0.0/2.0 = 0.0

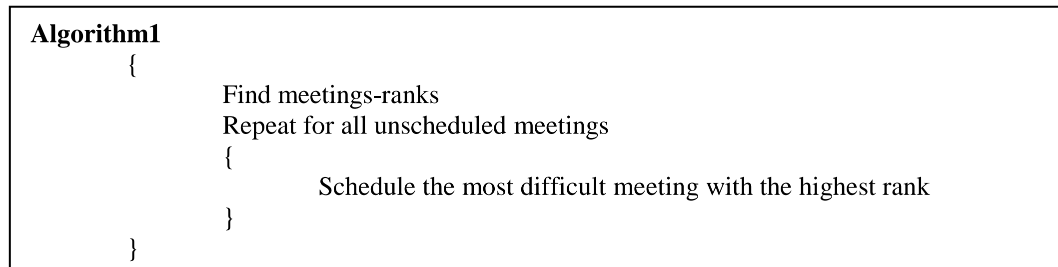
How_Much_Busy (A_2, D_2) = how busy A_2 is in D_2/ size of D_2= 0.0/2.0 = 0.0

How_Much_Busy (A_1, D_3) = how busy A_1 is in D3/ size of D3= 1.0/2.0 = 0.5

How_Much_Busy (A2, D_3) = how busy A_1 is in D3/ size of D3= 0.0/2.0 = 0.0

Algorithm 1 in Fig. 7 shows the prioritised scheduling algorithm used by the SMAs to accomplish the scheduling task; the algorithm starts by calculating the rank for each

meeting according to equation (6.1); and then orders the meetings accordingly. After which it schedules the meetings by rank by giving scheduling priority to the most difficult meetings, which is the one with the maximal rank value (Fig. 7).



```
Algorithm1
    {
        Find meetings-ranks
        Repeat for all unscheduled meetings
        {
            Schedule the most difficult meeting with the highest rank
        }
    }
```

Fig. 7: prioritised scheduling

So the ranks for the meetings are:

$$Rank\ (X_2) = ((0.9*0.0) + (0.1*0.0))$$
$$Rank\ (X_2) = 0.0$$

$$Rank\ (X_3) = ((0.4*0.5) + (0.6*0.0))$$
$$Rank\ (X_3) = 0.2$$

In this research project, two algorithms have been proposed:

(i) Optimisation problem solving: Fig. 8 shows Algorithm 2; the scheduling pseudo code for optimisation problems. In this algorithm the initiator starts scheduling the most difficult meeting (i.e. with maximum rank which in example2 is X_3), as shown in Fig. 8; the initiator sends first timeslot as a proposal for the meeting and waits for replies from the attendees. Each attendee will reply with Reply-Violation message, which contains a number indicating how many meeting he has got in the same proposed timeslot i.e. Number 0 indicates that there are no other meetings in the proposed

timeslot. The more meetings the attendee has in the specific timeslot, the higher Reply-Violation value sent.

The initiator tries to find the timeslot from the domain for the corresponding meeting with violation 0, the initiator calculates the violation according to the following equation (6.2):

$$\text{Domain}(x_i) = \{d_{i1}, d_{i2}\ldots d_{in}\} \text{ where the } d_{ij} \text{ are timeslots}$$
$$\text{Violation}(d_{ij}) = \sum (\text{rank}(\text{attendee}) * \text{Reply_Violation}(\text{attendee}, d_{ij})) \quad (6.2)$$

Each Reply_Violation value received from specific attendee is multiplied by the rank of the corresponding attendee, and the summation of this multiplication for the corresponding meeting's attendees gives the corresponding meeting's violation. The higher Reply_Violation values increase the meeting violation, on the other hand the more ranked attendee affect more this violation.

```
Algorithm2
Schedule (Meeting M)
{
        Choose first time slot from the domain as Best_Proposal;
        Send Best_Proposal to all the attendees;
        Calculate the Best_Proposal_Violation;
        Loop until Best_Proposal_Violation is 0 or no more timeslots in the domain
        {
                Choose next time slot from the domain as Proposal2;
                Send Proposal2 to all the attendees;
                Calculate the Proposal2_Violation;
                If ((Proposal2_Violation <= Best_Proposal_Violation)
                {
                        Best_Proposal = Proposal2;
                        Best_Proposal_Violation =Proposal2_Violation;
                }



        }
        Send Best_Proposal to all the attendees as confirmation

}
```

Fig. 8: Scheduling Pseudo Code for Optimisation problems

In Example 2, for meeting X_3 the violations of its domains (1st June, 3rd June) are:

$$violation(D1) = (0.4 \times 1) + (0.6 \times 0)$$
$$violation(D1) = 0.4$$

$$violation(D2) = (0.4 \times 0) + (0.6 \times 0)$$
$$violation(D2) = 0$$

Once the algorithm finds a timeslot with violation 0 (which is D_2), it assigns that meeting to that timeslot. Otherwise, it assigns the meeting to the timeslot with the minimal violation. The initiator agent then sends a confirmation message with the assignment to all attendees. Each attendee receives this confirmation message, adds the corresponding meeting to its local calendar at the specified timeslot. The other meetings already scheduled at the same timeslot are called "affected meetings".

The Reply_Violation value for those affected meetings (if they exist) are updated in order to consider the new confirmed meeting as a violation for those affected meetings; and notification messages to the initiators for those affected meetings are sent; in order to update the violation of the meetings that are scheduled at the same timeslot.

This process is reiterated for each meeting until all meetings have been assigned. In Example 2, therefore, at the end of this stage the meeting X_2, X_3 would be assigned to 3rd and 2nd June respectively.

(ii) Search problem solving: Fig. 9 shows Algorithm 3; scheduling pseudo code for search problems. As can be seen in the figure the initiator starts by node_consistency. Node consistency is the method used for *unary constraint.* It determines if a value is consistent or not, if it is inconsistent, then that value is removed from the domain of the variable.

Node_consistency in MSP means exclude unavailable timeslots. In Example 2; node_consistency would remove the timeslot "1st June" from X_3 domain; this is because A_1 has got meeting X_1 in "1st June"; hence this timeslot is unavailable for A_1. Node_consistency is applied in search problem because there is no need to propose timeslots that are not available for the initiator itself.

After node_consistency the initiator starts as shown in Fig. 9 with the most difficult meeting ;X_3 in example2 ; sends a proposal from the available domain (3rd June), and waits for replies from the attendees. Each time the initiator receives Reply_Violation message from specific attendee not equals zero, then executes **"excluding (timeslot, attendee)**" function that excludes *this timeslot* from the rest of the meetings *if the corresponding attendee attends*.

Example 3: agent A wants to schedule two meetings with another agent B: M_1 with domain {1st June, 2nd June}, and M_2 with domain {1st June, 2nd June, 3rd June} with agent B, where M_1 is more difficult than M_2. Agent A starts with M_1, proposes firstly (1st June), if A receives Reply_Violation not equals zero –means B has got a meeting in 1st June- A removes 1st June from the domain of M_1 and M_2; so he would not propose 1st June during the scheduling of M_2. This due to the fact that B is also attending M_2 and A already knows –during scheduling M_1-that B is busy in 1st June, hence no need to propose 1st June again while already know the answer. This would reduce the scheduling time and load; hence no need to ask about the same timeslot the same attendee twice.

After calculating violation not equals zero, the initiator tries the next time slot until finding timeslot with violation zero, otherwise the corresponding meeting would not be schedule. Each meeting he schedules a meeting, he performs arc_consistency. Arc_consistency is used for *binary constraints*; given two variables X and Y then the constraint graph is consistent if for every possible value of X there is a value of Y that satisfies the constraint between X and Y.

Arc_consistency in MSP excludes the already assigned timeslot from the rest of the meeting; since they become unavailable to the initiator itself. Returning to Example 2; when the initiator proposes “3rd June” for X_3, and receives Reply_Violation messages from the all the attendees equal zero, then the initiator schedule X_3 in “3rd June”, and performs arc_consistency that would remove timeslot “3rd June” form the domain of the rest of the meetings (e.g. form the domain of X_2). This is because “3rd June” becomes unavailable to the initiator itself.

In Example 3; when scheduling meeting M_1 in 2nd June, arc_consistency removes "2nd June" form the domain of M_2; then when scheduling M_2 the initiator already removed timeslots {1st June (node_consistency), 2nd June (arc_consistency)}since he already has answers for them, then the initiator proposes just one timeslot in scheduling M_2 which is "3rd June".

```
Algorithm3
Schedule (Meeting M)
{

        Loop until Best_proposal_violation is 0 or no more timeslots in the domain
        {
                Choose timeslot from the domain as Best_proposal
                Send Best_proposal for all the attendees
                Calculate the Violation of Best_proposal
                If ((Best_proposal_violation  is 0)
                {

                        Send Best_Proposal for all the attendees as confirmation
                        Arc_consistency (timeslot)
                }

                Else
                {       For all attendees that Reply-Violation is not zero
                        Excluding (timeslot; attendee)
                }


        }
```

Fig. 9: Scheduling Pseudo Code for Search problems

But why not excluding timeslots that are not available in "optimisation problem"? This is because in dealing with the problem as optimisation problem, the initiator has to schedule the meeting in the minimum violated timeslot, which means even if the timeslot is not available for some attendees, there is a probability that this time slot is the minimum violated one.

6.3. A Simple Local Search Repair Strategy

As has been discussed in the previous section, *the Prioritised/Ranked-Meetings Scheduling heuristic* for optimisation problem solving does schedule all the meetings with the minimum violation. In order to search for better scheduling that may reduce the violation; a simple Local Search Approach (LSA) repair strategy has been developed.

In LSA, the search for an approximate solution is conducted with respect to a neighbourhood structure defined on the set of feasible solutions F. For every $x \in F, N(x) \subseteq F$ is a neighbourhood function. Feasible solutions in $N(x)$ are called neighbours of x, or solutions adjacent to x. For example the Simplex algorithm is a local search algorithm where $N(x)$ consists of all basic feasible solutions which differ from x in only one basic column [44].

LSA starts from an initial solution X and repeatedly replaces X with a better solution in its neighbourhood $N(x)$ until no better solution is found in $N(x)$, where $N(x)$ is a set of solutions obtainable by slight perturbations.

The iterative steps for LSA are based on modifications of a single solution as follows [41]:

- Start with a solution
- Improve it, aiming at a better solution

In the proposed repair strategy, a local search neighbourhood structure has been defined using simple moves involving only timeslots and meetings. The move involves two overlapped meetings (X_i, X_j). Two meetings X and Y are considered to be overlapped if

each meeting has in its domain the timeslot which the other meeting is assigned to. Swapping two overlapped meetings X and Y means reassigning X the timeslot that Y is already assigned to, and reassigning Y the timeslot that X was assigned to before the swap.

```
Algorithm4
{
        List of violated meetings;
        Calculate the total violation;
        Loop for all violated meetings

          Find Neighbourhood of M;
          Loop for all neighbour X of M
          {
                    Swap X with M;
                    Recalculate the violation;
                    If the new violation is less than the previous violation
                    {
                              Exit loop;
                              Restart the local search process

                    }


                    Else if the new violation is greater than the previous violation
                    {
                              Undo-swap
                    }
          }
}
```

Fig. 10: Local Search Pseudo Code

Fig. 10 shows the pseudo code for Algorithm 4 "the local search repair strategy"; the algorithm loops for all the violated meetings, starting by the most violated ones respectively. Firstly, it finds the neighbourhoods for the most violated meeting M, then swaps M with one of its neighbours and recalculates the total violation. If the violation decreases, then confirm this swapping and restart the local search process. If the

swapping increases the violation, then undo this swapping and try to swap M with the next neighbour.

This is iterated for all the neighbours of the corresponding meeting. Then another local search for better neighbourhood is starting by swapping the next violated meeting with one of its neighbourhood. The neighbourhood of meeting M is generated by the function in Fig. 11.

```
Algorithm5 Find_Neighbourhood (M)
{
        i=0;
        For (d=M.domain[0]  to  M.domain[final]) //search in domain (M)
        {
                For (M1=meeting[0] to all-meetings)  //search in all the meetings
                {
                        If (M1.assignment ==d) //if meeting (M1) assigned in the timeslot (d)
                        {
                        For (d1=M1.domain [0] to M1.domain[final]) // search in domain (M1)
                        {
                                If (M.assignment==d1) //if meeting (M) assigned in timeslot (d1)
                                {
                                M.Neighbourhood[i]=M1; //then M1 is neighbour to M
                                i++;
                                Break;
                                }
                        }
                        }

                }
        }
}
```

Fig. 11: Neighbourhood Function

The meeting M_1 is considered to be one of M neighbourhood, if M_1 is assigned to timeslot which is in M domain; and M is assigned in to timeslot that is in M_1 domain.

In Example 2, we have the followings:

Meeting X_1 has domain {1st June, 2nd June, 3rd June} and is assigned to (1st June)

Meeting X_2 has domain {1st June, 3rd June} and is assigned to (3rd June)

Meeting X_3 has domain {2nd June, 3rd June} and is assigned to (2nd June)

Then the neighbourhood of X_1 is {X_2}. So the algorithm could swap X_1 with X_2 (overlapped in {1st June and 3 of June}); if needed.

Each agent first assigns all the meetings to timeslots, and then performs this simple local search repair strategy on the solution in order to improve the solution (i.e. to find a better solution with a small number of violations).

6.4. Implementation Platform

The computer world currently has many platforms, and it has become increasingly difficult to produce software that runs on all of them. The Java Platform, however, provides an ideal solution to this with its Java Virtual Machine [2, 71, 77].

In this research project Java has been adopted for developing a prototype for the HMAA. Some of this platform’s advantages will now be discussed. According to JavaSoft's White Paper [26], the Java Base Platform is currently embedded in the most widely used Internet browser, Netscape Navigator, Microsoft Internet Explorer (applet), workstation and network operating systems [12, 75].

Java is, at its most basic, a programming language created by Sun Microsystems. However, it has developed from being just a programming language into a platform designed for running highly interactive, dynamic and secure applets and applications on networked computer systems [26]. Being interactive and dynamic, the Java Platform has benefits not only for the developer and support personnel, but also for the end user.

Developers can write object-oriented, multithreaded, dynamically linked graphical end-user applications using the Java language. The platform has built-in security, exception handling and automatic garbage collection called GC (a schema of memory management that automatically frees up space, based on the reachability of the object blocks of memory, sometimes after all references to the memory have been redirected), so that designers need not worry about cleaning up dead memory, because the responsibility for releasing unused memory has been moved from the developer to GC. This would reduce the probability of memory leaks [26].

Java is:

- Cross-Platform
- Object-oriented
- Multithreaded
- Distributed

One of the key components of Java's success in addition to the excellent programming language and cross-platform support is its approach to distributed programming. Java code can be downloaded dynamically from remote servers and interpreted within a local application or applet. A compiled Java program is distributed as a set of files known as class files (one Java class per file) and is generally run through an interpreter (known as the Java Virtual Machine, or JVM) on the client.

The JVM handles the platform-specific calls such as GUI, file-system and networking calls, and also performs run-time garbage collection to remove unused objects from the memory. This garbage collection process -as have been mentioned-removes the burden

of memory management from the programmer's shoulders, resulting in drastically fewer runtime programming errors.

Java also supports distributed object programming (the initiation and use of objects running on other servers) through its native protocol, Java RMI, and also through CORBA. This means that a client application running on a Palm organiser could initiate remote objects written in any CORBA-compliant language on nearly all operating platforms [31].

6.5 Summary

The chapter illustrates the Small Agent task of finding timeslots from each domain to assign the corresponding meeting to, where all the attendees accept that assignment. The goal is to find feasible or optimal (depends on the chosen framework) solution. The proposed heuristic -used by small agents- is prioritised/ ranked heuristic that gives initial solutions for the systems. The heuristic starts by ranking the meetings –according to proposed equation -in order to schedule the most difficult ones –with highest rank- respectively.

After that the proposed local search repair strategy for the violated solutions has been illustrated. A Local search neighbourhood structure has been defined using simple moves involving only timeslots and meetings. The move involves two overlapped meetings, it swaps these two overlapped meetings, each meeting in the swapped meetings has in its domain the timeslot for which the other meeting has been assigned to.

Chapter 7

Super Agents Construction

Objectives

- To illustrate the performance of SUA in HMAA.
- To present two types of SUA (LGP and LSP), with two SUAs for each type.
- To present and illustrate the algorithm for each SUA.
- To present case that illustrates SUAs algorithms and results.

7.1. Introduction

The super agent task is fired when one of HMAA small agents failed to find a zero violation solution, using its predefined heuristic. Its task is to generate a new heuristic for the corresponding agent, which produces the optimal solution for the current situation. Because a super agent has more computational power, it can implement evolutionary approaches in order to achieve its goal. The predefined small agent heuristic is used as a parent heuristic in the evolution process. Super agent keeps generating and trying new generations of heuristic until the one that produces optimal solution is generated. Then it sends the optimal heuristic to the corresponding small agent to be used to overcome the failure.

Two types of super-agent have been implemented in the HMAA, with two SUAs for each type. Section 7.2 discusses the first type that consists of two SUAs called LGP_SUAs namely; superagentLGP and superagentLGP_SP; these SUAs use the LGP approach to generate new heuristics. Section 7.3 illustrates the second type formed by the LSP_SUAs; superagentLSP and superagentLSP_SP that use the LSP approach to generate new heuristics. Screenshots for the implemented HMAA prototype is in the Appendix A.

7.2. LGP_SUAs (superagentLGP/superagentLGP_SP)

In HMAA, the SUA's task starts after receiving from the SMA two messages: the first one includes a list of ***meetings*** with all their details, and the second one contains the ***heuristic*** of the SMA (used to accomplish the scheduling process) represented as a program.

In HMAA, two LGP_SUAs using the LGP approach have been implemented (superagentLGP and superagentLGP_SP). As we have mention in Chapter 7; LGP needs to crossover two parents' heuristics in order to generate new heuristics' children. The two mentioned SUAs receive one of the parent heuristics from the SMAs and generate the second parent by reversing the received one.

As mentioned in Chapter 5.1, the SMA heuristic is a Prioritised/Ranked-Meetings Scheduling heuristic that ranks meetings according to the discussed properties using the Rank-Meeting function, and then uses the Schedule function, which schedules the meetings according to some defined specifications, for all the meetings.

The following are the preparatory steps followed in generating LGP heuristics:

1) The set of terminals: the meetings defined by the *domain* and the list of ranked *attendees*.

2) The set of primitive functions consists of:

    i) the ranking function: “Rank-Meeting “

    ii) the scheduling function: “Schedule”

3) The fitness measure minimises the total violation (i.e. The number of overlapping meetings), and the algorithm preserves the children with the minimum violation as parents for the production of the next generation.

4) The termination criterion: the rule for stopping is either finding a program which solves the problem, or stopping after a given number of generations, or stopping if there are no new parents with fewer violations (for superagentLGP_SP).

```
Initialise population;
Evaluate population;
Loop until the termination criterion is satisfied
{
        Select parents for reproduction;
        Perform recombination and mutation;
        Evaluate population;

}
```

Fig. 12: Pseudo Code for LGP

Fig. 12 is a pseudo-code for LGP. LGP starts by initialising a population of solutions, and evaluating the solutions in the population, after which the algorithm loops until the termination criterion is satisfied: firstly it selects parents for the reproduction of the next population, then recombines and mutates the chosen parents, finally evaluates the generated population in order either to terminate the search or select the best solutions for the next iteration.

7.2.1. Parents' Heuristics

For example, suppose that the user enters the followings six meetings to be scheduled, with their specifications of the attendees' ranks and domains:

{meeting0, meeting1, meeting2, meeting3, meeting4, meeting5}

The SMA default heuristic to solve this scheduling problem is shown in Fig. 13, where the algorithm firstly ranks the meetings and loops until scheduling the six meeting:

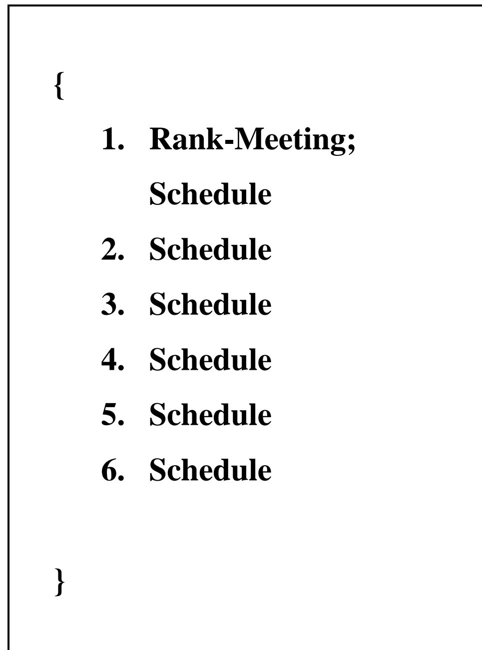


Fig. 13: LGP_parent1 heuristic

The LGP_SUAs consider this heuristic as one of the parents' heuristics (for the superagentLGP and superagentLGP_SP) received from the SMAs. Both LGP_SUAs generate the second parent heuristic by reversing the first, as Fig. 14:

```
{
    1. Schedule
    2. Schedule
    3. Schedule
    4. Schedule
    5. Schedule
    6. Rank-Meeting;
       Schedule

}
```

Fig. 14: LGP_parent2 heuristic

The algorithm in Fig. 14 performs "schedule" five times, then ranks the meetings then calls the scheduling function once more to schedule the last unscheduled meeting.

7.2.2. Crossover Operations

The *Crossover Operation* is the most important primary operation. It is used to modify the structures in GP. In the crossover operation, two existing solutions are sexually combined to form two new ones.

For LGP_SUAs, the middle point (the middle step/function or statement) of the heuristic has been chosen to crossover the two parents. The algorithm reaches this point by dividing the steps of the linear heuristics by two. This point cuts the two parents from the middle element, after which the algorithm crosses over the ***first*** part of $parent_1$ with the ***second*** part of $parent_2$ and the ***first*** part of $parent_2$ with the ***second*** part of $parent_1$.

In our example, the crossover point is 6/2=3; Fig. 15 shows the two parents and details the crossing-over operation for LGP_SUAs, while Fig. 16 shows the generations/children result from this crossing-over operations.

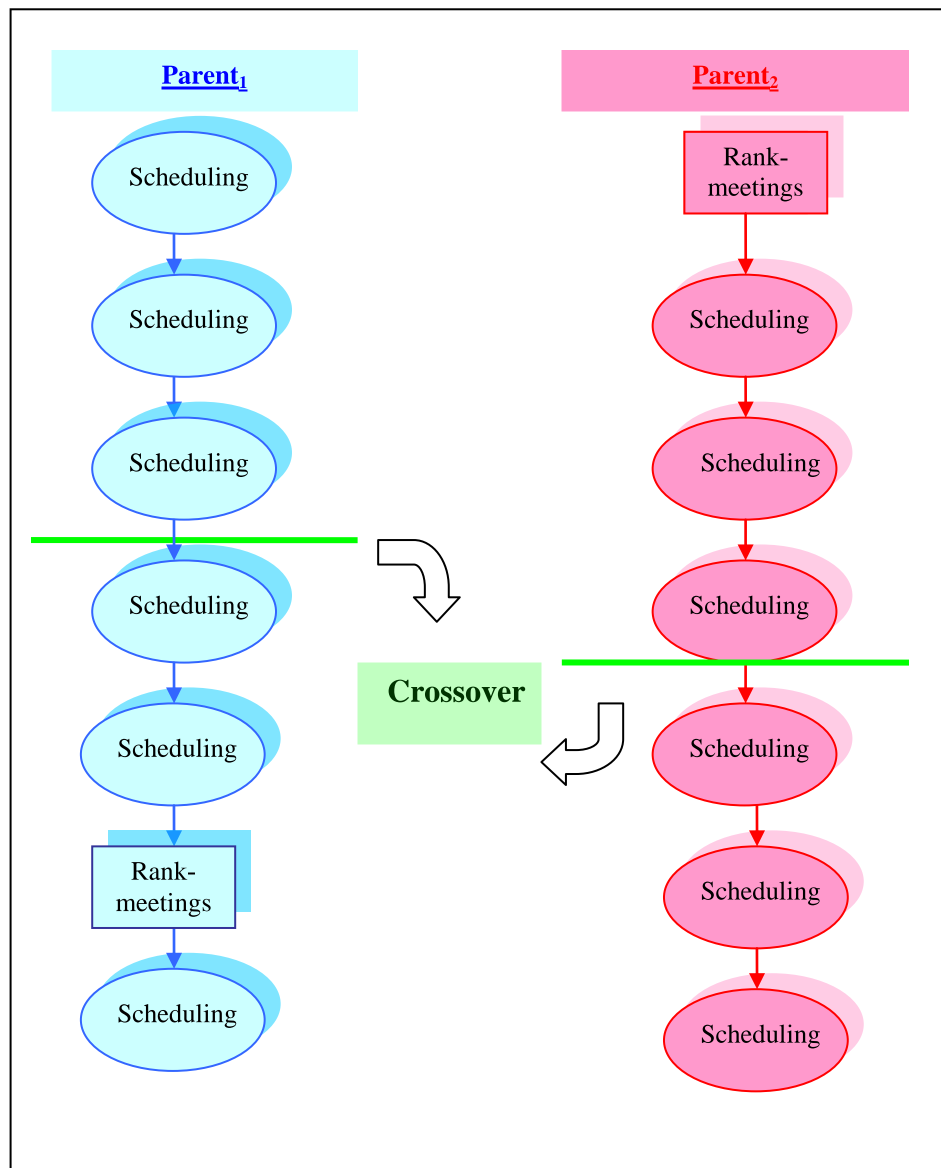


Fig. 15: LGP Crossover - the Parents

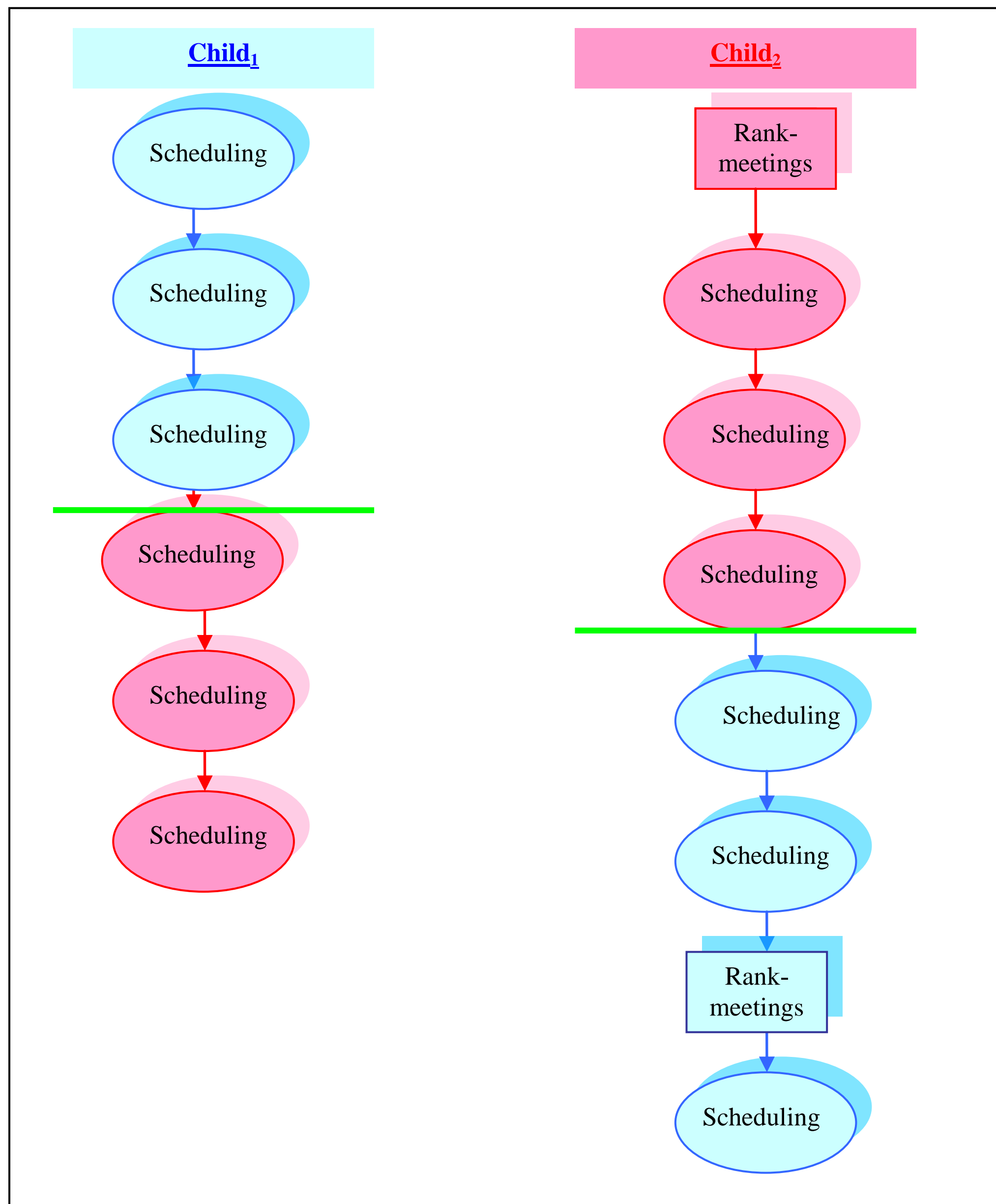


Fig. 16: LGP crossover - the Children

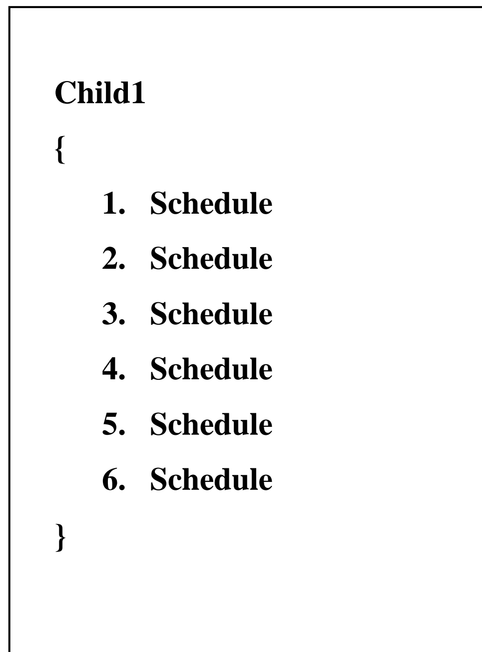


Fig. 17: LGP crossover - Child1

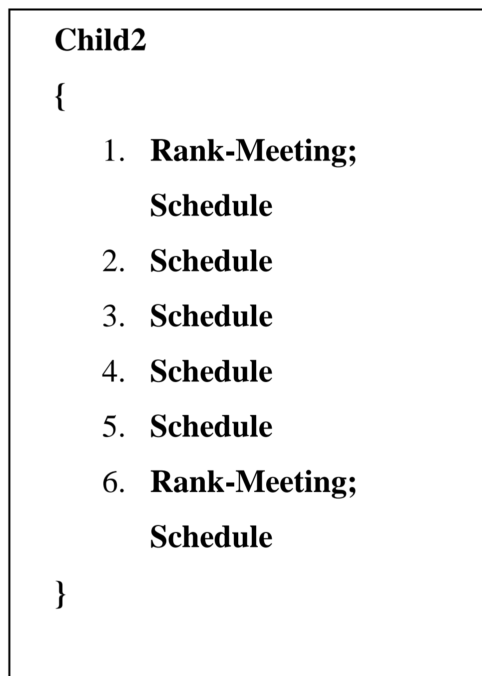


Fig. 18: LGP crossover - Child2

The first generation/ child heuristic resulted from crossover operation is presented in Fig. 17. It loops for scheduling six times. The other child is shown in Fig. 18 which loops for

scheduling six times and ranks the meetings two times in these six iterations; once in iteration/step one and once in iteration/step six.

7.2.3. Mutation Operation

Mutation is an important feature of GP. It creates a new child program by randomly altering a chosen section of a selected parent program. After the crossover operation, LGP performs the mutation operation on the children programs. The proposed mutation operation works as follows: (a) the second element in the heuristic is replaced with a replication of the last element; and (b) the third element in the heuristic is replaced by a replication (copy) of the element preceding the last element of the heuristic

Fig. 19 presents the mutation operation that has been done on child two of Fig. 18, while Fig. 20 presents the output child after this mutation.

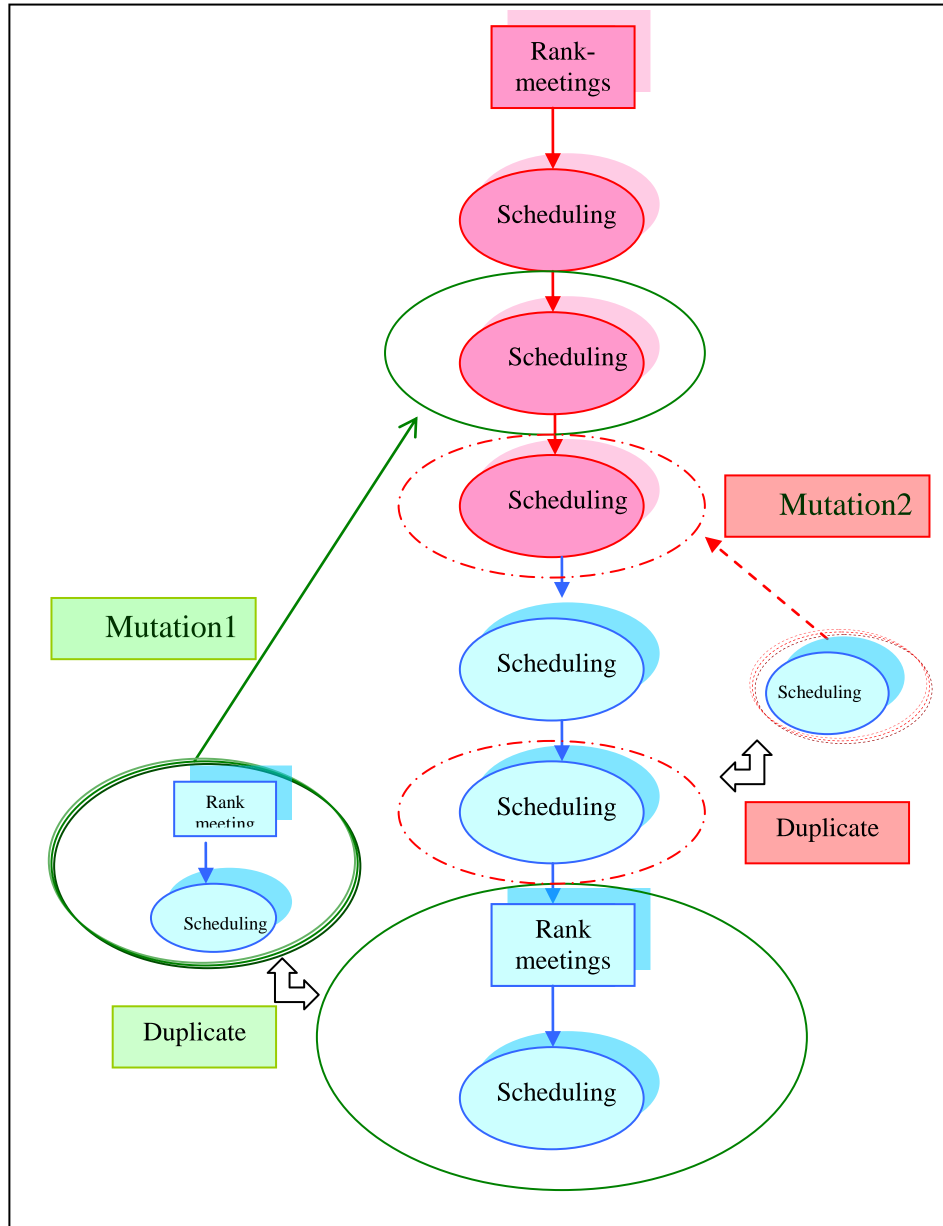


Fig. 19: LGP before mutation - Child2

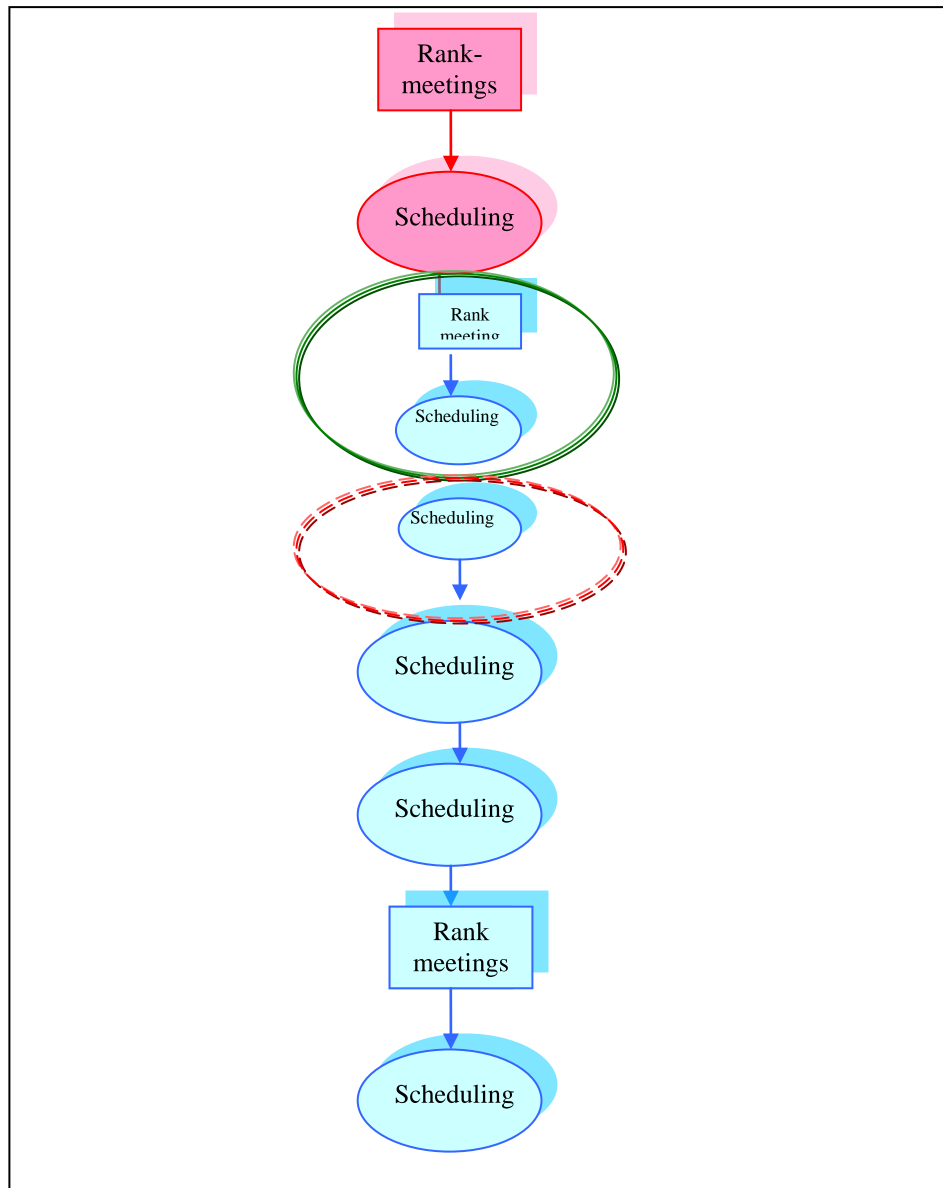


Fig. 20: LGP after mutation-Child2

Child1

{

1. **Schedule**
6. **Schedule**
5. **Schedule**
4. **Schedule**
5. **Schedule**
6. **Schedule**

}

Fig. 21: LGP mutation - Child1

Child2

{

1. **Rank-Meeting;**
   Schedule
6. **Rank-Meeting**
   Schedule;
5. **Schedule**
4. **Schedule**
5. **Schedule**
6. **Rank-Meeting;**
   Schedule

}

Fig. 22: LGP mutation - Child2

Figs. 21 and 22 present the two children/generations heuristics created by mutation. The first child (Fig. 21) loops six times for scheduling, the other one loops six times for scheduling while ranks three times in iterations one, two and six.

This mutation is due to the fact that certain changes are needed to prevent premature convergence to local optima. The experiments have also revealed that changing the first half of each parent is enough to prevent early local optima. This is because in the next iteration the children will be composed of the first half of one parent crossed over with the second half of the second. If the first half has been changed in proper way, then the probability of producing the same children is very small, even if the second part is the same. Therefore modifications of the first half with two elements from the second half have been proposed.

7.2.4. Termination Criterion

Each LGP_SUA evaluates the new generations/children, and the one with a violation total equalling zero becomes the solution heuristic. The algorithm then terminates and the solution heuristic is sent to the SMA to be applied and used to overcome the failure. If this happens, then LGP could generate a new SMA heuristic that solves the problem, which in fact achieves the main goal of HMAA, which is to automatically generate new heuristics for SMAs in order to solve the new problems.

If no child/generation gives a violation value of zero, the algorithm continues with the next iteration in which the children's heuristics with the minimum number of violations are chosen to be parents in the next iteration. The algorithm continues for the specified number of iterations (i.e. 500), then terminates. The best child heuristic, the one with the minimum number of violations, is sent to the SMA to be used.

As mentioned before, two SUAs have been implemented (*superagentLGP,* and *superagentLGP-SP*). They have similar responses to the followings conditions:

1) They terminate if one of the following occurs:
    a) Find a solution with number of violation equals 0
    b) Loop 500 rounds
2) Otherwise they continue looping and generating more generations. They select new parents according to the following conditions:
    a) If the two children's violations are less than the two parent's violations, then the two new parents for the next generation will be the two children generated.
    b) Otherwise: if only one of the children's violations is less than one or both parent's violations, then the two new parents for the next generation will be:
        i) the parent with the minimum number of violations between both of its parents
        ii) the child with the minimum number of violations between both children
    c) Otherwise: if one or two of the children's number of violations equals that of one or both parent, then the two new parents for the next generation will be:
        i) The parent with the minimum number of violations between both of its parents (if both parents have the same number of violations, either of them can be chosen).

ii) The child with the minimum number of violations between its children (if both children have the same number of violations, either of them can be chosen)

3) SuperagentLGP and SuperagentLGP_SP differ only in their response to the following condition:

a) Otherwise: if none of the children's violations number lower than at least one of the parent's,

i) SuperagentLGP's response will be: the two new parents for the next generation will be:

(1) the parent with the minimum number of violations between its parents (if both parents have the same number of violations, either of them can be chosen)

(2) the child with the minimum number of violations between its children (if both children have the same number of violations, either of them can be chosen)

(a) SuperagentLGP-SP: terminates.

7.3 LSP_SUAs (superagentLSP/ superagentLSP_SP)

A new method, *LSP*, is proposed for generating or modifying the existing heuristic. This method is inspired by both GP [49] and LSA [41]: the intent behind LSP is to generate new heuristics/programs using LSA instead of GA techniques.

The motivation for proposing LSP is that, while working on LGP_SUAs, it has been noticed that the two parents' programs have the same components (the same steps or functions). This implies that there is no need for two parents: one parent is enough to provide the desired solutions. LSP is therefore a method for automatically creating a working computer program using local search approaches by modifying one solution/parent of such a program. It iteratively transforms one solution to another chosen in its neighbourhoods. LSP includes *mutation*, *reproduction*, *duplication* and *deletion*.

As mention in the previous section, SUAs' task starts after receiving two messages from the SMA: the first includes a list of meetings with all their details, the second contains the heuristic the SMA follows in order to complete the scheduling process.

The LSP_SUAs (superagentLSP and superagentLSP-SP) use the ***LSP*** approach to generate new heuristics; the LSP needs one parent's heuristic in order to generate the new heuristics' children. The two LSP_SUAs receive the SMA heuristic from the SMAs and make some modifications to it before utilising it as a parent for the LSP process.

As mentioned before the SMA heuristic is a *Prioritised/Ranked-Meetings scheduling heuristic* that ranks meetings according to certain properties using the *Rank-Meeting* function and then using the *Schedule function*, that schedules meetings according to defined specifications, for all the meetings.

The *preparatory steps* for the basic version of LSP are the same as for LGP.

```
Initialise population (the solution is a mutation of the SMA heuristic);
Evaluate population;
Loop until one of the termination criteria is satisfied
{
        Select solution whose neighbourhood is to be searched
        Generate neighbourhood (crossover with itself) and mutations opertation
        Evaluate neighbours
}
```

Fig. 23: Pseudo code for LSP

Fig. 23 presents the pseudo code for LSP; it starts with an initial population/heuristic and evaluates this population; then the algorithm enters a loop until satisfying the terminations criterion. In each iteration the algorithm firstly selects a solution/heuristic to search in its neighbourhood for a better solution, then generate a neighbourhood for the selected solution, finally evaluate each neighbour in order either to terminate the search (if the termination criterion is met), or to choose the best solution in the next iteration.

7.3.1. Solution's Heuristic

An assumption is that the user enters six meetings to be scheduled, with their specification according to the attendees' ranks and domain.

$$\{meeting_0, meetign_1, meeting_2, meeting_3, meeting_4, meeting_5\}$$

The SMA heuristic received by the LSP_SUAs is shown in the following figure:

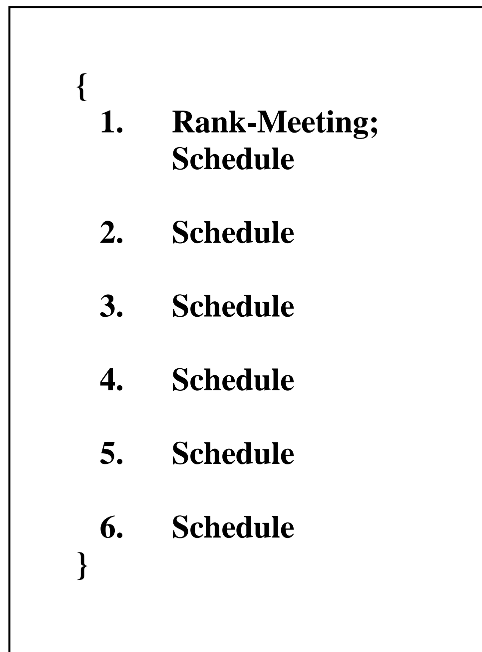
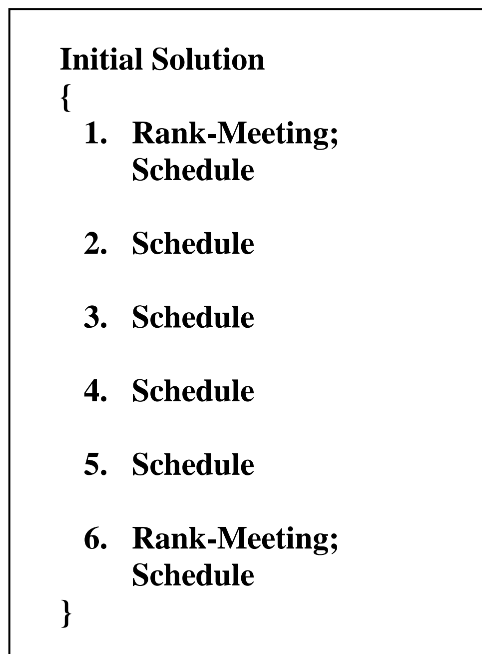


Fig. 24: SMA heuristic

It is proposed that the solution heuristic be initially generated in superagentLSP and superagentLSP-SP by Replacing the last element of the received heuristic with a copy of the first element of that heuristic, as shown in Fig. 25.



Initial Solution
{
1. **Rank-Meeting;**
   Schedule
2. **Schedule**
3. **Schedule**
4. **Schedule**
5. **Schedule**
6. **Rank-Meeting;**
   Schedule

}

Fig. 25: Solution Heuristic

This is a kind of mutation operation on the initial solution heuristic, which is to alter one gene value in the solution from its initial state. With this, the probability of the dissimilarity for the parent's two equals parts ($half_1$ and $half_2$) decreases; (the first element in the first part/half will be the same of the last element in the second part/half). Hence the algorithm will be able to find better solutions more quickly. Moreover, this mutation decreases the probability to reach to local optima from the beginning.

7.3.2. Crossover Operations

In the crossover operation, specific point has been chosen to crossover the solution in order to generate neighbours for the solution heuristic. The middle point (or the middle step/function or statement) of the heuristic has been proposed. The algorithm reaches this point by dividing the linear heuristic steps into two. This point cuts the parent form the middle element, after which the algorithm combines the first part of the solution

with itself in order to generate one neighbour, and the second part of the solution with itself to generate another as shown in Fig. 26; In this example the crossover point is 6/2=3

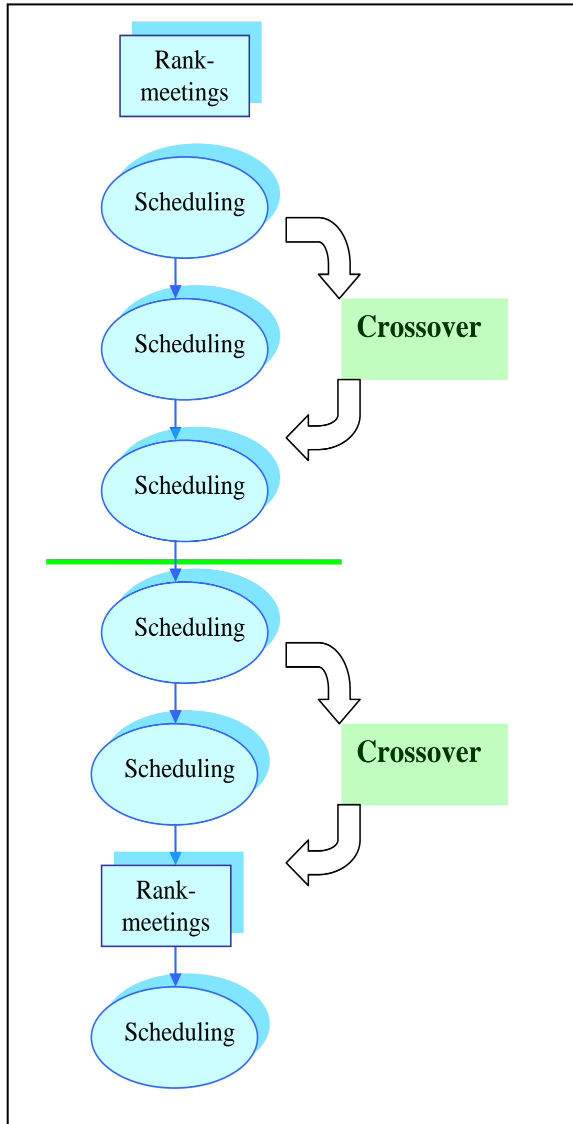


Fig. 26: LSP crossover

$Neighbour_1$
{

1. **Rank-Meeting**
   Schedule
2. **Schedule**
3. **Schedule**
4. **Rank-Meeting**
   Schedule
5. **Schedule**
6. **Schedule**

}

Fig. 27: LSP $crossover_neighbour_1$

$Neighbour_2$
{

1. **Schedule**
2. **Schedule**
3. **Rank-Meeting**
   Schedule
4. **Schedule**
5. **Schedule**
6. **Rank-Meeting**
   Schedule

}

Fig. 28: LSP $crossover_neighbour_2$

The neighbourhood outcomes from this crossover operation is shown in Figs. 27 and 28; the first neighbour heuristic (Fig. 27) loops the scheduling function six times and ranks meetings two time in iteration one and four. The other neighbour (Fig. 28) loops the scheduling function six times and ranks the meetings two times in iteration three and six.

7.3.3. Mutations Operation

After the crossover operation, LSP performs some mutation operations on the neighbourhood programs, by altering chosen parts of selected programs. The first mutation or transformation swaps two specified elements: the one in index (crossover point /2) with the one in index (last index – (crossover point/2)).

In this example the crossover point equals 3 then crossover point/2 is 3/2 =1.5; that means index "2" is the index of "crossover point/2". The two elements that will be swapped are: the first element is the one in index (2); and the other one is the one in index (6-2=4), hence swapping 2 and 4. Fig. 29 illustrates this mutation on the first neighbour:

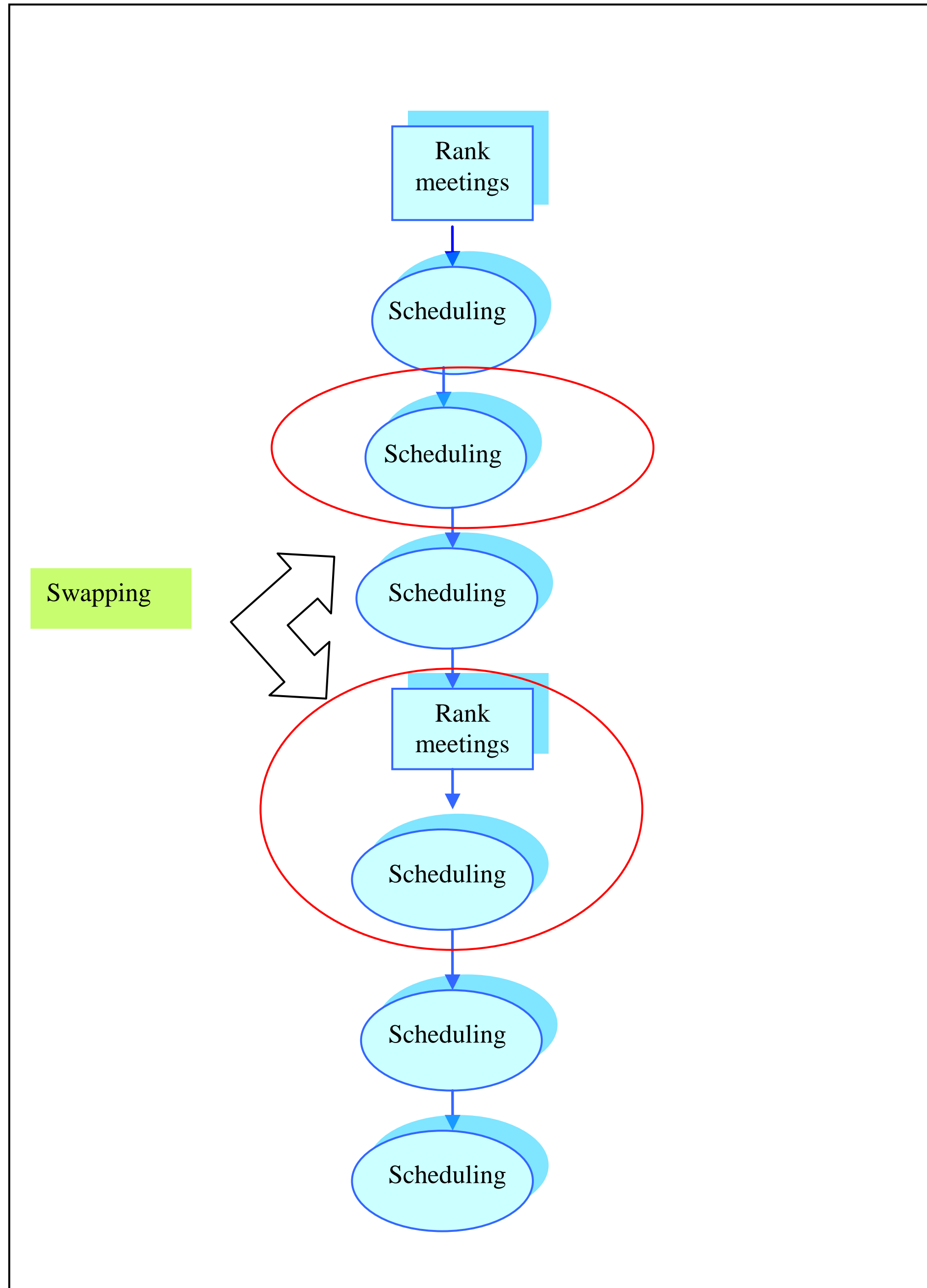


Fig. 29: LSP $mutation_1_neighbour_1$

$Mutation_1_Neighbour_1$

{

1. **Rank-Meeting**
   Schedule

4. **Rank-Meeting**
   Schedule

3. **Schedule**

2. **Schedule**

5. **Schedule**

6. **Schedule**

}

Fig. 30: LSP $mutation_1_neighbour_1$

$Mutation_1_Neighbour_2$

{

1. **Schedule**

4. **Schedule**

3. **Rank-Meeting**
   Schedule

2. **Schedule**

5. **Schedule**

6. **Rank-Meeting**
   Schedule

}

Fig. 31: LSP $mutation_1_neighbour_2$

Fig. 30 and 31 present the heuristics formed by applying the first mutation operation on $Neighbour_1$ and $Neighbour_2$, respectively.

This swap is performed because the two equal parts of the heuristic are identical, so mixing is needed between elements from both parts in order to differentiate them.

The second mutation is sliding the first half/part of each neighbour by one step as shown in Fig. 32; this sliding on one half of the heuristic is in order to decrease the opportunity for similarity between the two equal halfs/parts in the heuristic; Fig. 32 illustrates this mutation on neighbour one.

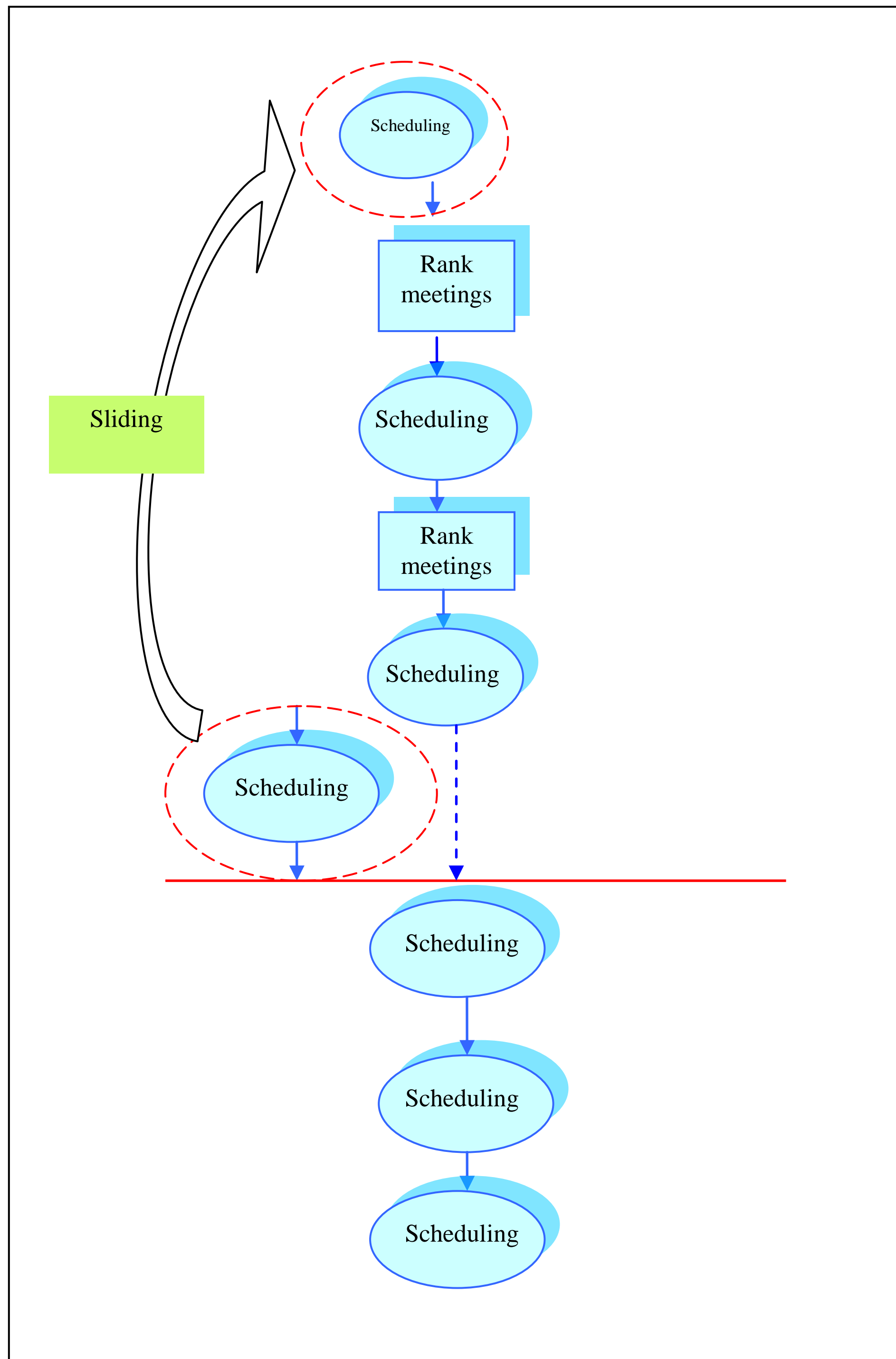


Fig. 32: LSP mutation$_2$_neighbour$_1$

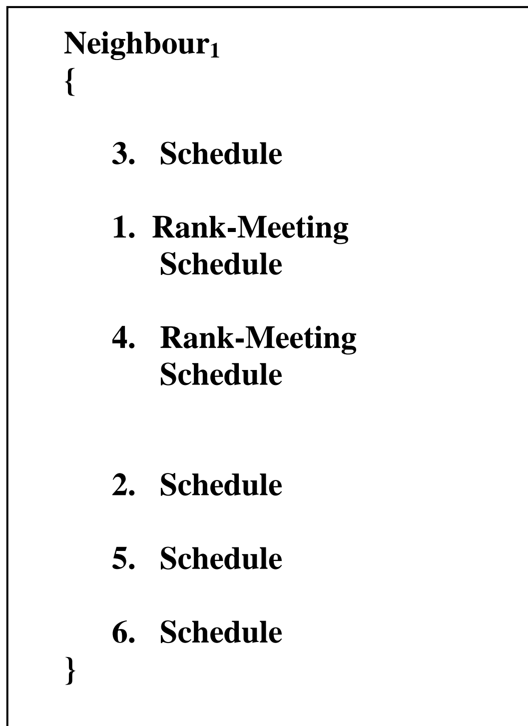


Fig. 33: LSP mutation$_2$_neighbour$_1$

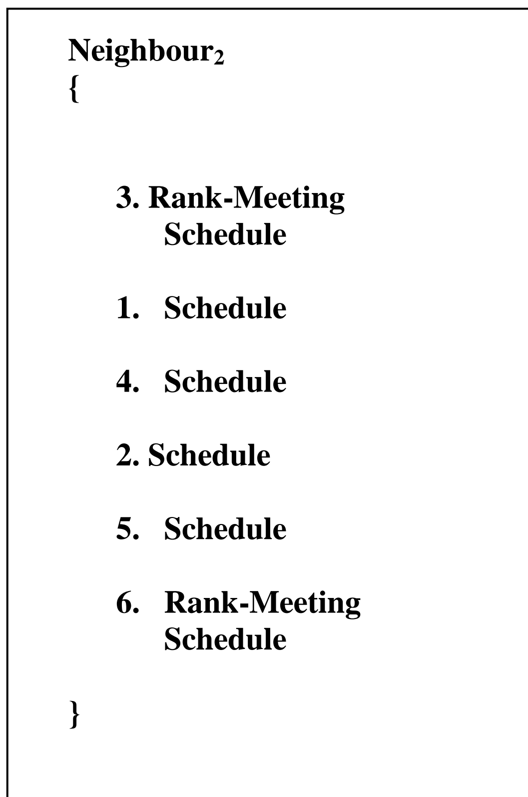


Fig. 34: LSP mutation$_2$_neighbour$_2$

The outcomes neighbourhood heuristics of this mutation is shown in Figs. 33 and 34.

The last mutation or transformation transposes all the algorithms or solution steps of each neighbour by two steps, so that enormous variants between the neighbours are generated in the next neighbourhood. Illustration for this mutation on the first obtained neighbour is shown in Fig.35.

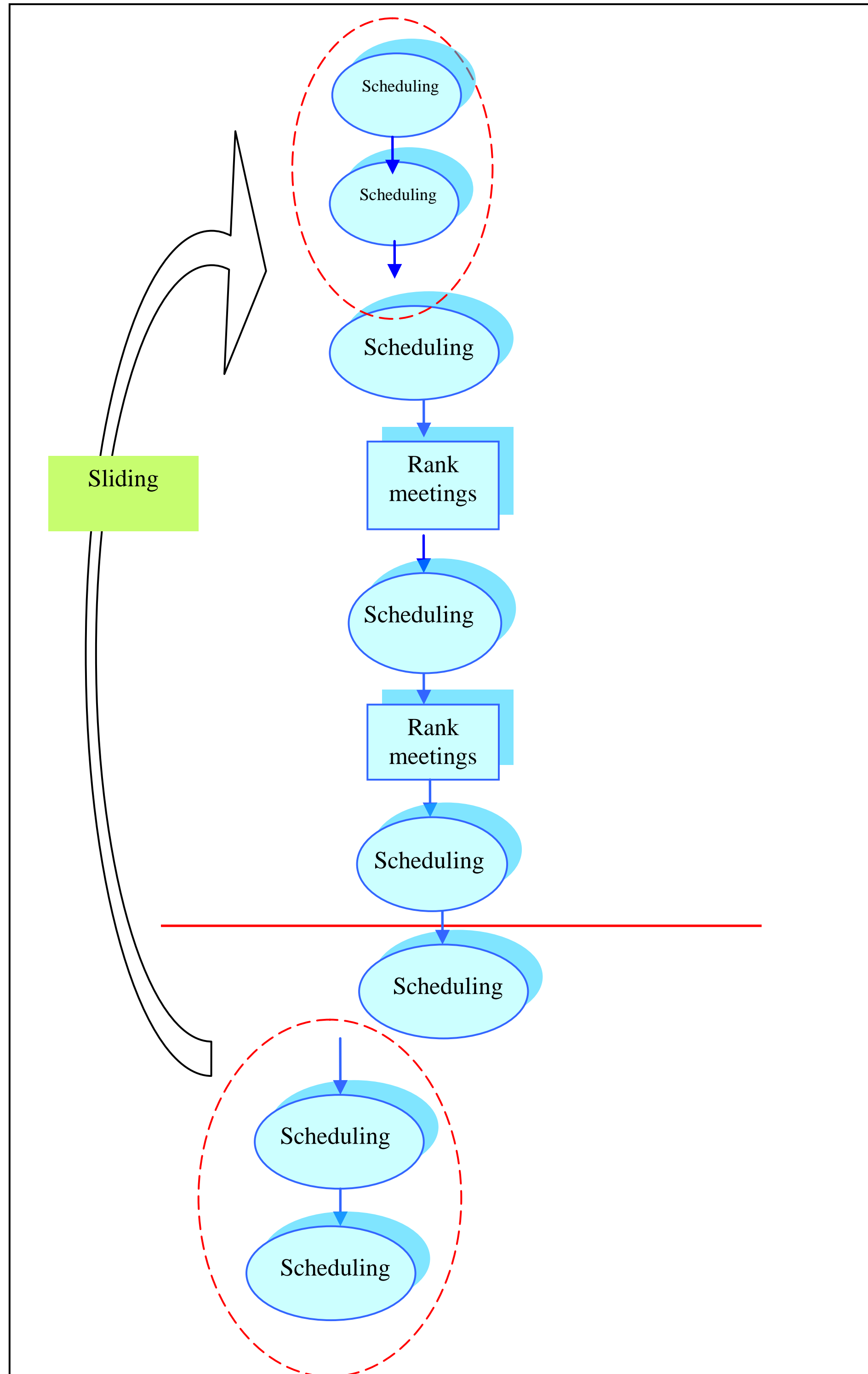


Fig. 35: LSP mutation$_3$_neighbour$_1$

Neighbour$_1$
{

5. Schedule

6. Schedule

3. Schedule

1. Rank-Meeting Schedule

4. Rank-Meeting Schedule

2. Schedule

}

Fig. 36: LSP mutation$_3$_neighbour$_1$

Neighbour$_2$
{

5. Schedule

6. Rank-Meeting Schedule

3. Rank-Meeting Schedule

1. Schedule

4. Schedule

2. Schedule

}

Fig. 37: LSP mutation$_3$_neighbour$_2$

At the end of these mutations, the obtained neighbours are shown in Figs. 36 and 37; the first neighbour heuristic (Fig. 36) loops for scheduling meeting six times through which it ranks the meeting two times; in iteration four and five. The second neighbour heuristic (Fig. 37) loops for scheduling meeting six times through which it ranks the meeting two times; in iteration two and three. $Mutation_{1,}$ $mutation_2$ and $mutation_3$ all of them are used as mutations operation in the proposed LSP.

7.3.4. Termination Criterion

The superagentLSP and superagentLSP_SP try the new generations/ neighbours; the one that returns the number of violations equalling zero is the solution heuristic, which will be send to the SMA to be used in order to overcome the failure. If no neighbour gives such a result, the algorithm continues with the next iteration, in which the neighbouring heuristic with the minimum number of violations is chosen as the solution whose neighbourhood is to be searched in the next iteration. The algorithm continues for the defined number of iterations (500), and the best neighbour heuristic (i.e. with the minimum violation) is sent to the SMA to be used.

As mentioned before, two LSP_SUAs have been implemented (*superagentLSP,* and *superagentLSP-SP*) each of which have similar responses to the followings conditions:

a) They terminate if one of the following occurs:

   i) A solution with violation =0 is found

   ii) 500 rounds are looped

b) Otherwise they continue looping and searching for better neighbourhoods, selecting new solutions according to the following criteria:

   i) If the two neighbours' violations total less than the solution violation, the new solution for the next iteration will be the neighbour with the minimum number of violations.

   ii) Otherwise, if one of the neighbour's violations is less than the solution violation, then the new solution for the next generation will be the neighbour with the minimum number of violations.

   iii) Otherwise, if one or two of the neighbour's violations equals the solution violations, then the new solution for the next generation will be the neighbour with the minimum number of violations.

SuperagentLSP and superagentLSP_SP only in their responses to the following condition:

   iv) Otherwise if none of the neighbour's violations is less than the solution's violation:

      (1) The SuperagentLSP response will be that the new solution for the next generation will be the neighbour with the minimum number of violations.

      (2) SuperagentLSP-SP: terminates.

7.4. Summary

The chapter illustrates t the SUAs part of HMAA, it presents SUAs main goals; where some of which *search for feasible solution heuristic* while the others *search to optimise the solution heuristic*. It discusses the algorithms they use; two types of EAs have been used (LGP and LSP); and it illustrates by example how each SUA computes its solution heuristic. The performance of these SUAs and the feasibility of EAs that are used by SUA in the HMAA will be measured in the next chapter.

Chapter 8

Experimental Results and Evaluation

Objectives

- To present the HMAA implementation.
- To present the developed experiments on HMAA.
- To evaluate the performance of HMAA.
- To evaluate the performance of SUAs that implement EAs.
- To measure the feasibly of LSP.

8.1 Introduction

This chapter illustrates some experiments performed on HMAA. The experiments have been done on both frameworks search and optimisation problem solving frameworks, where two stages of experiments have been conducted.

Stage one of experiments is done on small agent predefined heuristic (prioritised/ranked heuristic and local search repair strategy), where two groups of experiments have been done as shown in Section 8.2. The first group is measuring the feasibility of the proposed heuristic by assessing of the (ranking property) before scheduling, and the (local search repair strategy) to optimise the obtained solution. This assessing is done by generating number of meetings distributed amongst defined number of agents. And run

the heuristic on stages each of which executing and measuring one of the defined property.

The second group of experiments evaluates the SMA based approach mentioned in [7]. In this case a number of random meetings are generated, depending on the same parameters used in the approach.

In Stage two three groups of experiments are conducted. The first presents simple cases with differing numbers of attendees and a variety of situations and combinations of meetings. The aim of these experiments is to examine the feasibility of running the *HMAA for large number of attendees*.

The second group contains more complicated cases and susceptible situations with one user, who has chosen their data carefully (as will be seen below), and a large number of meetings. This group was constructed in order to measure the *feasibility of the system in very complicated cases when the domain range data is limited*. The cases measure the feasibility of *the local optima* in the searching process. This means that instead of continuing until the solution is found, or completing a specified number of rounds, the search ends when the number of violations of the new generations are more than that of the parents' violations (for LGP), or the neighbourhood’s violations are more than those of the current solution’s (for LSP).

The final group consists of randomly selected cases that measure the *feasibility of the proposed architecture in different situations*. It must be said that each meeting is determined by the specific *domain* of possible dates on which the meeting can be held, and the potential *attendees* with their *weight/significance*. This is because the HMAA aims partly to optimise scheduling, which means reducing the number of violations as

much as possible so that if there is no feasible schedule without overlaps, the system schedules the meeting by overlapping the meetings of the least significant attendees.

8.2. Stage One: SMA Experiments

Experiments Group 1

In these experiments, three agents have been initialised and thirty meetings distributed amongst them. Each agent initialises some of them with different domains, some of which overlap.

The experiments have been performed in four stages:

- Stage One- the agent *ranks* the meetings before scheduling them and *does not schedule violated meetings (search problem)*. (The agents arrange the meetings according to the mentioned equation –equation 5.1 - and then schedule the meetings according to their ranks. Once it finds a violation, the meeting will not be scheduled. The feasibility of the algorithm is measured by the number of unscheduled meetings).

- Stage Two- the agent *ranks* the meetings before scheduling them and *schedules the violated meeting (optimisation problem)*, after which it *performs a local search* (the agent arranges the meetings according to the mentioned equation 5.1, and then schedules them according to their ranks; it schedules all the meetings even if there is a violation, then perform the local search in order to optimise the violation. The feasibility of the algorithm is measured by the total violation).

- Stage Three- the agent *does not rank* the meeting, and *does not schedule the violated meetings (search problem)* (the agent schedules the meetings according to how the user has entered them; once there is a violation the algorithm will not schedule the meeting. The feasibility of the algorithm is measured by how many meetings are not scheduled).

- Stage Four- the agent *does not rank* the meeting but *does schedule the violated meetings (optimisation problem)*, after which it *performs a local search* (the agent schedules the meetings according to how the user has entered them; it schedules all the meetings even if there is a violation, then performs a local search in order to optimise the violation. The feasibility of the algorithm is measured by the total violation).

In these four stages of experiments, the feasibility of ranking the meetings before scheduling is measured by comparing the result of the experiment's first stage with those of the third (Fig. 38), and likewise the second with the fourth (Fig. 39), and then measuring the feasibility of the local search by comparing the results before and after that search in Stage Two (Fig. 40) and Stage Four (Fig. 41). The results of Stage Two and Stage Four are finally compared in order to measure the feasibility of the ranking before and after the local search (Fig. 42).

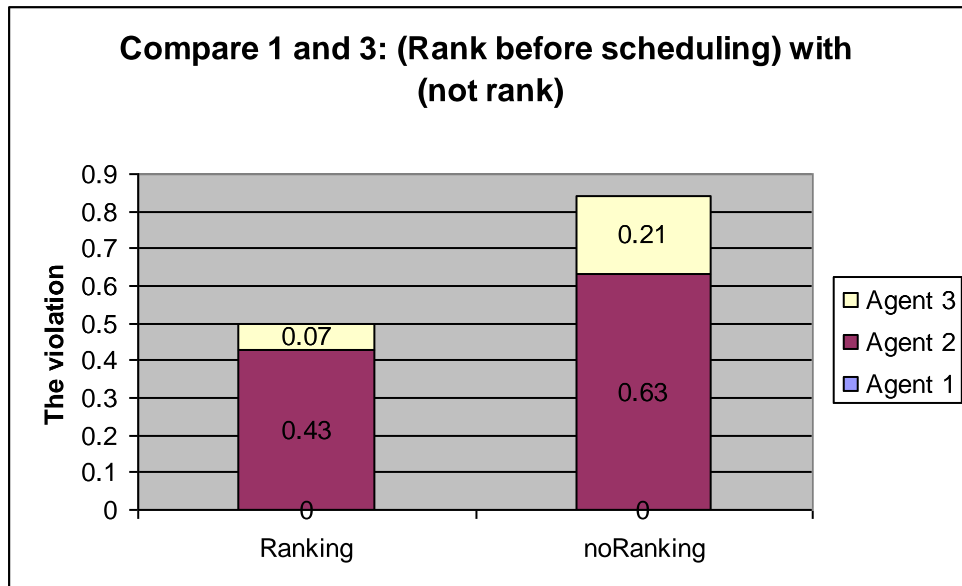


Fig. 38: The feasibility of the ranking (comparing Stages 1 and 3)

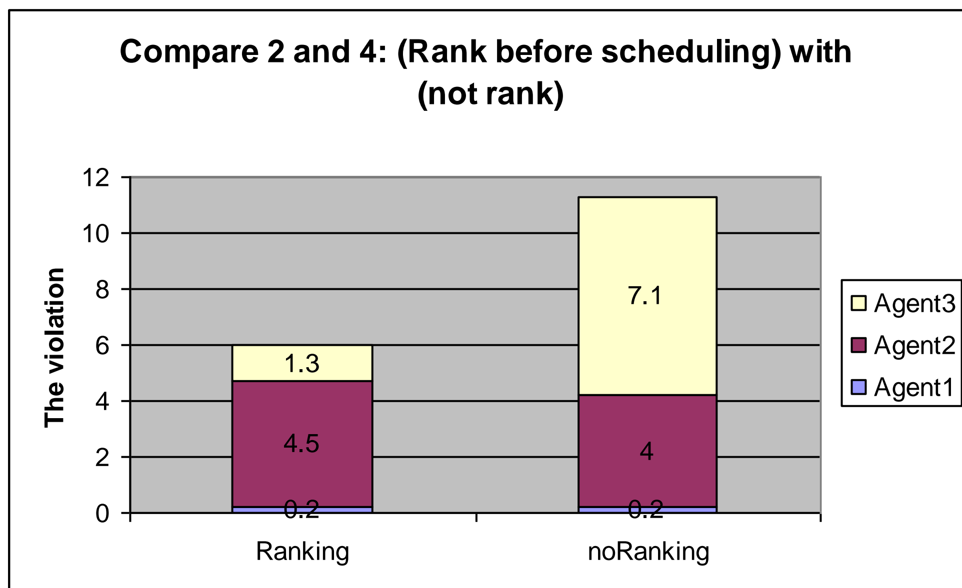


Fig. 39: The feasibility of the ranking (comparing Stages 2 and 4)

Figs. 38 and 39 show that there are tangible reductions in the violation when ranking the meetings before scheduling; the total violations for the three agents in Stages one and two were “00.50, 06.00” respectability; while it were in Stages three and four “00.84, 11.3” respectively, this indicates that the ranking process reduces the total violations

and so improves the scheduling results. This is due to the fact that the **ranking** property/function gives a glow to the agent about the difficulties to schedule the meetings, and then starts with the more difficult.

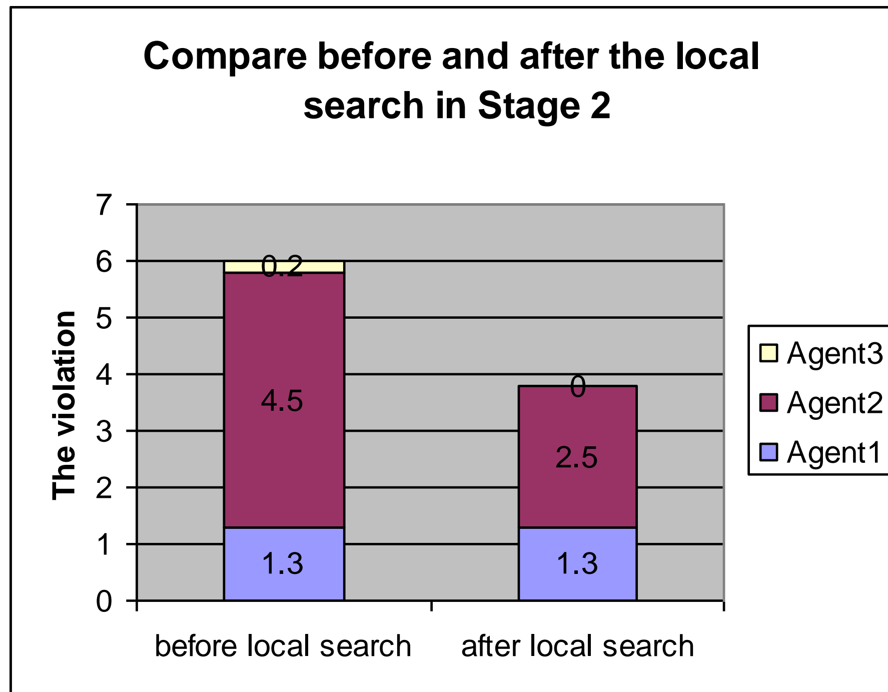


Fig. 40: the feasibility of local search in (2)

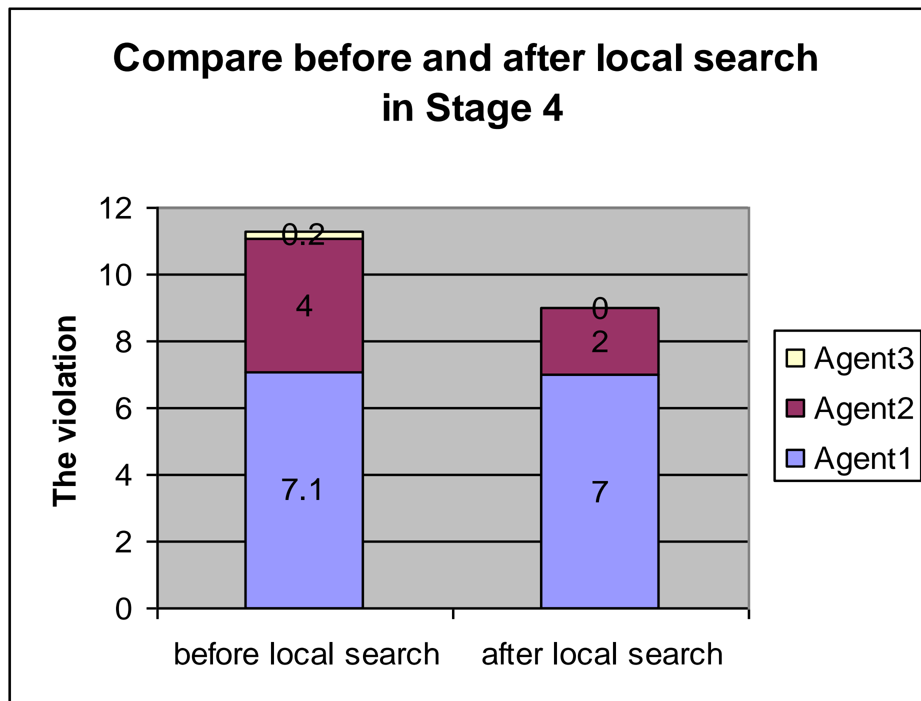


Fig. 41: the feasibility of local search in (4)

Figs. 40 and 41 show that the local search has improved the scheduling process, by decreasing the violation from “06.00” up to “03.80” in Fig 40; and from “11.30” up to “09.00” in Fig. 41; which means that it increases the opportunity of scheduling more meetings.

From the figures above, it is clear that the ranking and local search have improved the performance of the scheduling process; the following shows how combining these two algorithms affect the scheduling process.

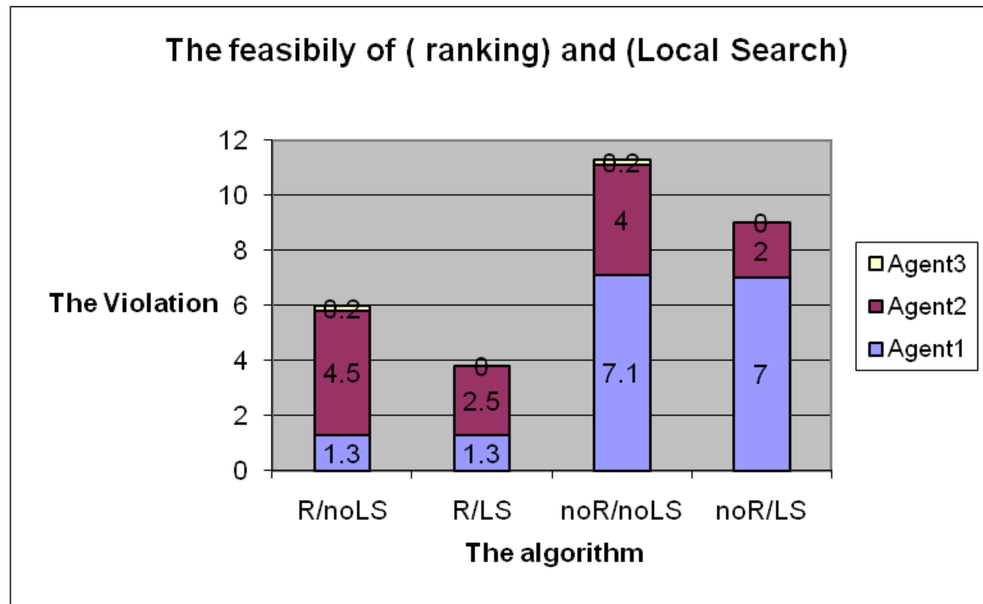


Fig. 42: The performance of the FMSH

Column (R/noLS) shows the performance of the algorithm that only ranks the meetings (Stage two before local search), column (R/LS) the performance of the algorithm that both ranks the meeting and performs local searches for the schedules (Stage two after local search; which is the main thrust behind the present work), column (noR/noLS) the performance of the algorithm that does not rank or perform local searches (Stage four before local search), and column (noR/LS) the performance of the algorithm that only performs local search for the schedules(Stage four after local search).

It is clear from Fig. 42 that (noR/noLS) has the maximum number of violations. This means that ranking or local searching would result in better scheduling. From the same figure, it can also be seen that applying both ranking and local searching together would offer the best scheduling capability. Column R/LS has the minimal violation, indicating that this algorithm that ranks the meetings and performs local searches has improved the performance of the SMA, and has the capacity to find better schedules for many

meetings without affecting the small size of the agent nor their ability to run on small devices.

Experiments Group 2

To evaluate the SMA proposed heuristics, a comparison has been done between:

- Approach 1: "Prioritised/Ranked-Meetings Scheduling heuristic for Search problem solving".
- Approach 2: "Prioritised/Ranked-Meetings Scheduling heuristic for Optimisation problem solving with local search repair strategy".
- And Approach 3: "Meetings scheduling solver enhancement with local consistency reinforcement" [7].

As has been clarified in Section 3.4.2 Hassine et al. in their work "Meetings scheduling solver enhancement with local consistency reinforcement"[7, 37, 38] have devised a new approach based on distributed reinforcement of node and arc consistency (DRAC) to solve MSPs.

The comparison is done by generating number of random meeting scheduling problems using the same parameters as in [7]. These are:

- n= 10 is the number of agents in the framework
- m=3 is number of new meetings to be scheduled
- Att $\in$ {3, 5, 7} is the number of the attendees in these meetings

- T=two months, which is equivalent to 60 time slots available in the calendar of each agent, $c_H \in \{20, 30, 40, 50\}$ are the hard constraints in each attendee calendar, these being the timeslots that could not be scheduled, and accordingly $d \in \{66\%, 50\%, 33\%, 16\%\}$ is the maximum possible number of timeslots available per T.

For each pair <Att, c_H > 9 instances were generated, after which the average of scheduled meetings using approach 1 and the average of un-violated meetings using approach 2 are measured and compared with the average of scheduled meetings using approach 3**.**

Table 1: Percentage of scheduled meetings of 108 cases

	<3,20>	<3,30>	<3,40>	<3,50>	<5,20>	<5,30>	<5,40>	<5,50>	<7,20>	<7,30>	<7,40>	<7,50>
Appraoch1	**100**	**70**	**20**	**7**	**35**	**30**	**8**	**7**	**14**	**12**	**2**	**2**
Appraoch2	**100**	**70.24**	**20.32**	**7.16**	**35.32**	**30.0**	**8.16**	**7.08**	**14.0**	**12.16**	**2**	**2**
Appraoch3	**100**	**50**	**10**	**3**	**50**	**10**	**3**	**0**	**15**	**3**	**3**	**0**

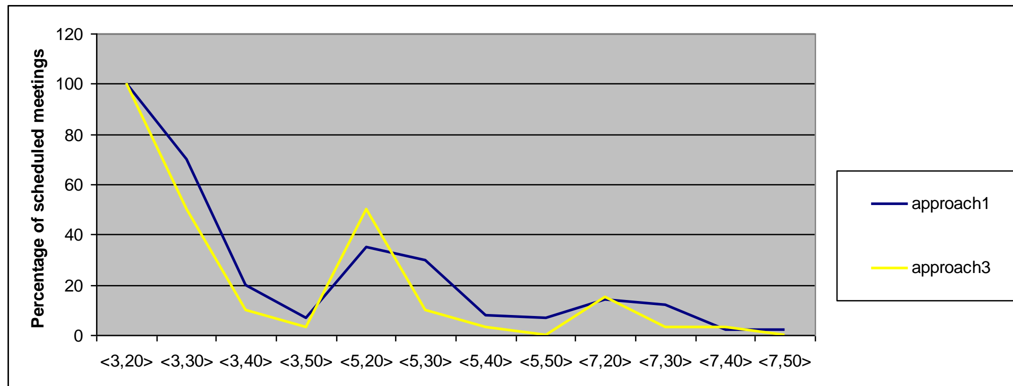


Fig. 43: The performance of Prioritised/ranked heuristic search problem compared with the local consistency approach

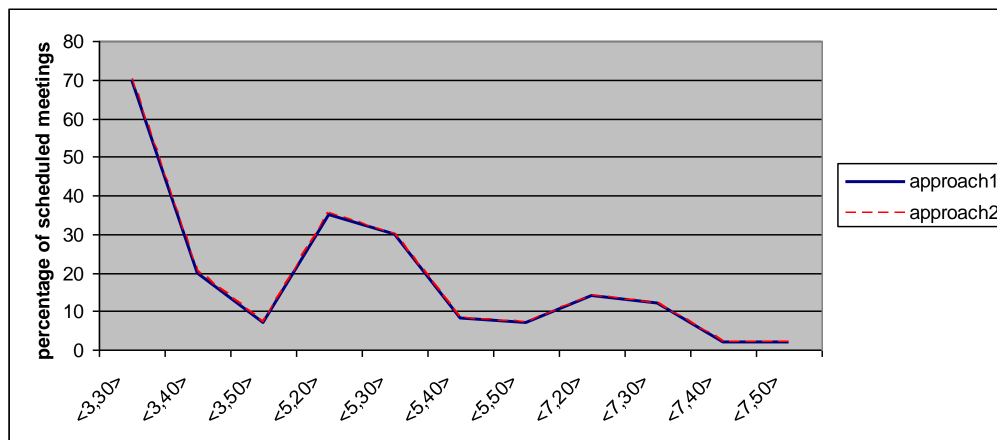


Fig. 44: Comparing the performance of Prioritised/Ranked-Meetings Scheduling heuristic for Search problem solving with Prioritised/Ranked-Meetings Scheduling heuristic for optimisation problem solving and local search repair strategy

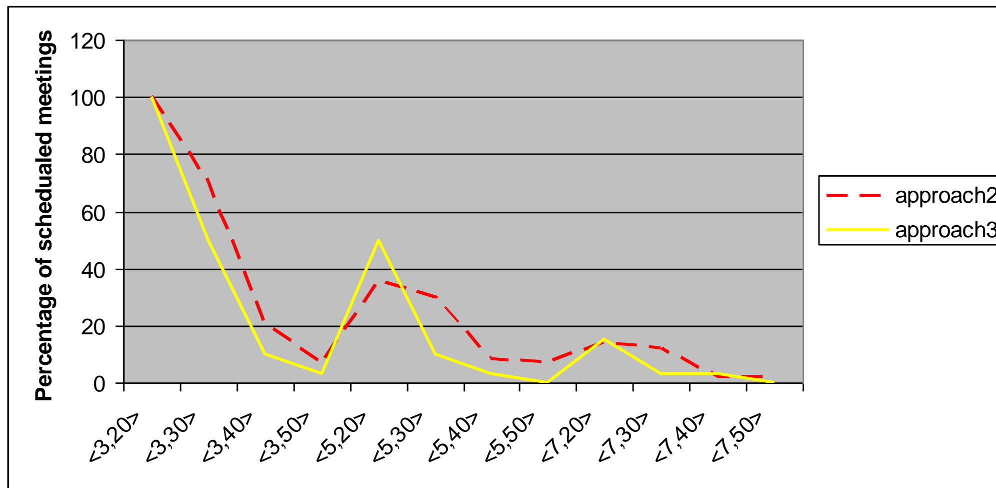


Fig. 45: Comparing the performance of Prioritised/ranked heuristic optimisation problem and local search with local consistency approach

In the Figs. 43, 44 and 45 the dark blue lines show the performance of SMAs using approach 1, the dashed red lines are the performance of SMAs using approach 2, and the yellow lines are the performance of SMAs using approach 3; the local consistency approach [7].

As shown in the Fig. 43, the performance of approach 1 in general is better than approach 3, since it considers the difficulty-complexity- of scheduling the meeting, and starts with the most difficult meetings in sequence.

It can be noticed in Fig. 44 that local search repair strategy used in approach 2 improves the performance of the SMAs compared with approach 1 that does not use this repair strategy.

Fig. 45 compares the performance of the SMAs using approach 1 with others using approach 3. The results show that approach 1 with its proposed approach is better than the local consistency approach. This is due to the fact that in SMA heuristic starts with

the most difficult meetings and local search optimises the obtained solution that would improve their performance.

The proposed SMA heuristics and repair strategy partially overcomes the limitation of fixed predefined heuristic. Since from the experiments above; it has been illuminated that applying local searching –an evolutionary approach- on the obtained solution/scheduling improves the scheduling, and in some cases produces new solutions/schedules that could not be obtained by the predefined heuristic.

This has motivated us to extend the system with super-agents (SUAs) that employ evolutionary approaches (EAs). These EAs are used by SUA in order to generate new heuristic for SMAs; where SUAs would propose new heuristics to be executed by the SMAs. This is aimed to overcome that failure and optimise the scheduling with the new hybrid architecture proposed in the present work.

8.3. Stage Two: HMAA Experiments

Experiments Group 1

These experiments have been done on both HMAA optimisation and search frameworks, and consist of simple cases with differing numbers of attendees and a variety of situations and combinations of meetings. The aim of these experiments is *to demonstrate the feasibility of the hybrid multi-agent architecture for any number of attendees*.

As it can be seen below, each case starts by defining the number and names of attendees. The doubled lined tables are data tables showing the meetings, the attendees

with their ranks and domains. The optimisation framework column specifies the dates found by the SMAs' predefined heuristics using optimisation HMAA, while the search framework column specifies the dates found by the SMAs' predefined heuristics using search HMAA.

This is followed by:

1) Firstly, SuperagentLGP results or outcomes are characterised by
    a) the generated heuristic of the superagentLGP,
    b) the rounds or loops needed to find the solution
    c) the final violation
2) Secondly, SuperagentLSP outcomes are characterised by
    a) the generated heuristic by the superagentLSP.
    b) its loops.
    c) its violation.

The dashed tables are the results of the framework, the rectangle around certain domain timeslots specify the dates that have been found by the SMA using the *superagentLGP* heuristic and the *superagentLSP* heuristic.

Case 1:

Number of Users=1: Serein (Ser)

Number of meetings= 7

Data table				
Meetings	**Attendees(rank)**	**Domain**	**Optimisation framework**	**Search framework**
Meeting1	Ser (1.0)	{1 Oct,2 Oct,3 Oct,5 Oct}	5 Oct	5 Oct
Meeting2	Ser (1.0)	{1 Oct,2 Oct,5 Oct,7 Oct}	7 Oct	violated
Meeting3	Ser (1.0)	{2 Oct,6 Oct,8 Oct}	2 Oct	2 Oct
Meeting4	Ser (1.0)	{3 Oct,7 Oct,8 Oct}	3 Oct	3 Oct
Meeting5	Ser (1.0)	{1 Oct,2 Oct,3 Oct}	1 Oct	1 Oct
Meeting6	Ser (1.0)	{1 Oct,2 Oct,7 Oct}	7 Oct	7 Oct
Meeting7	Ser (1.0)	{2 Oct,4 Oct,7 Oct}	4 Oct	4 Oct
Small agent violation is			2.0	1.0

Table 1: Case 1 data table

1) superagentLGP outcomes:

   a) The superagentLGP-generated heuristic is

      {

      Rank-Meetings

      Schedule

      Rank-Meetings

      Schedule

      Schedule

      Schedule

      Schedule

      Schedule

      Schedule

      }

   b) The total number of rounds = 0

   c) The violation after using the new heuristic = 0.0

2) superagentLSP outcomes:

   a) The superagentLSP Generated heuristic is

      {

      Rank-Meetings

      Schedule

      Rank-Meetings

      Schedule

      Schedule

Schedule

Schedule

Rank-Meetings

Schedule

Schedule

}

b) The total number of rounds=1

c) The violation after using the new heuristic (0.0)

Results table

Meetings	Attendees(rank)	Domain
Meeting1	Ser (1.0)	**{1 Oct,2 Oct,3 Oct, 5 Oct}**
Meeting2	Ser (1.0)	**{1 Oct,2 Oct,5 Oct,7 Oct}**
Meeting3	Ser (1.0)	**{2 Oct, 6 Oct,8 Oct}**
Meeting4	Ser (1.0)	**{3 Oct,7 Oct,8 Oct}**
Meeting5	Ser (1.0)	**{1 Oct,2 Oct,3 Oct}**
Meeting6	Ser (1.0)	**{1 Oct,2 Oct,7 Oct}**
Meeting7	Ser (1.0)	**{2 Oct,4 Oct,7 Oct}**

Small agent violation is **0.0**		

Table 2: Case 1 results Table

Case 2:

Number of Users=2: Serein (Ser), Ashraf (Ash)

Number of meetings=4

Data Table				
Meetings	**Attendees(rank)**	**Domain**	**Optimisation framework**	**Search framework**
Meeting1	Ser (1.0)	{1 Oct, 2 Oct, 3 Oct }	3 Oct	violated
Meeting2	Ser (0.2) Ash (0.8)	{2 Oct, 4 Oct }	2 Oct	2 Oct
Meeting3	Ser (0.4) Ash (0.6)	{1 Oct, 2 Oct }	1 Oct	1 Oct
Meeting4	Ser (1.0)	{3 Oct }	3 Oct	3 Oct
Small agent violation			2.0	1.0

Table 3: Case 2 data table

1) SuperagentLGP outcomes:

   a) The superagentLGP Generated heuristic is

{

Rank-Meeting

Schedule

Rank-Meeting

Schedule

Rank-Meeting

Schedule

Schedule

}

b) The total number of rounds=1

c) The violation after using the new heuristic =0.0

2) SuperagentLSP outcomes:

a) The superagentLSP Generated heuristic is

{

Schedule

Rank-Meeting

Schedule

Rank-Meeting

Schedule

Schedule

}

b) The total number of rounds =1

c) The violations after using the new heuristic =0.0

Results Table		
Meetings	**Attendees(rank)**	**Domain**
Meeting1	Ser (1.0)	**{1 Oct, 2 Oct, 3 Oct }**
Meeting2	Ser (0.2) Ash (0.8)	**{2 Oct, 4 Oct }**
Meeting3	Ser (0.4) Ash (0.6)	**{1 Oct, 2 Oct }**
Meeting4	Ser (1.0)	**{3 Oct }**
Small agent violation is 0.0		

Table 4: Case 2 results table

Case 3:

Number of Users=3: Serein (Ser), Ashraf (Ash), Omar (Omr)

Number of meetings=5

Data Table				
Meetings	**Attendees(ran**	**Domain**	**Optimisation**	**Search**

	k)		framework	framework
Meeting1	Ser (0.5) Ash (0.5)	{3 Oct }	3 Oct	3 Oct
Meeting2	Ser (0.8) Omr (0.2)	{2 Oct, 5 Oct }	2 Oct	2 Oct
Meeting3	Ser (0.1) Ash (0.2) Omr(0.7)	{2 Oct, 5 Oct }	5 Oct	5 Oct
Meeting4	Ser (0.1) Omr (0.9)	{2 Oct, 4 Oct, 5 Oct }	5 Oct	violated
Meeting5	Ser (0.7) Ash (0.3)	{4 Oct, 8 Oct }	4 Oct	4 Oct
Small agent violation			0.8	1

Table 5: Case 3 data table

1) SuperagentLGP outcomes:

   a) The superagentLGP Generated heuristic is

      {

      Rank-Meeting

      Schedule

      Rank-Meetings

      Schedule

      Rank-Meetings

      Schedule

      Schedule

Schedule

}

b) The total number of rounds =1

c) The violation after using the new heuristic =-0.0

2) SuperagentLSP outcomes:

a) The superagentLSP Generated heuristic is

{

Schedule

Schedule

Schedule

Rank-Meetings

Schedule

Schedule

}

b) The total number of rounds =1

c) The violation after using the new heuristic =0.0

Result Table		
Meetings	**Attendees(rank)**	**domain**
Meeting1	Ser (0.5) Ash (0.5)	{ 3 Oct }
Meeting2	Ser (0.8) Omr (0.2)	{ 2 Oct, 5 Oct }
Meeting3	Ser (0.1) Ash (0.2) Omr(0.7)	{2 Oct, 5 Oct }
Meeting4	Ser (0.1) Omr (0.9)	{2 Oct, 4 Oct, 5 Oct }
Meeting5	Ser (0.7) Ash (0.3)	{4 Oct, 8 Oct }
Small agent violation is 0.0		

Table 6: Case 3 results table

Case 4:

Number of Users=4: Serein (Ser), Ashraf (Ash), Omar (Omr), Abed (Abd)

Number of meetings=7

Data Table				
Meetings	**Attendees(rank)**	**Domain**	**Optimisation framework**	**Search framework**
Meeting1	Ser (0.1) Ash (0.5) Omr (0.3) Abd (0.1)	{1 Oct, 2 Oct , 5 Oct, 6 Oct }	5 Oct	5 Oct
Meeting2	Ser (0.8) Ash (0.2)	{1 Oct, 2 Oct, 6 Oct, 8 Oct }	8 Oct	8 Oct
Meeting3	Ser (0.5) Omr (0.5)	{3 Oct, 7 Oct }	7 Oct	7 Oct
Meeting4	Ser (1.0)	{1 Oct, 2 Oct, 3 Oct }	3 Oct	violated
Meeting5	Ser (0.5) Abd (0.5)	{1 Oct, 2 Oct }	1 Oct	1 Oct
Meeting6	Ser (0.3) Ash (0.2) Abd (0.5)	{3 Oct }	3 Oct	3 Oct
Meeting7	Ser (0.3) Ash (0.7)	{2 Oct, 4 Oct }	2 Oct	2 Oct
Small agent violation			1.3	1

Table 7: Case 4 data table

1) superagentLGP outcomes:
    a) The superagentLGP Generated heuristic is

        {

        Rank-Meetings

        Schedule

        Rank-Meetings

        Schedule

        Schedule

        Schedule

        Schedule

        Schedule

        Schedule

        }

    b) The total number of round=1
    c) The violation after using the new heuristic =0.0
2) superagentLSP outcomes:
    a) The superagentLSP Generated heuristic is

        {

        Schedule

        Rank-Meetings

        Schedule

        Rank-Meetings

Schedule

Schedule

Schedule

Schedule

Schedule

}

b) The total number of rounds =1

c) The violation after using the new heuristic =0.0.

Results Table

Meetings	**Attendees(rank)**	Domain
Meeting1	Ser (0.1) Ash (0.5) Omr (0.3) Abd (0.1)	{1 Oct, 2 Oct ,5 Oct, 6 Oct }
Meeting2	Ser (0.8) Ash (0.2)	{1 Oct, 2 Oct, 6 Oct, 8 Oct }
Meeting3	Ser (0.5) Omr (0.5)	{3 Oct, 7 Oct }
Meeting4	Ser (1.0)	{1 Oct, 2 Oct, 3 Oct }
Meeting5	Ser (0.5) Abd (0.5)	{1 Oct, 2 Oct }

Meeting6	Ser (0.3) Ash (0.2) Abd (0.5)	{3 Oct }
Meeting7	Ser (0.3) Ash (0.7)	{2 Oct, 4 Oct }
Small agent violation is **0.0**		

Table 8: Case 4 results table

Case 5:

Number of Users=5: Serein (Ser), Ashraf (Ash), Omar (Omr), Abed (Abd), Jon (Jon)

Number of meetings=8

Data Table				
Meetings	**Attendees(rank)**	**domain**	**Optimisation framework**	**Search framework**
Meeting1	Ser (0.2) Ash (0.2) Omr (0.2) Abd (0.2) Jon (0.2)	{1 Oct,2 Oct,3 Oct, 5 Oct,6 Oct,7 Oct}	3 Oct	3 Oct
Meeting2	Ser (0.1) Ash (0.9)	{1 Oct, 2 Oct,5 Oct, 7 Oct}	5 Oct	5 Oct

Meeting3	Ser (0.4) Ash (0.5) Omr (0.1)	{2 Oct, 6 Oct, 8 Oct}	6 Oct	6 Oct
Meeting4	Ash (1.0)	{3 Oct,7 Oct, 8 Oct}	3 Oct	3 Oct
Meeting5	Ser (0.5) Jon (0.5)	{1 Oct,2 Oct, 3 Oct}	2 Oct	violated
Meeting6	Ser (0.4) Ash (0.4) Omr (0.1) Abd (0.1)	{1 Oct, 2 Oct}	1 Oct	1 Oct
Meeting7	Ser (0.5) Ash (0.5)	{3 Oct,5 Oct, 6 Oct, 7 Oct}	7 Oct	7 Oct
Meeting8	Ash (1.0)	{2 Oct, 4 Oct}	2 Oct	2 Oct
Small agent violation			1.2	1

Table 9: Case 5 data table

1) SuperagentLGP outcomes:

   a) The superagentLGP Generated heuristic is

      {

      Rank-Meetings

      Schedule

      Rank-Meetings

      Schedule

Schedule

Schedule

Schedule

Schedule

Schedule

Schedule

}

b) The total number of rounds =0

c) The violation after using the new heuristic is =0.0

2) SuperagentLGP outcomes:

a) The superagentLSP Generated heuristic is

{

Schedule

Rank-Meetings

Schedule

Rank-Meetings

Schedule

Schedule

Schedule

Schedule

Schedule

Schedule

}

b) The total number of rounds =1

c) The violation after using the new heuristic is =0.0

Results Table

Meetings	**Attendees(rank)**	**Domain**
Meeting1	Ser (0.2) Ash (0.2) Omr (0.2) Abd (0.2) Jon (0.2)	{1 Oct,2 Oct,3 Oct,5 Oct, 6 Oct,7 Oct}
Meeting2	Ser (0.1) Ash (0.9)	{1 Oct,2 Oct, 5 Oct,7 Oct}
Meeting3	Ser (0.4) Ash (0.5) Omr (0.1)	{2 Oct,6 Oct, 8 Oct}
Meeting4	Ash (1.0)	{3 Oct, 7 Oct,8 Oct}
Meeting5	Ser (0.5) Jon (0.5)	{1 Oct,2 Oct,3 Oct}
Meeting6	Ser (0.4) Ash (0.4) Omr (0.1) Abd (0.1)	{1 Oct, 2 Oct}
Meeting7	Ser (0.5) Ash (0.5)	{3 Oct,5 Oct,6 Oct,7 Oct}

Meeting8	Ash (1.0)	{2 Oct, 4 Oct}
Small agent violation is 0.0		

Table 10: Case 5 results table

Analysis for Experiment Group 1

From these experiments, the following observations can be made:

1) It is feasible for the proposed framework to run any number of users without affecting its performance. HMAA has been run on up to ten users, and it could quite conceivably run hundreds. In this document, only five experiments are demonstrated since it is unfeasible to show all experiments.

2) It is feasible for SUAs to solve wide verity of situations. Both types of SUA (superagentLGP and superagentLSP) can automatically generate algorithms using LGP or LSP to overcome the failures in the situations previously outlined. These situations have been carefully selected because they are difficult for SMAs to solve.

3) Both types of SUA give the same components with different combinations. All the produced algorithms have the same components (i.e. the number of "Rank-Meetings" functions combined with the number of "Schedule" functions), but they have combined and arranged these two components in different forms.

Experiments Group 2

This group of experiments consists of more complicated cases and susceptible situations, having one user (Serein (Ser)) and different numbers of meetings with different domains and constraints. These experiments are run in order to see if one or all of the SUAs can reach a solution and what is *the feasibility of the system in very complicated and limited domains and constraints*.

Moreover, *the local optima* property has been added to the SUA algorithms. This means that, instead of continuing until the solution is found or the specified number of rounds is reached, searching ends when the neighbourhood's violations(for LSP) or the new generations' violations (for LGP) are greater than the solution’s/parent’s violation/s.

This could be achieved by defining another two SUAs (superagentLGP_SP, superagentLSP_SP), which exit searching when they reach the local optimum (i.e. where the neighbourhood's violations (superagentLSP_SP)/the next generations' violations (superagentLGP_SP) are greater than the current solution’s (superagentLSP_SP)/parents' (superagentLGP_SP) violations, in addition to the existing defined SUAs (superagentLGP, superagentLSP) which continue until they reach a solution or loop for 500 rounds. This is due to the fact that the feasibility of any system measured by: the ability to solve the problem, and the time needed to do; this time on our system measured by the number of rounds the algorithms loops to generate the solution heuristic. Sometimes local optima solution in an acceptable time is more feasible than global optima solution in a long time.

This group of experiments was conducted using both frameworks (optimisation and search frameworks). However, the results from the experiments that were performed using the optimisation framework is presented only. This is because the results were similar in both frameworks as shown in previous section.

As will be seen below, the double-lined data tables are those showing the meetings, attendees with their ranks, and domains. The underlined dates are those chosen by the default SMA heuristic; the red rectangles around specific dates are conflicting/overlapping ones. The last four columns of the table are the results of the previously mentioned four agents (SuperagentLGP, SuperagentLSP, SuperagentLGP_SP and SuperagentLSP_SP). The last row in the table represents the violations of the corresponding columns.

The following tables are the results tables for the four SUAs. They show the starting and ending violations and the number of rounds for each SUA algorithm. This information is presented in order to compare the performance of these SUAs.

Case 6:

In this case, sixteen meetings have to be scheduled; their domain is thirty days and %43.34 of this domain is constrained.

The number of the Meetings=16;

The number of constrained time slots=30 * %43.34 = 13 time slots constrained.

The number of available time slots =30 - 13 = 17 time slots available.

Hence ***sixteen meetings*** can be distributed among ***seventeen timeslots***. The results were as follows:

Data Table						
Meetings	**Attendees**	**Domain with SMA solution**	**Superagent LGP**	**Superagent LSP**	**Superagent LGP_SP**	**Superagent LSP_SP**
Meeting1	Ser (1)	**{2 Oct}**	**2 Oct**	**2 Oct**	**2 Oct**	**2 Oct**
Meeting2	Ser (1)	**{1 Oct, 4 Oct}**	**1 Oct**	**1 Oct**	**1 Oct**	**1 Oct**
Meeting3	Ser (1)	**{1 Oct, 4 Oct}**	**4 Oct**	**4 Oct**	**4 Oct**	**4 Oct**
Meeting4	Ser (1)	**{1 Oct,3 Oct, 4 Oct}**	**3 Oct**	**3 Oct**	**3 Oct**	**3 Oct**
Meeting5	Ser (1)	**{3 Oct,5 Oct}**	**5 Oct**	**5 Oct**	**5 Oct**	**5 Oct**
Meeting6	Ser (1)	**{6 Oct,7 Oct,8 Oct}**	**6 Oct**	**6 Oct**	**6 Oct**	**6 Oct**
Meeting7	Ser (1)	**{7 Oct,9 Oct}**	**9 Oct**	**9 Oct**	**9 Oct**	**7 Oct**
Meeting8	Ser (1)	**{6 Oct,7 Oct}**	**7 Oct**	**7 Oct**	**7 Oct**	**7 Oct**

Meeting9	Ser (1)	{8 Oct}	**8 Oct**	**8 Oct**	**8 Oct**	**8 Oct**
Meeting10	Ser (1)	**{10 Oct,11 Oct, 14 Oct, 15 Oct}**	**14 Oct**	**14 Oct**	**14 Oct**	**14 Oct**
Meeting11	Ser (1)	**{10 Oct,11 Oct, 15 Oct, 17 Oct}**	**15 Oct**	**15 Oct**	**15 Oct**	**15 Oct**
Meeting12	Ser (1)	**{12 Oct,16 Oct}**	**16 Oct**	**16 Oct**	**16 Oct**	**16 Oct**
Meeting13	Ser (1)	**{10 Oct,11 Oct, 12 Oct}**	**10 Oct**	**10 Oct**	**10 Oct**	**10 Oct**
Meeting14	Ser (1)	**{10 Oct,11 Oct}**	**11 Oct**	**11 Oct**	**11 Oct**	**11 Oct**
Meeting15	Ser (1)	{12 Oct}	**12 Oct**	**12 Oct**	**12 Oct**	**12 Oct**
Meeting16	Ser (1)	**{11 Oct,13 Oct}**	**13 Oct**	**13 Oct**	**13 Oct**	**13 Oct**
Violation =		**6**	**0**	**0**	**0**	**2**

Table 11: Case 6 data table

Case 7:

In this case, eighteen meetings have to be scheduled; their domain is thirty days and %33.34 of this domain is constrained.

The number of the Meetings=18 meetings;

The number of constrained time slots in the domain= 30 * %33.34=10 time slots constrained

The number of available time slots =30-10 = 20 time slots available.

Hence eighteen ***meetings*** can be distributed across ***twenty dates***. The results were as follows:

Data Table						
Meetings	**Attendees**	**Domain with SMA violation**	**Superagent LGP**	**Superagent LSP**	**Superagent LGP_SP**	**Superagent LSP_SP**
Meeting1	Ser (1)	**{2 Oct}**	**2 Oct**	**2 Oct**	**2 Oct**	**2 Oct**
Meeting2	Ser (1)	**{1 Oct, 4 Oct}**	**1 Oct**	**1 Oct**	**1 Oct**	**1 Oct**
Meeting3	Ser (1)	**{1 Oct, 4 Oct}**	**4 Oct**	**4 Oct**	**4 Oct**	**4 Oct**
Meeting4	Ser (1)	**{1 Oct,3 Oct, 4 Oct}**	**3 Oct**	**3 Oct**	**3 Oct**	**4 Oct**
Meeting5	Ser (1)	**{3 Oct,5 Oct}**	**5 Oct**	**5 Oct**	**5 Oct**	**3 Oct**
Meeting6	Ser (1)	**{6 Oct,7 Oct,10 Oct,11 Oct}**	**10 Oct**	**10 Oct**	**10 Oct**	**10 Oct**
Meeting7	Ser (1)	**{6 Oct, 7 Oct,11 Oct,**	**11 Oct**	**11 Oct**	**11 Oct**	**11 Oct**

		13 Oct}				
Meeting8	Ser (1)	**{8 Oct,12 Oct}**	**12 Oct**	**12 Oct**	**12 Oct**	**12 Oct**
Meeting9	Ser (1)	**{6 Oct, 7 Oct, 8 Oct}**	**6 Oct**	**6 Oct**	**6 Oct**	**6 Oct**
Meeting10	Ser (1)	**{6 Oct,7 Oct}**	**7 Oct**	**7 Oct**	**7 Oct**	**7 Oct**
Meeting11	Ser (1)	**{8 Oct}**	**8 Oct**	**8 Oct**	**8 Oct**	**8 Oct**
Meeting12	Ser (1)	**{7 Oct,9 Oct}**	**9 Oct**	**9 Oct**	**9 Oct**	**9 Oct**
Meeting13	Ser (1)	**{14 Oct,15 Oct, 18 Oct}**	**15 Oct**	**15 Oct**	**15 Oct**	**15 Oct**
Meeting14	Ser (1)	**{14 Oct,15 Oct,18 Oct, 19 Oct}**	**18 Oct**	**18 Oct**	**18 Oct**	**18 Oct**
Meeting15	Ser (1)	**{15 Oct, 19 Oct, 20 Oct}**	**19 Oct**	**19 Oct**	**19 Oct**	**19 Oct**
Meeting16	Ser (1)	**{16 Oct,19 Oct, 20 Oct}**	**20 Oct**	**20 Oct**	**16 Oct**	**20 Oct**
Meeting17	Ser (1)	**{ 14 Oct, 15 Oct, 16 Oct}**	**16 Oct**	**16 Oct**	**16 Oct**	**16 Oct**
Meeting18	Ser (1)	**{14 Oct,15 Oct}**	**14 Oct**	**14 Oct**	**14 Oct**	**14 Oct**
Violation =		**6**	**0**	**0**	**2**	**0**

Table 12: Case 7 data table

Cases 8 and 9:

In these cases, sixteen meetings have to be scheduled; their domain is thirty days and %46.67 of this domain is constrained.

The number of the Meetings=16 meetings;

The number of constrained time slots in the domain= 30 * %46.67=14 time slots

The number of available time slots =30-14 = 16 time slots available.

Hence, *sixteen meetings* can be distributed between *sixteen time slots*. The date and the performances for both SUAs are shown separately in the two following tables.

Data Table						
Meetings	**Attendees**	**Domain with SMA solution**	**Superagenet LGP**	**Superagent LSP**	**Superagent LGP_SP**	**Superagent LSP_SP**
Meeting1	Ser (1)	**{2 Oct}**	**2 Oct**	**2 Oct**	**2 Oct**	**2 Oct**
Meeting2	Ser (1)	**{1 Oct, 4 Oct}**	**1 Oct**	**1 Oct**	**1 Oct**	**1 Oct**
Meeting3	Ser (1)	**{1 Oct, 4 Oct}**	**4 Oct**	**4 Oct**	**4 Oct**	**4 Oct**
Meeting4	Ser (1)	**{1 Oct,3 Oct, 4 Oct}**	**4 Oct**	**4 Oct**	**4 Oct**	**3 Oct**
Meeting5	Ser (1)	**{3 Oct,5 Oct}**	**3 Oct**	**3 Oct**	**3 Oct**	**5 Oct**

Meeting6	Ser (1)	**{6 Oct,7 Oct,8 Oct}**	**6 Oct**	**6 Oct**	**6 Oct**	**6 Oct**
Meeting7	Ser (1)	**{7 Oct,9 Oct}**	**9 Oct**	**9 Oct**	**9 Oct**	**7 Oct**
Meeting8	Ser (1)	**{6 Oct,7 Oct}**	**7 Oct**	**7 Oct**	**7 Oct**	**7 Oct**
Meeting9	Ser (1)	**{8 Oct}**	**8 Oct**	**8 Oct**	**8 Oct**	**8 Oct**
Meeting10	Ser (1)	**{10 Oct,11 Oct, 14 Oct, 15 Oct}**	**14 Oct**	**14 Oct**	**14 Oct**	**14 Oct**
Meeting11	Ser (1)	**{10 Oct,11 Oct, 15 Oct, 16 Oct}**	**15 Oct**	**15 Oct**	**15 Oct**	**15 Oct**
Meeting12	Ser (1)	**{12 Oct,16 Oct}**	**16 Oct**	**16 Oct**	**16 Oct**	**16 Oct**
Meeting13	Ser (1)	**{10 Oct,11 Oct, 12 Oct}**	**10 Oct**	**10 Oct**	**10 Oct**	**10 Oct**
Meeting14	Ser (1)	**{10 Oct,11 Oct}**	**11 Oct**	**11 Oct**	**11 Oct**	**11 Oct**
Meeting15	Ser (1)	**{12 Oct}**	**12 Oct**	**12 Oct**	**12 Oct**	**12 Oct**
Meeting16	Ser (1)	**{11 Oct,13 Oct}**	**13 Oct**	**13 Oct**	**13 Oct**	**13 Oct**
Violation=		6	**2**	**2**	**2**	**2**

Table 13: Case 8 data table

Data Table						
Meetings	**Attendees**	**Domain with SMA solution**	**Superagent] LGP**	**Superagent LSP**	**superagentL GP_SP**	**superagentL SP_SP**
Meeting1	Ser (1)	**{2 Oct}**	**2 Oct**	**2 Oct**	**2 Oct**	**2 Oct**
Meeting2	Ser (1)	**{1 Oct, 4 Oct}**	**1 Oct**	**1 Oct**	**1 Oct**	**1 Oct**
Meeting3	Ser (1)	**{1 Oct, 4 Oct}**	**4 Oct**	**4 Oct**	**4 Oct**	**4 Oct**
Meeting4	Ser (1)	**{1 Oct,3 Oct, 4 Oct}**	**3 Oct**	**3 Oct**	**3 Oct**	**4 Oct**
Meeting5	Ser (1)	**{3 Oct,5 Oct}**	**5 Oct**	**5 Oct**	**5 Oct**	**3 Oct**
Meeting6	Ser (1)	**{6 Oct,7 Oct,8 Oct}**	**6 Oct**	**6 Oct**	**6 Oct**	**8 Oct**
Meeting7	Ser (1)	**{7 Oct,9 Oct}**	**9 Oct**	**7 Oct**	**9 Oct**	**7 Oct**
Meeting8	Ser (1)	**{6 Oct,7 Oct}**	**7 Oct**	**7 Oct**	**7 Oct**	**6 Oct**
Meeting9	Ser (1)	**{8 Oct}**	**8 Oct**	**8 Oct**	**8 Oct**	**8 Oct**
Meeting10	Ser (1)	**{10 Oct,11 Oct, 12 Oct, 14 Oct}**	**12 Oct**	**12 Oct**	**12 Oct**	**12 Oct**
Meeting11	Ser (1)	**{10 Oct,11 Oct, 14 Oct,**	**14 Oct**	**15 Oct**	**14 Oct**	**14 Oct**

		<u>15 Oct</u>}				
Meeting12	Ser (1)	**{<u>11 Oct</u>,15 Oct, 16 Oct}**	**11 Oct**	**11 Oct**	**11 Oct**	**11 Oct**
Meeting13	Ser (1)	**{<u>12 Oct</u>,15 Oct, 16 Oct}**	**16 Oct**	**16 Oct**	**16 Oct**	**16 Oct**
Meeting14	Ser (1)	**{<u>10 Oct</u>,11 Oct, 12 Oct}**	**10 Oct**	**10 Oct**	**10 Oct**	**10 Oct**
Meeting15	Ser (1)	**{10 Oct,11 Oct, <u>15 Oct</u>}**	**15 Oct**	**15 Oct**	**15 Oct**	**15 Oct**
Meeting16	Ser (1)	**{11 Oct,<u>13 Oct</u>, 15 Oct**	**13 Oct**	**13 Oct**	**13 Oct**	**13 Oct**
Violation		**6**	**0**	**0**	**0**	**4**

Table 14: Case 9 data table

	superagentLGP	superagentLSP	superagentLGP_SP local optimum	superagentLSP_SP local optimum
Starting violation	6	6	6	6
Ending violation	0	0	0	2
Number of rounds	5	13	5	2

Table 15: Case 6 results table

	superagentLGP	superagentLSP	superagentLGP_SP local optimum	superagentLSP local optimum
Starting violation	6	6	6	6
Ending violation	0	0	2	0
Number of rounds	11	2	7	2

Table 16: Case 7 results table

	superagentLGP	superagentLSP	superagentLGP_SP local optimum	superagentLSP_SP local optimum
Starting violation	6	6	6	6
Ending violation	2	2	2	2
Number of rounds	500	500	7	2

Table 17: Case 8 results table

	superagentLGP	superagentLSP	superagentLGP_SP local optimum	superagentLSP_SP local optimum
Starting	6	6	6	6

violation				
Ending violation	0	0	0	4
Number of rounds	5	10	5	3

Table 18: Case 9 results table

	Small agent	superagentLGP	superagentLSP	superagentLGP_SP local optimum	superagentLSP_SP local optimum
Violation percentage	36%	3%	3%	6%	12%
Reduction in violation	0%	33%	33%	30%	24%

Table 19: Results table

Analysis of Experiment Group 2

From the experiments above, the following observations can be made:

1) HMAA in this group has been examined on very complicated situations; where the meetings' domains are very constrained. We have documented four cases of these situations:

a) Case 6: we have (16) meetings and (30) domain timeslots; (%43.34) of this domain is unavailable (constrained); hence (%56.66) of this domain is available. On the other hand the percentage of the domain needed to schedule these 16 meetings is (16/30=%53.34). Then the meetings would be scheduled in %53.34 of the domain where %56.66 is available.

b) Case 7: (18) meetings and the domain is (30) timeslots where %33.34 is unavailable. These (16) meetings need (%53.33) of the domain to be scheduled, where just %66.66 is available.

c) Cases 8 and 9: there are (16) meetings; and (30) timeslots domain where (%46.67) of this domain is unavailable. These 16 meetings need (%53.33) of the domain to be scheduled where exactly (%53.33) of this domain is unconstrained or available.

The results of this group of experiments show that the percentage of the scheduled meetings' violation using SMA default heuristic is (%36), and this percentage is considered small since these cases are very restricted and constrained situations. On the other hand, this violation has been reduced more in the cases of using the help of one of the SUAs. Using superagentLGP, superagentLSP, superagentLGP_SP, superagentLSP_SP generated heuristics have reduced the violation up to 3%, 3%, 6%, 12% respectively. Hence it is perfectly feasible for the four SUAs in HMAA to solve very susceptible and complicated situations; they reduced the violations by a huge margin of the SMA's violation (33 per cent, 33 per cent, 30 per cent, and 24 per cent respectively). And this level of reduction is a great achievement.

2) The feasibility of the system is measured by the number of violations as well as the time taken or the number of rounds needed. Sometimes local optimum is more feasible than global optima, as in Tables 15, 17 and 18. In Table 17 the superagentLSP_SP and superagentLGP_SP reached the same violation of superagentLSP and superagentLGP –which equals 2- in a very small number of rounds “2”; while superagentLSP and superagentLGP needed “500” rounds for that. In Tables 15 and 18, superagentLSP_SP reached to violations (2, 4) in small number of rounds (2, 3); while superagentLSP reached to “zero” violations but in number of rounds equals (13 and 10) respectively. The same for superagentLGP_SP, in Table 16; it reached violation equals 2 in “7” rounds, while superagentLGP reached “zero” violations in 11 rounds. Hence, in some cases in is more feasible to reach acceptable violation in acceptable time; than “zero” violation in very long time.

Experiments Group 3

In these experiments, three agents are initialised and thirty meetings with different domain distributed between them. Some of these domains overlap.

The experiments have been carried out in three stages:

- Stage 1 — the SMA stage. The SMA attempts to resolve problems by using its heuristic to scheduling meetings with a minimum of violations.
- Stage 2 — the superagentLGP. The SMA asks the superagentLGP to generate new heuristics to solve the violations. The superagentLGP uses its LGP algorithm to generate new heuristics for the SMAs.

- Stage 3 — superagentLSP. The SMA asks the superagentLSP to generate new heuristics to solve the violation. The superagentLSP uses its LSP algorithm to generate a new heuristic for the SMAs.

These experiments can measure the feasibility of the system by comparing the results of Experimental Stage 1 with Stage 2 and Stage 3. As it can be seen in Fig. 46, the SUAs in the implemented HMAA are responsible for a big reduction in the number of violations. The violation of the SMAs using its heuristic was 94, which was reduced to between 18 and 20 using one of the SUA-generated heuristics. This means that the new HMAA can reduce violations by nearly 75 per cent.

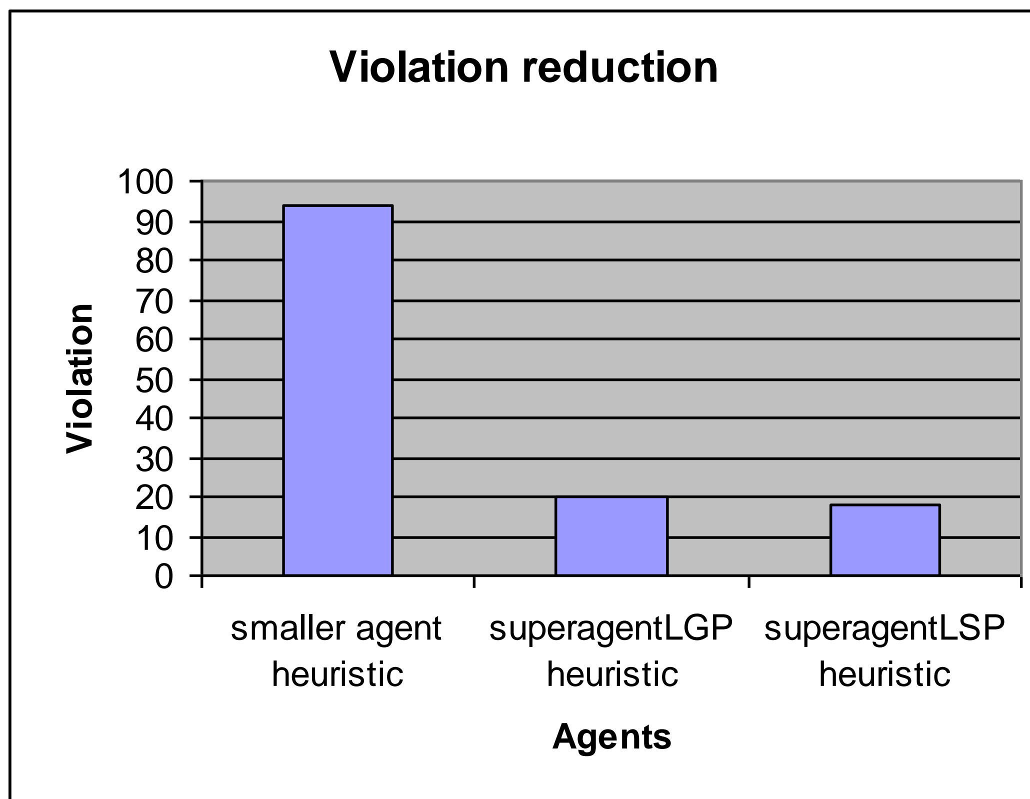


Fig. 46: Violation reduction

By comparing results of Stage 2 with those of Stage 3, the feasibility of the LSP can be measured as a proposed evolutionary approach and compared with the LGP-based one of existed evolutionary approaches. The superagentLGP produces algorithms that produced violations totalling 20, while the superagentLSP equivalent was 18; this reveals that superagentLSP (using LSP) is better that superagentLGP (which uses LGP). This does not mean that LSP is better than linear search programming, however.

Fig. 47 shows that the superagentLSP loops 60 rounds while superagentLGP loops 54: LSP loops 10 per cent more rounds. This is the reason why superagentLSP's violations are fewer in number than those of superagentLGP'. This is due to the proposed algorithms for both SUAs: since they have the same components (as mentioned in the analysis of Experiment Groups 1 and 2), so the proposed algorithms for superagentLSP loop more than those for superagentLGP. But if they have the same components, then LSP and LGP are both good enough to solve many complicated and sensitive situations, and both SUA algorithms could be modified and altered to generate better solutions.

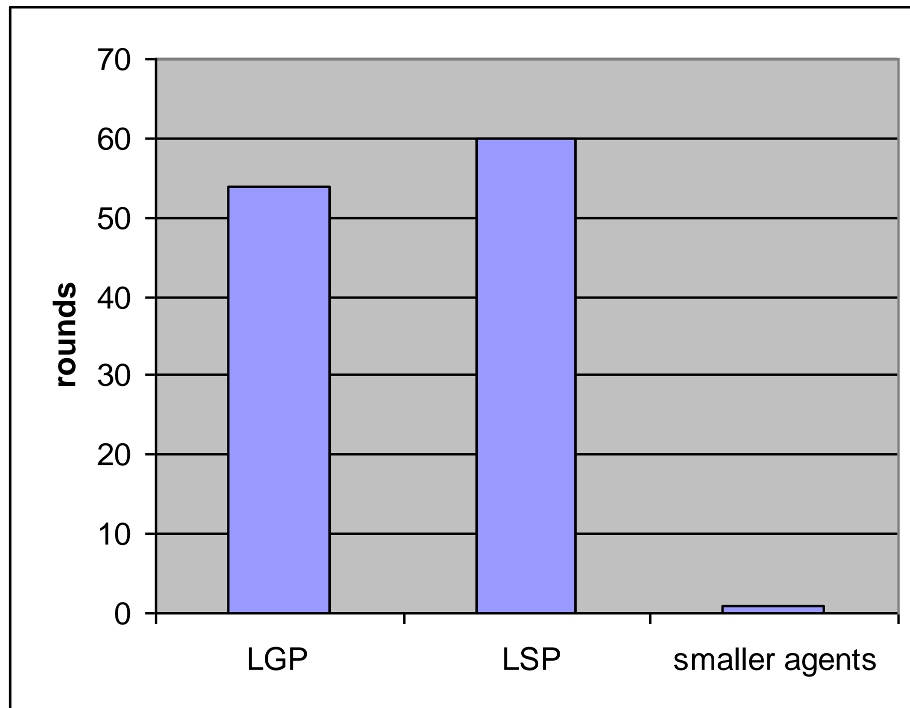


Fig. 47: Measurement of rounds/time

In Fig. 47; LSP rounds are more than LGP that does not mean LSP needs longer time frame than LGP. This is due to the fact that in each round LSP generates two heuristics and tries them to measure their feasibility; while LGP generates and tries four heuristics in each round; which means each round in LSP needs half time frame needed for LGP. On the other hand the probability of finding a solution in four options of heuristics is more than in two heuristics which is the case of LSP; that does not means LGP finds solution faster, since it depends mainly on the strength of the heuristic itself.

8.6. Summary

The chapter illustrates experiments for HMAA, where two stages of experiments have been conducted. Stage one is done on small agent predefined heuristic; where two groups of experiments have been done.

Stage two conducted three groups of experiments; experiments group 1 demonstrated the feasibility of HMAA on large number of users; and wide variety of situations. Experiments group 2 conducted to measure the feasibility of the system for acceptable level of violations and time frames; where some times searching local optima solution is more feasible than computing the global one. The final group is for measuring the feasibility of LSP compared with LGP as a new concept to EAs research; and the result that LSP performance was very good and equals to LGP approach.

Further modifications and more expanding to HMAA are discussed in the following chapter.

Chapter 9

Conclusion and Future Work

9.1 Summary

Recently AI researchers tend to move away from machine-orientation views of programming toward concepts and metaphors that more closely reflect human beings orientations; this evolved trend to *Agent* research area where high-level more human-oriented abstraction software are developed. In order to simplify the development of such software, it has been discovered after several years that distributed systems are easier to understand and build; specially that real-life problems are usually physically or functionally distributed, and sometimes the problems are too large to be solved by a centralised agent; hence further development in AI research toward distribution has led *Multiagent systems (MAS)*.

MASs are loosely-coupled network of problem solvers that work together to solve problems that are beyond their individual capabilities. Despite the big advantages raised from distributing the problem solving process, it cannot be denied that distributed frameworks have several difficulties; neither up to date information nor the complete range of resources are available to all agents. Consequently information and computation is localised; diversified goals also present significant challenges to the design of systems capable of achieving high levels of global utility where independent decisions may result in conflicts.

In this research we have proposed a new hybrid multi-agent architecture (HMAA), the architecture is "*semi-distributed/semi-centralised*" where problems are firstly distributed among small agents that implement predefined heuristics to solve problems. However, heuristics are not absolutely guaranteed to provide the best (i.e. optimal) solutions, or even to find a solution at all. Therefore optimisation techniques such as EAs are needed to solve such a situation.

HMAA has a centralised control; a central agent possessing much greater computational power and more intensive algorithms and optimisation techniques; becomes active when those small agents become stuck, and generates new heuristic that enable those small agents to solve their problems. The Meeting Scheduling Problem (MSP) has been adopted and investigated in order to examine and validate the idea.

The results reveal that this architecture is applicable to many different application domains because of its simplicity and efficiency. Its performance is better than those of many existing meeting scheduling frameworks. It preserves small agents' mobility (i.e. the ability to run on small devices) while implementing evolutionary algorithms. HMAA is very robust in that it can implement more than one optimisation technique without affecting mobility.

9.2 Contributions

This research has produced the following results:

(1) Hybrid Multi-Agent Architecture (HMAA) for solving many NP-hard problems: this architecture is *hybrid* because it is a "semi-distributed/semi-centralised" architecture. In the proposed HMAA, variables and constraints are distributed among small agents

exactly as in distributed architectures. But when the small agents become stuck, a centralised control is activated, in which the variables are transferred to a super agent that has a central view of the whole system and possesses much more computational power and intensive algorithms (such as evolutionary algorithms) that enable it to find an optimal solution . It does this by defining different classes of agents including super-agents and small agents. Fixed and limited heuristics of small agents can be altered by the super-agents; that generate new skills/heuristics using evolutionary approaches. This architecture is used for examining the feasibility of running computationally intensive algorithms (such as EA) on multi-agent architectures, while preserving the small agent size and their ability to run on small devices.

(2) Two types of HMAA have been implemented: the first deals with the NP-problems as optimisation problem DCOP, in which the goal is to find an optimal solution that minimises the violation; the other deals with NP-hard problems as search problem DisCSP, where the goal is to find a feasible solution. The implementation of these two types of HMAA is due to the fact that some applications' domains need to implement the optimal solution, with the minimum violations, leaving the final decision about the results obtained to the user. Other applications require feasible solutions without any violations, leaving the users the opportunity to enter the unsolved problem into the system again, using new options (domains) in order to retry the attempt at a solution.

(3) New SMA meeting scheduling heuristic: this prioritised/ranked heuristic takes into account two parameters: a set of domains and a set of ranked attendees. These parameters are necessary in order to measure the difficulty of a meetings' scheduling. The heuristic starts by ranking the meetings, in order to schedule the most difficult ones

respectively. This gives meetings that do not have many feasible options in their time domains the opportunity to be scheduled before the others. The aim of the prioritised/ranked heuristic is to find timeslots from each domain to assign the corresponding meeting to, where all the attendees accept that assignment. The goal is to find a feasible or optimal solution depending on the chosen framework.

(4) New SMA repair strategy for the scheduling process: this is activated when the scheduling ends with some violations. The repair strategy implements a local search algorithm to search for a better schedule. It starts with the meeting with the greatest number of violations, and defines its neighbourhood structure using simple moves involving only two-meeting timeslots. Each move involves the meeting with the most violations and one of the meetings overlapping with it, so that each has in its domain the timeslot to which the other meeting has been assigned. It swaps these two meetings in order to find a better schedule; the process is repeated for all the meetings with violations until the optimal solution with minimal violations. This repair strategy applies just to the DCOP framework. In the DisCSP framework the agents do not have to deal with violations of meetings, but only with unscheduled meetings.

(5) Small Agent (SMA) for meeting scheduling has been developed. This SMA is responsible for accomplishing the scheduling process on behalf the individual it represents; by utilising the proposed prioritised/ranked meeting scheduling heuristic and local search repair strategy. This SMA is small size and limited capabilities agent, and it could be implemented and run on small devices such as phone mobile device or PDAs. Hence, the users could manage meetings and know their calendar through small mobile devices that implement this proposed SMA.

(6) "Local Search Programming" (LSP), a new concept for evolutionary approaches, has been introduced. This is a method for automatically creating a working computer program that modifies one solution technique (heuristic/computer program) using local search approaches. This method is inspired by genetic programming and by local search techniques. LSP tries to generate new heuristics/programs using local searching instead of genetic algorithm (GA) techniques. Local searching optimises the current heuristic by moving from one heuristic/algorithm to one of its neighbours. The neighbourhood is composed of those heuristics that can be obtained by simple local changes to the current heuristic. Trial programs were evaluated against their parent/origin and the best one selected in order to continue searching for the optimal solution. This search pattern is repeated until the optimal program is produced. The main difference between local search algorithms and local search programming is that the former optimises the *solution* while the latter optimises *computer programs as the solution*. The difference between genetic programming and local search programming is that GP applies *GAs to a population of programs*, while LSP applies *LSA to one program*, meaning that *GP needs two parents* to crossover them, while *LSP works on one solution in order to modify it*.

(7) Two types of super-agent (LGP_SUA and LSP_SUA) have been implemented in the HMAA, and two SUAs (local and global optima) have been implemented for each type. The first type is LGP_SUA (superagentLGP and superagentLGP_SP), it uses the LGP approach to generate new heuristics. The second is LSP_SUA (superagentLSP and superagentLSP_SP), it uses the LSP approach for the same purpose. In HMAA, the SUAs task starts after it receives a request from the SMA for a new heuristic. The SUAs' design incorporates powerful functions that generate new meeting scheduling

solutions, while applying one of the aforementioned EAs. Evolutionary strategies promise to find optimal solutions that minimise the number of violations. This is not always possible for relatively unsophisticated small/mobile agents, because they have fixed heuristics with limited capabilities, and may therefore sometimes fail to find the optimal solution.

(8) A prototype for HMAA has been implemented: this prototype employs the proposed meeting scheduling heuristic with the repair strategy on SMAs, and the four extensive algorithms on SUAs. This is in order to evaluate the SMA heuristic and the local search repair strategy, as well as to examine the feasibility of running the investigated computationally intensive algorithms on multi-agent architectures while preserving the small agents' size and ability to run on small devices. This examination is carried out by evaluating the performance of the SUAs.

9.3. Future work

In future work the following will be investigated:

- Generalising the HMAA and extending it to include many different distributed, large-scale, open, heterogeneous, dynamic applications, as well as many NP-hard problems. An exciting example of such problems is Multi-agent Resource Allocation (MARA), which merges computer science and economics. MARA is defined as the allocation of resources within a system of autonomous agents, which not only have preferences over alternative allocations of resources but also participate in computing an allocation. The objective of a resource allocation procedure is either to find an allocation that is feasible (e.g. search

problem); or to find an allocation that is optimal (e.g. optimisation problem). Another example is Task Allocation Problem (TAP), where the agents are connected in a social network and tasks arrive at the agents distributed over the network.

- Extending local search algorithm to cater for a larger neighbourhood structures that does not need great computational power. Extending the local search algorithm is supposed to be worth, since it would improve the performance of the SMA while preserving the ability to run on small devices.

- Implementing another types of evolutionary algorithms on super agents; which search to find the optimal heuristic or optimal solution. many discipline are grouped under evolutionary algorithms; these disciplines are: evolutionary strategy(Rechenberg 1964), evolutionary programming (Fogel, Owens and Walsh 1965), genetic algorithm (Holland 1975) and genetic programming (Koza 1992) the one that has been adopted to investigate and validate the proposed HMAA. Implementing other types of EAs may improve the performance of the proposed HMAA.

- Formalising heuristics that are used in solving NP-hard problem, and using them in evolutionary algorithm to generate new heuristics. This is due to the fact that if heuristic can be defined as sequence of blocks, it is easily for the EAs to generate new heuristics by simply perform mutation on these blocks.

References

[1] A. Omicini, F. Zambonelli, and M. Klusch. *Coordination of Internet Agents: Models Technology and Applications.* Chapter 13, R. Tolksdorf, Springer, 2001.

[2] A. Taivalsaari, B. Bush, and D. Simon. The Spotless System: Implementing a Java System for the Palm Connected Organizer. *Tech. report SMLI TR-99-77*, Sun Microsystems, Palo Alto, Calif., 1999.

[3] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer, Natural Computing, ISBN 3540401849, 2008.

[4] A. Abraham. *Evolutionary Computation, Handbook for Measurement, Systems Design.* John Wiley and Sons Ltd., London, ISBN 0-470-02143-8, pp. 920–931, 2005.

[5] A. Petcu and B. Faltings. A Distributed, Complete Method for Multi-agent Constraint Optimisation. In *the proceedings of the fifth international workshop on Distributed Constraint Reasoning (DCR04)*, Toronto, Canada, 2004.

[6] A. Hassine, T. Bao Ho. An Agent-based Approach to Solve Dynamic Meeting Scheduling Problems with Preferences. *Engineering Applications of Artificial Intelligence,* vol. 20 , issue 6, pp. 857-873, 2007.

[7] A. Hassine, T. B. Ho, and T. Ito. Meetings Scheduling Solver Enhancement with Local Consistency Reinforcement. *Applied Intelligence*, vol. 24, issue 2, pp. 143–154, 2006.

[8] A. Abraham, N. Nedjah, and L. Mourelle. *Evolutionary Computation: from Genetic Algorithms to Genetic Programming*. Springer, Germany, ISBN: 3-540-29849-5, 2006.

[9] A. Meisels. *Distributed Search by Constrained Agents Algorithms, Performance, Communication*. Springer, ISBN 1848000405, p19-26, 2008.

[10] T. Back. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, 1996.

[11] M. Brameier and W. Banzhaf. A Comparison of Linear Genetic Programming and Neural Networks in Medical Data Mining. *IEEE transactions on Evolutionary Computation,* pp. 17-26, 2001.

[12] B. Montague. JN: An Operating System for an Embedded Java Network Computer. *Technical Report UCSC-CRL-96-29*, 1997.

[13] C. Guilfoyle and E. Warner. *Intelligent Agents: the New Revolution in Software*. Ovum, 1994.

[14] C. Bernon, V. Chevrier, and V. Hilaire. Applications of Self-Organising Multi-Agent Systems: An Initial Framework for Comparison. *Informatica*, vol. 30, pp. 73–82, 2006.

[15] C. Voudouris, D. Lesaint, and D. Pothos. Solving Large Industrial Problems using Heuristic Search and Constraint Programming. *Intelligent System Research*, BT, 1998.

[16] C. Paolo and M. Wooldridge. Agent-oriented Software Engineering. In *the proceedings of the International Conference on Software Engineering,* pp. 816-817, 2000.

[17] P. I. Cowling, S. Ahmadi, P. C. Cheng and R. Barone. Combining Human and Machine Intelligence to Produce Effective Examination Timetables. *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning (SEAL2002)*, pp. 662-666, 2002.

[18] D. Beasley, D. Bull, and R. Martin. An Overview of Genetic Algorithms: Part 1, Fundamentals. *Technical report*, University of Purdue, volume 15(2), pp. 69-58, 1993.

[19] D. Goldberg. *Genetic Algorithms in Search, Optimisation and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1989.

[20] D. Partridge. *A new Guide to Artificial Intelligence*. Ablex Pub. Corp. (Norwood, N.J), 1991.

[21] L. M. Deschaine. Tackling Real-world Environmental Challenges with Linear Genetic Programming. *PCAI Magazine*, vol. 15, pp. 35-37, 2000.

[22] L. M. Deschaine, R. A. Hoover, and J. Skibinski. Using Machine Learning to Complement and Extend the Accuracy of UXO Discrimination Beyond the Best Reported Results at the Jefferson Proving Grounds. In *the proceedings of the Society for Modeling and Simulation International*, 2002.

[23] L. M. Deschaine, J. J. Patel, R. G. Guthrie, J. T. Grumski, and M. J. Ades. Using Linear Genetic Programming to Develop a C/C++ Simulation Model of a Waste Incinerator. *The Society for Modeling and Simulation International: Advanced Simulation Technology Conference*, pages 41-48, 2001.

[24] D. Li and Y. Du. *Artificial Intelligence with Uncertainty*. Chapman & Hall/CRC, 2008.

[25] M. Dianati, I. Song, and M. Treiber. An Introduction to Genetic Algorithms and Evolution Strategies. *Technical report*, University of Waterloo, Ontario, N2L 3G1, Canada, 2002.

[26] D. Kramer. The Java™ Platform: A White Paper. *JavaSoft White Paper Sun Microsystems*, California USA, 1996.

[27] E. S. K. Yu. Agent-Oriented Modelling: Software versus the World. In *the proceedings of AOSE'01*, LNCS 2222, pp. 206-225. Springer-Verlag, 2001.

[28] E. Mezura-montes, A. Carlos, and C. Coello. An Improved Diversity Mechanism for Solving Constrained Optimisation Problems using a Multimembered Evolution Strategy. In *the proceedings of GECCO*, 2004.

[29] E. Anderson. Playing Smart – Artificial Intelligence in Computer Games. In *the proceedings of zfxCON03 Conference on Game Development, ZFX - 3D Entertainment*, Braunschweig, Germany, 2003.

[30] E. Shakshuki, H. Koo, D. Benoit, and D. Silver. A Distributed Multi-agent Meeting Scheduler. *Journal of Computer and System Sciences archive,* vol. 74, pp. 279-296, 2008.

[31] E. Evans and D. Rogers. Using Java Applets and CORBA for Multi-user Distributed Applications. *IEEE Internet Computing*, vol. 1, pp 43-55, 1997.

[32] F. Streichert. Introduction to Evolutionary Algorithms. *Frankfurt MathFinance Workshop*, University of Tuebingen, 2002.

[33] F. Zhang, P. B. Luh and E. Santos Jr. Performance Study of Multi- Agent Scheduling and Coordination Framework for Maintenance Networks. In *the proceedings of 2004 IEEEIRSI International Conference on intelligent Robots and Systems*, Sendai, Japan, 2004.

[34] J. Ferber. *Multi-agent systems: an introduction to distributed artificial intelligence*. Addison-Wesley, London, 1999.

[35] L. Garrido and K. Sycara. Multi-agent Meeting Scheduling: Preliminary Experimental Results. In *the proceedings of 1st International Conference on Multi-Agent Systems (ICMAS).* Pp. 95 – 102, 1996.

[36] G. Weiss. *Multiagent Systems: a Modern Approach to Distributed Artificial Intelligence*. The MIT Press, 1999.

[37] A. Hassine, T. Ito, and T. B. Ho. A new Distributed Approach to Solve Meeting Scheduling Problems. In *the proceedings of IEEE/WIC Int. Conf. IAT*, 2003.

[38] A. Hassine, X. Defago, and T. B. Ho. Agent-based Approach to Dynamic Meeting Scheduling Problems. *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, vol. 3, 2004.

[39] J. Holland. *Adaptation in Natural and Artificial Systems*. Ann Harbor: University of Michigan Press, 1975.

[40] H. Nwana and D. Nduma. A Perspective on Software Agents Research. *The Knowledge Engineering Review*, 1999.

[41] I. Devarenne, H. Mabed, and A. Caminada. Intelligent Neighbourhood Exploration in Local Search Heuristics. In *the proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence*, pp. 144-150 , 2006.

[42] I. Demirel and N. Erdogan. Meeting Scheduling with Multi-agent Systems: Design and Implementation. In *the proceedings of the 6th WSEAS International Conference on Software Engineering*, pp. 92-97, 2007.

[43] R. Jain, F. Anjum, and A. Umar. A Comparison of Mobile Agent and Client-Server Paradigms for Information Retrieval Tasks in Virtual Enterprises. In *the proceedings of AiWoRC Workshop,* Buffalo, New York, 2000.

[44] J. Orlin, A. Punnen, and A. Schulz. Approximate Local Search in Combinatorial Optimisation. *SIAM Journal on Computing*, Vol. 33, pp. 1201-1214, 2004.

[45] N. R. Jennings and M. Wooldridge. Applications of Intelligent Agents. *Agent Technology: Foundations, Applications, and Markets.* Springer-Verlag, Heidelberg, Germany, 1998.

[46] N. R. Jennings, K. Sycara, and M. Wooldridge. A Roadmap of Agent Research and Development. *Vivek*, vol. 12, no. 3-4, pp. 38-66, 1999.

[47] J. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. The MIT Press, ISBN: 026258116, 1992.

[48] J. McCarthy. John McCarthy: father of AI. *Intelligent Systems, IEEE*, Volume 17, Issue 5, pp. 84 – 85, 2002.

[49] J. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection*. The MIT press, Cambridge, MA, 1992.

[50] J. Leung. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Chapman & Hall; 1 edition, Mathematics, pp. 2-9, 2004.

[51] K. Andersen and J. Debenham . Database and Expert Systems Applications. In *the proceedings of the 16th international conference on Database and Expert Systems Applications (DEXA)*, Denmark, p. 339, 2005.

[52] R. Kohout and K. Erol. Achieving High Quality Solutions in Distributed Agent-Based Control Systems. In *the proceedings of the IASTED International Conference on Artificial Intelligence and Soft Computing,* August 9-12, Honolulu, Hawaii, 1999.

[53] D. B. Lange and M. Oshima. Seven Good Reasons for Mobile Agents. *Communications of the ACM*, vol. 42, no. 3, pp. 88-8, 1999.

[54] Y. C. Law and J. H. M. Lee. Algebraic Properties of CSP Model Operators. In *the proceedings of the International Workshop on Reformulating Constraint Satisfaction Problems: Towards Systematisation and Automation*, 2002.

[55] M. Sugumaran, K. Easawarakumar, and P. Narayanasamy. An Effective Approach for Distributed Meeting Scheduler. *International Journal of Information Technology* , Vol. 12, No. 8, 2006.

[56] M. Heywood and A. Zincir-Heywood. Dynamic Page Based Crossover in Linear Genetic Programming. *IEEE transactions on Systems, Man, and Cybernetics*, Part B, Vol. 32, pp. 380-388, 2002.

[57] M. Huhns and L. Stephens. Multiagent Systems and Societies of Agents. *In Multiagent systems: a modern approach to distributed artificial intelligence*, G. Weiss (Ed.), MIT Press, pp. 79-120, 1999.

[58] M. Gervasio, M. Moffitt, M. Pollack, J. Taylor, and T. Uribe. Active Preference Learning for Personalised Calendar Scheduling Assistance. In *the proceedings of the International Conference on Intelligent User Interfaces,* San Diego, CA, Jan 2005.

[59] M. Wooldridge. Agent-based Software Engineering. *IEE Proc. on Software Engineering*, 37-26 (1) 144, 1997.

[60] M. Wooldridge and N. Jennings. Software Engineering with Agents: Pitfalls and Pratfalls. *IEEE Internet Computing*, Vol. 3, pp. 20-27, 1999.

[61] M. Wooldridge and P.Ciancarini. Agent-Oriented Software Engineering: The State of the Art. *Agent-Oriented Software Engineering*, Springer-Verlag, 2001.

[62] R. Mailler and V. Lesser. Solving Distributed Constraint Optimisation Problems using Cooperative Mediation. In *the proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems AAMAS*, pp. 438–445, 2004.

[63] M. Yokoo, E. Durfee, and K. Kuwabara. The Distributed Constraint Satisfaction Problem: Formalisation and Algorithms. *IEEE Transactions on knowledge and data engineering,* vol. 10, no. 5, 1998.

[64] M. Brameier and W. Banzhaf. *Linear Genetic Programming (Genetic and Evolutionary Computation)*. Springer, 2006.

[65] A. Meisels and O. Lavee. Using Additional Information in Discsps Search. *Distributed Constraint Reasoning Workshop (DCR)*, 2004.

[66] M. O'Niell. *Grammatical Evolution, Evolutionary Automatic Programming in an Arbitrary Language*. Springer, ISBN 1402074441, 2003.

[67] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer; 3rd ed., p. 1 and p. 26, 2008.

[68] M. Wooldridge. *An Introduction to MultiAgent Systems.* John Wiley & Sons, ISBN 047149691X, p. 23, 2002.

[69] M. Oltean. Encoding Multiple Solutions in a Linear Genetic Programming Chromosome. In *the proceedings of the 4th International Conference on Computational Science*, Part III, Springer-Verlag, 2004.

[70] P. Modi, W. Shen, M. Tambe, and M. Yokoo. An Asynchronous Complete Method for Distributed Constraint Optimisation. In *the proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems AAMAS*, pp. 161–168, 2003.

[71] D. Mulchandani. Java for Embedded Systems. *IEEE Internet Computing*, Vol. 2, pp. 30-39, 1998.

[72] N. Jennings and M. Wooldridge. Applications of Intelligent Agents. *Agent Technology: Foundations, Applications, and Markets*, 1998.

[73] Ovum Report. *Intelligent agents: the new revolution in software*. London: Ovum Publications, 1994.

[74] P. Jackson. *Introduction to Expert Systems*. Addison-Wesley: Reading, MA, 1986.

[75] P. Tullmann, M. Hibler, and J. Lepreau. Janos: A Java-oriented OS for Active Networks. *IEEE Journal on Selected Areas of Communications*, March 2001.

[76] Adrian. Dynamic Distributed Optimisation for Planning and Scheduling. *AAAI Workshop* - Technical Report, pp. 52-53, v WS-05-06, 2005.

[77] P. Dobble. *Real-time java platform programming*. Prentice Hall PTR, 1st edition, Page 13, 2000.

[78] P. Modi and M. Veloso. Multiagent Meeting Scheduling with Rescheduling. In *the proceedings of the fifth Workshop on Distributed Constraint Reasoning* (DCR), 2004.

[79] P. Modi, M. Veloso, S. Smith, and J. Oh. CMRADAR: A Personal Assistant Agent for Calendar Management. *Agent Oriented Information Systems, (AOIS)*, 2004.

[80] Profit Scheduler for MeetingsTM.
http://www.profitpt.com/profit_meeting_scheduler.asp, accessed on Jun 2009.

[81] R. Gregory. *The Oxford Companion to the Mind*. Oxford University Press, Oxford UK, 1998.

[82] R. Poli, W. Langdon, and N. McPhee. *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd, ISBN 1409200736, March 2008.

[83] R. Gray. Agent Tcl: A Flexible and Secure Mobile-agent System. In *the proceedings of the fourth Annual Tcl/Tk Workshop,* 1996.

[84] R. Barták and H. Rudová . Integrated Modelling for Planning, scheduling, and Timetabling Problems. In *the proceedings of PLANSIG*, Edinburgh, UK, December 2001.

[85] S. Russell and P. Norvig. *Artifical intelligence: a modern approach*. Prentice Hall, 2nd edition, 2003.

[86] S. Ahmadi, R. Barone, E. Burke, P. Cheng, P. Cowling and B. McCollum. Integrating Human Abilities and Automated Systems for Timetabling: a Competition using Stark and Hussh Representations at the PATAT 2002 Conference. In *the proceedings of the 4th international conference on the practice and theory of automated timetabling (PATAT),* pp. 265-273, 2002.

[87] S. Sivanandam, S. Deepa. *Introduction to Genetic Algorithms.* Springer; 1 edition, ISBN 354073189X, 2007.

[88] S. Al-Ratrout and S. Ahmadi. Learning the Effect of Parameters in Timetabling Process. In *the proceedings of the international conference International Conference on Recent Advances in Soft Computing (RASC)*, 2006.

[89] S. Legg and M. Hutter. Universal Intelligence: A Definition of Machine Intelligence. *Minds and Machines*. Vol. 17, pp. 391-444, 2007.

[90] Y. Shoham. Agent-oriented Programming. *Artificial Intelligence*, 60(1):51-92, 1993.

[91] V. Silva, A. Garcia, A. Brandao, C. Chavez, C. Lucena, and P. Alencar. Taming Agents and Objects in Software Engineering. *Software engineering for large-scale multi-agent systems, Research issues and practical applications*, pp. 1-26, 2003.

[92] Snap Schedule Employee Scheduling Software, http://www.bmscentral.com/products/schedule/overview.aspx, accessed on 21 June 2009.

[93] K. P. Sycara. Multiagent Systems. *AI Magazine*, vol. 19, no. 2, pp. 79-92, 1998.

[94] TimeBridge, http://www.timebridge.com/home.php, accessed on 21 June 2009.

[95] H. Tomoyuki, M. Mitsunori, and S. Hisashi. Optimisation Problem Solving System using Grid RPC. *In the Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGr]\id 2003),* Japan, 2003.

[96] R. J. Wallace and E. C. Freuder. Constraint-based Reasoning and Privacy/Efficiency Tradeoffs in Multi-agent Problem Solving. *Artificial Intelligence*, vol. 161, no. 1-2, Distributed Constraint Satisfaction, pp. 209-227, January, 2005.

[97] W. Zhang. Modelling and Solving a Resource Allocation Problem with Soft Constraint Techniques. *Technical Report: WUCS*, 2002.

[98] W. Langdon and W. Banzhaf. Repeated Sequences in Linear Genetic Programming Genomes. *Complex Systems publications*, 15 (2005) 285–306; Inc. 2005.

[99] W. Banzhaf, P. Nordin, R. Keller, and F. Francone. *Genetic Programming An Introduction on the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann Publishers, 1st edition, ISBN 155860510X, 1998.

Appendix A

An HMAA Screen Shots

After running the FMAF, the first user interface (Fig. 48) asks the user to choose the type (formalisation) of the scheduling framework; either *search heuristic* framework, where the FMAF searches for scheduling all un-violated meetings, leaving the violated meetings without scheduling. The other framework is *optimisation heuristic* framework where FMAF search for optimal scheduling with the minimum violation.

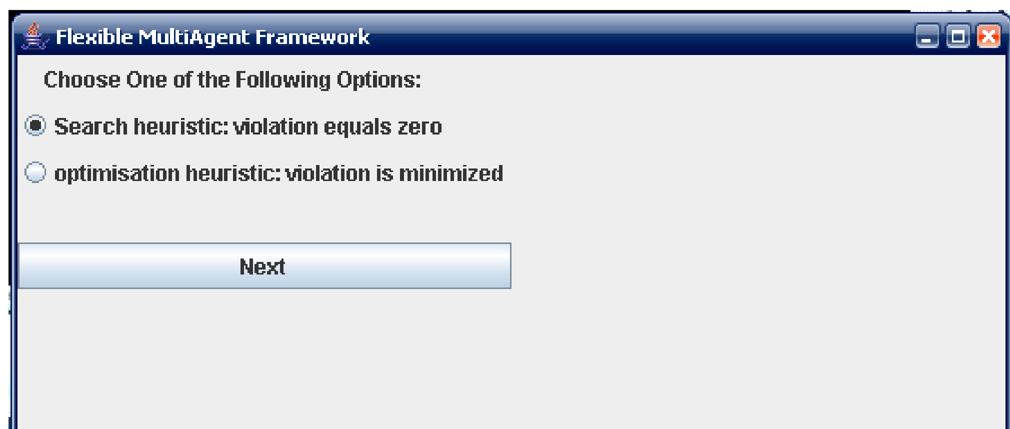


Fig. 48: First User Interface for FMAF

After choosing the framework type, a list of registered users will be displayed (fig. 49), and easily the initiator user selects the attendees with whom he wants to arrange the meetings. Accordingly their agents will be activated to participate in this scheduling problem solving.

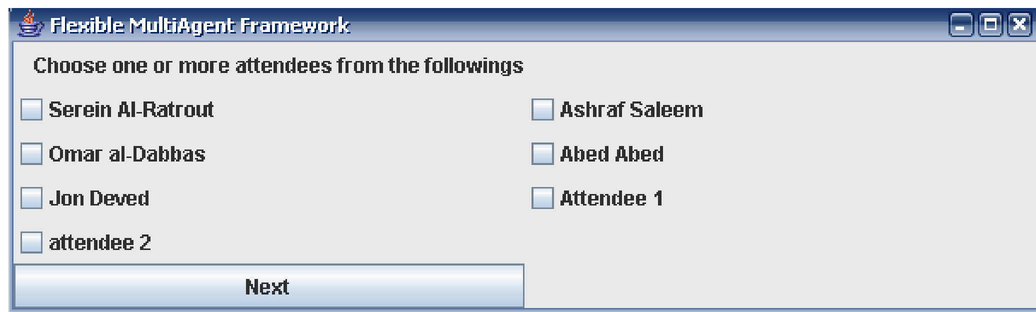


Fig. 49: a list of registered users

The next interface (Fig. 50) asks the user to choose one or more of the SUAs, in order to help the SMA if it fails to find un-violated scheduling; by generating new SMA's heuristic.

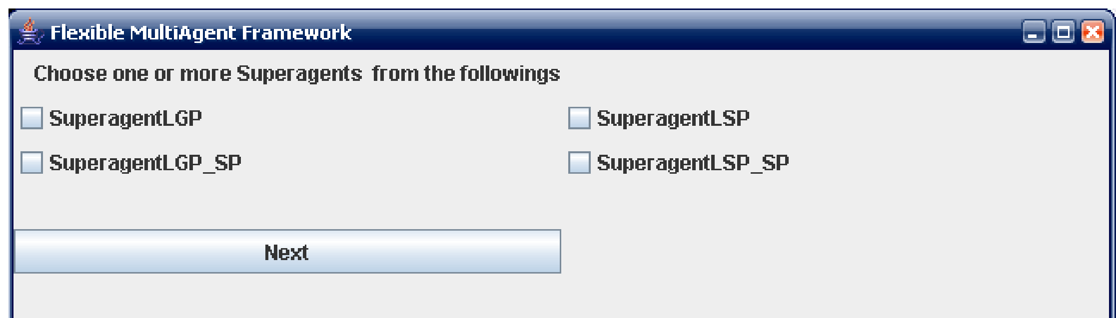


Fig. 50: the available SUAs.

When users are logged on, they see the following interface (Fig. 51):

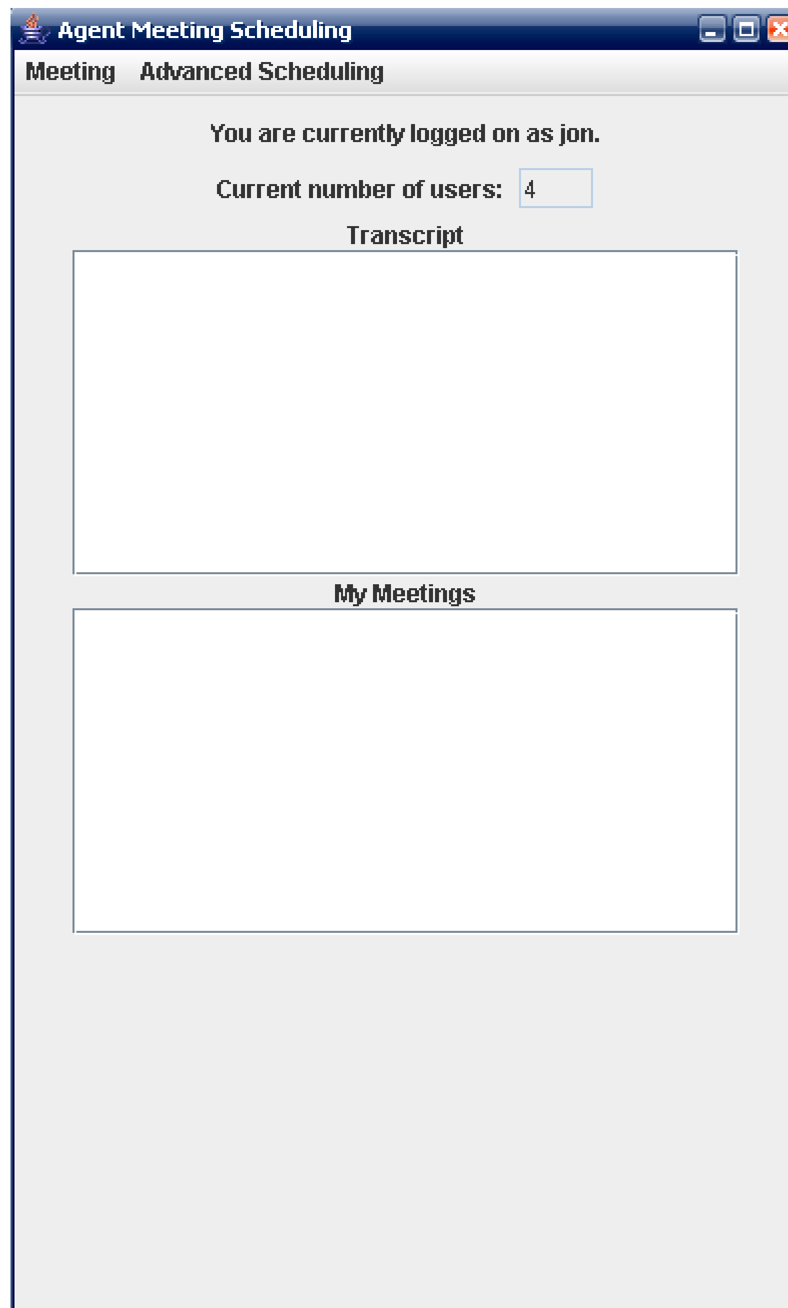


Fig. 51: SMA Main User Interface

This interface shows

- two menus: **Meeting** and **Advanced Scheduling**
- the user name for the corresponding user (e.g. Tom)
- how many users are logged on

- the **transcriptions** text book (the messages exchanged while arranging meetings)
- My **meeting** text book, where the user can see the final schedule for all the meetings.

When users open the "**Meeting**" menu, they see the followings menu items (Fig. 52):

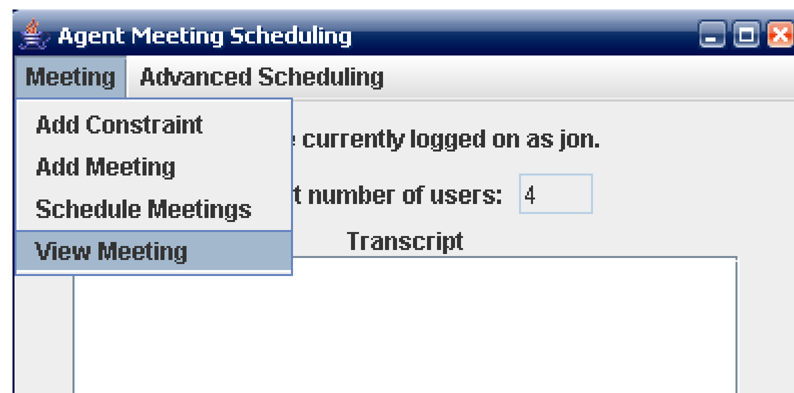


Fig. 52: The meeting menu

- *Add Constraint:* in order to add unavailable dates
- *Add a meeting*: in order to enter the meeting's corresponding data
- *Schedule meetings*: to perform the scheduling process
- *View meeting*: to see the corresponding schedule

When the "Add Constraint" menu item is pressed a new interface appears containing (Fig. 53):

- Calendar: to choose unavailable dates
- Finish Constraint: to save constrained/unavailable dates
- Stop constraint: this button stops the listener property in order to navigate between months or years without registering any dates.
- Start constraint: starts the listener property after it has been stopped; listener property is a calendar property used in order to record unavailable dates.

Fig. 53: Add Constraint Interface

When the user presses the “Add Meeting” menu item, a new interface appears containing the menu “Meeting properties” (Fig. 54).

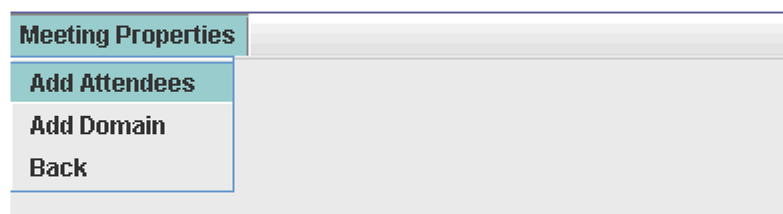


Fig. 54: Add Meeting Interface

This interface allows the user to enter the related data about the **attendees** and **time domains** for the corresponding meeting (Fig. 54), in order to enable the agent to perform the required calculations for the meeting scheduling heuristic.

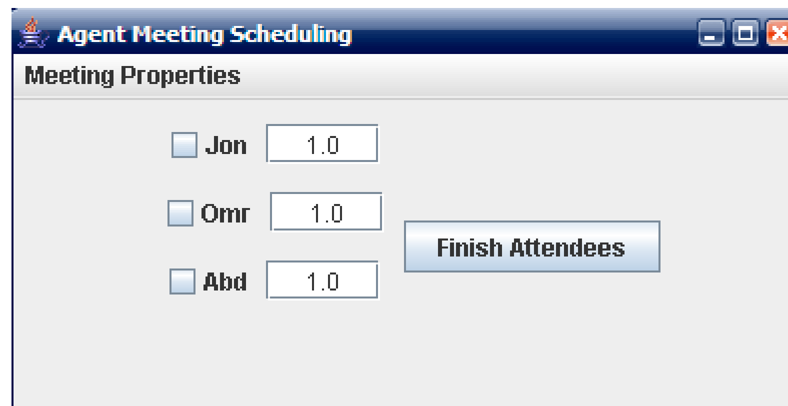


Fig. 55: Add Attendees Interface

By pressing “Add Attendees”, all the logged users will be presented on the screen (Fig. 55), so the user can easily rank the corresponding attendees (how important it is for each attendee to be present at the meeting), and then check the check box for the related users. The user finally presses the “Finish Attendees” button to save this information in the system.

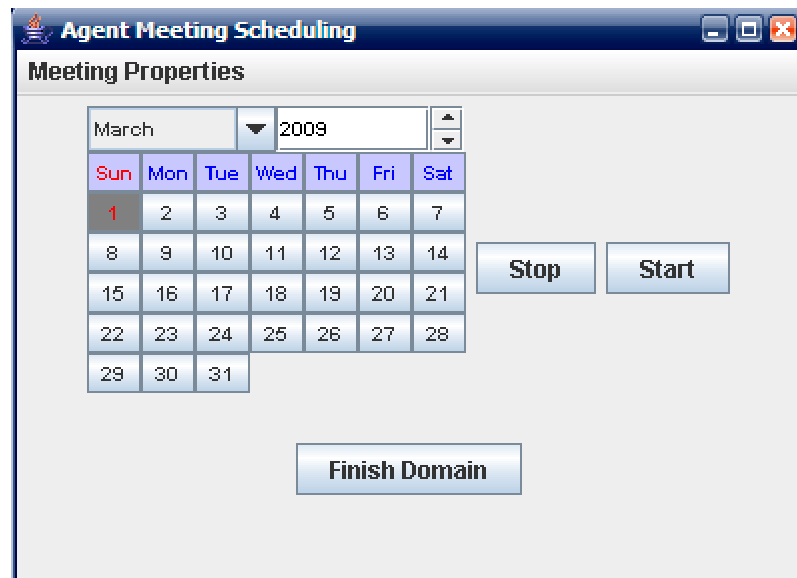


Fig. 56: Add Domain Interface

.

When the menu item "Add Domain" is pressed a calendar is presented (Fig. 56) on which the user can easily choose the possible dates for the meeting, and then press "Finish Domain" to save the domain to the corresponding meeting. Stop and Start buttons stop the listener to enable navigation between months and years, and start it again when the user is ready to enter the unavailable dates.

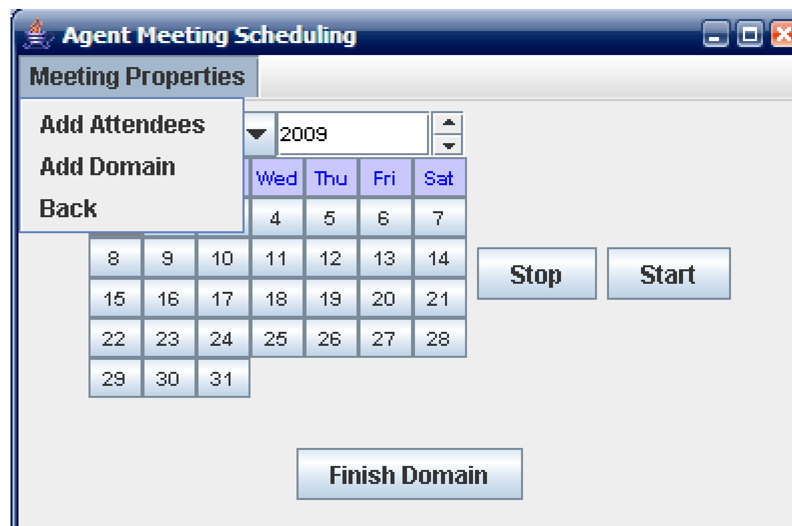


Fig. 57: Add Meeting Menu Item

When all the meeting properties have been set, the “Back” menu item from the “Meeting Properties” menu is chosen (Fig. 57). By doing this, the meeting is created and the corresponding data entered. This meeting now awaits the scheduling request, after which the user will be returned to the main user interface (Fig. 51).

If any data is missing such as if the user pressed "Back" without entering the domain or the attendees or both, the meeting will not be saved and a message will be displayed on the transcript text box. The user then has to repeat all the steps (Fig. 58).

By choosing “Schedule Meeting” (Fig. 52), the agent starts the scheduling process. Each message sent or received by the agent will be displayed on the transcript text box (Fig. 58):

The standard form for any message on the transcript text box is the following; where the underlined bold words are variables:

Sender name sends **a *type of message*** for a meeting: a ***meeting name time slot***

- *Sender name:* one of the agents responsible for scheduling the corresponding meeting
- *Type of message:* proposal, confirmation
- *Meeting name:* set by the system
- *Time slot:* date from the domain.

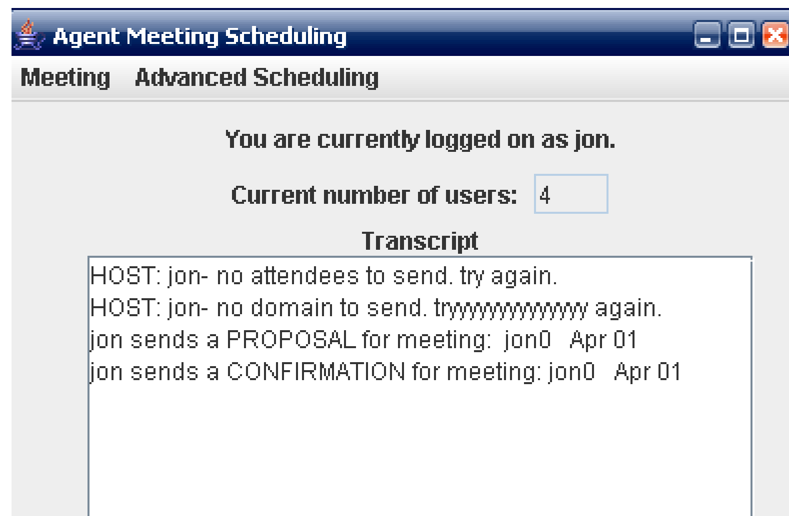


Fig. 58: Transcript Text Box

By choosing “View Meetings” (Fig. 52) from the “Meeting” menu, the schedule of meetings will be displayed in the My Meeting text box (Fig. 59).

My Meetings

MEETING NAME ~~~~	ASSIGNMENT ~~~~	VIOLATION
jon0	Apr 01	1.0
jon1	Apr 01	0.9
		~~
	total of the violations=	1.9

Fig. 59: View Meetings Text Box

As can be seen in Fig. 59, the system sometimes schedules meetings while there are violations (conflicts). The “Advanced Scheduling” menu contains the “Local Search Repair” item that executes the repair strategy in order to find better schedules (Fig. 60).

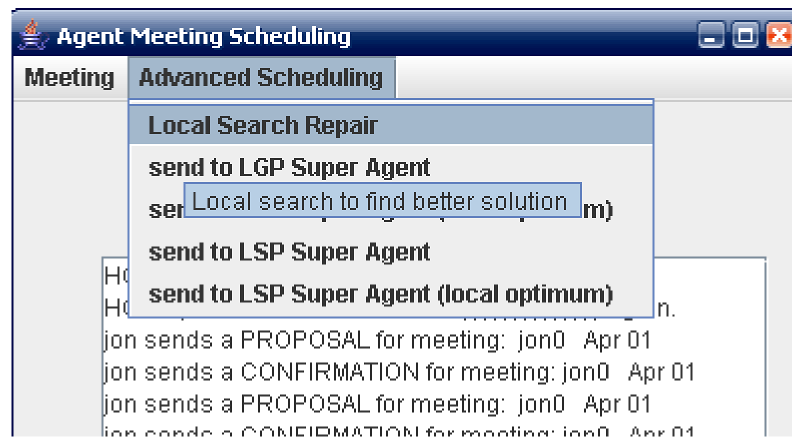


Fig. 60: Local Search Menu Item

After choosing the users and SUAs, all of which are activated in order to participate in scheduling problem solving.

When the user choose "View meetings" from "Meeting" menu item; the scheduled meetings with their total violation will be displayed in "My Meetings" text area.

If the violation is more than "0", then the user can ask one or more of the activated SUAs to generate new heuristic, by selecting the corresponding SUA's interface and clicking on "Generate" menu item (Fig.61, 62, 63 and 64).

Fig. 61: SuperagentLGP Interface

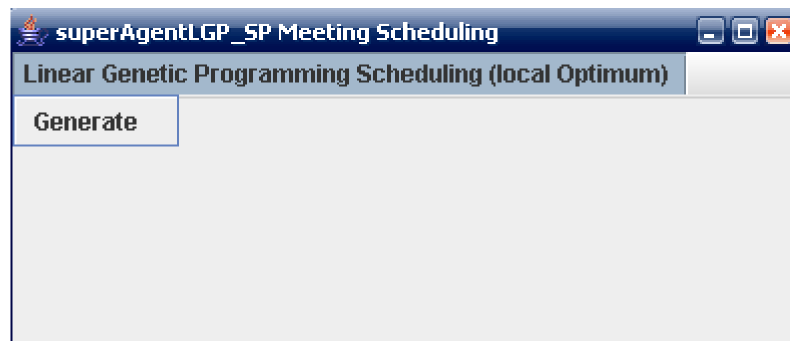


Fig. 62: SuperagentLGP_SP Interface

Fig. 63: SuperagentLSP Inteface

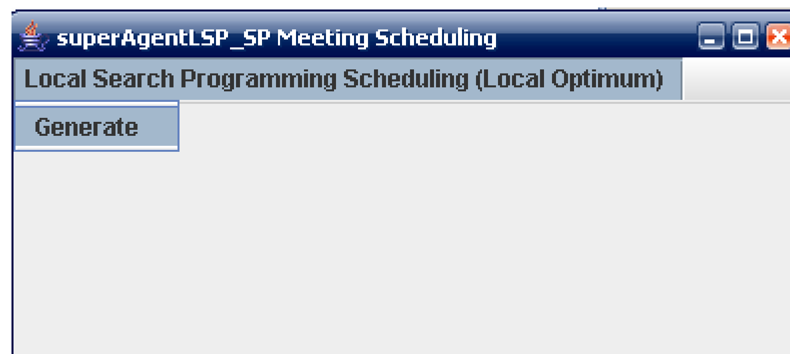


Fig. 64: SuperagentLSP_SP Inteface

Searching for new better heuristic will be starting and the best one will be send to the corresponding SMA to use.

Appendix B

Java Code of HMAA implementation

```
public class Client0 extends UnicastRemoteObject implements MessageClient0
 {
  public static String host;
  public static String chatName;
  public static MessageServer0 server0;
  public static int listening = 1;
  public static Client0 client0;
  public static View view;
  public static Vector users = new Vector();
  public static Shared0[] meeting=new Shared0[100];
  public static int no=-1;
  public static int no1=-1;

  public static Vector clientHeuristic=new Vector();
  public


  Client0(View view) throws RemoteException
  {
    try
    {
      this.view = view;
      Registry registry = LocateRegistry.getRegistry(host);
      server0 = (MessageServer0) registry.lookup(MessageServer0.REGISTRY_NAME);
      System.out.println("Registering with server...");
      server0.register(this);
      System.out.println("Registration complete");
    }
     catch (Exception e) { }


  }


  /*
   Allow the server to send the client a Shared0 message object.
   The method then passes the message to the update() method.
   */
  public void sendMessage(Shared0 msg)
  {
    view.update(msg);
```

```
 }


 /*
   Allows the server to retrieve the user name for this
   specific user.
  */
 public String getUserName()
 {
   return chatName;
 }


 /*.
  Allows the server to retrive the state of whether this
  client is, or is not listening.
  */

 public int getListening()
 {
   return listening;
 }



  static class  View extends JFrame {


         JMenuBar bar=new JMenuBar();
         JMenuBar bar1=new JMenuBar();
         JMenuBar bar2=new JMenuBar();
         JMenuBar calBar=new JMenuBar();
         JMenu meetingMenu=new JMenu("Meeting");
         JMenuItem addMeetingMenuItem =new JMenuItem("Add Meeting");
         JMenuItem addConstraintMenuItem =new JMenuItem("Add Constraint");
         JMenuItem schedMeetingMenuItem =new JMenuItem("Schedule Meetings");
         JMenuItem viewMeetingMenuItem =new JMenuItem("View Meeting");

         JMenu advancedSchedMenu=new JMenu("Advanced Scheduling");
         JMenuItem localSearchMenuItem=new JMenuItem("Local Search Repair");


         JMenuItem LGPMenuItem=new JMenuItem("send to LGP Super Agent");
         JMenuItem LGPSPMenuItem=new JMenuItem("send to LGP Super Agent
(local optimum)");
         JMenuItem LSPMenuItem=new JMenuItem("send to LSP Super Agent");
         JMenuItem LSPSPMenuItem=new JMenuItem("send to LSP Super Agent (local
optimum)");
```

```
        JMenu finishAddAttendeesMenu=new JMenu("Meeting Properties");
        JMenu finishAddConstaintMenu=new JMenu("Constraints");
        JMenuItem viewAttendeesMenuItem=new JMenuItem("Add Attendees");
        JMenuItem addDomainMenuItem=new JMenuItem("Add Domain");
        JMenuItem finishMenuItem=new JMenuItem("Back");
        JMenuItem finishMenuItemConstraint=new JMenuItem("Back");

        JButton finishAddDomainButton=new JButton("Finish Domain");
        JButton finishAddConstraintButton=new JButton("Finish Constraint");
        JButton finishAddAttendeesButton=new JButton("Finish Attendees");
        JButton stopListenerButton=new JButton("Stop");
        JButton startListenerButton=new JButton("Start");
        JButton stopListenerButtonConstraint=new JButton("Stop constraint");
        JButton startListenerButtonConstraint=new JButton("Start constraint");


        JCheckBox [] usersCheckBox=new JCheckBox[100];
        JTextField [] usersRankTextField=new JTextField[100];
        int user_no=0;
        int attendeesNo=0;
        int user_rank_sum=0;


        String [][] domain=new String [100][2];
        int iii;//**********************************************
        String [][] receiverName=new String[100][2];
        String [][]solution;
        String [] solution_index;
        String[][] max_violation;
        int max_v_l;
        double tot=0,tot_max_v=0,tot_max_v1=0;
        int find;
        //int [] index_max_violation;
        double [] rankReceiver;
    Lis s=new Lis();
    Lis1 s1=new Lis1();
    checkBoxItemListener checkBoxHandler=new checkBoxItemListener();
    private static final int transcriptRows = 10;
    private static final int transcriptColumns = 30;
    private static final int inputRows = 10;
    private static final int inputColumns = 30;
    private String[] whisperingtoMany=new String[0],copyws=new String[0];
    private String[][] whisperingToMany_r=new
String[0][0],whisperingToMany_r1=new String[0][0],whisperingToMany_r2=new
String[0][0];

  //  double violation=0,violation_no=0,violation_c=0,violation_no_c=0;
    int numberAttendees;
```

```
private JCalendar mycalendar1;
private JTextArea transcript = new JTextArea(transcriptRows, transcriptColumns);
private JTextArea input = new JTextArea(inputRows, inputColumns);
private JTextField roomCount = new JTextField(3);
private JTextField sendToField = new JTextField("Attendees");
private JTextField rankSendToField = new JTextField("Ranks");

JScrollPane scrollPane;
private JLabel nameLabel = new JLabel("You are currently logged on as "
                                   + chatName + ".");
private JLabel roomCountLabel = new JLabel("Current number of users: ");

private JPanel namePanel = new JPanel();

private JPanel infoPanel = new JPanel();

private JPanel panel = new JPanel();
private JPanel panel1 = new JPanel();

private JPanel [] panel22;
private JPanel panel21 = new JPanel();
private JPanel panel2 = new JPanel();

private JPanel calPanel = new JPanel();
private JPanel calPanel1 = new JPanel();
private JPanel calPanel2 = new JPanel();

private JPanel buttonPanel1 = new JPanel();

private JPanel buttonPanel2 = new JPanel();


View () {
  super("Agent Meeting Scheduling ");

  transcript.setEditable(false);
  roomCount.setEditable(false);
  transcript.setLineWrap(true);



  mycalendar1= new JCalendar();
        mycalendar1.setFont(new Font("Dialog", Font.BOLD, 10));

        setJMenuBar(bar);
```

```
        meetingMenu.add(addConstraintMenuItem);
        meetingMenu.add(addMeetingMenuItem);
        meetingMenu.add(schedMeetingMenuItem);
        meetingMenu.add(viewMeetingMenuItem);
        bar.add(meetingMenu);

        advancedSchedMenu.add(localSearchMenuItem);

        advancedSchedMenu.add(LGPMenuItem);
        advancedSchedMenu.add(LGPSPMenuItem);
        advancedSchedMenu.add(LSPMenuItem);
        advancedSchedMenu.add(LSPSPMenuItem);
        bar.add(advancedSchedMenu);

        finishAddAttendeesMenu.add(viewAttendeesMenuItem);
        finishAddAttendeesMenu.add(addDomainMenuItem);
        finishAddAttendeesMenu.add(finishMenuItem);
        bar1.add(finishAddAttendeesMenu);


        finishAddConstaintMenu.add(finishMenuItemConstraint);
        bar2.add(finishAddConstaintMenu);



    buttonPanel1.setBackground(Color.yellow);
    buttonPanel2.setBackground(Color.red);


    localSearchMenuItem.setToolTipText("Local search to find better solution " );


    final JDesktopPane theDesktop=new JDesktopPane();
    getContentPane().add(theDesktop);
    namePanel.add(nameLabel);
    infoPanel.add(roomCountLabel);
    infoPanel.add(roomCount);
    buttonPanel1.setLayout(new FlowLayout());


    final Component[] components =
    {

            namePanel,

            infoPanel,

            new JLabel("Transcript"),
```

```
                new JScrollPane(transcript),
                new JLabel("My Meetings"),
                new JScrollPane(input)
                };

    /**
     * When the user closes the window, this method will send off one
     * final message to the server letting it know that this client
     * is leaving.
     */

    addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
          Shared0 msg = new Shared0(chatName, "", listening);
          try {
            server0.deregister(msg);
          } catch (RemoteException f) { }
          finally { System.exit(0); }
        }
      });



                finishMenuItemConstraint.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e)
                {
                            setJMenuBar(bar);


       panel.add(SwingUtil.vBox(components, SwingUtil.CENTER));
       setContentPane(panel);
    //panel.setBackground(Color.blue);
    pack();
                      }});



                finishMenuItem.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e)
                {


                      String [][] receiverName_c=new String[attendeesNo][2];

                      for(int i=0;i<attendeesNo;i++)
                      {
                            receiverName_c[i][0]=receiverName[i][0];
```

```
                        receiverName_c[i][1]=receiverName[i][1];
                }


                receiverName=new String[attendeesNo][2];

                for(int i=0;i<attendeesNo;i++)
                {
                        receiverName[i][0]=receiverName_c[i][0];
                        receiverName[i][1]=receiverName_c[i][1];
                }

                no++;
                meeting[no] = new Shared0(chatName, domain,
listening,iii,"meeting");
                meeting[no].setName(chatName+no);
                meeting[no].setWhisperingToMany(receiverName);
                meeting[no].setMessageType("proposal");
                meeting[no].dont_loop=0;
                meeting[no].index=0;
                meeting[no].violation=0;
                meeting[no].violation_c=0;
                meeting[no].violation_no=0;
                meeting[no].violation_no_c=0;
        //      meeting[no].rank=100-no;
//(1_2)**************************************************the first entered
meeting the first one scheduled in order to discard the meetings ranks



                numberAttendees = whisperingtoMany.length;

                if(meeting[no].whisperingToMany.length==0)
                {

                        transcript.append("HOST: " + meeting[no].getInitiator() +
                        "- no attendees to send. try again.\n");
                        no--;
                }
                else
                {
                        if (meeting[no].messageArray.length==0)
                        {

                                transcript.append("HOST: " +
meeting[no].getInitiator() +
                                "- no domain to send. tryyyyyyyyyyyyyy
again.\n");
```

```
                                no--;
                            }}


                    setJMenuBar(bar);


        panel.add(SwingUtil.vBox(components, SwingUtil.CENTER));
        setContentPane(panel);
    //panel.setBackground(Color.blue);
    pack();

                    }});


            viewAttendeesMenuItem.addActionListener(new ActionListener() {
             public void actionPerformed(ActionEvent e) {


            setContentPane(panel2);

        pack();

         }
        });

        finishAddAttendeesButton.addActionListener(new ActionListener()
    {
          public void actionPerformed(ActionEvent e)
          {


    //setJMenuBar(bar);


   // panel2.add(SwingUtil.vBox(components, SwingUtil.CENTER));
    setContentPane(panel1);
    //panel.setBackground(Color.blue);
    pack();

  }});



    addConstraintMenuItem.addActionListener(new ActionListener()
    {
          public void actionPerformed(ActionEvent e)
```

```
        {

            setJMenuBar(bar2);
            calPanel2=new JPanel();
            calPanel2.add(mycalendar1);
            calPanel2.add(finishAddConstraintButton);
            calPanel2.add(stopListenerButtonConstraint);
            calPanel2.add(startListenerButtonConstraint);




     setContentPane(calPanel2);
            pack();
            mycalendar1.addPropertyChangeListener(s1);



            }});


   addMeetingMenuItem.addActionListener(new ActionListener()
   {
         public void actionPerformed(ActionEvent e)
         {

     panel1=new JPanel();

            panel21=new JPanel();
            panel2=new JPanel();


         panel22 = new JPanel[10];


            attendeesNo=0;
            receiverName=new String[100][2];
            Enumeration en = users.elements();

         user_no=0;
         Object[] userInfo;
         String currName;
         int currState;
         while(en.hasMoreElements()) {
```

```
            userInfo = (Object[]) en.nextElement();

            if(!( userInfo[0].equals("lsp")) && !( userInfo[0].equals("lgp"))&&!(
userInfo[0].equals("lgpsp"))&&!( userInfo[0].equals("lspsp"))){

            currName = (String) userInfo[0];
            currState = (int) ((Integer) userInfo[1]).intValue();


            usersCheckBox[user_no]=new JCheckBox(currName);
            usersRankTextField[user_no]=new JTextField("      1.0      ");
            user_rank_sum=0;


            usersCheckBox[user_no].addItemListener(checkBoxHandler);

           panel22[user_no]=new JPanel();
            panel22[user_no].add(usersCheckBox[user_no]);

            panel22[user_no].add(usersRankTextField[user_no]);
            user_no++;

           }
           }

           panel21.setLayout(new GridLayout(user_no+1,2));
           for(int i=0;i<user_no;i++)
           {

           panel21.add(panel22[i]);
           }
           panel2.add(panel21);
                panel2.add(finishAddAttendeesButton);


        iii=0;

        calPanel=new JPanel();
                calPanel.add(mycalendar1);
                calPanel.add(stopListenerButton);
                calPanel.add(startListenerButton);
        calPanel.add(finishAddDomainButton);


        domain=new String[100][2];


        setJMenuBar(bar1);
```

```
        setContentPane(panel1);
        pack();

          }
});


     viewMeetingMenuItem.addActionListener(new ActionListener()
     {
          public void actionPerformed(ActionEvent e)
          {
                tot= 0;
                input.setText("MEETING NAME ~~~~  ASSIGNMENT  ~~~~
VIOLATION \n");
                input.append(
"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
                for(int i=0;i<=no;i++)
        {
                if(meeting[i].violationAssignment!=100)
                {
                input.append("        "+meeting[i].getName()+"
"+meeting[i].getAssignment());
        if((meeting[i].getInitiator()).equals(chatName))
                {
                input.append("                    "+meeting[i].violationAssignment+"\n");

                tot=tot+(meeting[i].violationAssignment);

                tot=round(tot,2);
                }
                else
                {
                        input.append("\n");
                }
                }

        }
        input.append( "                                                   ~~~~~~~\n");
        input.append( "                              total of the violations=  "+tot+"\n");

        System.out.println("tot         =         "+tot);


          }
         });
```

```
        finishAddDomainButton.addActionListener(new ActionListener()
  {
        public void actionPerformed(ActionEvent e)
        {


                    mycalendar1.removePropertyChangeListener(s);

        panel2 = new JPanel();
 // panel.add(SwingUtil.vBox(components, SwingUtil.CENTER));
  setContentPane(panel2);
  //panel.setBackground(Color.blue);
  pack();

              }});


               finishAddConstraintButton.addActionListener(new ActionListener()
  {
        public void actionPerformed(ActionEvent e)
        {


                    mycalendar1.removePropertyChangeListener(s1);

        panel2 = new JPanel();
 // panel.add(SwingUtil.vBox(components, SwingUtil.CENTER));
  setContentPane(panel2);
  //panel.setBackground(Color.blue);
  pack();

              }});

              stopListenerButton.addActionListener(new ActionListener()
  {
        public void actionPerformed(ActionEvent e)
        {


                    mycalendar1.removePropertyChangeListener(s);



              }});

                    startListenerButton.addActionListener(new ActionListener()
  {
        public void actionPerformed(ActionEvent e)
```

```
        {

                    mycalendar1.addPropertyChangeListener(s);



            }});


            stopListenerButtonConstraint.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent e)
        {

                    mycalendar1.removePropertyChangeListener(s1);



            }});

                    startListenerButtonConstraint.addActionListener(new
ActionListener()
    {
        public void actionPerformed(ActionEvent e)
        {

                    mycalendar1.addPropertyChangeListener(s1);



            }});



     addDomainMenuItem.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent e)
        {



        setContentPane(calPanel);
        pack();
        mycalendar1.addPropertyChangeListener(s);
```

```
}
    }       );

   ;



    schedMeetingMenuItem.addActionListener(new ActionListener()
{
     public void actionPerformed(ActionEvent e)
     {


          node_consistency();

           if(clientHeuristic.size()<=no)
          {
                         for(int a=clientHeuristic.size();a<=no;a++)
                 {

                         if(a==0)
                         {
                                 clientHeuristic.add("rs");
                         }

                         else
                         {
                         clientHeuristic.add("s");

                         }

            }
   }



    for(int a=0;a<clientHeuristic.size();a++)
     {
          System.out.println("step========"+clientHeuristic.get(a));

          }



   System.out.println("no1="+no1+"    no="+no);

   clearMeetings();
     for(int ii=0;ii<=no;ii++)
```

```
                {
                System.out.println("ii="+ii);

                Object o=clientHeuristic.get(ii);
                String os=(String)o;
                if((os).equals("rs"))
                {
                        System.out.println("rs");
                        findMeetingsRankandschedulingProcess(no);//(2-
2)*****************************************call the ranking function in
order to ranks the meetings
                }
                else
                {
                        System.out.println("s");
                        schedulingProcess();
                }

            }
          }
         }
        );


        localSearchMenuItem.addActionListener(new ActionListener()
        {
                public void actionPerformed (ActionEvent e)
                {
                        System.out.println("local search no=    "+no);
                        solution=new String[no+1][2];
                        solution_index=new String[no+1];
                        max_v_l=0;
                        find=0;
                        tot_max_v=tot;

                        System.out.println("local search tot ="+tot+"    tot_max_v
="+tot_max_v);

                        neigbourhood();



                }
        });

    LGPMenuItem.addActionListener(new ActionListener()
        {
                public void actionPerformed (ActionEvent e)
```

```
                {

                 Shared0 msg_heu=new
Shared0(chatName,"lgp",clientHeuristic,listening);
                 msg_heu.setMessageType("msg_type_c_h");
                 sendTo(msg_heu);

                 Shared0 msg_lsp=new
Shared0(chatName,"lgp",meeting,listening,no+1);
                 msg_lsp.setMessageType("msg_type_lsp");
                 sendTo(msg_lsp);
                        }});


           LGPSPMenuItem.addActionListener(new ActionListener()
           {
                public void actionPerformed (ActionEvent e)
                {

                 Shared0 msg_heu=new
Shared0(chatName,"lgpsp",clientHeuristic,listening);
                 msg_heu.setMessageType("msg_type_c_h");
                 sendTo(msg_heu);

                 Shared0 msg_lsp=new
Shared0(chatName,"lgpsp",meeting,listening,no+1);
                 msg_lsp.setMessageType("msg_type_lsp");
                 sendTo(msg_lsp);
                        }});



           LSPMenuItem.addActionListener(new ActionListener()
           {
                public void actionPerformed (ActionEvent e)
                {

                 Shared0 msg_heu=new
Shared0(chatName,"lsp",clientHeuristic,listening);
                 msg_heu.setMessageType("msg_type_c_h");
                 sendTo(msg_heu);

                 Shared0 msg_lsp=new
Shared0(chatName,"lsp",meeting,listening,no+1);
                 msg_lsp.setMessageType("msg_type_lsp");
                 sendTo(msg_lsp);
                        }});
```

```
            LSPSPMenuItem.addActionListener(new ActionListener()
            {
                    public void actionPerformed (ActionEvent e)
                    {

                     Shared0 msg_heu=new
Shared0(chatName,"lspsp",clientHeuristic,listening);
                     msg_heu.setMessageType("msg_type_c_h");
                     sendTo(msg_heu);

                     Shared0 msg_lsp=new
Shared0(chatName,"lspsp",meeting,listening,no+1);
                     msg_lsp.setMessageType("msg_type_lsp");
                     sendTo(msg_lsp);
                             }});



    /*
     * This section deals with action events from the "Who's Here" button.
     */



    JPanel panel = new JPanel();
    panel.add(SwingUtil.vBox(components, SwingUtil.CENTER));
    setContentPane(panel);
    //panel.setBackground(Color.blue);
    pack();
    setVisible(true);
    input.requestFocus();
    input.setLineWrap(true);
    setResizable(true);
  }

  public void solutionRepresentation()
  {
        max_v_l=0;
        tot_max_v=0;

        for(int i=0;i<=no;i++)
        {

                    System.out.println("  meeting name      "+meeting[i].getName()+"
initiator =          "+meeting[i].getInitiator()+"          me     "+chatName);
                    if((meeting[i].getInitiator()).equals(chatName))
                    {
```

```
                //System.out.println("        "+meeting[i].getName()+"        initiator
"+meeting[i].getInitiator()+"        me        "+chatName);
                solution [max_v_l][0]=meeting[i].getAssignment();

                solution
[max_v_l][1]=((Double)meeting[i].violationAssignment).toString();

                solution_index[max_v_l]=""+i;

                //if((Double)meeting[i].violationAssignment!=100)
                //{
                        max_v_l++;
                //}
                System.out.println("solution"+ solution[i][0]+"
violation"+solution[i][1]);
                }

        }

        max_violation=new String[max_v_l][2];

        //System.out.println("_________________________________"+max_v_l);
                int s=0;
        for(int i=0;i<max_v_l;i++)
        {

        System.out.println(""+meeting[i].getName()+"violationAssignment="+meeting[
i].violationAssignment);

        //      if((Double)meeting[i].violationAssignment!=100)
        //      {
        //              System.out.println("true");
                        max_violation[s][0]=solution[i][1];
                        max_violation[s][1]=solution_index[i];
                        s++;
        //      }
                }

                String temp,temp_index;
        System.out.println("max_violation.length="+max_violation.length);
                for (int i = (max_violation.length-1); i >= 0; i--)
       for (int j = 0; j < i; j++)
         if (Double.parseDouble(max_violation[j][0]) <
Double.parseDouble(max_violation[j + 1][0]))
         {
           temp = max_violation[j][0];
           temp_index=max_violation[j][1];
           max_violation[j][0] = max_violation[j + 1][0];//the value of the max violation
```

```
                max_violation[j][1] = max_violation[j+1][1];// the index of tha max violation
in the solution representation
                max_violation[j + 1][0] = temp;
                max_violation[j + 1][1] = temp_index;

            }
//          System.out.println("Max_violarion list is ready");



                for(int i=0;i<max_violation.length;i++)
            {
                    System.out.println("max violation"+max_violation[i][0]+
"index"+max_violation[i][1]+" assigmnet
"+solution[Integer.parseInt(max_violation[i][1])][0]);
                    tot_max_v=tot_max_v+Double.parseDouble(max_violation[i][0]);
                    tot_max_v=round(tot_max_v,2);
            }
            System.out.println(" total max_violation= list "+tot_max_v);

            if((tot_max_v==0)||(Double.parseDouble(max_violation[0][0])==0))
            {
                    find++;
                    System.out.println("~~~~~optimal solution found cause
max_violation=o~~~~~~~~~~~~");
            }



        }

        public void neigbourhood()
        {
            System.out.println("neigbor");


            solutionRepresentation();
                    System.out.println("solutoin ");
            tot_max_v1=tot_max_v;
            System.out.println("tot_max_l         ======="+tot_max_v1);

            int r=-1;

            while((tot_max_v!=0)&&(Double.parseDouble(max_violation[0][0])!=0)&&(r<
(max_violation.length)-1)&&(Double.parseDouble(max_violation[r+1][0])!=0))
            {
```

```
        System.out.println("r="+r+"Double.parseDouble(max_violation[r+1][0])"+Doub
le.parseDouble(max_violation[r+1][0]));
                find=0;


                r=-1;

                while ((find==0)&&(r<(max_violation.length)-
1)&&(Double.parseDouble(max_violation[r+1][0])!=0))
                {

        System.out.println("rrrrrrrrr="+r+"Double.parseDouble(max_violation[r+1][0])
"+Double.parseDouble(max_violation[r+1][0]));
                        r++;
                        String [] domain_max_violation =new String
[meeting[Integer.parseInt(max_violation[r][1])].messageArray.length];
                        System.out.println("domain length max violation is
"+domain_max_violation.length);
                        for(int i=0;i<domain_max_violation.length;i++)// find the
domain for the max violated meeting
                        {

        domain_max_violation[i]=meeting[Integer.parseInt(max_violation[r][1])].messa
geArray[i][0];
                                System.out.println("domain max violation is
"+domain_max_violation[i]);
                        }


                        for(int i=0;i<=no;i++)//search in all meetings
                        {

                                for(int j=0;j<domain_max_violation.length;j++)//search
in all domain of the most violated meeting
                                {
                        //
        System.out.println("meeting[i].getAssignment()"+meeting[i].getAssignment()+"
domain_max_violation[j]"+domain_max_violation[j]+"equals"+(meeting[i].getAssign
ment()).equals(domain_max_violation[j]));
                        //
        System.out.println("((meeting[i].getInitiator()).equals(chatName))"+((meeting[i]
.getInitiator()).equals(chatName)));

        if(((meeting[i].getInitiator()).equals(chatName))&&((meeting[i].getAssignment(
)).equals(domain_max_violation[j])))//find meeting assigned to domain for MVM
                                        {
```

```
                                        for(int
k=0;k<meeting[i].messageArray.length;k++)// search in the domain of that meeting
                                        {
                                    //
    System.out.println("meeting[i].getAssignment()"+meeting[i].getAssignment()+"
meeting[Integer.parseInt(max_violation[r][1])].getAssignment()"+meeting[Integer.parse
Int(max_violation[r][1])].getAssignment());

    if((!(meeting[i].getAssignment()).equals(meeting[Integer.parseInt(max_violatio
n[r][1])].getAssignment()))&&((meeting[i].messageArray[k]).equals(solution[Integer.p
arseInt(max_violation[r][1])][0]))&&(!(meeting[i].getName()).equals(meeting[Integer.p
arseInt(max_violation[r][1])].getName()))&&(!(meeting[i].getAssignment()).equals(me
eting[Integer.parseInt(max_violation[r][1])].getAssignment())))
                                            {
                                                //System.out.println("true");

    //System.out.println("tot_max_v before swap"+tot_max_v);
                                                tot_max_v1=tot_max_v;

                                                System.out.println("swap
"+meeting[i].getName()+" with
"+meeting[Integer.parseInt(max_violation[r][1])].getName());
                                                Shared0 m1=new Shared0
(),m2=new Shared0();
                                            //  m1=meeting[i];
                                        //
    m2=meeting[Integer.parseInt(max_violation[r][1])];


    swap(meeting[i],meeting[Integer.parseInt(max_violation[r][1])]);
                                                solutionRepresentation();


    if(tot_max_v1<=tot_max_v)
                                                {

    System.out.println("tot_max_v1="+tot_max_v1+"tot_max_v= after
swap"+tot_max_v+"so re swap");

    swap(meeting[Integer.parseInt(max_violation[r][1])],meeting[i]);

    solutionRepresentation();


    System.out.println("tot_max_v after canceling swap"+tot_max_v);
                                                }
                                                else
                                                {
```

```
                                                                find++;


          System.out.println("~~~~~better solution solution has been
found~~~~~~~~~~~~");
                                                                break;
                                                          }

                                              }

                                        }

                                        if(find!=0)
                                        {
                                              System.out.println("break");
                                              break;
                                        }


                              }
                        }
                        if(find!=0)
                        {
                              System.out.println("break");
                              break;
                        }


                  }

            }

      }//while max_violation[0][0]!=0


      }


      public void swap(Shared0 s_i,Shared0 s_index_max_violation)
      {
            String s=new String(s_i.getAssignment());
            String s2=new String(s_index_max_violation.getAssignment());


            s_i.setMessageType("confirm_delete");
            s_index_max_violation.setMessageType("confirm_delete");
```

```
          sendTo(s_i);//to delete the meeting
          sendTo(s_index_max_violation);// to delete the meeting




                    // reply(s_i);
          System.out.println("*********************meeting  "+s_i.getName()+"
violation assignment "+s_i.violationAssignment);
   //     reply(s_index_max_violation);
          System.out.println("***********************meeting
22222222222"+s_index_max_violation.getName()+" violation assignment
"+s_index_max_violation.violationAssignment);


          s_i.setMessageType("confirm");
          s_i.setMessage(s2);
          s_i.setAssignment(s2);


          s_index_max_violation.setMessageType("confirm");
          s_index_max_violation.setMessage(s);
          s_index_max_violation.setAssignment(s);




                    sendTo(s_i);

                    System.out.println("++++++++++++++++++meeting  "+s_i.getName()+"
violation assignment "+s_i.violationAssignment);
                    sendTo(s_index_max_violation);
          System.out.println("++++++++++++++++++++meeting
22222222222"+s_index_max_violation.getName()+" violation assignment
"+s_index_max_violation.violationAssignment);




   }



   public Dimension getPreferredSize() {
    return (new Dimension(400, 800));
   }
```

```
public void schedualing(Shared0 meeting )
{



    int l=copyws.length;


            try
            {


   if((meeting.messageArray[meeting.index][1]).equals("yes"))

                        {
                            String proposal =
meeting.messageArray[meeting.index][0];
                        meeting.setMessage(proposal);
                        sendTo(meeting);
                        }
                        else
                        {
                            meeting.index++;
                        if(meeting.index<(meeting.messageArray.length))
                            {
                                schedualing(meeting);

                            }
                            else
                            {

   if((meeting.getAssignment()).equals(""))
                                 {
                                    meeting.setAssignment("");
                                meeting.violationAssignment=1;
                                meeting.scheduled=false;
                                }


                            }
                        }




        }
```

```
                        catch(Exception e)
                        {
                                transcript.append("HOST: " + meeting.getInitiator() +
                                "- no domain tooooooooooo send. try again.\n");
                        }



    }



    public void sendTo(Shared0 msg)
    {

        //System.out.println("SEND To");
        String from = msg.getInitiator();



        msg.setUsers(users);
        Enumeration en = users.elements();
        Object[] currUserInfo;

        while(en.hasMoreElements())
        {
                currUserInfo =  (Object[]) en.nextElement();
     try{

                for(int i=0;i<msg.whisperingToMany.length;i++)

                {

                        if (currUserInfo[0].equals(msg.whisperingToMany[i][0]) )

                        try
                        {
                                ((MessageClient0) currUserInfo[2]).sendMessage(msg);
                        }
                                catch (RemoteException e) { System.out.println("error
send to");}

                        }
                }//try
                        catch(Exception e){System.out.println("widesprd exception");}
        }
        }
```

```
/*
* Allows clients to public messages that will be sent to every
* person in the room that is listening at that time.
*/
    public void sendAll(Shared0 msg)
    {
    Enumeration en= users.elements();
    Object[] currUserInfo;
       msg.setUsers(users);

    while(en.hasMoreElements())
    {

            currUserInfo =  (Object[]) en.nextElement();

            if (((Integer) currUserInfo[1]).intValue() == 1)

            {
            try
                    {
                    ((MessageClient0) currUserInfo[2]).sendMessage(msg);

                    }

            catch (RemoteException e)
                    {
                    }
            }
    }
    }


/**
 * Appends all incoming "chat" messages (not "state" messages)
 * to the transcript window, and updates this client with the
 * current chat room user information.
 */
public void update(Shared0 msg)
{


    //System.out.println("UPDATE ");

try{
    if ((listening == 1) || msg.getWhispering())
    if((msg.msgType).equals("msg_type_s_h"))
    {
            System.out.println("heuristic received");
```

```
        clientHeuristic=msg.heuristic;
        }
                else if((msg.msgType).equals("confirm"))
        transcript.append(msg.getInitiator() + " sends a CONFIRMATION for meeting:
 "+msg.getName()+"   " + msg.getMessage()+"\n");
        else if((msg.msgType).equals("proposal"))
        transcript.append(msg.getInitiator() +" sends a PROPOSAL for meeting:  "+
msg.getName()+"   "+ msg.getMessage()+"\n");
        roomCount.setText((String) msg.getRoomSize());
        users = (Vector) msg.getUsers();
    }
    catch(Exception e){System.out.println("listtttttttttttning");}

        try{


                if((msg.msgType).equals("empty"))
        {
                System.out.println("received empty msg");
                System.out.println("no before empty="+no);
        /*              meeting=new Shared0[100];
                        no=-1;
          no1=-1;*/

          for(int t=0;t<=no;t++)
          {
                if((meeting[t].meeting_constraint).equals("constraint"))
                {

                }
                else
                {
                        no=t-1;
                        no1=no;
                        break;
                }
          }

         System.out.println("no after empty="+no);
        }
                else if((msg.msgType).equals("msg_type_lsp1"))
        {

                        meeting=new Shared0[msg.meetings.length];
                        for(int s=0;s<msg.meetings.length;s++)
                        {
```

```
                        meeting[s]=msg.meetings[s];

                        meeting[s].setInitiator(chatName);
                        System.out.println("meeting==="+meeting[s].getName()+"
initiator  "+meeting[s].getInitiator());
                        no++;
                        }

        }

    else            if((msg.msgType).equals("proposal"))
        {

                        reply(msg);

        }


        else if((msg.msgType).equals("reply"))
        {

                int i=0;
                try{

                        for(i=0;i<=no;i++)//find this rply for which meeting
                        {

                                if((meeting[i].getName()).equals(msg.getName()))
                                {


                                        meeting[i].violation_no++;// we received one rply



                                                for(int
k=0;k<meeting[i].whisperingToMany.length;k++)
                                                {
                                                        if
((meeting[i].whisperingToMany[k][0].equals(msg.getInitiator())))
                                                        {


        meeting[i].violation=meeting[i].violation+(Double.parseDouble(meeting[i].whis
peringToMany[k][1]))*Double.parseDouble(msg.getMessage());

        arc_consistency(meeting[i].messageArray[meeting[i].index][0],msg.getInitiator(
));
```

```
                                                break;

                                            }
                                        }

                                //}

                            break;//we found for which this reply is
                            }
                        }
                    }
                    catch(Exception e)
                    {
                        System.out.println("error in finding to which proposal this
reply");
                    }

                    try{



        if(meeting[i].violation_no==meeting[i].whisperingToMany.length)//all replies
have been received
                        {


                            try
                            {


                                if(meeting[i].violation==0)
                                {


        meeting[i].setAssignment(meeting[i].getMessage());
                                    meeting[i].violationAssignment=0;


                                }

                                else
                                {
                                //  System.out.println("violatiojn/violation
no="+l);
                                    if
(meeting[i].violationAssignment>=meeting[i].violation)
                                    {
```

```
meeting[i].setAssignment(meeting[i].getMessage());

meeting[i].violationAssignment=meeting[i].violation;

meeting[i].violationAssignment=round(meeting[i].violationAssignment,2);

                                        //System.out.println("meeting "+
meeting[i].getName()+"    propose schedualed so rank =");



                                    }

                                    meeting[i].index++;

                                    //System.out.println("iii="+iii);

if(meeting[i].index<(meeting[i].messageArray.length))
                                    {
                                        meeting[i].violation=0;
                                        meeting[i].violation_no=0;
                                        schedualing(meeting[i]);

                                    }

                                }

                            }
                            catch(Exception e)
                            {
                                System.out.println("4444444444444444");
                            }

                            try
                            {

                            if(meeting[i].dont_loop==0)
                                {
                                    no1++;

                                    System.out.println("meeting"+i+"
.violationAssignment="+meeting[i].violationAssignment);
                                    if((meeting[i].violationAssignment<=0))
                                    {
                                    System.out.println("if is true");
                                    Shared0 conf_msg=new
Shared0(chatName,meeting[i].getAssignment(),listening);
```

```
                                                  conf_msg.setName(meeting[i].getName());

                                                  conf_msg.setMessageType("confirm");


conf_msg.setWhisperingToMany(meeting[i].whisperingToMany);

arc_consistency(meeting[i].getAssignment());
                                                  meeting[i].dont_loop++;
                                                  sendTo(conf_msg);
                                                  }
                                                  else
                                                  {
                                                       meeting[i].setAssignment("");
                                                       meeting[i].violationAssignment=1;
                                                       meeting[i].scheduled=false;
                                                  }


                                        }
                              }

                              catch(Exception e)
                              {
                                   System.out.println("8888888888");
                              }

                    }

          }
          catch(Exception e)
          {
               System.out.println("222222222222222222");
          }
//        System.out.println("hiiiiiiiiiii");

}
else if ((msg.msgType).equals("confirm"))
{

System.out.println("***********received confirm msg");
          boolean found=false;
          try
          {
          // search in the meeting if this confirmed meeting is exist
               int i;
               for(i=0;i<=no;i++)
```

```
                        {
                        System.out.println("my meetings are no"+i+" its
name"+meeting[i].getName());
                                if ((msg.getName()).equals(meeting[i].getName()))
                                {
                                        System.out.println("+++++++++++++++++name
for this meeting is"+msg.getName());
                                        meeting[i].setAssignment(msg.getMessage());
                                        meeting[i].violationAssignment=0;
                                        meeting[i].attend=true;
                                        meeting[i].scheduled=true;
                                        found=true;
                                        //no1++;

                                                //System.out.println("meeting name
"+meeting[i].getName()+"violation assignmt     "+meeting[i].violationAssignment);
                                        break;
                                }
                        }

                        //this meeting is not exist then add it
                        if (!found)
                        {       System.out.println("this meeting is not exist i will add it i
have meeting number"+no);

                                no++;
                                no1++;
                                meeting[no]=new
Shared0(msg.getInitiator(),msg.getMessage(),listening);
                                meeting[no].setName(msg.getName());
                                meeting[no].setAssignment(msg.getMessage());
                                meeting[no].violationAssignment=0;
                                meeting[no].attend=true;
                                meeting[no].scheduled=true;
                                System.out.println("now ihave meeting no "+no+" and the
assgment"+meeting[no].getAssignment());


                        }
                //reply(msg);
                }
                catch(Exception e)
                {
                        System.out.println(" can not confirm this meeting");
                }
        }
```

```
                else if ((msg.msgType).equals("confirm_delete"))
        {


                try
                {
                System.out.println("CONFIRM DELETE  to"+chatName+"      regarding
meeting"+msg.getName());

                reply(msg);
                }
                catch(Exception e)
                {
                        System.out.println(" can not confirm this meeting");
                }
        }



        else if ((msg.msgType).equals("confirm_reply"))
        {

        //System.out.println("msg reply for confirm received
from"+msg.getInitiator()+"regarding meeting"+msg.getName()+"with
value"+msg.getMessage());
        int i=0;
        try{
                        for( i=0;i<=no;i++)//find this rply for which meeting
                        {

                                if((meeting[i].getName()).equals(msg.getName()))
                                {
                                        meeting[i].violation_no_c++;// we received one
rply


                                                for(int
k=0;k<meeting[i].whisperingToMany.length;k++)
                                                {
                                                        if
((meeting[i].whisperingToMany[k][0].equals(msg.getInitiator())))//find sender
                                                        {
                                                        //System.out.println("msg
received"+msg.getMessage()+" from "+msg.getInitiator()+"regarding meeting
"+meeting[i].getName()+"
rank"+(Double.parseDouble(meeting[i].whisperingToMany[k][1]))+"old violation
```

```
"+meeting[i].violationAssignment+"violation_no_c="+meeting[i].violation_no_c+"
whidspreadtomany="+meeting[i].whisperingToMany.length);

        meeting[i].violation_c=meeting[i].violation_c+(Double.parseDouble(meeting[i].
whisperingToMany[k][1]))*(Double.parseDouble(msg.getMessage()));//take sender
rank

        meeting[i].violation_c=round(meeting[i].violation_c,2);

        //System.out.println("===========new violation_c"+meeting[i].violation_c);
                                                        break;
                                                        }
                                                }


                                //      }

                                break;//we found for which this reply is
                                }
                        }
                  }
                catch(Exception e)
                {
                        System.out.println("||||||||||||||||||||error in finding to which
confirmation this reply");
                }


        if(meeting[i].violation_no_c==meeting[i].whisperingToMany.length)//all replies
have been received
                        {


                                try
                                {

        meeting[i].violationAssignment=meeting[i].violation_c;

        meeting[i].violationAssignment=round(meeting[i].violationAssignment,2);
                                        //System.out.println("++++++++++++meeting
"+meeting[i].getName()+"assigned
to"+meeting[i].getAssignment()+"violation="+meeting[i].violationAssignment);
                                        meeting[i].violation_c=0;
                                        meeting[i].violation_no_c=0;

                                }
                                catch(Exception e)
                                {
```

```
                                        System.out.println("error in violation calculation
");
                                }

                        }

                        else
                        {

                        }

    }

        else if ((msg.msgType).equals("confirm_reply_update"))
          {
          System.out.println("msg confirm reply update  received
from"+msg.getInitiator()+"regarding meeting"+msg.getName()+"  with
value"+msg.getMessage());
          int i=0;
          try{
                System.out.println(chatName+"received update confirm for
meeting"+msg.getName());
                        for( i=0;i<=no;i++)//find this rply for which meeting
                        {
                        System.out.println("existed meetings :  "+meeting[i].getName());
                                if((meeting[i].getName()).equals(msg.getName()))
                                {
                                        System.out.println("meeting found");
                                        for(int
k=0;k<meeting[i].whisperingToMany.length;k++)
                                        {
                                                if
((meeting[i].whisperingToMany[k][0].equals(msg.getInitiator())))//find sender
                                                {
                                        //System.out.println("msg received confirm update
for"+meeting[i].getName()
                                        //        +"from
"+msg.getInitiator()+"value"+msg.getMessage()
                                        //        +"
rank"+(Double.parseDouble(meeting[i].whisperingToMany[k][1]))+"old violation
"+meeting[i].violationAssignment);



          meeting[i].violationAssignment=(meeting[i].violationAssignment)

          +(Double.parseDouble(meeting[i].whisperingToMany[k][1]))*(Double.parseDo
uble(msg.getMessage()));
```

```
                                                //take sender rank


        meeting[i].violationAssignment=round(meeting[i].violationAssignment,2);

        //System.out.println("############ meeting
name"+meeting[i].getName()+"new
violationAssignment"+meeting[i].violationAssignment);
                                                break;
                                                }
                                        }

                                        break;//we found for which this reply is

                        }


                }

        }

        catch(Exception e)
        {
                System.out.println("error in finding to which confirmation this
reply");
        }






}




else if((msg.msgType).equals("busy"))
{

        System.out.println("busy ");
        double free=0;

        String [][]wsr=new String[1][2];
        wsr[0][0]=msg.getInitiator();
        wsr[0][1]="1";
```

```
        for(int i=0;i<msg.messageArray.length;i++)
        {
                System.out.println("-------------i="+i+" i have number of
meetings="+no);


                for(int j=0;j<=no;j++)
                {
                System.out.println("-------------meeting="+j+"
scheduled"+meeting[j].scheduled);

                        if((meeting[j].scheduled))
                        {
                                System.out.println("i have scheduled al ready
meeting"+j+"    assigment        "+meeting[j].getAssignment()+"
msg.messageArray[i][0]"+msg.messageArray[i][0]);


                        if((meeting[j].getAssignment()).equals(msg.messageArray[i][0]))
                        {
                                System.out.println("the same date");
                                free++;
                                break;
                        }
                        }
                        }
                        }
    System.out.println("free="+free);
        free=free/msg.messageArray.length;

        Shared0 busyReply=new Shared0(chatName,""+free,1);




        busyReply.setWhisperingToMany(wsr);

        busyReply.msgType="busy_reply";

        busyReply.setName(msg.getName());


        sendTo(busyReply);
    }

    else if((msg.msgType).equals("busy_reply"))
    {
```

```
int i=0;
try{
                for( i=0;i<=no;i++)//find this rply for which meeting
                {

                        if((meeting[i].getName()).equals(msg.getName()))
                        {
                                meeting[i].busyReplyAll++;// we received one
rply


                                        for(int
k=0;k<meeting[i].whisperingToMany.length;k++)
                                        {
                                                if
((meeting[i].whisperingToMany[k][0].equals(msg.getInitiator())))//find sender
                                                {
                                //              System.out.println("busy_reply
msg received="+msg.getMessage()+" from "+msg.getInitiator()+"regarding meeting
"+meeting[i].getName()+"
rank"+(Double.parseDouble(meeting[i].whisperingToMany[k][1]))+"old busy
"+meeting[i].busyReplyRank+"violation_no_c="+meeting[i].violation_no_c+"
whidspreadtomany="+meeting[i].whisperingToMany.length);

        meeting[i].busyReplyRank=meeting[i].busyReplyRank+((Double.parseDouble(
meeting[i].whisperingToMany[k][1]))*(round((Double.parseDouble(msg.getMessage())
),2))*10);//take sender rank

        meeting[i].busyReplyRank=round(meeting[i].busyReplyRank,2);
                                        //
        System.out.println("===========new
busyReplyRank"+meeting[i].busyReplyRank);
                                                        break;
                                                        }
                                                }



                        //      }

                        break;//we found for which this reply is
                        }
                }
        }
        catch(Exception e)
        {
```

```
                    System.out.println("error in finding to which meeting this busy
reply");
              }



          if(meeting[i].busyReplyAll==meeting[i].whisperingToMany.length)//all replies
have been received
                    {


                         try
                         {

          //System.out.println(""+meeting[i].getName()+"received all busy replies");
                              meeting[i].rank=meeting[i].busyReplyRank;
                              //meeting[i].rank=round(meeting[i].rank,2);
                              //System.out.println("meeting
"+meeting[i].getName()+"ranked to"+meeting[i].rank);
                              meeting[i].busyReplyAll=0;
                              meeting[i].busyReplyRank=0;

                         }
                         catch(Exception e)
                         {
                              System.out.println("error in rank calculation ");
                         }

                    }

                    else
                    {

                    }



  }

  }//try

  catch(Exception e)
  {
       System.out.println("erroor update");
  }
  }

  public void reply(Shared0 msg)
```

```
{

        int r=0;

        whisperingToMany_r=new String[1][2];
        whisperingToMany_r1=new String[1][2];

        whisperingToMany_r[0][0]=msg.getInitiator();
        whisperingToMany_r[0][1]="1";


        int [] otherEffectedMeetings=new
int[no+1],otherEffectedMeetings1=new int[no+1];
        int effectedMeetings=0,effectedMeetings1=0;
        int k;


                //System.out.println("   check all meeting");
        for(int i=0;i<=no;i++)//search in all meetings
        {


    //  System.out.println("meeting   "+meeting[i].getName()+" assigned to
"+meeting[i].getAssignment()
     //  +"msg.getmessge  "+msg.getMessage());

        if ((meeting[i].getAssignment()).equals(msg.getMessage()))//if the
meeting has the same assignment
        {
                if(meeting[i].attend)//I am attend this meeting
                {
                        if((meeting[i].getName()).equals(msg.getName()))// it is
the same meeting
                        {
                        /*      if ((msg.msgType).equals("confirm_delete"))
                                {

                                otherEffectedMeetings[effectedMeetings]=i;
                                effectedMeetings++;
                                }*/
                        //System.out.println("*** meeting
"+meeting[i].getName()+" has the same assignemnt"+meeting[i].getAssignment());
                        }
                        else// there is another meeting with this assignment
                        {
```

```
                    /*      if((msg.msgType).equals("confirm")||
(msg.msgType).equals("confirm_delete"))
                            {

                            //System.out.println("meeting
"+meeting[i].getName()+" has the same assignemnt"+meeting[i].getAssignment());
                            otherEffectedMeetings[effectedMeetings]=i;
                            effectedMeetings++;
                            //System.out.println("effected meeting no
="+effectedMeetings+
//                          " name
"+meeting[otherEffectedMeetings[effectedMeetings]].getName());



                            }*/
                            r++;//how many meeting have at the same time
                    //      System.out.println("rrrrrrrrrrrrrrrrrrrrrrrr"+r);

                    //      break;
                    }
                }
            }
            }
              //System.out.println("msg type "+msg.msgType+"r="+r);
/*      for(int t=0;t<effectedMeetings;t++)
        {
            System.out.println("EEEEEEEEEEEEEEEeffected meeting"+
meeting[otherEffectedMeetings[t]].getName());
        }   */

   //   System.out.println("fish effected meetings");

   /*   if((msg.msgType).equals("confirm")||(msg.msgType).equals("confirm_delete"))
            {

                    if((msg.msgType).equals("confirm"))
                    {
                            for(int i=0;i<effectedMeetings;i++)
                            {

                            Shared0 msg2 = new Shared0(chatName, "1",
listening);
                            System.out.println("msg.msgType =
"+msg.msgType);
                            msg2.setMessage("1");
```

```
                              msg2.setMessageType("confirm_reply_update");


        msg2.setName(meeting[otherEffectedMeetings[i]].getName());
                              whisperingToMany_r1[0][0] =
meeting[otherEffectedMeetings[i]].getInitiator();
                                      whisperingToMany_r1[0][1]="1";
                              msg2.setWhisperingToMany(whisperingToMany_r1);


                              sendTo(msg2);
                      //      System.out.println("msg sent");
                              }
                      }
                      else if((msg.msgType).equals("confirm_delete"))
                              {

                                      if(effectedMeetings>1)
                                      {


                                              {

                                              for(int i=0;i<effectedMeetings;i++)
                                                      {

                                                      Shared0 msg2 = new
Shared0(chatName, "-1", listening);
                                                      System.out.println("msg.msgType
=  "+msg.msgType);
                                                      msg2.setMessage("-1");

        msg2.setMessageType("confirm_reply_update");


        msg2.setName(meeting[otherEffectedMeetings[i]].getName());

                                                      whisperingToMany_r1[0][0] =
meeting[otherEffectedMeetings[i]].getInitiator();

        whisperingToMany_r1[0][1]="1";

        msg2.setWhisperingToMany(whisperingToMany_r1);
                                                      sendTo(msg2);
                      //      System.out.println("msg sent");
                                                      }
                                      }
```

```
                    }

                }

            }*/

            if ((msg.msgType).equals("confirm_delete"))
            {}
            else
            {


            Shared0 msg1 = new Shared0(chatName, ((Integer)r).toString(),
listening);

            if((msg.msgType).equals("proposal"))
            msg1.setMessageType("reply");
            else
            msg1.setMessageType("confirm_reply");

            msg1.setName(msg.getName());
            msg1.setWhisperingToMany(whisperingToMany_r);
            sendTo(msg1);
            }


    }



        public void findMeetingsRankandschedulingProcess(int m)
        {

            Shared0 [] busy=new Shared0[no+1];

            //System.out.println("m=   "+m);

            for(int a=0;a<=no;a++)
            {
                //System.out.println("aaaa="+a);

                if(((meeting[a].getInitiator()).equals(chatName))&&
meeting[a].rank!=-2)// i am the initiator and not schedualed yet
                {
                    //System.out.println("111");

                meeting[a].busyReplyRank=0;
```

```
                    //System.out.println("2222");

                busy[a]=new
Shared0(chatName,meeting[a].messageArray,1,meeting[a].messageArray.length,"meeti
ng");
                //      System.out.println("333");

                busy[a].setWhisperingToMany(meeting[a].whisperingToMany);
            //  System.out.println("444");
                busy[a].setName(meeting[a].getName());
                //System.out.println("1555511");
                busy[a].msgType="busy";
                //      System.out.println("16661");

                sendTo(busy[a]);
            //          System.out.println("7771");

                }
            }
            schedulingProcess();
            //no=no-1;
        }


public void schedulingProcess()
{


                int max_rank_meeting=0;


            for (int j = 0; j <=no; j++)
        {

                if( (meeting[j].getInitiator()).equals(chatName ))
                {

                    if ((meeting[j].rank) >(meeting[max_rank_meeting
].rank))
                {


                max_rank_meeting = j;

                }

                else if (((meeting[j].rank) == (meeting[max_rank_meeting
].rank))&& (j!=max_rank_meeting) )
```

```
                    {


                                    if ((meeting[j].messageArray.length)
<(meeting[max_rank_meeting].messageArray.length))
                              {

                                    max_rank_meeting=j;




                              }
                    }
                    }

               }





                    if(meeting[max_rank_meeting].rank>=-1)
                    {


               System.out.println("scheduling meeting
"+meeting[max_rank_meeting].getName()+"
Rank=="+meeting[max_rank_meeting].rank);
                    //
System.out.println("meeting[max_rank_meeting].scheduled"+meeting[max_rank_meeti
ng].scheduled);
               meeting[max_rank_meeting].rank=-2;
               schedualing(meeting[max_rank_meeting]);

               }
}
   public String[] effect(Shared0 m)
   {
        String []e_meetings=new String[10];
        return e_meetings;
   }

   public void node_consistency()
   {
        for(int m=0;m<=no;m++)//loop for all entered new meetings
```

```
        {
                if((meeting[m].getInitiator()).equals(chatName))

                for(int d=0;d<meeting[m].messageArray.length;d++)//loop for the
domain of this new meeting
                {
                        for(int mm=0;mm<=no1;mm++)//loop for all the scheduled
meetings
                        {

        if((meeting[mm].getAssignment()).equals(meeting[m].messageArray[d][0]))
                                {
                                        meeting[m].messageArray[d][1]="no";
                                }

                        }

                }
        }
    }

    public void arc_consistency(String date1, String att1)
    {
        for(int m=0;m<=no;m++)//loop for all entered new meetings
        {
                if((meeting[m].getInitiator()).equals(chatName))
                if(! meeting[m].scheduled)
                for(int d=0;d<meeting[m].messageArray.length;d++)//loop for the
domain of this new meeting
                {
                        if((meeting[m].messageArray[d][0]).equals(date1))
                        {
                                for(int a=0;a<meeting[m].whisperingToMany.length;a++)
                                {

        if((meeting[m].whisperingToMany[a][0]).equals(att1))
                                        meeting[m].messageArray[d][1]="no";
                                }



                        }

                }
        }
    }
  public void arc_consistency(String date1)
    {
```

```
        for(int m=0;m<=no;m++)//loop for all entered new meetings
        {
                if((meeting[m].getInitiator()).equals(chatName))
                if(! meeting[m].scheduled)
                for(int d=0;d<meeting[m].messageArray.length;d++)//loop for the
domain of this new meeting
                {


                        if((meeting[m].messageArray[d][0]).equals(date1))
                        {

                                meeting[m].messageArray[d][1]="nos";

                        }

                }
        }
    }

    public void clearMeetings()
 {

        no1=-1;
        System.out.println("clear meetings ="+no);
         for(int g=0;g<=no;g++)
          {

          if((meeting[g].meeting_constraint).equals("constraint"))
                {
                        no1++;

                }
                else
                {
                        meeting[g].assignment="";
                                meeting[g].violationAssignment=100;
                                meeting[g].rank=-1;
                                meeting[g].violation=0;
                                meeting[g].violation_c=0;
                                meeting[g].violation_no=0;
                                meeting[g].violation_no_c=0;
                                meeting[g].dont_loop=0;
                                meeting[g].index=0;
                                meeting[g].scheduled=false;

                }
      }
```

```
  }



double round(double value, int decimalPlace) {
double power_of_ten = 1;
while (decimalPlace-- > 0)
  power_of_ten *= 10.0;
return Math.round(value * power_of_ten)
  / power_of_ten;
}


String getInput()
{

      String text = input.getText();
      if (!text.equals(""))
      if (text.charAt(text.length() - 1) != '\n') text = text + "\n";
      input.setText("");
      input.requestFocus();
      return text;
}



class checkBoxItemListener implements ItemListener
{
      public void itemStateChanged (ItemEvent e)
      {

            //JCheckBox s=new JCheckBox();

            for(int y=0;y<user_no;y++)
            {

                  if (e.getSource()==usersCheckBox[y])

                        if(e.getStateChange()==ItemEvent.SELECTED)
                        {


      receiverName[attendeesNo][0]=usersCheckBox[y].getText();
```

```
        receiverName[attendeesNo][1]=usersRankTextField[y].getText();
                                attendeesNo++;
                                }
                                else
                                {
                                        for(int h=0;h<attendeesNo;h++)
                                        {

        if((usersCheckBox[y].getText().equals(receiverName[h][0])))
                                                {
                                                        for(int o=y;o<attendeesNo;o++)
                                                        {

        receiverName[o][0]=receiverName[o+1][0];

        receiverName[o][1]=receiverName[o+1][1];

                                                        }
                                                        attendeesNo--;
                                                        user_rank_sum=user_rank_sum-
Integer.parseInt(receiverName[attendeesNo][1]);

                                                }}}}}

        }
  class Lis implements PropertyChangeListener {
        public void propertyChange(PropertyChangeEvent e) {
                        java.util.Calendar c = mycalendar1.getCalendar();
                        domain[iii][0]=(c.getTime().toString()).substring(4,10);

        domain[iii][1]="yes";//System.out.println("domain[iii]="+domain[iii]);
                        iii++;


        }
}


  class Lis1 implements PropertyChangeListener {
        public void propertyChange(PropertyChangeEvent e) {
                        java.util.Calendar c = mycalendar1.getCalendar();
                        domain[0][0]=(c.getTime().toString()).substring(4,10);
                        domain[0][1]="yes";
                        System.out.println("domain="+domain[0][0]);

                        receiverName=new String[1][2];
                receiverName[0][0]=chatName;
```

```
                    receiverName[0][1]="1.0";
                              no++;

                              meeting[no] = new Shared0(chatName, domain,
listening,1,"constraint");
                              meeting[no].setName(chatName+no);
                              meeting[no].setWhisperingToMany(receiverName);
                              meeting[no].setMessageType("proposal");
                              meeting[no].dont_loop=0;
                              meeting[no].index=0;
                              meeting[no].violation=0;
                              meeting[no].violation_c=0;
                              meeting[no].violation_no=0;
                              meeting[no].violation_no_c=0;








}
  }
}

  /**
   * Creates the view object first and then the client.
   * Will not allow user names longer then 10 characters.
   */
  public  static void main (String[] args) {

    host = args[0];
    chatName = args[1];
    if (chatName.length() > 10)
    {
      System.out.println("Shorter name required. Please try again.");
      System.exit(1);
    }
    view = new View();
    try
    {
      client0 = new Client0(view);
    }
    catch (RemoteException e) {  }
  }
}


import java.util.*;
```

```
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.*;
import CalendarBean.*;
import CalendarBean.JCalendar.*;
import java.beans.*;
import java.util.Collections;






public class superAgentLGP extends UnicastRemoteObject  implements MessageClient
 {
  public static String host;
  public static String chatName;
  public static MessageServer server;
  public static int listening = 1;
  public static superAgentLGP superagentLGP;
  public static View view;
  public static Vector users = new Vector();
  public static Shared[] meeting=new Shared[100];
  public static int no=-1;
  public static int no1=-1;
  public static Vector heu_c=new Vector();
  public static Vector heu_s=new Vector();

  public static Vector [] parent=new Vector[2];



  public static Vector [] children=new Vector[4];

  public static double [] parent_v=new double[2];

  public static double [] children_v=new double[4];
  public static String smalleragentname;

  superAgentLGP(View view) throws RemoteException
  {
```

```
  try
  {
    this.view = view;
    Registry registry = LocateRegistry.getRegistry(host);
    server = (MessageServer) registry.lookup(MessageServer.REGISTRY_NAME);
    System.out.println("Registering with server...");
    server.register(this);
    System.out.println("Registration complete");
  }
  catch (Exception e) { }
}

public void run()
{
        System.out.println("runjnnnnnnnnnnnnnnnnnnnnn");
}

/*
 Allow the server to send the client a Shared message object.
 The method then passes the message to the update() method.
 */
public void sendMessage(Shared msg)
{
  view.update(msg);
}


/*
  Allows the server to retrieve the user name for this
  specific user.
 */
public String getUserName()
{
  return chatName;
}


/*.
 Allows the server to retrive the state of whether this
 client is, or is not listening.
 */

public int getListening()
{
  return listening;
}
```

```
static class  View extends JFrame {


        JMenuBar bar=new JMenuBar();
        JMenuBar bar1=new JMenuBar();
        JMenuBar calBar=new JMenuBar();
        JMenu meetingMenu=new JMenu("Linear Genetic Programming Scheduling ");
        JMenuItem addMeetingMenuItem =new JMenuItem("Add Meeting");
        JMenuItem schedMeetingMenuItem =new JMenuItem("Generate");
        JMenuItem viewMeetingMenuItem =new JMenuItem("View Meeting");

        JMenu advancedSchedMenu=new JMenu("Advanced Schedualing");
        JMenuItem localSearchMenuItem=new JMenuItem("Local Search Repair");

        JMenu finishAddAttendeesMenu=new JMenu("Meeting Properties");
        JMenuItem viewAttendeesMenuItem=new JMenuItem("Add Attendees");
        JMenuItem addDomainMenuItem=new JMenuItem("Add Domain");
//      JMenuItem finishMenuItem=new JMenuItem("Back");

        JButton finishAddDomainButton=new JButton("Finish Domain");
        JButton finishAddAttendeesButton=new JButton("Finish Attendees");

        JCheckBox [] usersCheckBox=new JCheckBox[100];
        JTextField [] usersRankTextField=new JTextField[100];
        int user_no=0;
        int attendeesNo=0;
        int user_rank_sum=0;



        String [] domain=new String [10];
        int iii;//***********************************************
        String [][] receiverName=new String[100][2];
        String [][]solution;
        String [] solution_index;
        String[][] max_violation;
        int max_v_l;
        double tot=0,tot_max_v=0,tot_max_v1=0;
        int find;
        //int [] index_max_violation;
        double [] rankReceiver;
  Lis s=new Lis();
  checkBoxItemListener checkBoxHandler=new checkBoxItemListener();
  private static final int transcriptRows = 10;
  private static final int transcriptColumns = 30;
  private static final int inputRows = 10;
  private static final int inputColumns = 30;
  private String[] whisperingtoMany=new String[0],copyws=new String[0];
```

```
   private String[][] whisperingToMany_r=new
String[0][0],whisperingToMany_r1=new String[0][0],whisperingToMany_r2=new
String[0][0];

 //  double violation=0,violation_no=0,violation_c=0,violation_no_c=0;
   int numberAttendees;


   private JCalendar mycalendar1;
   private JTextArea transcript = new JTextArea(transcriptRows, transcriptColumns);
   private JTextArea input = new JTextArea(inputRows, inputColumns);
   private JTextField roomCount = new JTextField(3);
   private JTextField sendToField = new JTextField("Attendees");
   private JTextField rankSendToField = new JTextField("Ranks");

   JScrollPane scrollPane;
   private JLabel nameLabel = new JLabel("You are currently logged on as "
                                        + chatName + ".");
   private JLabel roomCountLabel = new JLabel("Current number of users: ");

   private JPanel namePanel = new JPanel();

   private JPanel infoPanel = new JPanel();

   private JPanel panel = new JPanel();
   private JPanel panel1 = new JPanel();

   private JPanel [] panel22;
   private JPanel panel21 = new JPanel();
   private JPanel panel2 = new JPanel();

   private JPanel calPanel = new JPanel();

   private JPanel buttonPanel1 = new JPanel();

   private JPanel buttonPanel2 = new JPanel();


   View () {
     super("superAgentLGP Meeting Scheduling ");

     transcript.setEditable(false);
     roomCount.setEditable(false);
     transcript.setLineWrap(true);



     mycalendar1= new JCalendar();
```

```
        mycalendar1.setFont(new Font("Dialog", Font.BOLD, 10));

        setJMenuBar(bar);

      //  meetingMenu.add(addMeetingMenuItem);
        meetingMenu.add(schedMeetingMenuItem);
      //  meetingMenu.add(viewMeetingMenuItem);
        bar.add(meetingMenu);

        advancedSchedMenu.add(localSearchMenuItem);
        //bar.add(advancedSchedMenu);

        finishAddAttendeesMenu.add(viewAttendeesMenuItem);
        finishAddAttendeesMenu.add(addDomainMenuItem);
        //finishAddAttendeesMenu.add(finishMenuItem);
        bar1.add(finishAddAttendeesMenu);

buttonPanel1.setBackground(Color.yellow);
buttonPanel2.setBackground(Color.red);


localSearchMenuItem.setToolTipText("Local search to find better solution " );

final JDesktopPane theDesktop=new JDesktopPane();
getContentPane().add(theDesktop);
namePanel.add(nameLabel);
infoPanel.add(roomCountLabel);
infoPanel.add(roomCount);
buttonPanel1.setLayout(new FlowLayout());


final Component[] components =
{

      };

addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
          Shared msg = new Shared(chatName, "", listening);
          try {
            server.deregister(msg);
          } catch (RemoteException f) { }
          finally { System.exit(0); }
        }
      });
```

```
viewAttendeesMenuItem.addActionListener(new ActionListener() {
 public void actionPerformed(ActionEvent e) {


setContentPane(panel2);

pack();

}
});









schedMeetingMenuItem.addActionListener(new ActionListener()
{
 public void actionPerformed(ActionEvent e)
 {

int crossover_point=0;

no=meeting.length;

Shared msg_emty=new Shared ();

msg_emty.setMessageType("empty");

Enumeration en = users.elements();

String [][] rn=new String[(users.size())-1][2];

Object[] currUserInfo;
int i =0;

while(en.hasMoreElements())
{
  currUserInfo =  (Object[]) en.nextElement();
 try{

   if (!currUserInfo[0].equals("lsp") &&!currUserInfo[0].equals("lgp"))
```

```
                        {
                                rn[i][0]=(String)currUserInfo[0];
                                rn[i][1]="1";
                                i++;
                        }

                msg_emty.setWhisperingToMany(rn);

        }//t

                catch(Exception ee)
                        {System.out.println("problem in LSP scheduling EMPTY");}

         }




  for(int a1=0;a1<2;a1++)
  {


        sendTo(msg_emty);

        clearMeetings();


          for(int ii=0;ii<parent[a1].size();ii++)
                {


                Object o=parent[a1].get(ii);
                String os=(String)o;
                if((os).equals("rs"))
                {

                        findMeetingsRankandschedulingProcess(no);//(2-
2)******************************************call the ranking function in
order to ranks the meetings
                }
                else
                {

                        schedulingProcess();
                }

                }
```

```
            tot= 0;
                 for(int i1=0;i1<no;i1++)
          {
                 if(meeting[i1].violationAssignment!=100)
                 {

                 if((meeting[i1].getInitiator()).equals(chatName))
                 {

                 tot=tot+(meeting[i1].violationAssignment);

                 tot=round(tot,2);
                 }

                 }
          }

          parent_v[a1]=tot;
   System.out.println("violation parent "+a1+"="+tot);
     }





  Vector solution_h =new Vector();
  double solution_h_v;
  if(parent_v[0]<parent_v[1])
   {
   //     System.out.println("(parent_v[0]<parent_v[1]
          solution_h=(Vector)parent[0].clone();");

          solution_h=(Vector)parent[0].clone();
          solution_h_v=parent_v[0];

   }
   else
   {
   //             System.out.println("(parent_v[0]>parent_v[1]
          solution_h=(Vector)parent[1].clone();");
          solution_h=(Vector)parent[1].clone();
          solution_h_v=parent_v[1];

          parent[1]=(Vector)parent[0].clone();
          parent_v[1]=parent_v[0];
```

```
        parent[0]=(Vector)solution_h.clone();
        parent_v[0]=solution_h_v;

  }




   for(int i2=0;i2<children.length;i2++)
{
        children[i2]=new Vector();
}

int rounds=0;
int q=0;
boolean sameparent=false;

while((solution_h_v>0)&&(rounds<50))//&&(!sameparent))
{


rounds++;
children[0]=(Vector)parent[0].clone();
children_v[0]=parent_v[0];

children[1]=(Vector)parent[1].clone();
children_v[1]=parent_v[1];

children[2]=(Vector)parent[0].clone();
children[3]=(Vector)parent[1].clone();


crossover_point=children[2].size()/2;
System.out.println("");
System.out.println("crossover in point =" +crossover_point);
System.out.println("");




for(int i4=crossover_point+1;i4<children[2].size();i4++)
{

        children[2].set(i4,parent[1].get(i4));
        children[3].set(i4,parent[0].get(i4));

}
```

```
Object o1=new Object();
Object o2=new Object();
 for(int u=2;u>=1;u--)
 {

        o1=children[2].get(children[2].size()-u);
        children[2].remove(children[2].size()-u);

        children[2].insertElementAt(o1,u);

        o2=children[3].get(children[3].size()-u);
        children[3].remove(children[3].size()-u);

        children[3].insertElementAt(o2,u);

 }



System.out.println("");
System.out.println("***************Round number ="+rounds);
System.out.println("");
for(int a1=2;a1<4;a1++)
{

        System.out.println("child no "+a1+" is the following heuristic");
        System.out.println("");



        sendTo(msg_emty);

        clearMeetings();


          for(int ii=0;ii<children[a1].size();ii++)
                {
                        System.out.print("     ("+(ii+1)+")  ");

                Object o=children[a1].get(ii);
                String os=(String)o;
                if((os).equals("rs"))
                {
                  System.out.println(" findMeetingsRank");
                  System.out.println("           schedulingProcess");
                  System.out.println("");
```

```
                    findMeetingsRankandschedulingProcess(no);//(2-
2)*****************************************call the ranking function in
order to ranks the meetings
               }
               else
               {
                    System.out.println(" schedulingProcess");
                 System.out.println("");
                    schedulingProcess();
               }

               }


          tot= 0;
               for(int i1=0;i1<no;i1++)
          {
               if(meeting[i1].violationAssignment!=100)
               {

               if((meeting[i1].getInitiator()).equals(chatName))
               {

               tot=tot+(meeting[i1].violationAssignment);

               tot=round(tot,2);
               }

               }
          }

          children_v[a1]=tot;

     }




  sameparent=true;
  boolean anychangeonparent=false;
   for(int h1=2;h1<4;h1++)
   {

               if(children_v[h1]<=parent_v[1])
          {
               sameparent=false;
                    anychangeonparent=true;
```

```
                if(children_v[h1]<parent_v[0])
                {

                        parent[1]=(Vector)parent[0].clone();
                                parent_v[1]=parent_v[0];

                                parent[0]=(Vector)children[h1].clone();
                                parent_v[0]=children_v[h1];


                                solution_h=(Vector)children[h1].clone();
                                solution_h_v=children_v[h1];

                        }
                  else
                  {
                        parent[1]=(Vector)children[h1].clone();
                  parent_v[1]=children_v[h1];

                  }

        }


  }



if(!anychangeonparent)// no change on parent
{

        System.out.println("children have no less violation");
        if(children_v[2]<=children_v[3])
        {

                parent[1]=(Vector)children[2].clone();
        parent_v[1]=children_v[2];

        }
        else
        {
                parent[1]=(Vector)children[3].clone();
        parent_v[1]=children_v[3];

        }
}
```

```
            for(int j=0;j<4;j++)
            {
                  System.out.println("");
                  System.out.println("violation for child no"+j+"  is equals==
"+children_v[j]);
            }





              }    /***********************/
        /**********************/

  System.out.println("");
  System.out.println("***************at the end the total Rounds  ="+rounds);
  System.out.println("");
  sendTo(msg_emty);
  clearMeetings();
  System.out.println("");
  System.out.println("the solution heuristic/ algorithm is");
  System.out.println("");
  for(int o=0;o<solution_h.size();o++)
  {
        System.out.print("     ("+(o+1)+")  ");
              if((solution_h.get(o)).equals("s"))
                    {
                          System.out.println("        schedulingProcess");
                          System.out.println("");
                    }

              else if((solution_h.get(o)).equals("rs"))
              {

                    System.out.println("        findMeetingsRankand");
                    System.out.println("                schedulingProcess");
                    System.out.println("");
               }

}

              Shared msg_heu1=new Shared(chatName,
smalleragentname,solution_h,listening);
               msg_heu1.setMessageType("msg_type_s_h");
               sendTo(msg_heu1);
```

```
                    System.out.println("this  heuristic has been sent to the smaller agents");

                    Shared msg_lsp1=new
Shared(chatName,smalleragentname,meeting,listening,no);

                    msg_lsp1.setMessageType("msg_type_lsp1");


                    sendTo(msg_lsp1);


                }
               }
              );







    /*
     * This section deals with action events from the "Who's Here" button.
     */



    JPanel panel = new JPanel();
    panel.add(SwingUtil.vBox(components, SwingUtil.CENTER));
    setContentPane(panel);
    //panel.setBackground(Color.blue);
    pack();
    setVisible(true);
    input.requestFocus();
    input.setLineWrap(true);
    setResizable(true);
  }






  public Dimension getPreferredSize() {
    return (new Dimension(400, 800));
  }
```

```
public void schedualing(Shared meeting )
{



        int l=copyws.length;


                try
                {

                        String proposal = meeting.messageArray[meeting.index];

//                      System.out.println(meeting.getName());

                        meeting.setMessage(proposal);


                        sendTo(meeting);


                }

                catch(Exception e)
                {
                        transcript.append("HOST: " + meeting.getInitiator() +
                        "- no domain tooooooooooo send. try again.\n");
                }



}



public void sendTo(Shared msg)
{

    //System.out.println("SEND To");
    String from = msg.getInitiator();



    msg.setUsers(users);
    Enumeration en = users.elements();
    Object[] currUserInfo;

    while(en.hasMoreElements())
```

```
        {
                currUserInfo =  (Object[]) en.nextElement();
    try{

                for(int i=0;i<msg.whisperingToMany.length;i++)

                {

                        if (currUserInfo[0].equals(msg.whisperingToMany[i][0]) )

                        try
                        {
                                ((MessageClient) currUserInfo[2]).sendMessage(msg);
                        }
                                catch (RemoteException e) { System.out.println("error
send to");}

                        }
                }//try
                        catch(Exception e){System.out.println("widesprd exception");}
        }
        }

         /*
   * Allows clients to public messages that will be sent to every
   * person in the room that is listening at that time.
   */
        public void sendAll(Shared msg)
        {
        Enumeration en= users.elements();
        Object[] currUserInfo;
           msg.setUsers(users);

        while(en.hasMoreElements())
        {

                currUserInfo =  (Object[]) en.nextElement();

                if (((Integer) currUserInfo[1]).intValue() == 1)

                {
                try
                        {
                        ((MessageClient) currUserInfo[2]).sendMessage(msg);

                        }

                catch (RemoteException e)
```

```
                    {
                    }
               }
          }
          }


    /**
     * Appends all incoming "chat" messages (not "state" messages)
     * to the transcript window, and updates this client with the
     * current chat room user information.
     */
    public void update(Shared msg)
    {



          //System.out.println("UPDATE ");

    try{
          if ((listening == 1) || msg.getWhispering())
               if((msg.msgType).equals("confirm"))
          transcript.append(msg.getInitiator() + " sends a CONFIRMATION for meeting:
"+msg.getName()+"   " + msg.getMessage()+"\n");
          else if((msg.msgType).equals("proposal"))
          transcript.append(msg.getInitiator() +" sends a PROPOSAL for meeting:  "+
msg.getName()+"   "+ msg.getMessage()+"\n");
          roomCount.setText((String) msg.getRoomSize());
          users = (Vector) msg.getUsers();
     }
     catch(Exception e){System.out.println("listtttttttttttttning");}

          try{

          if((msg.msgType).equals("msg_type_c_h"))
          {
               smalleragentname=msg.getInitiator();
               heu_c=msg.heuristic;
               heu_s=(Vector)heu_c.clone();



               heu_c.set(1,heu_c.get(0));

               heu_s.set(heu_s.size()-1,heu_s.get(0));
          Collections.reverse(heu_s);
       for(int iii=0;iii<2;iii++)
       {
          parent[iii]=new Vector();
```

```
        }
          parent[0]=heu_c;
                  parent[1]=heu_s;



                  System.out.println("");

                  System.out.println("parent[1]-the smaller agent heuristic- is the
following heuristic");
                  System.out.println("");

                  for(int y=0;y<parent[0].size();y++)
                  {
                          System.out.print("     ("+(y+1)+")  ");
                          if((parent[0].get(y)).equals("s"))
                          {
                                  System.out.println("     schedulingProcess");
                                  System.out.println("");
                          }

                  else if((parent[0].get(y)).equals("rs"))
                  {

                          System.out.println("    findMeetingsRank");
                          System.out.println("               schedulingProcess");
                          System.out.println("");
                    }
                  }
                  System.out.println("");

                  System.out.println("parent[2]-the reverse of parent[1]- is the following
heuristic");
                  System.out.println("");

                          for(int y=0;y<parent[1].size();y++)
                          {
                                  System.out.print("     ("+(y+1)+")  ");
                          if((parent[1].get(y)).equals("s"))
                          {
                                  System.out.println("     schedulingProcess");
                                  System.out.println("");
                          }

                  else if((parent[1].get(y)).equals("rs"))
                  {

                          System.out.println("     findMeetingsRankand");
```

```
                    System.out.println("            schedulingProcess");
                    System.out.println("");
                }
            }

    System.out.println("");
    System.out.println("");


        /*********************/
        /*********************/





        }

        else if((msg.msgType).equals("msg_type_lsp"))
        {
                    meeting=new Shared[msg.meetings.length];
                    for(int s=0;s<msg.meetings.length;s++)
                    {


                    meeting[s]=msg.meetings[s];

                    meeting[s].setInitiator(chatName);

        //JOptionPane.showMessageDialog(null,"meeting==="+meeting[s].getName()+
"  initiator  "+meeting[s].getInitiator());

                    }

        }


    else        if((msg.msgType).equals("proposal"))
        {

                    reply(msg);

        }


        else if((msg.msgType).equals("reply"))
        {
```

```
//      System.out.println("received reply");

                int i=0;
                try{

                        for(i=0;i<=no;i++)//find this rply for which meeting
                        {
                //
        System.out.println("meeting[i].getName()="+meeting[i].getName()+"
msg.getName()="+msg.getName());
                                if((meeting[i].getName()).equals(msg.getName()))
                                {
                                //System.out.println("trueee and
meeting[i].violation_no="+meeting[i].violation_no);

                                        meeting[i].violation_no++;// we received one rply


        //System.out.println("meeting[i].violation_no="+meeting[i].violation_no);

                                                for(int
k=0;k<meeting[i].whisperingToMany.length;k++)
                                                {

        //System.out.println("meeting[i].whisperingToMany[k][0]="+meeting[i].whispe
ringToMany[k][0]+"    msg.getInitiator()"+msg.getInitiator());
                                                        if
((meeting[i].whisperingToMany[k][0].equals(msg.getInitiator())))
                                                        {
                                                //      System.out.println("the old
violation"+meeting[i].violation);

        meeting[i].violation=meeting[i].violation+(Double.parseDouble(meeting[i].whis
peringToMany[k][1]))*Double.parseDouble(msg.getMessage());
                                                                //System.out.println("
proposal  "+meeting[i].getMessage()+"  received rply for prop from "+msg.getInitiator()
                                                        //      +"  rank for the
replier"+meeting[i].whisperingToMany[k][1]+" so violation="+meeting[i].violation);
                                                                break;
                                                                }
                                                        }

                                        //}

                                break;//we found for which this reply is
                                }
                        }
                }
```

```
                    catch(Exception e)
                    {
                            //System.out.println("error in finding to which proposal this
reply");
                    }

                    try{


        //System.out.println("meeting[i].violation_no="+meeting[i].violation_no+"
meeting[i].whisperingToMany.length="+meeting[i].whisperingToMany.length);

        if(meeting[i].violation_no==meeting[i].whisperingToMany.length)//all replies
have been received
                            {
                                    //System.out.println("true");

                                    try
                                    {


                                            if(meeting[i].violation==0)
                                            {

        //System.out.println("meeting[i].violation==0");

        meeting[i].setAssignment(meeting[i].getMessage());
                                                    meeting[i].violationAssignment=0;


                                            }

                                            else
                                            {
                                            //      System.out.println("violatiojn/violation
no="+l);
                                                    if
(meeting[i].violationAssignment>=meeting[i].violation)
                                                    {

        meeting[i].setAssignment(meeting[i].getMessage());

        meeting[i].violationAssignment=meeting[i].violation;

        meeting[i].violationAssignment=round(meeting[i].violationAssignment,2);

                                                            //System.out.println("meeting "+
meeting[i].getName()+"    propose schedualed so rank =");
```

```
                                        }

                                        meeting[i].index++;

                                        //System.out.println("iii="+iii);

        if(meeting[i].index<(meeting[i].messageArray.length))
                                        {
                                                meeting[i].violation=0;
                                                meeting[i].violation_no=0;
                                                schedualing(meeting[i]);

                                        }

                                }

                        }
                        catch(Exception e)
                        {
                                System.out.println("4444444444444444");
                        }

                        try
                        {
                //System.out.println(meeting[i].dont_loop);
                        if(meeting[i].dont_loop==0)
                                {
                                        //System.out.println("trueeeee");
                                        Shared conf_msg=new
Shared(chatName,meeting[i].getAssignment(),listening);

                                        conf_msg.setName(meeting[i].getName());

                                        conf_msg.setMessageType("confirm");


        conf_msg.setWhisperingToMany(meeting[i].whisperingToMany);



                        //              meeting[i].violation_no=0;
                                        meeting[i].dont_loop++;
                        //
        if((meeting[i].violationAssignment>0))//(1-
6)++++++++++++++++++++++++++++++++++++not sced violated meeting
```

```
                                        //                    {//(2-
6)+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++n
ot sced violated meeting
                                        //                              meeting[i].setAssignment(" can
not");//(3-6)++++++++++++++++++++++++++++++++++not sced violated meeting
                                        //
          meeting[i].violationAssignment=0;//(4-
6)+++++++++++++++++++++++++++++++++++++not sced violated meeting

                                        //                    }//(5-
6)+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++n
ot sced violated meeting
                                        //                    else
if((meeting[i].violationAssignment==0))//(6-
6)+++++++++++++++++++++++++++++++not sced violated meeting
                                                              sendTo(conf_msg);
                                        //                    no1++;
                                                    }
                                        }

                                        catch(Exception e)
                                        {
                                                    System.out.println("8888888888");
                                        }

                              }

                    }
                    catch(Exception e)
                    {
                              System.out.println("222222222222222222");
                    }
          //        System.out.println("hiiiiiiiiiii");

          }
          else if ((msg.msgType).equals("confirm"))
          {


                    boolean found=false;
                    try
                    {
                    // search in the meeting if this confirmed meeting is exist
                              int i;
                              for(i=0;i<=no;i++)
                              {
                              //System.out.println("my meetings are no"+i+" its
name"+meeting[i].getName());
```

```
                    if ((msg.getName()).equals(meeting[i].getName()))
                    {

        //System.out.println("++++++++++++++++++name for this meeting
is"+msg.getName());
                        meeting[i].setAssignment(msg.getMessage());
                        meeting[i].violationAssignment=0;
                        meeting[i].attend=true;
                        found=true;
                        //      System.out.println("meeting name
"+meeting[i].getName()+"violation assignmt     "+meeting[i].violationAssignment);
                        break;
                    }
                }

                //this meeting is not exist then add it
                if (!found)
                {       //System.out.println("this meeting is not exist i will add it
i have meeting number"+no);

                    no++;
                    no1++;
                    meeting[no]=new
Shared(msg.getInitiator(),msg.getMessage(),listening);
                    meeting[no].setName(msg.getName());
                    meeting[no].setAssignment(msg.getMessage());
                    meeting[no].violationAssignment=0;
                    meeting[no].attend=true;
                //      System.out.println("now ihave meeting no "+no+" and the
assgment"+meeting[no].getAssignment());


                }
            reply(msg);
            }
            catch(Exception e)
            {
                System.out.println(" can not confirm this meeting");
            }
        }




            else if ((msg.msgType).equals("confirm_delete"))
        {
```

```
try
{
System.out.println("CONFIRM DELETE  to"+chatName+"      regarding
meeting"+msg.getName());

reply(msg);
}
catch(Exception e)
{
        System.out.println(" can not confirm this meeting");
}
}



else if ((msg.msgType).equals("confirm_reply"))
{

//System.out.println("msg reply for confirm received
from"+msg.getInitiator()+"regarding meeting"+msg.getName()+"with
value"+msg.getMessage());
int i=0;
try{
        for( i=0;i<=no;i++)//find this rply for which meeting
        {

                if((meeting[i].getName()).equals(msg.getName()))
                {
                        meeting[i].violation_no_c++;// we received one
rply


                                for(int
k=0;k<meeting[i].whisperingToMany.length;k++)
                                {
                                        if
((meeting[i].whisperingToMany[k][0].equals(msg.getInitiator())))//find sender
                                        {
                                        //System.out.println("msg
received"+msg.getMessage()+" from "+msg.getInitiator()+"regarding meeting
"+meeting[i].getName()+"
rank"+(Double.parseDouble(meeting[i].whisperingToMany[k][1]))+"old violation
"+meeting[i].violationAssignment+"violation_no_c="+meeting[i].violation_no_c+"
whidspreadtomany="+meeting[i].whisperingToMany.length);

        meeting[i].violation_c=meeting[i].violation_c+(Double.parseDouble(meeting[i].
whisperingToMany[k][1]))*(Double.parseDouble(msg.getMessage()));//take sender
rank
```

```
            meeting[i].violation_c=round(meeting[i].violation_c,2);

            //System.out.println("===========new violation_c"+meeting[i].violation_c);
                                                                    break;
                                                                    }
                                                            }


                                    //          }

                                    break;//we found for which this reply is
                                    }
                            }
                    }
                    catch(Exception e)
                    {
                            System.out.println("||||||||||||||||||||||error in finding to which
confirmation this reply");
                    }


            if(meeting[i].violation_no_c==meeting[i].whisperingToMany.length)//all replies
have been received
                            {


                                    try
                                    {

            meeting[i].violationAssignment=meeting[i].violation_c;

            meeting[i].violationAssignment=round(meeting[i].violationAssignment,2);
                                            //System.out.println("++++++++++++meeting
"+meeting[i].getName()+"assigned
to"+meeting[i].getAssignment()+"violation="+meeting[i].violationAssignment);
                                            meeting[i].violation_c=0;
                                            meeting[i].violation_no_c=0;

            //System.out.println("&&&&&&&&&&&&&meeting[i].scheduled="+meeting[
i].scheduled);
                                            meeting[i].scheduled=true;

            //System.out.println("^^^^^^^^^^^^^^^^^^^^^^^meeting[i].scheduled="+meeti
ng[i].scheduled);
                                    }
                                    catch(Exception e)
                                    {
```

```
                                        System.out.println("error in violation calculation
");
                                }

                        }

                        else
                        {

                        }

    }

        else if ((msg.msgType).equals("confirm_reply_update"))
          {
          //System.out.println("msg confirm reply update  received
from"+msg.getInitiator()+"regarding meeting"+msg.getName()+"  with
value"+msg.getMessage());
          int i=0;
          try{
                //System.out.println(chatName+"received update confirm for
meeting"+msg.getName());
                        for( i=0;i<=no;i++)//find this rply for which meeting
                        {
                        //System.out.println("existed meetings :
"+meeting[i].getName());
                                if((meeting[i].getName()).equals(msg.getName()))
                                {
                                        //System.out.println("meeting found");
                                        for(int
k=0;k<meeting[i].whisperingToMany.length;k++)
                                        {
                                                if
((meeting[i].whisperingToMany[k][0].equals(msg.getInitiator())))//find sender
                                                {
                                        //System.out.println("msg received confirm update
for"+meeting[i].getName()
                                        //          +"from
"+msg.getInitiator()+"value"+msg.getMessage()
//                                              +"
rank"+(Double.parseDouble(meeting[i].whisperingToMany[k][1]))+"old violation
"+meeting[i].violationAssignment);



          meeting[i].violationAssignment=(meeting[i].violationAssignment)
```

```
    +(Double.parseDouble(meeting[i].whisperingToMany[k][1]))*(Double.parseDo
uble(msg.getMessage()));
                                                    //take sender rank


    meeting[i].violationAssignment=round(meeting[i].violationAssignment,2);

    //System.out.println("############ meeting
name"+meeting[i].getName()+"new
violationAssignment"+meeting[i].violationAssignment);
                                                    break;
                                                    }
                                               }

                                               break;//we found for which this reply is

                              }


                         }

                    }

                    catch(Exception e)
                    {
                         System.out.println("error in finding to which confirmation this
reply");
                    }




  }




  else if((msg.msgType).equals("busy"))
  {
    double free=0;

    String [][]wsr=new String[1][2];
    wsr[0][0]=msg.getInitiator();
    wsr[0][1]="1";
```

```
        for(int i=0;i<msg.messageArray.length;i++)
        {
                for(int j=0;j<=no1;j++)
                {
                        if((meeting[j].getAssignment()).equals(msg.messageArray[i]))
                        {
                                free++;
                                break;
                        }
                }
        }

        free=free/msg.messageArray.length;

        Shared busyReply=new Shared(chatName,""+free,1);
//      System.out.println("*********free="+free);



        busyReply.setWhisperingToMany(wsr);

        busyReply.msgType="busy_reply";

        busyReply.setName(msg.getName());


        sendTo(busyReply);
    }

    else if((msg.msgType).equals("busy_reply"))
    {


        int i=0;
        try{
                        for( i=0;i<=no;i++)//find this rply for which meeting
                        {

                                if((meeting[i].getName()).equals(msg.getName()))
                                {
                                        meeting[i].busyReplyAll++;// we received one
rply


                                                for(int
k=0;k<meeting[i].whisperingToMany.length;k++)
```

```
                                        {
                                                if
((meeting[i].whisperingToMany[k][0].equals(msg.getInitiator())))//find sender
                                                {
                        //              System.out.println("msg
received"+msg.getMessage()+" from "+msg.getInitiator()+"regarding meeting
"+meeting[i].getName()+"
rank"+(Double.parseDouble(meeting[i].whisperingToMany[k][1]))+"old busy
"+meeting[i].busyReplyRank);

        meeting[i].busyReplyRank=meeting[i].busyReplyRank+((Double.parseDouble(
meeting[i].whisperingToMany[k][1]))*(round((Double.parseDouble(msg.getMessage())
),2))*10);//take sender rank

        meeting[i].busyReplyRank=round(meeting[i].busyReplyRank,2);
                                        //
        System.out.println("===========new
busyReplyRank"+meeting[i].busyReplyRank);
                                                        break;
                                                        }
                                                }


                        //      }

                        break;//we found for which this reply is
                        }
                }
          }
          catch(Exception e)
          {
                System.out.println("error in finding to which meeting this busy
reply");
          }



        if(meeting[i].busyReplyAll==meeting[i].whisperingToMany.length)//all replies
have been received
                {


                        try
                        {

        //System.out.println(""+meeting[i].getName()+"received all busy replies");
                                meeting[i].rank=meeting[i].busyReplyRank;
                                //meeting[i].rank=round(meeting[i].rank,2);
```

```
                                        //System.out.println("meeting
"+meeting[i].getName()+"ranked to"+meeting[i].rank);
                                        meeting[i].busyReplyAll=0;
                                        meeting[i].busyReplyRank=0;

                                }
                                catch(Exception e)
                                {
                                        System.out.println("error in rank calculation ");
                                }

                        }

                        else
                        {

                        }



    }

    }//try

    catch(Exception e)
    {
        System.out.println("erroor update");
    }
    }

    public void reply(Shared msg)
    {

                int r=0;

                whisperingToMany_r=new String[1][2];
                whisperingToMany_r1=new String[1][2];

                whisperingToMany_r[0][0]=msg.getInitiator();
                whisperingToMany_r[0][1]="1";


                int [] otherEffectedMeetings=new
int[no+1],otherEffectedMeetings1=new int[no+1];
                int effectedMeetings=0,effectedMeetings1=0;
                int k;
```

```
                //System.out.println("   check all meeting");
            for(int i=0;i<=no;i++)//search in all meetings
            {


            //System.out.println("meeting   "+meeting[i].getName()+" assigned to
"+meeting[i].getAssignment()
            //+"msg.getmessge  "+msg.getMessage());

            if ((meeting[i].getAssignment()).equals(msg.getMessage()))//if the
meeting has the same assignment
            {
                if(meeting[i].attend)//I am attend this meeting
                {
                    if((meeting[i].getName()).equals(msg.getName()))// it is
the same meeting
                    {
                        if ((msg.msgType).equals("confirm_delete"))
                        {

                        otherEffectedMeetings[effectedMeetings]=i;
                        effectedMeetings++;
                        }
                    //System.out.println("*** meeting
"+meeting[i].getName()+" has the same assignemnt"+meeting[i].getAssignment());
                    }
                    else// there is another meeting with this assignment
                    {


                        if((msg.msgType).equals("confirm")||
(msg.msgType).equals("confirm_delete"))
                        {

                        //System.out.println("meeting
"+meeting[i].getName()+" has the same assignemnt"+meeting[i].getAssignment());
                        otherEffectedMeetings[effectedMeetings]=i;
                        effectedMeetings++;
                        //System.out.println("effected meeting no
="+effectedMeetings+
//                      " name
"+meeting[otherEffectedMeetings[effectedMeetings]].getName());



                        }
                        r++;//how many meeting have at the same time
                    //  System.out.println("rrrrrrrrrrrrrrrrrrrrrrrr"+r);
```

```
                                        //          break;
                                        }
                                }
                        }
                        }
                        //  System.out.println("msg type "+msg.msgType+"r="+r);
                for(int t=0;t<effectedMeetings;t++)
                {
                        //System.out.println("EEEEEEEEEEEEEEEEeffected meeting"+
meeting[otherEffectedMeetings[t]].getName());
                }

   //     System.out.println("fish effected meetings");

        if((msg.msgType).equals("confirm")||(msg.msgType).equals("confirm_delete"))
                {

                                        if((msg.msgType).equals("confirm"))
                                        {
                                                for(int i=0;i<effectedMeetings;i++)
                                                {

                                                Shared msg2 = new Shared(chatName, "1",
listening);
                                                //System.out.println("msg.msgType =
"+msg.msgType);
                                                msg2.setMessage("1");



                                        msg2.setMessageType("confirm_reply_update");


        msg2.setName(meeting[otherEffectedMeetings[i]].getName());
                                        whisperingToMany_r1[0][0] =
meeting[otherEffectedMeetings[i]].getInitiator();
                                                whisperingToMany_r1[0][1]="1";
                                        msg2.setWhisperingToMany(whisperingToMany_r1);


                                        sendTo(msg2);
                                //          System.out.println("msg sent");
                                        }
                                }
                                else if((msg.msgType).equals("confirm_delete"))
                                        {
```

```
                                        if(effectedMeetings>1)
                                        {


                                             {

                                             for(int i=0;i<effectedMeetings;i++)
                                                  {

                                                  Shared msg2 = new
Shared(chatName, "-1", listening);
                                                  //System.out.println("msg.msgType
=  "+msg.msgType);
                                                  msg2.setMessage("-1");

     msg2.setMessageType("confirm_reply_update");


     msg2.setName(meeting[otherEffectedMeetings[i]].getName());

                                                  whisperingToMany_r1[0][0] =
meeting[otherEffectedMeetings[i]].getInitiator();

     whisperingToMany_r1[0][1]="1";

     msg2.setWhisperingToMany(whisperingToMany_r1);
                                                  sendTo(msg2);
               //        System.out.println("msg sent");
                                                  }
                                        }

                                   }

                              }

                    }

                    if ((msg.msgType).equals("confirm_delete"))
                    {}
                    else
                    {


                    Shared msg1 = new Shared(chatName, ((Integer)r).toString(), listening);

                    if((msg.msgType).equals("proposal"))
                    msg1.setMessageType("reply");
                    else
```

```
                msg1.setMessageType("confirm_reply");

                msg1.setName(msg.getName());
                msg1.setWhisperingToMany(whisperingToMany_r);
                sendTo(msg1);
                }


   }



        public void findMeetingsRankandschedulingProcess(int m)
        {

                Shared [] busy=new Shared[no];

                //System.out.println("1111111111");

                for(int a=0;a<no;a++)
                {

                        if(((meeting[a].getInitiator()).equals(chatName))&&
meeting[a].rank!=-2)// i am the initiator and not schedualed yet
                        {

                        meeting[a].busyReplyRank=0;

                        busy[a]=new
Shared(chatName,meeting[a].messageArray,1,meeting[a].messageArray.length,"meetin
g");

                        busy[a].setWhisperingToMany(meeting[a].whisperingToMany);

                        busy[a].setName(meeting[a].getName());

                        busy[a].msgType="busy";

                        sendTo(busy[a]);


                        }
                }
                schedulingProcess();
                //no=no-1;
        }

public void schedulingProcess()
```

```
{

                //      System.out.println("enter schedulingProcess");
                        int max_rank_meeting=0;

                //      System.out.println("no============="+no);
                for (int j = 0; j <no; j++)
        {
   //     System.out.println("j="+j+"   meeting[j].rank  ="+meeting[j].rank+"
meeting[j].messageArray.length="+meeting[j].messageArray.length);
   //     System.out.println("meeting[j].scheduled="+meeting[j].scheduled);

                        if( (meeting[j].getInitiator()).equals(chatName ))
                        {

                                if ((meeting[j].rank) >(meeting[max_rank_meeting
].rank))
                        {


                        max_rank_meeting = j;

                        }

                        else if (((meeting[j].rank) == (meeting[max_rank_meeting
].rank))&& (j)!=max_rank_meeting )

                        {


                                        if ((meeting[j].messageArray.length)
<(meeting[max_rank_meeting].messageArray.length))
                                {

                                        max_rank_meeting=j;




                                }
                        }
                        }

                }
```

```
                              if(meeting[max_rank_meeting].rank>=-1)
                              {
//              System.out.println("scheduling meeting
"+meeting[max_rank_meeting].getName()+"
Rank=="+meeting[max_rank_meeting].rank);
                              meeting[max_rank_meeting].rank=-2;
                    schedualing(meeting[max_rank_meeting]);




                    }
}

public void clearMeetings()
{

//          System.out.println("clear meetings ="+no);
           for(int g=0;g<no;g++)
            {
                     meeting[g].assignment="";
        meeting[g].violationAssignment=100;
        meeting[g].rank=-1;
        meeting[g].violation=0;
        meeting[g].violation_c=0;
        meeting[g].violation_no=0;
        meeting[g].violation_no_c=0;
        meeting[g].dont_loop=0;
        meeting[g].index=0;
        meeting[g].scheduled=false;
      }
}

   /*
    Returns the typed message from the input text field.

    */

   public String[] effect(Shared m)
   {String []e_meetings=new String[10];
   return e_meetings;
   }


   double round(double value, int decimalPlace) {
   double power_of_ten = 1;
```

```
while (decimalPlace-- > 0)
  power_of_ten *= 10.0;
return Math.round(value * power_of_ten)
  / power_of_ten;
}


String getInput()
{

       String text = input.getText();
       if (!text.equals(""))
       if (text.charAt(text.length() - 1) != '\n') text = text + "\n";
       input.setText("");
       input.requestFocus();
       return text;
}



class checkBoxItemListener implements ItemListener
{
       public void itemStateChanged (ItemEvent e)
       {


              JCheckBox s=new JCheckBox();

              for(int y=0;y<user_no;y++)
              {



                     if (e.getSource()==usersCheckBox[y])

                            if(e.getStateChange()==ItemEvent.SELECTED)
                            {



       receiverName[attendeesNo][0]=usersCheckBox[y].getText();


       receiverName[attendeesNo][1]=usersRankTextField[y].getText();
                                   attendeesNo++;
                            }
                            else
                            {
```

```
                                        for(int h=0;h<attendeesNo;h++)
                                        {

          if((usersCheckBox[y].getText().equals(receiverName[h][0])))
                                                {
                                                        for(int o=y;o<attendeesNo;o++)
                                                        {

          receiverName[o][0]=receiverName[o+1][0];

          receiverName[o][1]=receiverName[o+1][1];

                                                        }
                                                        attendeesNo--;
                                                        user_rank_sum=user_rank_sum-
Integer.parseInt(receiverName[attendeesNo][1]);


                                                }

                                        }

                                }
                        }

   }

          }
   class Lis implements PropertyChangeListener {
          public void propertyChange(PropertyChangeEvent e) {



                        java.util.Calendar c = mycalendar1.getCalendar();


                        domain[iii]=(c.getTime().toString()).substring(4,10);

                        //System.out.println("domain[iii]="+domain[iii]);
                        iii++;




          }
}
  }
```

```
/**
 * Creates the view object first and then the client.
 * Will not allow user names longer then 10 characters.
 */
public  static void main (String[] args) {

  host = args[0];
  chatName = args[1];
  if (chatName.length() > 10)
  {
    System.out.println("Shorter name required. Please try again.");
    System.exit(1);
  }
  view = new View();
  try
  {
    superagentLGP = new superAgentLGP(view);
  }
  catch (RemoteException e) { }
 }
}
```