An agent based co-operative preference model

Rashid Jayousi

Doctor of Philosophy

October 2003

Keele University

Abstract

Distributed problem solving is often characterised by multiple valid solutions, a solution being considered to be valid if it meets all the constraints. When there are multiple valid solutions, the user has a choice, which can be specified in terms of preferences on different desirable aspects (i.e. resources) of the solution. In that event, the best solution is the one that meets as many preferences as possible.

In this thesis, a multi-agent approach has been used for distributed problem-solving, each autonomous agent (task agent), under the supervision of a relevant task coordinator, solves its part of the subtask, in cooperation with other task agents. To resolve contention in the preferences for the same resources, a market-based payment scheme is applied for the preferences to be bought and sold by the contending task agents through their coordinators. The best solution is achieved for a task, when further iteration does not increase its total preference value, that is a convergence is achieved.

This thesis presents a preference model that includes a preference specification strategy, a preference processing technique, and a theoretical performance model, the latter describes the quantitative behaviour of the preference model. The thesis also presents a simulation study to show that the preference model works satisfactory and according to the theoretical performance model.

For the simulation study we used the problem of distributed scheduling in the manufacturing domain. The results of the study show that our agent based strategy not only reaches convergence on the final preference value for the whole system, that value is also independent of initial order of subtask allocation. The results verify the validity of our approach handsomely.

Contents

List of Figures

List of Tables

Acknowledgments

Thanks are mainly due to my supervisor Professor S. Misbah Deen for his active support and guidance throughout this research project. His insights into the research, his constant encouragement, and his continued academic and personal advice enabled me to complete this work, I am honoured to be one of his students.

I am grateful for Al-Quds university in Jerusalem for its financial support and its cooperation in enabling me to complete this research.

I would also like to thank the members of the Data and Knowledge Engineering group for their valuable contributions and advice throughout our discussions, namely Thomas Neligwa, Peter Granby, James Cole, Martyn Fletcher, Kapila Ponnamperuma, and Ryad Soobhany. The work would not have been possible without the assistance of the Computer Science Departments at Keele University who provided equipment, software and technical assistance.

I would like to thank Michael Alcock and James Cole for the time they have devoted to the task of proof-reading my thesis.

My special thanks to my parents, brother, sisters, other members of my family, and friends for their support and encouragement during my period of study at Keele University.

Most of all, I owe a huge debt of gratitude to my wife Suhad and daughters (Zenah, Farah, and Israa) who always gave me the strength and resolve I need.

Chapter 1

Introduction

In a distributed environment cooperative autonomous agents may work together to solve a joint task that often has a set of valid solutions. To help the cooperating agents in reaching a balanced decision, the choice of a solution can be expressed in the form of preferences on certain desired features of the solution. If the most preferred solution cannot be found, then the one that meets maximum amount of preferences can be accepted as the "best solution".

Therefore, associated with each task we identify a set of desired features, which can be resources or attributes. In this thesis we regard a resource as something that has a limited supply and can be consumed during the processing, whereas an attribute has no such limitation. The user requirement for preferences can vary widely depending on the application. For example, if the task is to schedule the manufacturing process of a product, the resources might be machine, time or labour, while attribute might be the colour of the product or the degree of polishing. The user may have preferences on a machine and on colour. If the task is the design of a web-page, we might think of the page size as a resource and the font-size as an attribute. The user may have preferences on the size of the web-page and the font-size as well as the layout of the web-page. If the resources are limited and multiple tasks are being executed concurrently, there will be inevitable contention. Contention can arise on desired resources but not on attributes. Resolving such contentions require not only negotiation and compromises among the agents responsible, but also enforcement of control for timely termination, avoiding explosive branches.

If preferences are allocated only on the currently available resources, then there will not be any conflict. If a desired resource is not available, then that preference cannot simply be met and some preferred solutions may not be feasible due to the non-availability of the preferred resources. If a resource has already been allocated to a task, and then if a later task arrives with a higher preference on that resource, then we have two possible approaches: either the previous allocations cannot be changed, or the previous allocations can be pre-empted. The first approach leads to a tree model where the allocation of a preference to a parent affects only the children, and hence it terminates in a finite time. The second case leads to a network model, where the allocation to a node (task) affects all other nodes (tasks) that have a preference on that resource. The cascading effect of pre-emption and reallocation leads to branch explosions. In this thesis we shall consider the second case, which is more challenging, and show how a set of agents can cooperate together to produce a solution that converges. An introductory version of this network model has been published by us in [DEEN02].

Thus in this thesis we address the problem of how to derive a preference-based solution in the presence of contention, where awarding preferences to the solution of one task can only be done by depriving/removing preferences from that of another in negotiation. Deprivation, and particularly removal, of preferences creates a high non-linearity leading to non-convergence. We propose a general preference model that produces a "reasonable" solution for trading and satisfying preferences and enforces timely termination in a "fairly competitive" market for cooperative agent-based scheduling.

As the preference model outlined above is highly non-linear, verification using available mathematical techniques is difficult. Therefore, based on the qualitative behaviour, of this model we also present in this thesis a theoretical performance model, which can predict the preference values that can be achieved by using our preference model.

To verify the validity of the theoretical model we performed a simulation study. In this study we have used the scenario of distributed scheduling, where a set of global tasks are resolved into subtasks by agents called Coordinators, and then these subtasks are allocated to Assembler agents through cooperation and negotiation, in which preferred resources are exchanged with payments. Agent-based systems support distributed scheduling, contrary to traditional centralised scheduling for manufacturing systems. We based our simulation for the multi-agent system on Cooperating Knowledge-Based Systems (CKBS) (see chapter 3).

1.1 Motivation

This work was inspired directly by our participation in the Holonic Manufacturing Systems (HMS) project (see section 2.4.2) and the current research activities in our DREAM (Dake Research into an Engineering Approach to Multi-agents) theme which has been applied to agent-based manufacturing and e-commerce and supply chain applications. Current research is focused on Cooperating Knowledge Based Systems (CKBS), where autonomous knowledge based systems, or agents, cooperate when solving problems in distributed applications. To solve a global task (joint task), the relevant agents normally form what we call a Cooperation Block (CB), where one agent acts as the coordinator and the other agents as cohorts (see section 3.3). The coordinator can be viewed as the user agent, since the user's requirements on the global task are expressed through this coordinator. Cooperation implies an enforceable agreement among the participating agents such that one agent can demand another to act within that agreement. Negotiation is the process by which agents agree on a mutually acceptable solution subject to constraints, meeting as many preferences as possible.

Our work in this thesis is an extension of the work on the contention-free (but not constraint free) solution that was published in [DEEN99]. This work is an extension and generalisation of these ideas to produce a generalised preference model for cooperative agent-based problem-solving and is not restricted to scheduling, even though scheduling offers most interesting examples for application.

1.2 Research objectives

The principal objective of this thesis is to develop a model for preference handling in a distributed agent based environment. In more specific terms we can state our objectives as follows:

1) To investigate the issues of preferences in task execution.
2) To study a mechanism for preference execution with convergence.
3) To develop an execution-cost model to control exploding branches.
4) To develop a theoretical performance model.
5) To implement a demonstrator to verify the approach and the validity of the theoretical model.

1.3 Thesis Structure

The remainder of the thesis consists of seven chapters. Chapter 2 is a literature review focusing on the current work on conflict resolution, the different applications that use preferences and current approaches in scheduling manufacturing systems.

Chapter 3 presents an overview of Cooperating Knowledge Based Systems (CKBS). This chapter describes the basic elements of the CKBS model, the cooperation environment and the operational architecture.

Chapter 4 presents our market-based approach to preference handling. It contains a detailed description of our preference and cost models. It describes the allocation process as well as the algorithm used for implementing the preference model among cooperating agents.

Chapter 5 presents a theoretical performance model. The model can roughly estimate the minimum remaining preference value. We study the effect of the clustering of requests for the same resource instances. The effect of the uniform distribution of these clusters on the developed formula is discussed first. Then we show how the formula is affected by the non uniform distribution of these clusters, or what we call a skewed distribution.

Chapter 6 presents our design and implementation of a Preference Model Simulator (PMS). In this chapter we outline the PMS architecture, discuss the platform of our implementation, and give a detailed description of the implementation.

Chapters 7 and 8 present the results obtained from a simulation study using the simulator discussed in chapter 6. We used scheduling in a distributed manufacturing environment as a case study. In chapter 7 we present the results from the exploratory experiments that were conducted to study the preference model convergence and its characteristics. These experiments led us to the theoretical performance model discussed in chapter 5. Chapter 8 presents further experiments that explored the working behaviour of the preference model and verified the theoretical performance model. Also, we present a comparison of the predicted remaining preferences values (i.e. the inverse of the preference gain) after the final iteration with the actual values obtained from the simulation experiments is presented in this chapter.

The thesis is concluded in chapter 9 that presents a review and evaluation of the concepts presented in this dissertation. It also discusses possible future research directions and value to industry.

Chapter 2

Literature Review

One of our main objectives in this thesis, as outlined in the introduction, is to devise a model that can resolve preferential conflicts among cooperating agents in multi-agent systems. The main area of application for this agent based preference model is in solving problems in distributed systems where multiple solutions usually exist. In agent-based distributed systems negotiation among agents is the method used to arrive at a decision regarding the choice of a solution. Therefore, before we embark on presenting our model (Chapter 4) we start by reviewing the different techniques used to resolve conflicts in the area of distributed agent-based systems. Next we outline the different applications that have applied the concept of preferences. Such applications include manufacturing systems, product design, and distributed meeting scheduling. As a manufacturing system is one of the most complex and interesting application of these, we have applied our technique on distributed manufacturing scheduling. Thus we also present in this chapter a review of some work in this research area. Then we discuss the main differences between our approach and previous approaches.

2.1 Conflict Resolution

Researchers in the field of cooperative distributed problem and multi-agent systems have developed various techniques for conflict resolution in multi-agent systems. Some of these common techniques [LIU98] include case-based reasoning, voting, constraint relaxation, and market-based approaches as outlined below.

2.1.1 Case-Based Reasoning

Case-based reasoning (CBR) is an experience-based technique for knowledge acquisition and refinement [SEN99]. It draws upon previous solutions that worked well for similar problems and uses them to solve new problems [WATS94]. Whenever a new problem is encountered, it is matched against previously stored cases, similar cases are retrieved and used as a starting point for solving the problem. When the system fails to find similar cases, a solution is generated from scratch. All problems and their solutions are stored as cases in what is called the case base. The processes involved in CBR can be described as a cyclical process comprising the four *REs*:

- *RETRIEVE* the most similar cases;
- *REUSE* the cases to attempt to solve the problem;
- *REVISE* the proposed solution if necessary, and
- *RETAIN* the new solution as a part of a new case.

To match cases a measure of similarity between the task attributes is used. Very often this cycle requires human intervention and most CBR encourages human collaboration in decision support [WATS94]. The model integrates case-based reasoning and the multi-attribute utility theory [KEEN93]. One system that uses case-based reasoning to resolve conflicts during negotiation is the PERSUADER [SYCA89, SYCA89A, SYCA90] that has been used to resolve conflicts in labour relations.

2.1.2 Voting

Voting is a mechanism whereby a number of equals indicate a choice from several conflicting alternatives by some voting mechanism (majority, two thirds, etc.). Participants are obliged to accept the result of the voting. Voting strategies are widely used in social sciences such as political science and economics and have their roots in Game Theory.

In the domain of multi-agent systems, often, a group of agents has to make a common decision, yet they have different preferences about which decision to take. For such cases, voting theory has been used as a technique for reaching consensus in a negotiation process and group-decision making [SAND99]. Agents can perform voting on certain decisions, the result of the voting is obligatory for all agents participating in the voting. Several protocols for voting are presented in [SAND99], some of these protocols are presented below:

- Plurality protocol: This is a majority protocol where all alternatives are evaluated simultaneously and the one accepted by the maximal number of agents wins.
- Binary protocol: It uses the majority protocol but alternatives are evaluated in pairs. The one, which gets fewer votes is eliminated and the one which receives the majority is accepted and stays for the next round of voting.
- Bord's protocol: Each agent votes for the list of alternatives that is associated with a value. After the negotiations the values from the lists of all agents are added up and the alternative which gets the best value from all agents wins.

Voting has been used in several multi-agent decision making problems, such as distributed meeting scheduling [EPHR94, EPHR96, SEN97, SEN98] and collaborative filtering [PENN00, BREE98]. In the domain of distributed meeting scheduling all agents vote on all possible proposals to reach consensus on an acceptable time for the meeting (see section 2.3.4). In Collaborative filtering, a number of algorithms based on voting mechanisms attempt to predict the preferences of one user based on the preferences of a group of users. For example, given the user's ratings for several books and a database of other users' ratings, the system predicts how the user would rate unread books.

A key problem voting mechanisms are confronted with is that of manipulation by the voters, that is when an agent votes strategically and it does not sincerely declare its true preferences on the different alternatives so as to manipulate the outcome to be more favourable to the agent [CONI03]. Several mechanisms have been used to eliminate manipulative voting, one popular mechanism is the *Clarke Tax Mechanism* [EPHR94]. In this mechanism, a tax is imposed on those agents whose vote change the outcome and put other agents in a disadvantage.

2.1.3 Constraint Relaxation

Constraint satisfaction is an AI paradigm that represents problems as a finite number of variables and finite number of constraints [Kocj00, SYCA91]. Each variable has a domain with the possible values for that variable, and each constraint that hold the relation between the values of the variables on which only certain combinations of values are acceptable. A solution to a constraint satisfaction problem (CSP) is an assignment of values to the variables such that the constraints are satisfied. An example of a constraint-based representation is the map colouring problem in which a given a map is to be coloured using three colours, green, red, and blue, and that no two neighbouring countries have the same colour. Each variable can be assigned a colour from a limited spectrum and each constraint defines that a particular pair of variables can not have the same colour. A distributed CSP is a CSP in which the variables and constraints are distributed among autonomous agents in multi-agent systems. Each agent has one or multiple variables and tries to determine the possible values for these variables.

Constraint relaxation modifies the relationship defined by a constraint allowing wider range of relationships. The modification changes the problem definition, allowing a superset of the original solutions [BECK94A], for example in the map colouring problem, described above, we might decide that a particular adjacent variable pair can be different colours or can both be red.

Agents in multi-agent systems make use of a constraint relaxation approach to resolve conflicts by making particular changes to the definition of the constraint, i.e. relaxing the constraints. Thus altering the problem and allowing relationships that were not allowed in the original problem to become acceptable. An example of using this technique in the supply chain domain has been shown in [BECK94].

2.1.4 Market Based Approach

The market-based approach, otherwise known as market-oriented programming [WELL93] [WELL96], is based on market price mechanism. In this approach, agents try to solve a conflict in a distributed system by computing the competitive equilibrium of an artificial economy. In the general equilibrium theory agents are regarded as consumers and producers and their tasks are defined in terms of production and consumption of goods. Agents choose strategies for production and consumption of goods based on the going market price, their capabilities, and their preferences.

Cheng and Wellman have developed an algorithm that utilises this approach, and named it the *Wallras algorithm* [CHEN98]. Wallras algorithm calculates the competitive equilibrium using a price-adjustment process where an excess demand triggers price increases, and excess supply triggers price decreases - this is called the *tatonnemen process* that was originally expressed by Leon Walras [WAL1874]. The change in the price depends on excess demand. The Wallras algorithm has been used for practical applications such as transportation planning [WELL93] and allocating computational resources [BOGA94, DOYL94].

2.2 Using Preferences

Preferences are being used in many applications including advanced information retrieval [SEO00], quality of service provision [PARK98], product design [KEIN97], electronic commerce [JOO00], distributed meeting scheduling [SEN97], concurrent engineering [DARR94], fuzzy ranking [WANG01], agent-based routing [ROGE99], cooperative decision-making [WONG94], computer-supported cooperative work (CSCW) [MCCAc98], document ordering, learning and storage in an Web-based environment [BURK01], and designing an intelligent environment [NAGE99]. Some of these applications try to learn user preferences, some utilise preferences as a method for solving a problem and others are mixture of both learning and at the same time utilising preference to find a solution. Most of these applications share similar concepts for using and building user preference models. We outline some of these concepts by describing some of these applications in this section.

2.2.1 Advanced Information Retrieval

Information in the digital libraries and the internet is growing and the need to retrieve and refine relevant information effectively has prompted researchers to develop different models for information retrieval. Some of these models are based on sorting the information according to the user preferences that are stored in a user profile [PAZZ96]. [TAN98, GLOV98] proposed retrieval systems that use a user profile to filter information from the web, the user profile is not modified automatically. [SEO00] proposed a different information retrieval system, WAIR (Web-Agents for Information Retrieval), which learns user preferences by observing user behaviour during the user interaction with the system.

2.2.2 Concurrent Engineering

The design process of large-scale products involves consideration of hundreds or thousands of often competing concerns such as manufacturability, testability, cost, etc. [DARR94]. Concurrent Engineering (CE) is a product design methodology which aims to enhance productivity and to improve overall designs concerned with integrating all of the functions involved in the whole product development lifecycle. It has been used in the construction industry, electronic design, and manufacturing system design. A widely quoted example is the Boeing 777 jet was developed using concurrent engineering concepts [LOCH00].

One system that is used in the construction industry and that uses the concept of preferences is the Designer Fabricator Interpreter (DFI) tool introduced by Werkman [WERK92]. DFI is a computer tool used in steel-connection design. The DFI provides a multi-agent architecture which models design, fabrication and construction processes. In

DFI each agent is an expert in its domain with its own preferences. The system considers preferences and issues that are important to each participating agent and produces a cooperative solution through negotiation. DFI uses an arbitrator as a means of central control through which agents communicate. There is no direct interaction between agents.

In the Automated Configuration-Design Service (ACDS) introduced by DARR and Birmingham [DARR94, DARR96], the system helps designers during the design phase of the product to select components from catalogues. Catalogue agents use system preferences to select components subject to performance and feasibility constraints.

An agent-based framework to solve hierarchical CE problems is presented in [ARBO96]. In this approach preferences and constraints of a design supervisor are distributed to design subordinates, who are expected to achieve global coordination by using their local expertise. The solution of the design problem is affected by the preferences of all decision-makers participating in the design process.

2.2.3 Electronic Commerce (E-Commerce)

The rapid development of the internet as an electronic market has encouraged the development of systems and services that support users to locate, buy, and sell goods on the World Wide Web [DAST01]. For such systems to be used effectively, it is essential for the system developers to pay attention to the effective representation and utilisation of user preferences [MUKH01]. This can be achieved by building a user preference model. Three approaches for modelling user preferences in e-commerce system are shown in [DAST01]. In the first approach, *collaborative-based approach*, the user preference model is constructed on the rating of items previously used by users thought to be statistically similar, as in [MOVIFINDER]. In the second approach, *content-based approach*, the user preference model is constructed on the rating of previous items used and rated by the user, as in [BARGAINFINDER]. The third approach uses an integrated approach of the previous two approaches as in [BASU98].

An example of an e-commerce system that utilises user preferences is an agent-based grocery shopping system such as that was presented in [JOO00]. This system aims to automate grocery shopping by using agents that select products on the basis of information collected from different stores and comparing it with the user preferences. The agent learns and updates user preferences through the user evaluation of the previous results. This system demonstrated the effectiveness of using user preferences when selecting products.

2.2.4 Distributed Meeting Scheduling

The basic objective of meeting scheduling is to set a meeting time that is acceptable to all potential participants in the meeting. One key question that researchers in the area of distributed meeting scheduling are trying to find an answer to is how to choose an appropriate time slot from these feasible for a meeting [EPHR94]. Research in distributed meeting scheduling presented in [EPHR94, SEN97, GARR96, SHIN00] has used preferences to design models and systems that try to solve problems found in distributed meeting scheduling.

Ephrati et al. [EPHR94] introduced an approach that uses a primitive economic market, where the users indicate their preferences by assigning points to the different proposed solutions. Then the proposal with maximum points is taken as the accepted proposal. They analysed tradeoffs between mechanism complicity and information preferences using three centralised monetary-based meeting scheduling systems.

Garido and Sycara [GARR96] presented an agent-based meeting scheduling system in which each agent in the system knows its user preferences and calendar availability. They presented experiments that show the meeting scheduling performance is stable when information on user preferences and calendar are kept private. However, they did not show how to reach a compromise solution with other agents.

Sen et al. [HAYN97, SEN98] developed an agent-based system that can automate scheduling meetings between group of users. It utilises user preferences to find a suitable meeting time. User preference dimensions such as meeting topics, duration, participants, host, etc. are rated by the user. Then each preference dimension is given a number of votes depending on the user rating and proposals with enough votes are accepted [SEN97].

Shintani et al. [SHIN00] used a mechanism that includes multiple preference revision to design a distributed meeting scheduler. In this mechanism the invitee agent, that is the agent invited to the meeting tries to revise its preferences so that its most preferable alternative is the same as the host agent, that is the agent calling for a meeting. The negotiating agents try to reach a compromise by revising their preference until an agreement is reached.

2.3 Manufacturing Systems and Scheduling

The subject of scheduling has attracted researchers in many areas, such as manufacturing [SHEN99], project management [KIM01], and public transport [WREN99], as well as the production of timetables in schools, universities [SCHA95]. Resource allocation in manufacturing systems is a typical representative scheduling and resource allocation application where various types of preferences need to be involved to obtain any acceptable solution. Also the manufacturing-scheduling problem has received considerable attention because of its highly combinatorial aspects (NP-hard[1]), dynamic nature, and practical interest for industrial applications [SHEN02].

Scheduling plays an important part in manufacturing systems and several approaches and techniques have been proposed to solve the manufacturing-scheduling problem. Such techniques can be categorised into traditional approaches and agent-based approaches [SHEN02]. In the following subsections we first outline the traditional approaches in scheduling and then we discuss agent-based manufacturing systems with specific focus on scheduling.

2.3.1 Traditional Approaches in Scheduling

Such scheduling algorithms usually use search strategies to solve problems where the calculation of all possible solution is not possible with available computing equipment. In the research community different classifications for such algorithms have been proposed. For example [BROW95] classifies scheduling algorithms as artificial and computational intelligence methods, [BAKE98] divides them according to the degree of optimality sought, e.g. nearly-optimal scheduling, towards optimal scheduling and heuristic scheduling, and [GeYI99] categorises them as analytical, heuristic and AI approaches. In their most recent study [SHEN02] traditional algorithms are classified into analytical, heuristic and metaheuristic. We outline these categories in this section.

- **Analytical Approaches:** These techniques, such as queue theory, linear programming, branch and bound algorithm, dynamic programming etc., try to produce the exact

[1] That is, finding an optimal solution is impossible without using an essentially enumerative algorithm, and the computation time increases exponentially with the problem size

solutions. However, they are only effective for solving small problems or problems that are not NP-complete.

- **Heuristic Approaches:** These algorithms try to replace exhaustive search algorithms by trying to discover solutions that point to the optimum solution, although they might miss the optimal solution. Such algorithms include forward/backward scheduling and deterministic simulation.

- **Metaheuristic Artificial Intelligence Approaches:** Under this name, the following algorithms are grouped: simulated annealing [JOHN89]; genetic algorithms [GOLD94]; neuro scheduling [RABE93]; tabu search [HERT95, DAM94].

All the traditional methods outlined above use simplified theoretical methods and are essentially centralised [SHEN02]. As we shall show below it is difficult to use them for modern manufacturing systems.

2.3.2 Agent-Based Approaches in Manufacturing Systems

In order to remain competitive modern manufacturing systems are required to adapt automatically to changes in the environment [BREN02]. Such changes can be market changes or the emergence of new technology. Manufacturing systems are required to deal with unpredictable demand for different products in small batches. Nowadays, many industries are changing from mass production to mass customisation [CHORA01]. The requirements for modern manufacturing systems are as follows [SHEN99]:

- **Integration:** The manufacturing systems of a manufacturing organisation should be integrated with their related management systems via networks to support global competitiveness and rapid market responsiveness.
- **Interoperability:** Manufacturing systems may need to accommodate heterogeneous software and hardware in both their manufacturing and information environments. The components of such heterogeneous environments should interoperate in an efficient manner.
- **Cooperation:** Manufacturing systems need to cooperate with their suppliers, partners, and customers for material supply, parts fabrication, marketing and so on.
- **Dynamism:** It must be possible to schedule and reschedule without stopping and reinitialising the working environment.
- **Flexibility:** The system must handle different orders and deal with changing machine characteristics.
- **Scalability:** The system must handle any increase in orders or resources without disrupting any organisational links that were previously established.
- **Fault Tolerance:** The system should detect system failures at all levels and should be able to have a recovery procedure that can avoid system collapse or a reduction in system throughput.

Adapting traditional centralised manufacturing systems to fulfil the requirements above is cumbersome. In recent years researchers have been applying concepts in multi-agent systems to manufacturing systems and have developed new type of manufacturing systems such as Holonic Manufacturing Systems (HMS) [DEEN03A]. HMS is an international project called Intelligent Manufacturing Systems (IMS), which started in 1993 as a ten-year programme by Australia, Japan, Europe, Canada and the USA, and is supported by respective Governments. The HMS project is focused on what might be described as an agent-based manufacturing system particularly suited to low-volume high-variety manufacturing. HMS can be viewed as a distributed system consisting of autonomous, cooperative, and recursive functional units called *holons* that can be viewed as special kinds of agents.

Agent-based manufacturing scheduling systems support distributed scheduling, in contrast to traditional manufacturing scheduling systems which support centralised schedulers. In the Agent-based approach each agent can locally handle its schedule. A global schedule can be produced through a given negotiation mechanism and protocol. A number of researchers have used agent technology to resolve the manufacturing scheduling problem. A survey by [SHEN99] reports on 30 projects using agent-based approaches for manufacturing planning, scheduling and execution control where agents represent physical entities, processes, operations, parts, etc. We overview some of the of recent projects and focus on how they deal with the scheduling problem.

- **AARIA** (Autonomous Agents at Rock Island Arsenal) [AARIA]: An implemented system that runs in a real or simulated mode for the U.S. Army facilities at Rock Island Arsenal. The system implements the following functionality: finite capacity scheduling, basic planning, order entry, purchasing, bill-of-materials management, inventory management, resource management, personnel management, integrated financials, and reporting [PARU97]. AARIA uses agents representing the manufacturing capabilities of the system. When these agents are connected together, they self-configure to provide a full system functionality. AARIA demonstrates the interactive insertion of new jobs into a distributed schedule. It allows the dialog with customers and suppliers to optimise schedules and react to disturbances in the system.

When a job arrives, it causes bid requests, bids, purchase orders, and commitments to propagate through the network [SOUSA99]. Through this process, the system finds the minimum-cost schedule for each new job, assuming already-scheduled jobs cannot be moved.

- **IBM Paper Mill Scheduling** [IBM]: The IBM mill scheduling system is an *interactive* decision-support system for scheduling operations in a multi-mill multi-machine in the paper industry from manufacturing to product delivery [MURT97]. The system architecture is based on asynchronous team (A-Team) of cooperating agents. This architecture consists of a population of solutions and three types of agents, which create and modify this population. These three agents are *Constructors, Improvers, and Destroyers* [AKKI98]. The Constructors create initial solutions, then Improvers select existing solutions and modify them to produce a new solution, which is then added to the current population while the original solution is preserved. The main function of the Destroyers is to keep the size of the population of solutions in check, which is done by deleting clearly bad or redundant schedules.

- **PROSA** (Product –Resource –Order –Staff Architecture) [PROSA]: This is a reference architecture for HMS (see above). In the PROSA architecture a manufacturing system is built from three basic holons: order holon, product holon, and resource holon [WYNS99]. A product holon holds the process and product knowledge to ensure the correct fabrication of the product and acts as an information server to the other holons in the HMS. A resource holon represents the physical part that controls the resource and holds the methods to allocate the production resources, and the knowledge and procedures to organise, use and control these production resources to drive production

[BRUS98]. An order holon is responsible for performing the work correctly and on time, it represents a manufacturing order. A staff holon whose mission is to assist and advise the basic holons can be added. An example of a staff holon is the scheduler. [BONG95] describes the schedule execution for this environment.

- **Contract-Net Protocol for HMS** [SOUS99]: This is a proposed dynamic-scheduling system architecture based on the Contract-Net Protocol [DAVI83], The proposed architecture adapted this negotiation protocol between all parts of the system to handle conflicts in the decision problem, and dynamic changes in the system. This architecture is composed of holons representing tasks and holons representing resources.

- **HOLOS / MASSYVE**: The HOLOS architecture developed at the New University of Lisbon is concerned about scheduling in Virtual Enterprises [RABE94]. A unique global and comprehensive schedule does not exist, but rather a collection of distributed and interrelated pieces of smaller schedules. HOLOS uses the Contract-Net Protocol co-ordinating mechanism to support information exchange among agents during the generation of the scheduler. It also uses the tandem agent architecture to support integration with legacy systems [LEIT01]. This work was later on extended to distributed multi-site manufacturing systems and virtual enterprises in the framework of the INCO MASSYVE project [RABE99]

- **MASCADA** (Manufacturing Control Systems Capable of Managing Production Change and Disturbances) [MASCADA]: The focus of this project is to develop a manufacturing control system that is able to manage production change and disturbance both effectively and efficiently [MASCADA]. The manufacturing system in the MASCADA approach is composed of communicating local intelligent autonomous agents. These agents are based on the HMS - PROSA reference architecture (see above). Within the MASCADA project the manAge architecture [HEIK99], which is an agent architecture for manufacturing control, has been tested.

- **DEDEMAS** (Decentralised Decision Making and Scheduling) [DEDEMAS]: The DEDEMAS system prototype, developed for the integration of distributed systems providing mechanisms for decentralised decision making and scheduling, covers both multi-site operations of one company and its chain of external suppliers [TOENOO]. The decision-making mechanism is based on the extended contract net and several monitoring schemes and rules support companies to optimise their processes. In that sense it is similar to HOLOS/MASSYVE approach (see above). It uses XML messages to support a high degree of transparency of the business processes and system integration.

- **MetaMorphII**: This is a multi-agent architecture for intelligent manufacturing developed at University of Calgary [METAMORPH2]. Its objective is to integrate the manufacturing enterprise's activities such as design, planning, scheduling, simulation, execution, and so on, with those of its suppliers, customers and partners within a distributed intelligent open environment (MATU96). The project proposes a hybrid agent-based architecture, combining the Mediator and the autonomous agents approaches. MetaMorphII organises the manufacturing system at the highest level through subsystem mediators and each subsystem can be an agent-based system. Some of these agents may also be able to communicate directly with other subsystems or agents in other subsystems [METAMORPH2].

2.4 Discussion

We outlined in section 2.3 some applications, which utilise preferences for solving problems in different domains. We summarise our findings on their use of preferences as follows:

- Most of these applications use preferences as a simple ranking mechanisim.
- Most of the applications require human intervention at one point or another of the processing.
- Complex tasks with multi-level subtasks and preferences are not handled.
- Most of these applications do not guarantee the convergence of distributed computation.
- The issues of relaxing preferences to determine the maximum possible preference values that can be achieved have not been addressed.

We outlined the various techniques that have been developed by researchers in the field of cooperative distributed problem and multi-agent systems for conflict resolution in multi-agent systems in section 2.2. Some of these techniques are based on game theory. In practical applications, negotiation can not be treated as a game, as in games one party loses and others win, while in real world situation both parties must gain some benefits [MUDG00]. Other techniques are based on decision theory such as utility theory [NEUM44]. Most of these techniques depend on the availability of well-behaved quantitative data. Such data is difficult to obtain in most applications [WONG94].

In this thesis we propose a model to solve preferential conflict based on a simple market-based approach. It differs from the market-based approaches discussed in section

2.2.4 in that it does not use computational economics (general equilibrium theory). Our approach is based on a simple cost accounting approach. In our approach, agents can trade preferences for “money”, which enables agents to gain more preference values in subsequent iterations.

As the trend in modern manufacturing systems goes towards mass-customisation (section 2.3.4), customer preferences as well as manufacturer preferences play a major role when it comes to scheduling the product manufacturing process. We found that most manufacturing scheduling systems concentrate mainly on resolving constraints conflicts (referred to as hard constraints in some literature) and give only little consideration to solving preferential conflicts (referred to as soft constraints in some literature). We show how our preference model can be used for scheduling processes in distributed manufacturing systems.

We formulated a theoretical performance model based on the preference model presented in this thesis. For a given distribution of preference values over a number of resources the theoretical model can roughly estimate the minimum and maximum preference values that can be achieved. We did not find any performance models similar to this in previous research work on distributed manufacturing scheduling.

We apply our preference model for conflict resolution within the Cooperating Knowledge Based Systems (CKBS) paradigm. The CKBS approach is an engineering paradigm for solving real-world problems in distributed applications. This is in contrast to the mentalistic approaches used in current multi-agent systems (see section 3.1). We also use the cooperation model and the agent architecture as applied in CKBS. In the next chapters we outline the CKBS model and our preference model.

Chapter 3

Cooperating Knowledge Based Systems

The preference model, which is the focus of this thesis, is to be used as a means of resolving conflicts when solving problems in Multi-Agent Systems (MAS). From the various MAS architectures that are currently present we chose the Cooperating Knowledge Based Systems (CKBS) architecture as a framework to apply our preference model. CKBS is a research area in which the main objective is to develop good formalised solutions for computing problems in distributed application using an agent based approach. We use this chapter to outline the basic concepts in the CKBS architecture. In the following section we review the basic concepts in the CKBS model. Section 3.2 reviews the basic elements (Agents, Tasks, and Shadows) of the CKBS model. We outline the cooperation environment in section 3.3, while the operational architecture is outlined in section 3.4.

3.1 Basic Concepts of the CKBS Model

CKBS research overlaps with that of Multi-Agent Systems (MAS) of Distributed Artificial Intelligence (DAI) [DEEN96]. The CKBS approach emphasis is on solving real-world distributed problems, where effectiveness, performance, reliability, and usability are of utmost importance [DEEN96] [DEEN97]. In this respect, CKBS has been said to be an engineering approach to agent-based systems [DEEN96], this is in contrast to the DAI/MAS approach where models are formulated in terms of human behavioural concepts such as belief, intention, and desire (BDI) [WOOL99]. Concepts of belief, desire and intention (BDI) were introduced by Bratman et al [BRAT88] and developed further by Rao and Georgeff [RAO91] as the basis of single-agent architecture. An informal architectural frame for the CKBS was described in [DEEN96] and [DEEN97]. To support fault tolerance in a distributed environment the CKBS model was developed further and presented in [Deen98]. Deen and Johnson presented an abstract and a formulised version of the CKBS model in [DEEN99A] and [DEEN03] respectively.

In the CKBS approach an agent, is an autonomous knowledge based systems having a compulsory software component and an optional hardware or human component. The agent can work with other agents in cooperation to solve a joint task. The collection of agents together constitute a Cooperating Knowledge Based System (CKBS) [DEEN96] [DEEN97][DEEN99]. In this thesis an agent is treated wholly as a software-based system with no hardware or human component.

AS outlined by Deen in [DEEN96] and later by Fletcher [FLET97] the CKBS main objective is to develop systems for real world applications. Towards this objective CKBS offers the following for interagent activities within a transaction oriented CKBS [DEEN96]:

- A development environment that reduces the development time for new applications. CKBS provides well-defined structures and components as well as a friendly user interface such as a graphic user interface (GUI) that can help to achieve this goal.
- A framework to enable agents to manage and interpret cooperation strategies, and recovery mechanisms.
- A support for user interface tools and inter-agent communications to specify and invoke user-defined cooperation strategies for distinct tasks at different times.

- A provision of simplified Cooperation strategy specifications. This is provided by using distribution transparency, that is, multi-agent systems are viewed as a mono-agent system.
- A multilevel schema to accommodate different levels of users with different expertise.

The CKBS model has been applied in different application areas that includes Holonic Manufacturing Systems (HMS) [CHRI98][FLET98][FLET98A], Air Traffic Control [DEEN97A] [NDOV94], Telecommunications Network Management [FLET97] and agent-based interoperability with partial global ontologies [ALQA99].

It should be noted that the CKBS has been adopted as an abstract foundation for agent-based manufacturing within the HMS project [Deen03].

3.2 Basic Elements of the CKBS Model

Agents, tasks, and shadows are the key elements in the CKBS model. In the following subsections we review these basic elements and their role in the CKBS model.

3.2.1 CKBS Agents

As indicated earlier, agents in the CKBS model are described as a large grain entity, which have a compulsory software component and an optional hardware component [DEEN96]. While agents can participate in a joint task processing, so in that sense they are cooperative, they are autonomous, in the sense they can decide which joint task to participate in and then negotiating their role in that task [DEEN03]. Figure 3.1 shows the structure of the agent as defined in the CKBS model. The agent has a head connected to a body by a communication link (the neck).

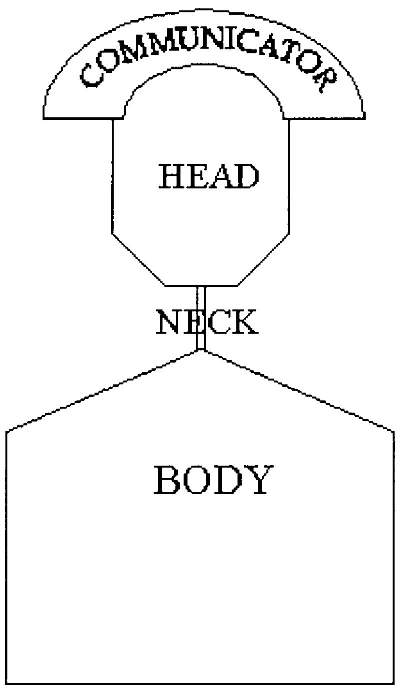


Fig. 3.1 Agent structure

Inter-agent activities take place in the agent head which holds knowledge about itself and knowledge about other agents in what is called the home model and the environment model respectively. Communications with the outside world is the responsibility of the communicator at the top of the head.

The body provides the individual skills of the agent as discussed later and makes all internal decisions regarding the skill execution.

Skill Classes

A skill is what an agent offers to other agents, it signifies the type the operations the agent can perform and the role it plays within its community [HAMA98]. An agent may possess more than one skill, though each agent must have at least one skill. Agents are grouped by the skill they posses, one agent can belong to more than one skill class. Each skill is represented in a skill class. A skill can may have more than one agent, agents in one skill class are called twins.

Agent Classification

To simplify the process of solving problems in multi-agent systems, CKBS classifies agents into several categories [DEEN98], the following are the ones which are relevant to our work in this thesis:

1. Coordinator agents: such agents have the skill to coordinate task execution.
2. Processing agents (cohort agents): such agents possess the skill to process a task.
3. Service provider agents: such agents provide additional services that are needed in MAS. As examples of such agents are directory agents, transport agents, monitor agents, etc.

Within each category multiple classes of agents can be found. For instance to handle breakdowns each class can have a set of agents that have the same capability and can replace one another in the event of failure, these are called *twins* [DEEN99]. A class of agents called *minders* whose responsibility is to supervise the replacement of the faulty agent and to schedule its twins.

3.2.2 Tasks

In MAS agents work together to solve a joint (i.e. global) task, say T^i, which can be decomposed into a lower-level tasks (subtasks) (T_1 ... T_n) [DEEN99A]. Subsequently any subtask can be decomposed to another set of subtasks. Each subtask requires a certain skill. Any agent that belongs to the skill class is a potential candidate to execute that subtask.

Tasks in the CKBS model are generally described using task schema, dependency schema, and preference schema. We outline these schemas below.

The Task Schema

The task schema describes the task procedure, the task parameters, and the pre/post conditions that apply to every instance of the schema [DEEN99]. Agents which possess the appropriate skills execute the task procedure and report the result of the execution. The task parameters such as start-time, duration, deadline, and success-parameter, are those associated with the execution of the task. Some parameters are set before task execution (*input parameters*), and others are set during execution. Some parameters remain static during execution and other parameters are subject to changes during execution. Pre-conditions can be system-dependent which guarantee that resources are available and in a fault-free state, or precedent-dependent that are derived dynamically from the dependency schema [DEEN99A]. Post_conditions are these conditions that the task must satisfy upon completion of the task, for example a quality check or that end-time is less than deadline.

Dependency Schema

Each subtask will have a dependency schema that describes the relevant task dependencies. It shows what tasks should be executed before starting the task execution. These are static constraints and contribute to the pre-conditions discussed above. This schema can be integrated with the task schema, but in this thesis we choose to separate it for clarity purposes.

Preference Schema

Associated with each subtask is a set of preferences as well as the set of dependencies and constraints explained earlier. Preferences arise when multiple solutions are available for the task. In this case the user express his choice in the form of preferences on some desirable aspects of the solution. Preferences vary widely depending on the application, and each requires special presentation, such preferences can be a machine, time, labour, etc. It is not always possible to satisfy all preferences due to contention with preferences of other tasks.

Our focus in this thesis mainly, as pointed out in chapter 1, is how to derive a preference-based solution that satisfies as many preferences as possible in presence of contention.

3.2.3 Shadows

In order to keep the cost of communication as low as possible, we need to make sure that only the necessary information is exchanged amongst cooperative agents. The CKBS uses the concept of shadows [DEEN96] as means of exchanging information amongst agents. Information on what skills are available from an agent are called shadows of its skill relations. Shadows are derived from these relations, containing all the relevant attributes needed for the associated software function to be referenced or instantiated by a remote agent. Similarly, they are used in inter-agent activities, held at the upper head of the agent [DEEN97].

Shadows are quite similar to views in traditional relational databases, yet they provide strong distributed consistency and trigger schemes. A remote agent cannot directly update the external relations of other agents, but it sends a message to request an update to be done.

From the perspective of the agent, there are two types of shadows: export shadows and import shadows. Export shadows are the means by which an agent can publish its skills and capabilities to other agents. They are views created from external relations of the agents to be used by other agents as import shadows [ALQA99]. An export shadow *ES* for a skill *K* offered by agent *B* to *A* has its scheme derived from the skill relation *SR* (the source table of the shadow) and is exported to remote relations in A. The agent posts its export shadows for other agents to import. The corresponding source relation is responsible for keeping its export shadow up-to-date.

An agent builds acquaintances by importing suitable shadows from other agents. Agents access each other's shadows when communicating. Shadows that the agent collects form the partial global knowledge of the agent and can vary from time to time.

Multiple shadows, which are not necessarily identical, can be generated on a given skill. Import shadows received from agents with the same skill could be different, it would be the responsibility of the agent to map these shadows into one common import shadow for that skill for the convenience of operations at that agent.

3.3 Cooperation Environment

In this section we outline in general, with some details when needed, the basic concepts in the CKBS model, namely cooperation, coordination, inter-agent interactions, cooperation strategy and distribution transparency.

3.3.1 Cooperation

In [DEEN99] cooperation is defined as the process in which agents carry out dependent activities of a joint task (global task) and negotiation is the process by which agents agree on mutually acceptable solutions. CKBS agents are implicitly cooperative, they work together in what is called a cooperation block, CB. A CB can be created and terminated dynamically during the processing of the task. A basic CB has a coordinator and a set of other agents that carry on the actual task, referred to as *cohorts*[*]. The coordinator can be viewed as the user agent. The same agent can be a cohort in one CB and a coordinator in another CB, and it can participate in more one CB at the same time. This leads to a heterarchy as shown in Figure 3.2.

In this figure, agent A is the coordinator of a CB with agents B, C, and D as cohorts. Agents C and D are in turn the coordinators of a lower two CB's. Agent C is the coordinator of a CB with agents E and H as cohorts. Agent D is the coordinator of a CB with agents F, G and H as cohorts. Agents E is in turn the coordinator of a lower CB in which H, J, and I are cohorts. Note that H appears concurrently as a cohort in two CB's, D act as the coordinator in one CB and C acts as a coordinator in the other. In general an agent in a CB can be a coordinator of a lower level CB to execute a subtask with the help of cohort agents, or a cohort agent in a CB that executes a subtask with other cohorts. An agent can participate in more than one CB concurrently. An agent can act as a cohort in one CB and a coordinator in a lower level CB.

[*] This term is also widely used in distributed system processing and refer to processes that are working together

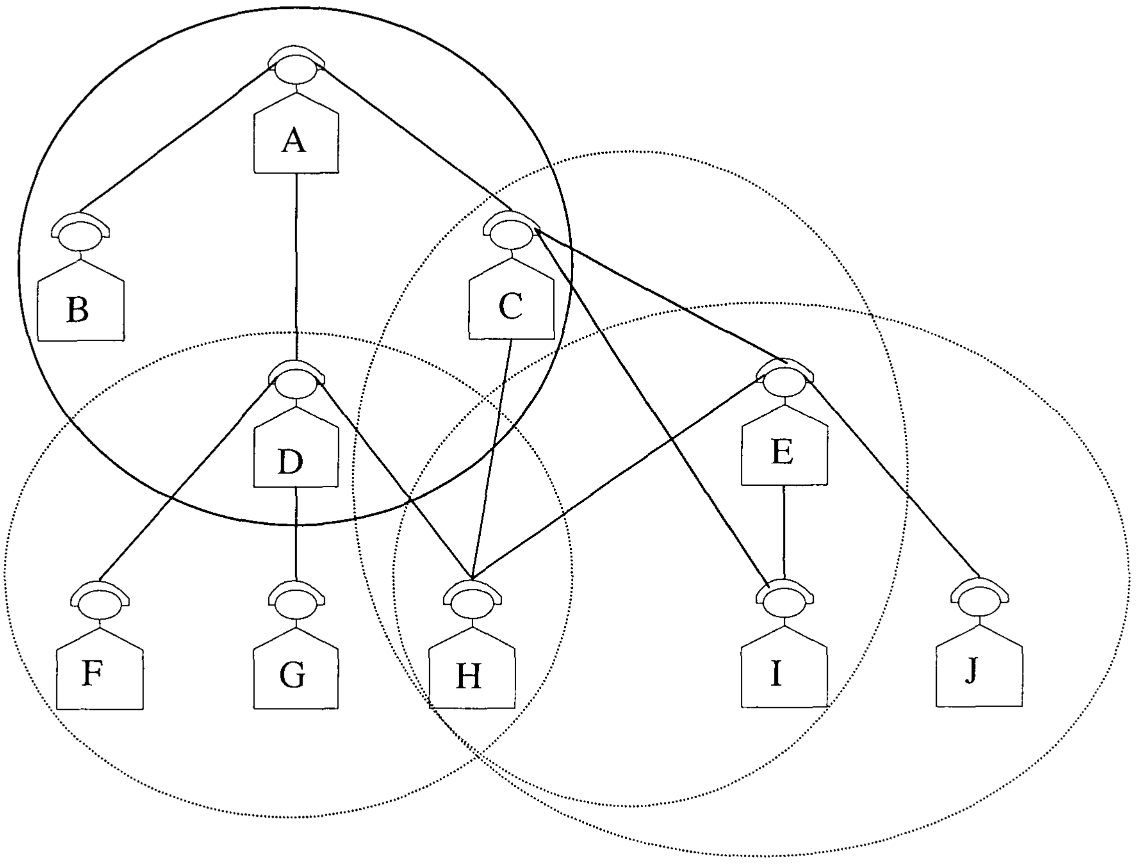


Fig. 3.2 A heterarchy of cooperation blocks

3.3.2 Coordination

As stated in [DEEN96] cooperation can be trivial or non-trivial. A trivial cooperation involves a simple information retrieval, as a simple retrieval query in a database. In a trivial cooperation the agent is committed to answer queries from other agents, even though the answer could be noncooperative, (e.g. Can not participate in executing the task). In a non-trivial cooperation an agent (cohort) enters, implicitly or explicitly into a CB through a three Stage Coordination, 3SC, protocol that was proposed in [DEEN94]. As stated in that proposal the 3SC protocol embodies the following three stages:

1- Agreement:

In this stage, agents enter into an enforceable agreement to perform a joint task subject to some constraints and preferences. The agreement is reached within a CB as agents in the CKBS model participate in a CB under a coordinator to solve a joint task. This agreement itself could be arrived through negotiation. Agents can enter this agreement one at a time or all at the same time depending on the application need.

2- Interactions:

The cohorts interact with each other to achieve the joint goal in accordance with the agreement. These interactions take place within a CB.

3- Termination

The task execution terminates when cohorts complete their tasks. Then the cohorts leave the CB one at a time, or all at the same time, depending on the application. The coordinator leaves last and then the CB terminates.

3.3.3 Inter-Agent Interactions

Effective communication is needed for a successful cooperation between agents. In a multi-agent research various proposals for inter-agent interactions have been made, most of these use psychological approaches by imitating biological organisation interactions and behaviours such as humans, insects and animals. One example is the BDI (Belief, Desire and Intention) that was introduced as a basis for research into multi-agent architectures [BRAT88][RAO95]. This BDI model defines the intra-agent behaviour in terms of mental faculties for the rational selection of action plans to satisfy goals. Some researchers

[MUEL96][FISH94][LUX97] in multi-agent systems are inspired by the Speech-Act Theory [SEAR69]. Speech-Act treats agent interaction and communication as a type of action to be incorporated into planning and reasoning processes. Primitives inspired by speech act theory include *propose, refuse, respond, inform*, etc. During the 1990s, DARPA introduced a LISP-based environment that integrates a Knowledge Interchange Format (KIF) [GENE92] and a Knowledge Querying and Manipulation Language (KQML) [FINI93]. More recently, FIPA (Foundation for Intelligent Physical Agents) [FIPA] defined a multi agent framework and an agent communication language (ACL) based on the Speech Act Theory and KQML. It uses modified BDI concepts and supports the Speech-Act inspired primitives mentioned above.

The CKBS takes a different approach based on a well-tried computer science concepts [Deen97]. A set of commands or communication primitives taken from the database domain, similar to the set of SQL commands used in the relational database model. The CKBS assumes the existence of six request primitives to reference inter-agent activities:

- *Retrieve* : to get information from another agent.
- *Perform* : to request an agent to carry out an action.
- *Modify* : to modify parameters on the agreed action.
- *Delete* : to delete an agreed action.
- *Confirm:* to confirm an action to proceed.
- *Abort*: to abort an action after its execution has started.

The MODIFY and DELETE represent error correction on a previous command.

The above commands are used in the agents' interactions to reference the exchanged shadows. For example, if an agent A has an import shadow S_b (section 3.2.3) of source external relation R_b at agent B, all operations requests by A on S_b are automatically dispatched by the underlying communication system. In the CKBS model these communication systems are treated as a blackbox, however Hammad in [HAMA98] shows one way of implementing such mechanism.

3.3.4 Cooperation Strategies

A cooperation strategy, as define in [DEEN97], is a user-defined specification describing how a task should be performed. It specifies, using a high level language, how the coordinator and the cohorts should behave during the execution of the task. The cooperation strategy is flexible in that it can change from time to time and not predefined, the agent can adopt different user-defined strategies.

To explain the specification of cooperation strategy we use an example of agent-based scheduling in a manufacturing environment [FLET00]. Usually in this kind of environment a number of operations (weld, screw, cut, polish, mount etc.) are required by a number of assemblers for the assemblage of a product of a given type. We assume we have three types of agents for manufacturing:

- *Coordinator agent* : coordinates the assemblage of a product type.
- *Assembler agent (cohort)*: responsible for the assembly operations of various kinds
- *Directory agent*: provide information on assemblers' skills.

We assume we have one Coordinator agent, a set of Assembler agents and one Directory agent. The basic steps for scheduling a task, using a contract net protocol [SMITH80] [DEEN97] proceeds as follows:

1. The Coordinotor invites tenders.
2. The Assemblers bid.

3. The Coordinator evaluates the bids, decides on which one to accept and inform the bidders of its decision.
4. The successful bidders confirm agreement.
5. The work is done and the protocol ends.

We assume that the allocation is incremental, that is allocation is requested when needed, hence the requested Assemblers slots may not be free, as they might be already scheduled to other subtasks. We also note that the allocation in one Assembler affects that in others due to precedence constraints. In the case of a request for a pre-allocated slot, a number of different options for the Assembler are available, such as:

1. send a NO, or
2. send next free slot, or
3. send all the subsequent slots

The Coordinator may choose to schedule the Assemblers one at a time or all at the same time using the contract net protocol outlined above. These will lead to different cooperation strategies to be defined by the Coordinator. The user may code any suitable cooperation strategy, subject to the capabilities of the assembler. One such strategy [DEEN97] is outlined in the steps below:

1. The Coordinator requests information from the directory agent about agents that posses the required skill.
2. The Coordinator preselects suitable Assembler agents
3. The Coordinator requests them to schedule the task.

4. The Coordinator receives the results of the requests, which is success or failure.
5. Based on the Assemblers' replies, the Coordinator selects which assembler to process the task.
6. The Coordinator informs the Assemblers of the acceptance and wait for confirmation. It is assumed that all Assemblers accept. It also posts its rejection to these rejected Assemblers.

We show in Figure 3.3 a scenario for allocating three subtasks, T_1, T_2, and T_3, in that order, in three Assemblers. Shaded boxes indicate pre-allocated slots. We assume the duration for each subtask is one slot. With respect to the Coordinator, Assembler 1 is regarded as prior Assembler and Assembler 3 is regarded as next Assembler. Usually, there will be different prior and next assemblers with respect to different Coordinators for different assemblies. T_1 is allocated to the first available slot in Assembler 1, slot 3. Next T_2 is scheduled at slot 5 of Assembler 2, though slot 3 of this assembler is free, this is due to the effect of precedent constraint mentioned above. Then T_3 is allocated to slot 7 at Assembler 3.

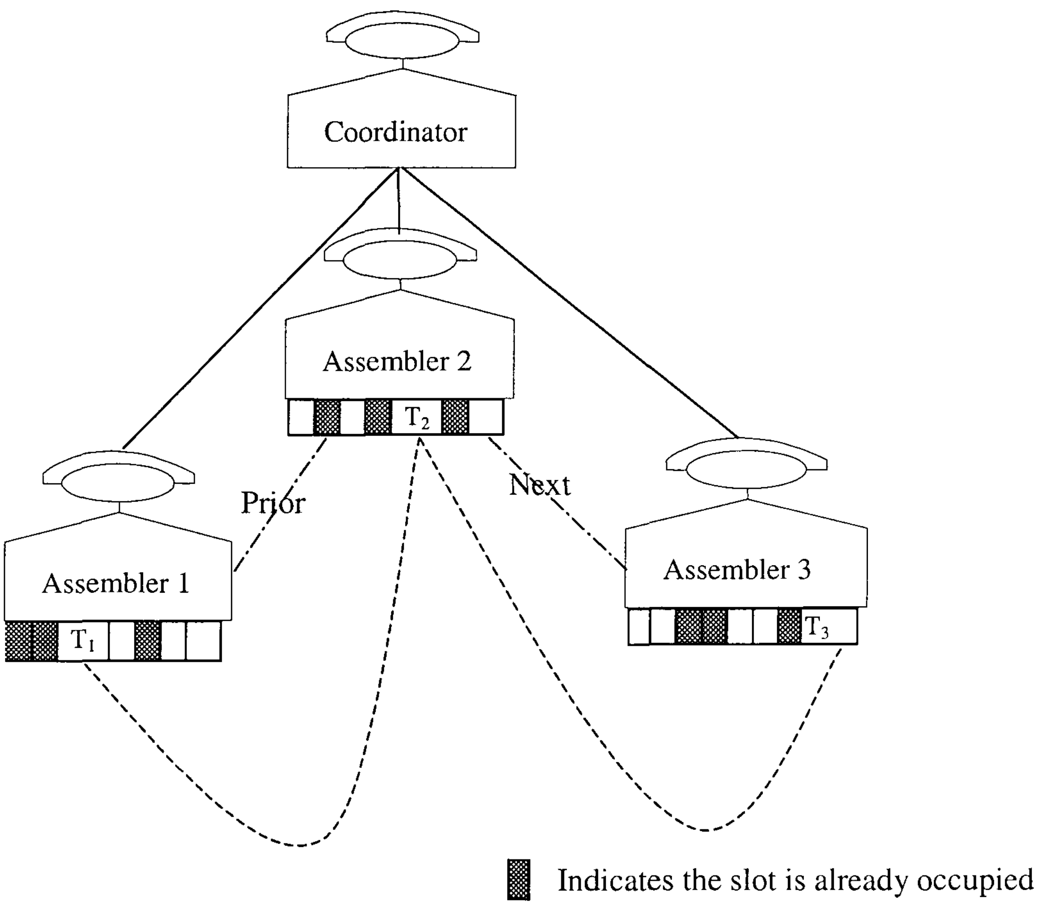


Fig. 3.3 Cooperative scheduling

In the above scenario, no consideration was given to the subtasks' preferences during scheduling the subtasks. Precedent constraints were the only pre-condition that was considered. In the next chapters we present a model that takes preferences into consideration during the scheduling process.

3.3.5 Distribution Transparency

The concept of distribution transparency [Deen97] is one whereby the user views the multi-agent system as a mono-agent system, giving the illusion that the coordinator is the only agent in the system and is capable of performing the task locally. In the CKBS model the task is decomposed into subtasks and processed by different cohorts. The user specifies the cooperation strategy (section 3.3.4) at the coordinator level. The Coordinator, using a lower-level system-software that identifies the remote agents, dynamically dispatches the sub-cooperation strategies for the relevant cohorts. Sub-strategies are communicated as parameters along with the subtask request. Figure 3.4 illustrates this concept.

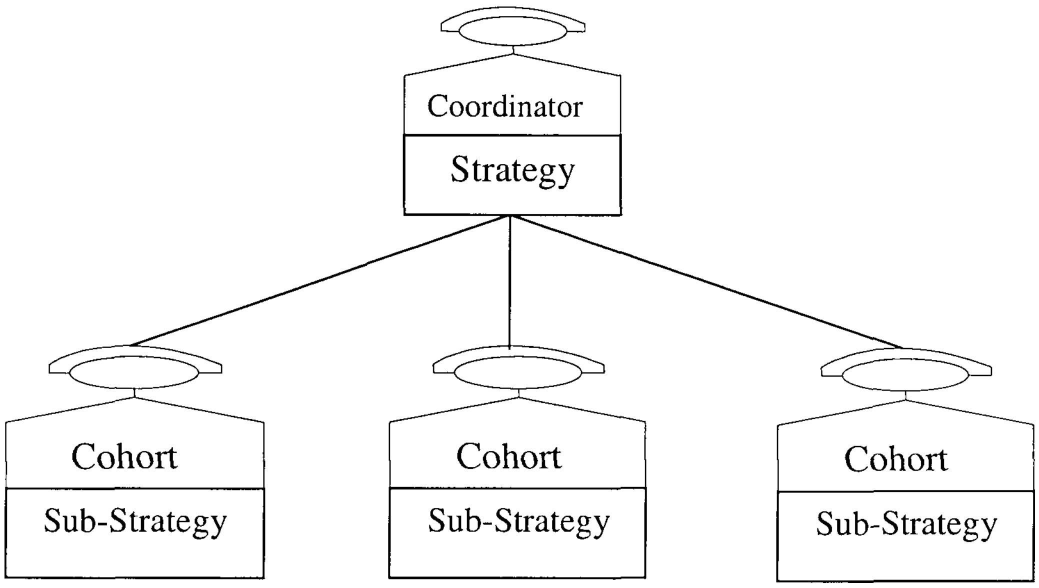


Fig. 3.4 Distribution transparency

The end-users can benefit from the distribution transparency in a number of ways, some of these benefits are outlined below:

1. From the user point view, the system appears to perform as a mono system, without having to know unnecessary system details, such as communication network, protocols and message handling is not needed.
2. The user specifies strategies as a normal program without the need to know the operational details.

3.4 Operational Architecture

In this section we outline the scenario for the steps taken by the different agents to process a global task as outlined in [DEEN00]. Initially the coordinator forms an initial cooperation block, then it decomposes the global task into subtasks, and determines the skills needed for each subtask. It enters the selection process for the appropriate cohort as shown in Figure 3.5. Using the directory agent, the coordinator finds the appropriate cohorts that are available to execute the subtasks (Figure 3.5(a)). The coordinator then enters into negotiation to select one of these cohorts (Figure 3.5(b)). After a successful negotiation the coordinator records and handles all communications to and from that cohort. If a contracted cohort becomes unable to meet its commitment, prior to the start of execution, the coordinator is informed so that rescheduling can take place. Each cohort reports its progress to the coordinator. In the case of delays the coordinator may negotiate with another cohort that has the same skill and thus change the membership of the CB dynamically. In the event of a cohort's breakdown during execution, the cohort will rollback to a recoverable point and informs the coordinator. The coordinator then aborts the processing and advises the other affected cohorts to rollback.

Cohort to cohort communication in the same CB may be needed in some cases. This includes messages on results, task-sharing, and resource-sharing. If the tasks of two cohorts have common subtasks, then they may negotiate with each other on such subtasks. Also common resources can be shared subject to some negotiation during task execution. Such negotiation may affect the execution order of subtasks, within each cohort.

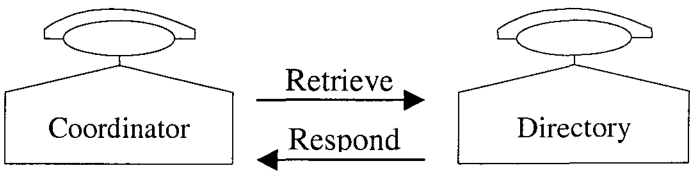


Fig. 3.5(a). Coordinator locates the appropriate cohorts

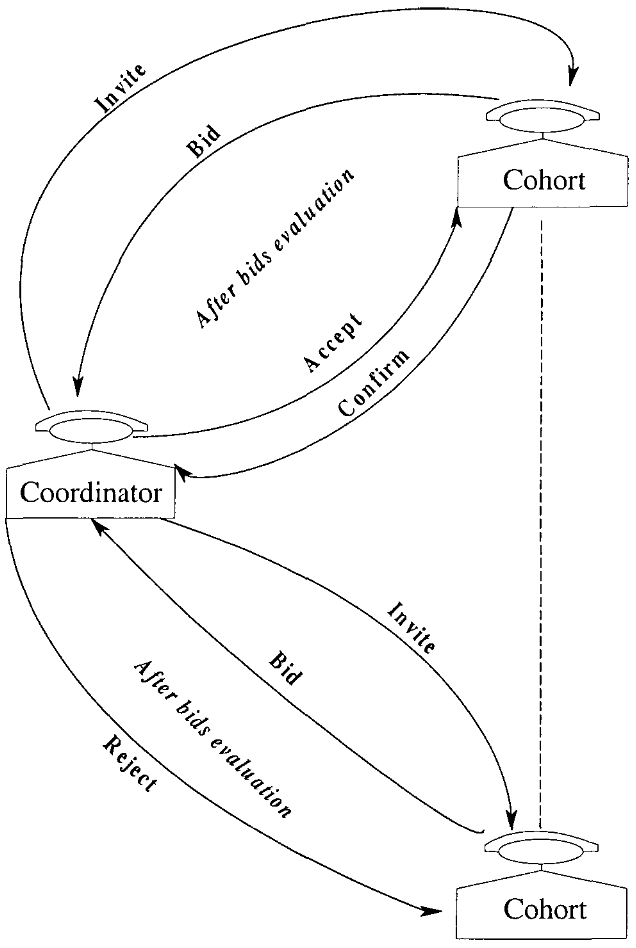


Fig. 3.5(b). Coordinator selects final cohort through negotiation

Fig. 3.5 Cohort selection process

3.5 Summary

In this chapter we outlined the CKBS approach for solving distributed real-world problems. The CKBS model is based on an engineering paradigm in contrast to the DAI/MAS approach where models are based on human behavioural concepts such as belief, intention and desire. The CKBS approach is based on well-established computer science concepts. It uses a layered architecture where the details of distribution and inter-agent communication are hidden from the end-user. We have outlined the structure of the CKBS agent and the cooperation environment concepts found in the CKBS model, namely, cooperation, coordination, inter-agent interactions, cooperation strategy and distribution transparency.

In the next chapter we present a preference model to resolve preferential conflicts among cooperating agents.

Chapter 4

The Preference Model

In cooperative processing, agents work together as cohorts to solve a joint (i.e. global) task T, subdivided to lower-level subtasks (T_1 ... T_n) (see chapter 3). Each task has both its own exclusive resources and some shared resources. Tasks have constraints and a set of preferences. In the introduction to this thesis (chapter 1) we explained the notion of preferences. Such preferences are expressed on resources such as machines, time, and labour. We use preferences, specified by the task, to choose a satisfactory solution which preserves as many preferences as possible. It is not always possible to satisfy all preferences due to contention with those of other agents. In this chapter we present a market-based approach for preference exchange and processing. An overview of our approach is given in section 4.2, while the model details are presented in section 4.2, and an algorithm for implementing the model is given in section 4.3.

4.1 Overview

The decomposition of the global task into lower-level tasks (see above) is shown in Figure 4.1. The figure shows a global task, T, decomposed into lower-level subtasks, (T_1 ... T_n), which are in turn decomposed into a lower-level subtasks. A task can have other tasks as precedents (indicated by dotted arrows in the figure). As can be seen from the figure this can lead to a heterarchical task dependency structure.

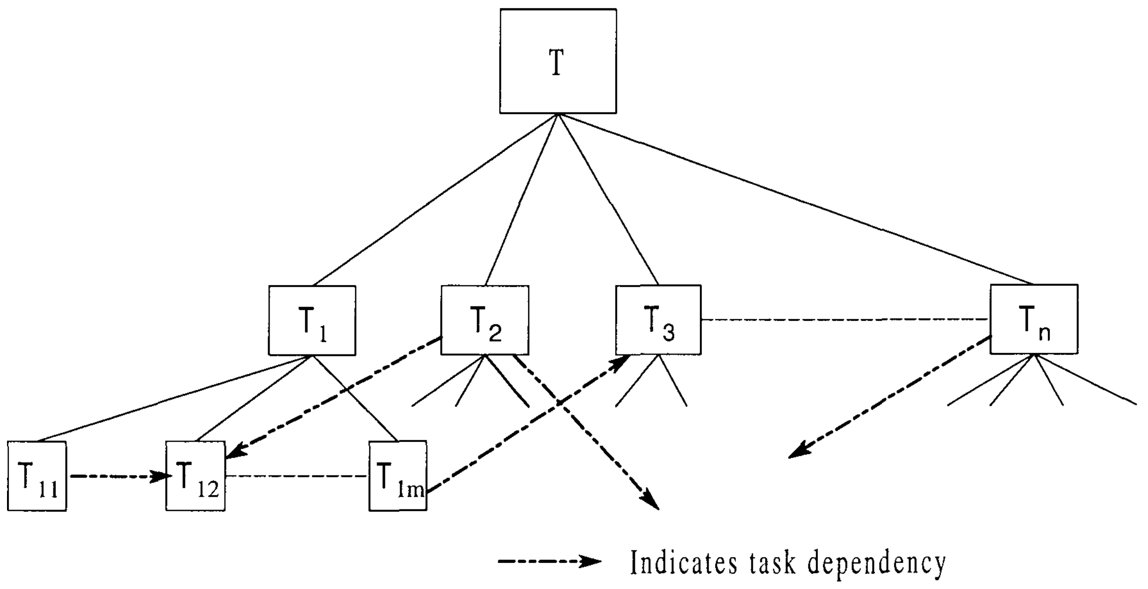


Fig. 4.1 Task decomposition & dependencies.

One agent might act as the task agent or the coordinator, responsible for the global task. If a resource has already been allocated to a task, and then if a later task arrives with a higher preference on that resource, then we may have to change the previous allocation. This leads to conflicts which can be represented in a network model, where the allocation to a node (task) affects all other nodes (tasks) that have preference on the same resource. The cascading effect of pre-emption and reallocation leads to branch explosions shown in figure 4.2. In this figure we show three subtasks, T_1, T_2, T_3, which are part of a global task T (not shown). Each subtask is decomposed into lower level subtasks. We assume that all subtasks are allocated. To reallocate subtask T_{15} requires reallocating T_{23} which in turn require reallocating T_{32}, which could require the reallocation of more subtasks.

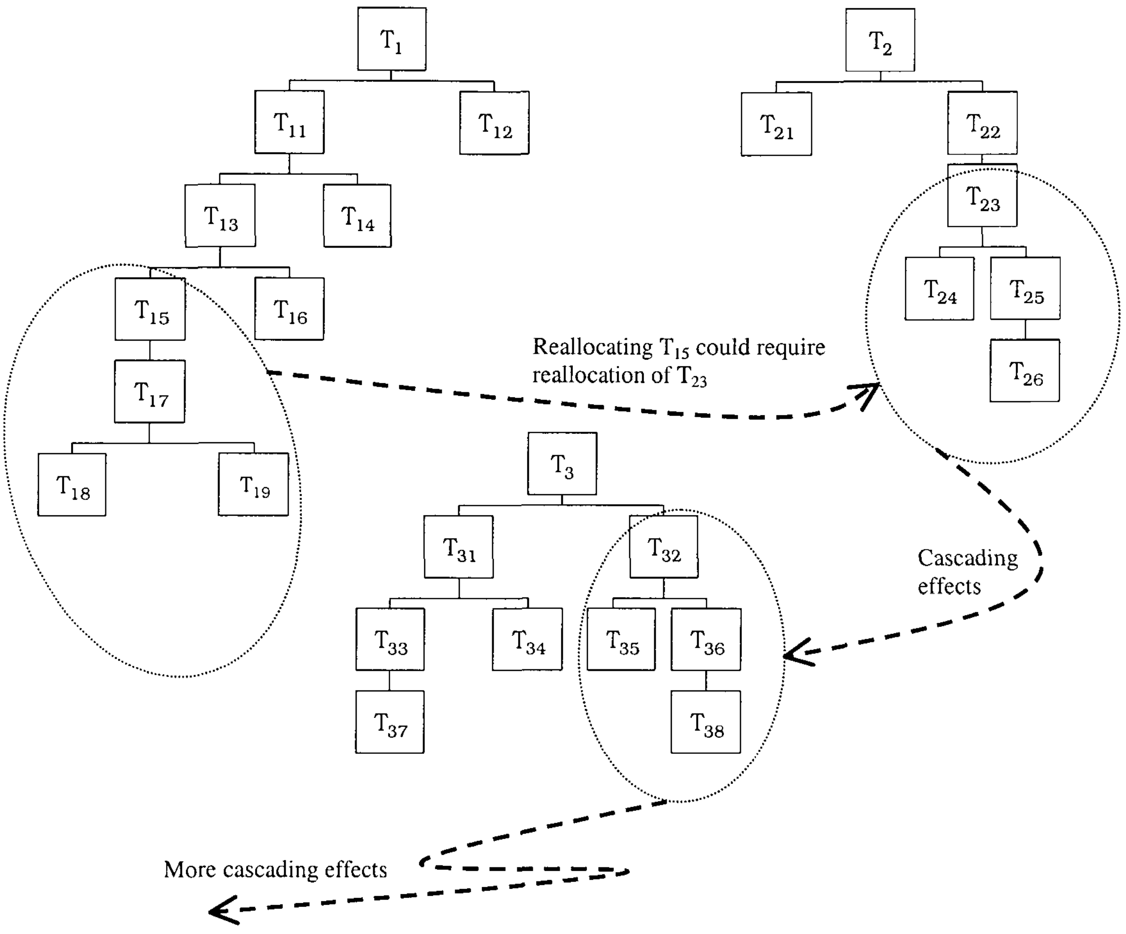


Fig. 4.2 The cascading effect of pre-emption and reallocation.

In this chapter we show how a set of agents can cooperate together to produce a solution that converges. This global solution space is the intersection of the local solution spaces. It may include more than one satisfactory global solution. In this case we accept the solution that preserves most preferences, we call this the best solution. It is often impossible to satisfy all the preferences due to one or more of the following reasons:

(i) contention with the preferences of other agents,

(ii) processing cost, and

(iii) intractability leading to non-convergence.

We shall also be presenting a market-based approach for preference exchange and processing, an outline of the allocation process, and an algorithm for implementing the preference model among cooperating agents.

4.2 Market-Based Approach

In order to assign preferences we categorise resources into two types:

- discrete, such as a machine or any physical items, and
- continuous, such as end-time, distance, and quality.

In discrete resources a list of alternative resources, each with its preferences, is specified. In contrast, in continuous resources an appropriate strategy for preference allocation is used, for example, if a preferred end-time cannot be met, the resultant end-time should be as close to the preferred end-time as possible. Preferences for both discrete and continuous resources for a task can be expressed as a number (i.e. a value), the higher the value, the higher the preference value. Since a user (through the coordinator of a task) can specify any preference values for the resources required by a task, an additional mechanism, in the form of what the user is prepared to pay to other agents for them to give up their preferences is introduced. We refer to these other agents as sellers. The sellers will sell their resources only if the price is right. An agent can accumulate such costs received and use them to buy preference values for itself later when needed. This market-based model also helps enforce convergence.

In some cases there could be resource type dependency. For example, task scheduling in machines, if a task has a preference on a machine and also on end time, the end-time will be dependent on the machine. In this case the evaluation process will have to evaluate both the combinations. If there are *r* dependent resource types, then the number of combinations will be *r!*. Thus resource type dependency will involve extra processing. In our subsequent discussion we will assume the preferred resource types of a task to be orthogonal to one another. By using this assumption we can restrict our discussion to a single resource without any loss of generality.

4.2.1 Preference Model

We define a preference satisfaction function (PSF) that returns a value to indicate the number of preference values that are not satisfied. If all preferences are satisfied then the PSF returns 0, otherwise it returns a negative value. PSF is expressed as follows:

$$PSF = -\sum_{i=1}^{n}\sum_{j=1}^{k} p_{ij} \qquad (4.1)$$

where p_{ij} is a preference loss function and can be expressed as follows:

$$p_{ij} = f(V_{ij}) \qquad (4.2)$$

where f is a user defined function on preference loss, V_{ij} is the value set by the task agent T_i for a preference j, k is the number of preferences, n is the number of tasks. We choose a positive cut-off (U) so that the acceptable *PSF* values are

$$-U \leq (PSF) \leq 0$$

For example if U has the value of 4 then the acceptable values of PSF are these between 0 and 4.

One way of expressing preference values in the case of continuous resources (e.g. end-time) is to use a user-defined *Preference Reduction Function*, ρ. ρ can be a fixed percentage reduction of the preference value, V, for each resource instance lower than the preferred instance. For example, if the resource type is time, and instances are hourly time-slots, then the time-slots on either side of the preferred time-slot have less preference value expressed as follows:

$$V(1 - s\rho) \qquad (4.3)$$

where s is the number of slots away from the preferred time-slot. For example if a preference value V is expressed for a 10 am time-slot, then it will be $V(1 - 2\rho)$ for the 8 am time-slot and $V(1 - \rho)$ for the 11 am time-slot. We can express ρ in other forms, for instance we can specify preference values separately for each resource instance of a resource type for a given subtask.

To avoid cycles that can lead to non-convergence we use a preference cut-off, φ. A solution is acceptable if it improves the overall preference value by φ. A smaller value of φ leads a larger number of exchanges and a higher final preference value. A larger value of φ leads to fewer iterations and a lower final preference value. The user sets the value of φ.

4.2.2 Cost Model

A task agent can specify very high preference values greedily. To control this greed, we use a market based cost model. The task agent must state how much it is prepared to pay to achieve its preference, thus a task T can be presented for T as:

$$T:: [(P_1/V_1/O_1, P_2/V_2/O_2, \ldots\ldots P_n/V_n/O_n]$$

where each triplet $P_i/V_i/O_i$ represents preference value V_i on resource P_i for which the task agent is willing to offer price O_i. Observe that O_i is the offer price in a negotiation, which is paid by this task agent to another agent in proportion to the percentage of V_i met by that agent (see later).

A number of alternative solutions are usually available for the task agent. Each solution is associated with a certain cost, which is meant to be covered by the offer price.

We use two types of cost: initial cost C_I, and a refinement cost C_R. C_I is the cost of finding a solution (acceptable PSF value) for the task without taking preferences into account. Such a solution might have satisfied some or all the preferences. If the minimum amount of preference values indicated by the cut-off value is not met, a refinement is needed to gain more preference values. In the refinement process, this task may be given a preferred resource, removing that resource from other tasks (some resources can be shared by several tasks). The cost of finding alternative resources for those affected tasks is the refinement cost C_R. There can be a cascading effect involving successive re-allocation due to task dependencies. The sum of C_R and C_I form the total cost C_T, This cost is meant to be covered by the offered price (O_i), as implied earlier.

Initial selection of candidate exchanges with likely preference gains and costs are made on estimates. The actual preference gain is found when the exchanges are actually carried out, and at that point the best result (the best gain) is selected, and payment from the offer price is made pro-rata to [(the actual preference gained)/ (the originally desired preference gain)], as indicated earlier. This cost is also used to terminate a branch if the agreed cost is exceeded.

Each agent can accumulate the payment it has received to pay for its preferences in the future. C_T can be less than the offer price, depending on the negotiation but can not exceed it, in other words $C_T \leq O_i$. The total cost C_T is expressed as follows:

$$C_T = C_I + \sum_{i=1}^{i=m} C_R \tag{4.4}$$

where m is the number of refinements needed to find a solution.

4.3 Algorithm

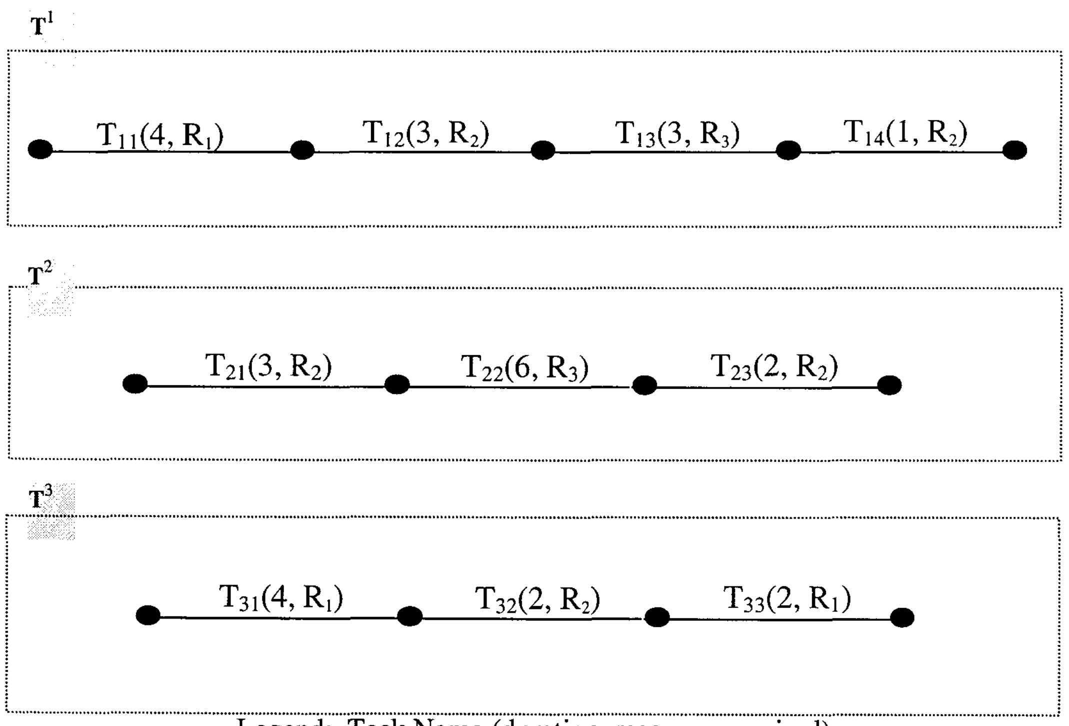
Using the market-based approach and the concepts outlined in the preference model described in the previous sections, we propose a scheduling algorithm that can be used by cooperative autonomous agents to resolve contention in the preferences for the same resource during subtasks scheduling. The following algorithm is a high level overview of implementing the preference model. An illustrative example to describe this algorithm is discussed in section 4.3.1, the lower level details are discussed later in section 4.3.2, and a practical implementation is discussed in more details in chapter 6.

```
Get an initial solution
  If(initial solution satisfies preferences cut-off)
            Accept solution
  else{
        Seek new allocation by negotiating with
        other coordinator agents.
        Find the gain in preference values and costs
        of new solutions.
        Find a solution with maximum (PSF)&(PSF $\geq$
        U)&($O_I \leq C_T$)
        If (such solution exists)
            Accept the obtained solution
        else
            Accept the initial solution
  }
```

4.3.1 Example

To illustrate the concepts outline in the preference model we introduce the following simple example. For the sake of clarity and not to lose focus, we use an example where each task is decomposed into one level of subtasks, and show the reallocation of two tasks. This should be adequate to illustrate the main concepts of the preference model.

In this example we assume that a task T is decomposed into three subtasks (T^1, T^2, and T^3) which are further decomposed into more subtasks, as shown in Figure 4.3. Three resources (R_1, R_2, and R_3) are required by the subtasks. The duration and the resource required by each task is shown in the figure, for example T_{11} requires resource R_1,and its duration is 4 slots.



Legend: Task Name (duration, resource required)

Fig. 4.3 The global task decomposition used in the example

To find an initial schedule we can use any of the various scheduling technigues used for optimal resource usage such as the Critical Path Method (CPM) [EPPEN84]. As our focus in this research is on satisfying as many task preferences as possible, it does not matter what technique we use for initial allocation. Therefore, we allocate tasks on a first come first serve basis. One possible schedule is as the one shown in Figure 4.4.

Resources	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
R_1	T_{11}	T_{11}	T_{11}	T_{11}	T_{31}	T_{31}	T_{31}	T_{31}			T_{33}	T_{33}			
R_2	T_{21}	T_{21}	T_{21}		T_{12}	T_{12}	T_{12}		T_{32}	T_{32}	T_{23}	T_{23}	T_{14}		
R_3				T_{22}	T_{22}	T_{22}	T_{22}	T_{22}	T_{22}	T_{13}	T_{13}	T_{13}			

Slots

Fig. 4.4 Initial allocation of all the tasks

Assume that the specified preference value, offer price and the preferred end-time slot for each task is as shown in Table 4.1. For simplicity reasons the end-time preference are stated for the global task (T^i), though preferences can be given for each subtask.

Table 4.1 Tasks Attributes (Figure 4.3)

Task	Preference value	Offer price	Preferred end-time slot
T^1	30	40	12
T^2	30	40	10
T^3	70	60	8

Table 4.1: This Table shows the task attributes from Figure 4.3

During the initial allocation, as shown in Figure 4.4, T^1 achieved its preferred end-time slot (slot 12); i.e. its preference value is totally satisfied, while the preferred end-time slot for T^3 is shifted three places (namely slot 11). Satisfying the total preference value of T^3 requires negotiation with T^1 to give up some of its satisfied preference value by reallocating its subtasks. Reallocation of T^1, according to the preference model, can only take place if the following conditions are satisfied:

1. Potential gain in total preference is greater or equal to the preference cut-off, φ.
2. The cost of reallocation is less than or equal to the T^3 offer price.
3. T^3 has enough funds to pay for the reallocation cost to T^1.

Therefore, we need to calculate the cost and preference gain using the equations described in the previous sections. Assuming the initial cost (C_I) is 5 units and the reallocation cost (C_R) is 10 units, the total cost (C_T) is :

$$5 + 10 \times 3 = 35 \text{ Cost units (using Equation (4.4))}$$

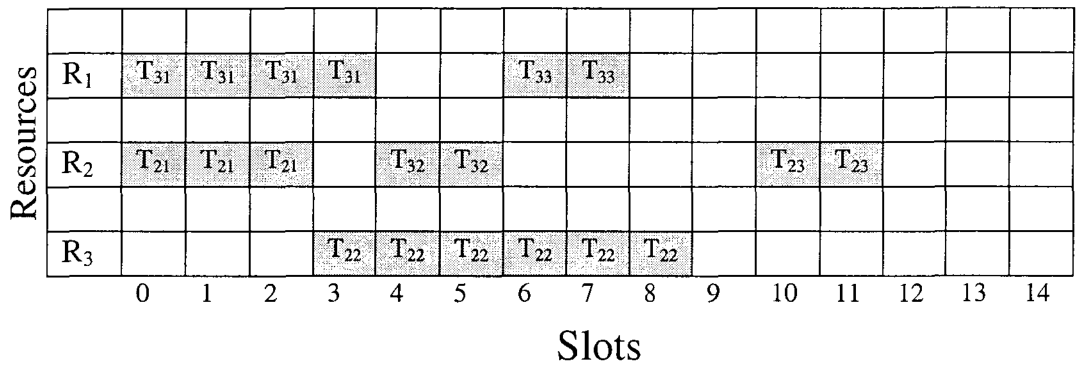
where 3 is the number of reallocated subtasks. Assuming the preference reduction function (ρ) is 0.1, the initial preference value for T^3 is:

$$70(1 - 0.1 \times 3) = 49 \text{ Preference value units (using Equation (4.3)).}$$

Therefore, if T^3 is reallocated to slot 0 the gain in preference value will be 21 (70 - 49). To estimate the overall gain, we need to calculate a new preference value for T^1. It is difficult to predict the new allocation of T^1 precisely, as this depends on the current status of all the tasks and the lack of the current knowledge available to the tasks. We assume that T^1 will be shifted to the right by the duration of the longest subtask (in this example, T_{33} is the longest duration, so T^1 is assumed to be shifted 4 slots to the right). Therefore, the predicted new preference value for T^1 is:

$$30(1 - 0.1 \times 4) = 18 \text{ Preference value units (using Equation (4.3)).}$$

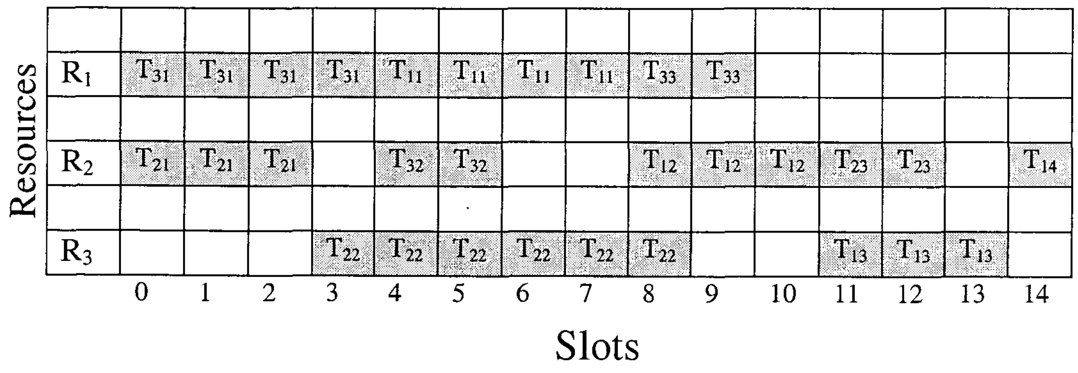
The predicted preference loss for T^1 is then 12 (30 - 18). Thus, the overall gain will be 9 (21 - 12) if this reallocation occurs. Assuming $\varphi \leq 9$, the reallocation takes place. The resulting schedule with T_3 of this reallocation is shown in Figure 4.5, (note T^1 is not yet reallocated). T^3 pays the reallocation cost T^1. Faulty predictions that cause drops in the overall preference value can be corrected in the subsequent iterations.


Resources
Slots

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
R_1	T_{31}	T_{31}	T_{31}	T_{31}			T_{33}	T_{33}							
R_2	T_{21}	T_{21}	T_{21}		T_{32}	T_{32}					T_{23}	T_{23}			
R_3				T_{22}	T_{22}	T_{22}	T_{22}	T_{22}	T_{22}						

Fig. 4.5 Reallocation of T^3 (T^1 not yet reallocated)

Reallocating T^1 using the first available slot yields an end-time for T^1 at slot 17 (T_{11} finishes at slot 11, T_{12} finishes at slot 14, T_{13} finishes at slot 17, and T_{14} finishes at 18). T^1 then needs to negotiate with T^3 and T^2 to regain all or some of its preferences. Subject to the aforementioned conditions, one option is shifting T_{33} two slots and shifting T_{23} one slot, both to the right as shown in figure 4.6. This can occur if all the aforementioned conditions are satisfied. T^1 pays both T^2 and T^3 the reallocation cost of two subtasks (T_{23} and T_{33}).


Resources
Slots

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
R_1	T_{31}	T_{31}	T_{31}	T_{31}	T_{11}	T_{11}	T_{11}	T_{11}	T_{33}	T_{33}					
R_2	T_{21}	T_{21}	T_{21}		T_{32}	T_{32}			T_{12}	T_{12}	T_{12}	T_{23}	T_{23}		T_{14}
R_3				T_{22}	T_{22}	T_{22}	T_{22}	T_{22}	T_{22}			T_{13}	T_{13}	T_{13}	

Fig. 4.6 Reallocation of T^1

Further negotiation can take place until no further gain in preferences can occur, then the iterations terminate.

4.3.2 Allocation Process

Below we shall use the capital letters U for coordinators (as user agents), T for global tasks, A for agents (i.e. cohorts), and R for the remaining preference values. The letters a, b, c used as superscripts in U and T are used to imply the specific instances of coordinators and global tasks, and the subscript letters i, j, k, are used to represent subtasks.

Initially the coordinator allocates preferences to its tasks, with the help of the cohorts, via a contract net protocol (CN). It invites all cohorts to bid for the allocation of the resource instances to all the tasks of the coordinator, which can then select the best bids. This allocation, called initial allocation *(Phase I),* is made without any pre-emption of resource instances and without using any cost or preference cut-offs, but otherwise an attempt is made to gain as much preference values as possible. This is iteration 0, for which no cost is paid to the cohorts, but each task gets its initial allocation. The agent that allocates a resource to a subtask becomes the *home agent* of that subtask. During phase I each subtask gets its home agent, which can change subsequently if the subtask is allocated another resource instance by another agent. In that event, this new agent becomes the new home agent for that subtask.

Once this Phase I is completed, each cohort takes over the reallocation of its tasks either to itself or to other cohorts (depending on the preferences of the tasks), using the coordinators for negotiation on cost. If a task is successfully reallocated to another cohort, the new cohort will take it over. The following are the steps in Phase II:

Step 0: [*setup*]

Coordinators are sorted according to their ID. Cost parameters are set and fund F is allocated to each coordinator. This assumes that a fund F is given for all the subtasks of the coordinator concerned, rather than individually to each subtask.

Step 1: [*start allocation process*]

Select the first coordinator in the ID order, say U^a. This selection is arbitrary, any coordinator can be selected.

Step 2: [*sort subtasks of current coordinators*]

All the subtasks of the current coordinator are sorted in decreasing order, using the remaining preference values ($R^a{}_i$), so that the cost available can be put to maximum benefit. This order will be referred to below as *the remainder order.* The coordinator selects the first subtask.

Step 3: [*resource reallocation*]

The coordinator then asks the relevant home agent A_i to reallocate the resource type.

Step 4: [*select a subtask*]

A subtask, along with its preferred resource instance is selected in the remainder order. Say this is $T^a{}_{iu(x)}$ wishing to acquire another instance y of resource u to gain preference $G^a{}_{iu(y)}$.

Step 5: [*invite bidding*]

The cohort invites all coordinators, including U^a, to bid.

Step 6: [*tentative bids*]

All coordinators having a subtask in possession of the resource instance z of u may bid. But if it has a subtask holding instance $y2$ of u (not z of u) such that $z > y2 > x$ *(in preference order)*, then it can bid for $y2$ with the potential preference gain $G^a{}_{iu(y2)}$ instead of $G^a{}_{iu(z)}$. Either way, we say the sacrificial task for this seller coordinator is T_j. Each coordinator will evaluate the precedent tree of T_j , and thus calculate the expected cost and the potential preference gain (which can be $G^a_{iu(y2)}$ instead of $G^a_{iu(z)}$), which are then sent them to the home agent concerned. This is a tentative bid, as there is no guarantee that the successful coordinator U^b will actually be able to deliver the bid.

Step 7: [*bid failures*]

If no coordinators have any bid, then this is *a bid-failure* (see section 4.2.4), and in that case the processing moves to Step 12. Otherwise, the home agent forwards the bids to the coordinator U^a.

Step 8: [*effect of cost cut-off μ or cost run-out*]

The coordinator U^a accepts the best bid that it can afford and advises the home agent to proceed with it. Say it is the task $T^b_{ju(y3)}$ and preference gain $G^a_{iu(y3)}$ for instance y3. This $G^a_{iu(y3)}$ is now *the expected gain.* The coordinator U^a may use a cost cut-off μ, that is, it may decide not to pay for any bid if the expected cost is > μ. Also, it may not have enough funds left in F^a to pay for this bid. In either case, the bid fails and control moves to step 12.

Step 9: [*single bid pre-selection*]

The home agent asks the coordinator U^b of subtask T^b_j to proceed. This step is refered to later as *single bid pre-selection.*

Step 10: [*reallocation of the subtasks*]

The coordinator U^b asks the relevant home agent, say A_j to reallocate. This cohort will ask the home agents of the precedent tasks of T^b_j to reallocate them. These will be recursive steps, carried out from the root (T^b_j) down to the leaves. We do not show them, but they have been described in the simulation (chapter 7). A further important point is this: although $T^b_{ju(y3)}$ has given up its resource instance y3, it may not be possible to secure y3 for task T^a_{iu} (e.g. if some precedent subtasks of T^b_j cannot be moved). In that event T^{a}_{iu} might end up getting some other lower instance of u, say $y < y3$, with less gain $G^{T}_{iu(y)} < G^{T}_{iu(y3)}$. In fact $G^{T}_{iu(y)}$ can be even be unacceptable to U^{a} (see below).

Therefore this agreement by U^{a} with U^{b} for this exchange is a gamble. It may be that some other rejected bidders could have provided a better gain. One may wonder if is it not possible to avoid this gamble by asking, in Step 9, for all the bidders [*multiple bids pre-selection*] to go ahead with their bids and then selecting the best from the final results. This should be possible, subject to the cost availability. However, if these bidders work in parallel, it will not work in general, since each bidder will potentially get in the way of the other bidders with contentions on the resources required by their respective precedent tasks. If the bidders work serially, then it should work, except for the high processing cost (including the time cost). The user can control the cost and exercise a choice. Once the best bid is selected, the rest of the processing will be the same – assuming the best bid to be the same as above, that is, U^{b} offering originally resource instance y3 of $T_{ju(y3)}$ with expected preference gain $G^{T}_{iu(y3)}$, but finally offering resource instance y with actual preference gain $G^{T}_{iu(y)}$.

Step 11: [*possible outcomes*]

Three possible outcomes can occur after processing step 10:

(a) The attempt fails due to the use of the preference cut-off φ. In this case iteration number, task and amount of the gain is recorded. The process returns to Step 12.

(b) The attempt succeeds partially, that is, instead of resource instance y3 it gets instance y, with preference gain $G^{a}_{iu(y)}$, The cost paid will be pro rata to the gain. Further explanation of this outcome is shown below.

(c) The attempt succeeds completely, getting the resource instance y3 with gain $G^{a}_{iu(y3)}$.

Further explanation on outcome (b):

If $G^{T}_{iu(y)} \leq \varphi$, where $G^{T}_{iu(y)}$ is the net preference gain by the whole system then we have subcase (b1), or else (b2).

Subcase (b1):

In this case the exchange has failed, and the processing returns to Step 13. We shall refer to this preference loss as being due to *delayed effect of preference cut-off* φ, or simply delayed preference cut-off.

Subcase (b2):

In this case the exchange is acceptable. But in this exchange the coordinator U^{b} might (but not necessarily) lose any preference value; that is, $G^{b}_{ju(y)\ (I)}$ for this iteration I could be negative.

The exchange has gained preference value $G^{a}_{ju(y)\ (I)}$, although it was expecting to gain $G^{a}_{ju(y3)\ (I)}$. This loss $\pi = G^{a}_{ju(y3)\ (I)} - G^{a}_{ju(y)\ (I)}$, which we shall refer to as the π loss.

The fund F^{a} of U^{a} would lose pro rata and those of the other coordinator involved will gain.

Step 12: [*check all subtasks of the current coordinator are processed*]

If all the subtasks of the current coordinator have been allocated, then the processing moves to Step 13, else it returns to Step 3 for the next subtask of the current coordinator.

Step 13: [*check all coordinators are processed*]

If all the coordinators have been processed, then processing moves to Step 14, or else it returns to Step 2 for the next coordinator.

Step 14: [*check preference gain*]

If there was no preference gain for any task in the system during the last iteration, then the process stops, otherwise it increments the iteration number by one and returns to Step 1.

The flow diagram shown in figure 4.3 shows an outline of the allocation process described above.

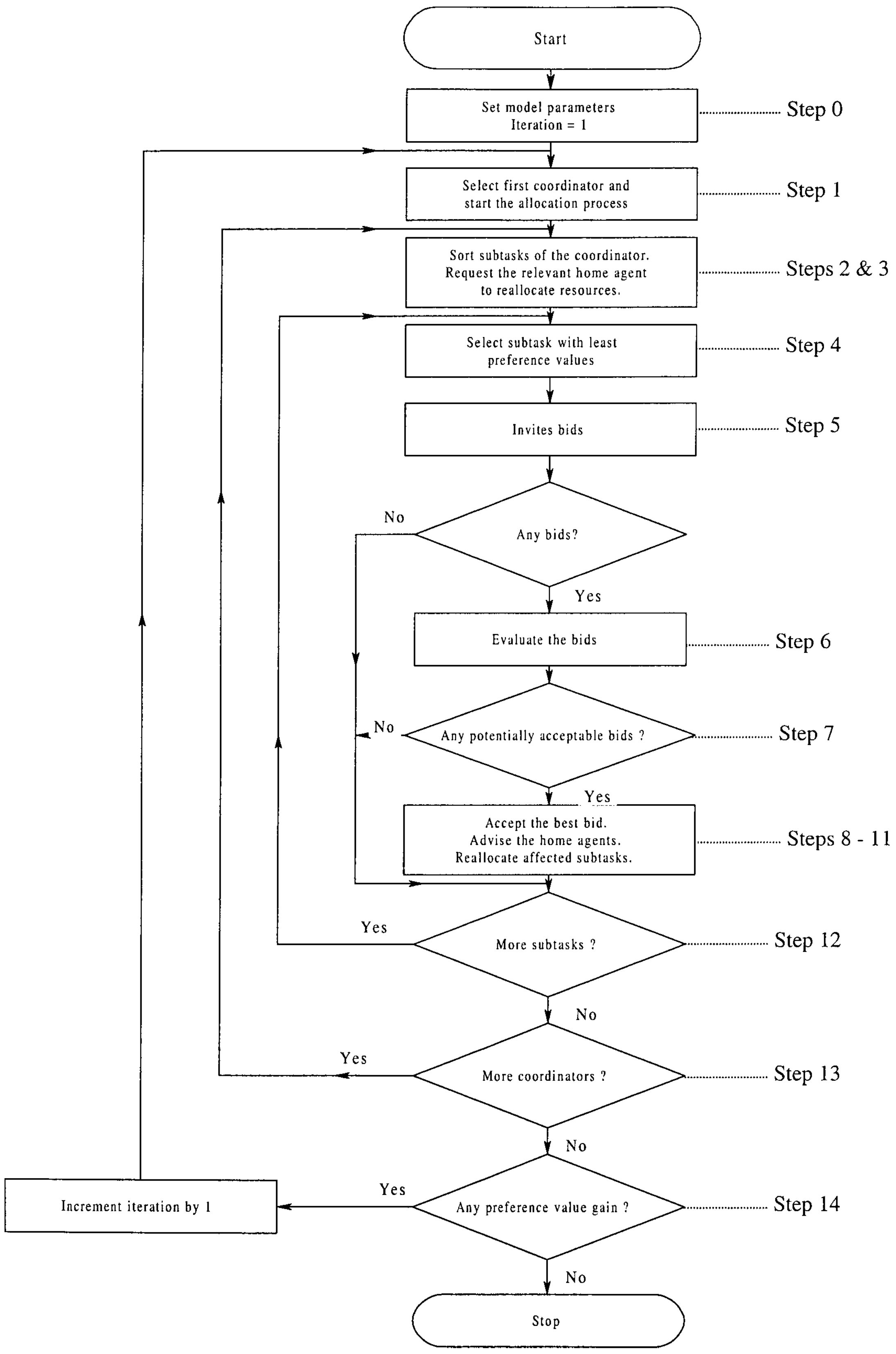

Start
Set model parameters
Iteration = 1
Step 0
Select first coordinator and start the allocation process
Step 1
Sort subtasks of the coordinator. Request the relevant home agent to reallocate resources.
Steps 2 & 3
Select subtask with least preference values
Step 4
Invites bids
Step 5
Any bids?
No
Yes
Evaluate the bids
Step 6
Any potentially acceptable bids ?
No
Yes
Step 7
Accept the best bid. Advise the home agents. Reallocate affected subtasks.
Steps 8 - 11
More subtasks ?
Yes
No
Step 12
More coordinators ?
Yes
No
Step 13
Any preference value gain ?
Yes
No
Step 14
Increment iteration by 1
Stop

Fig. 4.7. Outline of the allocation process

4.3.3 Comments on the allocation process

In the allocation process steps shown in the in the previous section, the home agent (see previous section) handles the resource allocation on behalf of the requester coordinator, contacting other coordinators and cohorts as necessary. The requester coordinator can also carry out this allocation directly, by inviting other coordinators to bid themselves. This will be an acceptable alternative, except that it could arguably be more cumbersome, since a coordinator will have to ask the cohorts for allocations, since only they can allocate and have the direct knowledge of which resources are held by which tasks. Therefore it seems more efficient for the home cohort to carry out the allocation directly, with the help of other cohorts as outlined above.

Also, instead of allocating one resource at a time, it is possible to allocate all the resources of a task at the same time (i.e. in the same loop) by inviting bids for all of the resource types in Phase II. In that case the coordinator of the task will ask all the home agents of the task, and each home cohort will invite the relevant coordinators, in parallel, to bid for its resource type. Eventually the bids will be forwarded to the requestor coordinator, which will tentatively accepts the best bid, subject to the availability of funds. The cohorts will execute and present the actual outcome of each resource type that meets the preference cut-off to the requester coordinator, which will pay pro-rata for each outcome.

Preference loss

In the steps outlined above we can attribute the preference loss either to the resource conflicts or to the lack of funds, these are discussed below.

I- Preference loss due to resource conflicts

We identify four sources of preference loss, these sources are:

(a)Bid failure *(Step 7)*

(b)preference cut-off φ *(Step 11(a))*

(c)Delayed cut-off φ *(Step 11(b1))*

(d)The π loss *(Step 11(b2))*

The bid failure (a) is purely due to the conflicts of subtasks on resources. The conflict is so severe that no movement is possible. If there were no conflict, we would have gained all the preference values, and if we did, then we would not have the sources b, c, and d either. The failure to exceed the preference cut-off, in both (b) and (c), implies conflicts are severe but not as bad as in (a). The situation in (d) is much better as the preference gain exceeds the cut-off, but not as much as expected. Therefore the conflicts on resources are better than in (b) and (c). These resources are not independent, the failure of a subtask to gain preference value due to any one of these four reasons might have enabled it to make much better gain later.

II- Preference loss due to lack of funds

This preference loss is due to cost cut-off *(Step 8)*. There are two separate situations where preference loss will occur due to lack of funds (i) when some exchanges are disallowed as being too expensive, this is to force a fast termination (μ cut-off) and (ii) when funds run out at later iterations (cost run-out). It is not easy to predict which exchanges would be considered too expensive and when the funds would actually run out, particularly since the coordinators both gain and lose funds from each other like money in a market during the processing.

4.4 Summary

This chapter presented a preference model to be used by cooperative agents that are working together towards solving a global task. The preference model is used to resolve conflicts among the cooperating agents. A preference satisfaction function (PSF) is defined, it returns a value that indicates the total preference values that are not satisfied. A cost model is presented. This model is used to ensure that a timely convergence, towards an acceptable solution, can occur. An allocation process, in which tasks are allocated for the agents according to task preferences, is shown. We presented a high level view of a scheduling algorithm based on this preference model and the allocation process. In the next chapter we present a mathematical formulation for this preference model, while in chapter 6 we outline an implementation of the algorithm presented in this chapter.

Chapter 5

Theoretical Distribution Model

Because the preference model, explained in the previous chapter, is highly nonlinear, verification using available mathematical techniques is difficult and hence it is quite hard to make a quantitative analysis. It is not possible to predict the preference values that can be achieved after executing phase I of the allocation process. We cannot relate preference values, cost elements and preference cut-offs in any mathematical formula, as these are arbitrary user choices. However, the allocation process described in the previous chapter is meant to improve the final global preference gradually over many iterations, with the preference gain G decreasing per iteration as the iteration I increases. In this chapter we try to model the qualitative behaviour of the preference gain over the iterations. We present a formula that can roughly estimate the minimum remaining preference value. We study the effect of the clustering of requests for the same resource instances. The effect of the uniform distribution of these clusters on the developed formula is discussed first. Then we show how the formula is affected by the non uniform distribution of these clusters (skewed distribution).

5.1 Basic Formulation

During the allocation process, described in chapter 4, the final global preference gain, G for all tasks, gradually decreases per iteration as the iteration I increases. This process will converge to a minimal value, R^T, which is the total remaining preference value. This we can describe by an exponential distribution function of the form:

$$R_I = A + Be^{-\lambda I} \qquad (5.1)$$

where A is the height of the plateau when $e^{-\lambda I}$ tends to zero, and hence $A = R_{min}$, the remaining minimum preference value at $e^{-\lambda I} = 0$. Constant B is R_{max}, which is equal to $(R_I - R_{min})$ and constant λ gives the curvature of the distribution and is related to the rate of preference gain. Thus the equation can be written as follows:

$$R_I = R_{min} + R_{max}\, e^{-\lambda I}. \qquad (5.2)$$

A graphical presentation of equation (5.2) is shown in Figure 5.1. for arbitrary values of R_{min}, R_{min}, and λ.

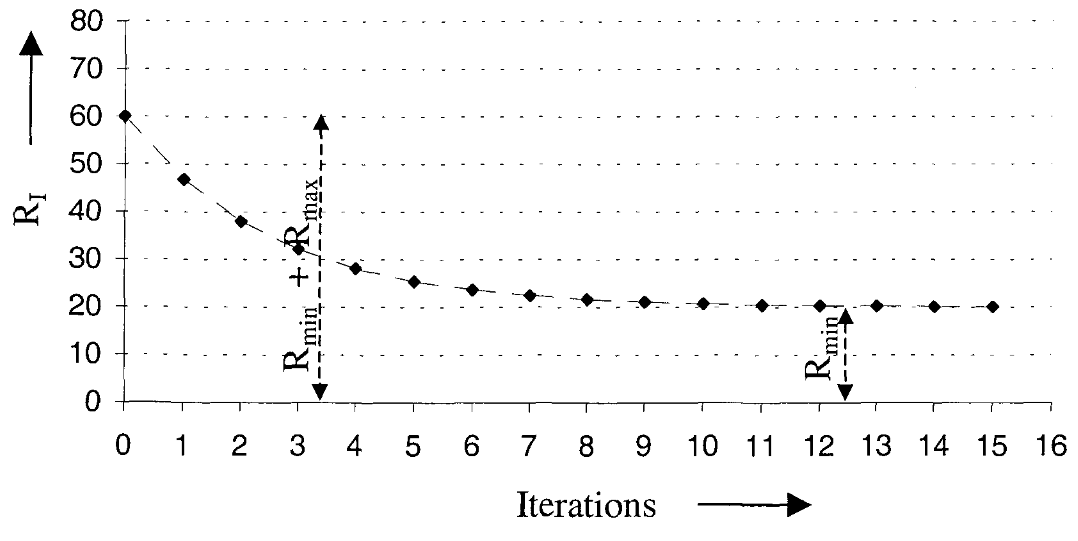


Fig. 5.1 A graphical presentation of the decreasing rate of preference gain

It can be seen that at I=0, that is after the initial allocation, where allocations are made without considering preference (see 4.2.3), $R_0 = R_{min} + R_{max}$. At I = 1, $R_I = R_{min} + R_{max} e^{-\lambda}$, and as I increases, i.e. $e^{-\lambda I} \rightarrow 0$, $R_I = R_{min}$.

In order to estimate a value R_{min} we have found that it is mainly affected by the distribution of preferences for the same resources. Next in this section we try to formulate an estimation for the value of R_{min}. In this formulation we use two values, the distribution density, d, and the *Preference Reduction Function*, ρ, that was described in the previous chapter. If we have t subtasks and all have preferences over the same m(<t) preferred resources, then the density d = t/m.

We assume we have t=2h subtasks distributed over m=2n resources (see figure 5.2). At each resource instance, as a time-slot, preference values will decrease on either side of the most preferred time-slot, this decrease is given by ρ. In order to find the preference loss formula we need to evaluate the average movement from slot 1 at the right side of the midpoint m_l for the right half slots and hence half the subtasks (t/2).

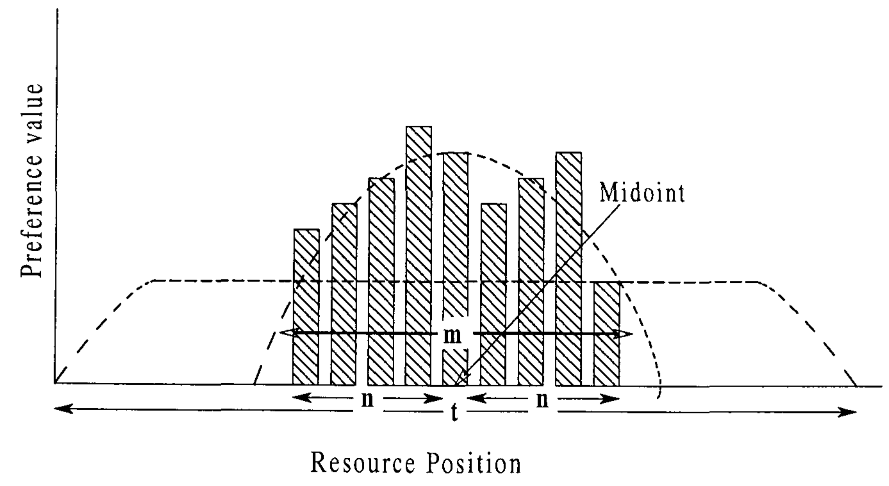


Fig. 5.2 Preference distribution

The first d subtasks at slot 1 move to slots 1,2,3,...d, with displacements [0, 1, 2, .., d-1], then the displaced d subtasks move to slots d+1, d+2, .. 2d, with displacements [d-1, d, d+1, .. 2(d-1)], and then the displaced d subtasks move to slots 2d+1, 2d+2, .. 3d, with displacements [2(d-1), .. 3(d-1)], and so on. Generally speaking:

The displaced d subtasks move to slots (n-1)d+1, (n-1)d+2, .. nd, with displacements [(n-1)(d-1), (n-1)d, (n-1)(d+1), .. n(d-1)].

The summation of the displacement of each sets of d subtasks gives:

(1) [0, 1, 2, .., d-1] = (d-1)d/2

(2) [d-1, d, d+1, .. 2(d-1)] = [(d-1)+2(d-1)]d/2= 3(d-1)d/2

(3) [2(d-1), .. 3(d-1)] = [2(d-1)+3(d-1)]d/2= 5(d-1)d/2

......

.....

(n) [(n-1)(d-1), .. n(d-1)] = [(n-1)(d-1)+n(d-1)]d/2= (2n-1)(d-1)d/2

If we sum the right-hand side we get

$$(1 + 3 + 5 + 7 + \ldots.. + (2n-1))(d-1)d/2 = [1 + 2n - 1](n/2)(d-1)d/2 = nn(d-1)d/2 \quad (1)$$

This also applies for the subtasks in the left half, hence the total displacement for the whole distribution is.

$$\text{Total} = nn(d-1)d \quad (2)$$

To get the average movement per subtask we divide by t = 2nd, and the average movement per subtask then becomes:

$$nn(d - 1)d / (2nd) = n(d - 1) / 2$$

If the preference loss per unit shift is ρ, then the preference loss per subtasks is:

$$\rho n(d - 1)/2 \tag{3}$$

which can be taken as the percentage preference loss for the subtasks in a target agent.

5.2 Skewed Distribution

In the previous section, we assumed that there are the same number of slots (h) available on either side of the midpoint. Let us suppose there are only y slots on one side, say the left side of the midpoint m_1, where y<h. This will lead to a skewed distribution as shown in figure 5.3.

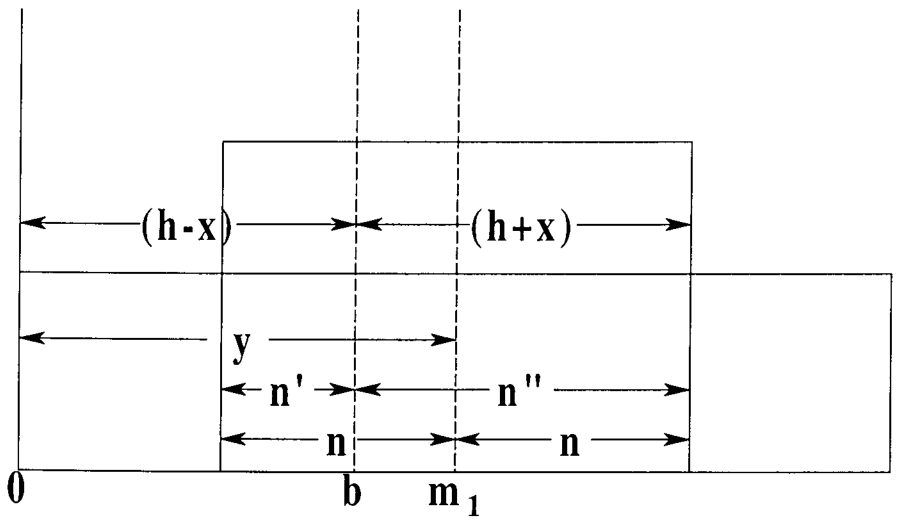


Fig. 5.3 Skewed distribution

We divide the m preferred slots into two unequal groups by a break point **b**, with a left group of width n' and a right group of width n", such that:

$$n' / n'' = (h - x)/(h + x) \tag{4}$$

where (h - x) = (y - n + n') is the number of slots available on the left side of the break point b. Note $y < h$, $n' + n'' = 2n = m$, therefore :

$$n'' = 2n - n'$$

$$x = h - y + n - n'$$

Replacing n" and x in Eqn (4)

$$n' / (2n - n') = (y - n + n') / (2h - y + n - n')$$

Solving this for n' we get:

$$n' = n(y - n) / (h - n) \qquad (5)$$

To evaluate the effect of this skewed distribution, we use eqn (1), replacing n by $n' = (h - x)/d$ for the left side and n by $n'' = (h + x)/d$ for the right side, then this eqn becomes:

$$\begin{aligned}\text{Total} &= n' n'(d - 1) d/ 2 + n'' n'' (d - 1) d/ 2\\ &= [(h-x)/d]^2 (d-1)/d/2 + [(h+x)/d]^2 (d-1)/d/2\\ &= (d-1)d[h^2 - 2hx + x^2 + h^2 + 2hx + x^2]/(2d^2)\\ &= (d-1)[h^2 + x^2]/d\\ &= (d-1)[(nd)^2 + x^2]/d\\ &= n^2(d-1) d + x^2 (d-1)/d\end{aligned}$$

Dividing it by t = 2h = 2nd, we get:

$$n(d - 1)/2 + x^2(d - 1)/2nd^2$$

Therefore the average shift

$$\rho[n(d - 1)/2 + x^2(d - 1)/2nd^2] \qquad (6)$$

The difference between equation (3) and equation (6) is the term $x^2(d - 1)/2nd^2$, and we refer to this term as the *correction term.* This term disappears when x=0, which is the case for a uniform distribution. It should be noted that when evaluating equation (6), in the case of uneven distribution, both m_l and b should be determined by counting the number of subtasks on the left and right side, rather than from the slots they occupy.

5.3 Effect of Multiple Target Agents

In the analysis discussed in the previous sections we only considered the case where there was one target agent, in this section we will consider the impact of having more than one target agent. If we assume the subtasks are to be distributed over **s** resources, where $s \geq t$, then the resultant distribution density **D** = s/m, in contrast to d = t/m.

The degree of impact of more than one target agent depends on task precedence dependencies. To determine the overall effect over q target agents we consider the following two cases: with no precedence dependencies and with precedence dependencies.

Case I: No precedence dependencies:

We assume here that there are no precedence dependencies among the subtasks, that is the subtasks do not have any precedence constraints and hence can be allocated freely. In that case we can calculate a weighted average of the total preference loss, which is the same as the remaining preference values, as follows:

$$R^T \text{ (case I)} = [\, P_1L_1 + P_2L_2 + \ldots\ldots + P_qL_q \,]/q \qquad (7)$$

where L_i is the preference loss (in percentage) pf agent A_i, calculated from equation (6) , and then P_i is the total preference value of agent A_i. This is the minimal loss based on density d_i in each agent A_i and hence is predicted to be the lowest limit.

Case II: With precedence dependencies:

In this case we take for each agent the number of slots over which the subtasks were distributed by phase I as representing an expression of dependency for that agent. We use this slot number to evaluate D, and we use this value of D in the preference loss evaluation as in case I. This is assumed to be the upper limit.

The final loss should lie between case I and case II. We should note that there will also be other factors such as preference cut-off, cost cut-off etc, which we have not taken into account in these estimates.

5.4 Summary

A theoretical formulation for the preference model was presented in this chapter. Two formulae were derived one for uniform distribution and one for uneven distribution, which can be regarded as a general case. Using the assumption whether precedence dependencies exists or not we have shown how to calculate the lower and upper limits for the remaining preference values when more than one resource is available.

In the following chapters we show our implementation for the preference model and the results collected from the experiments performed using this implementation. We compare those experimental results with those predicted using the formulae derived in this chapter. We have submitted this model and our results for publication in [DEEN03B].

Chapter 6

Design and Implementation

In the course of this research we developed a market based preference model to be used in resolving conflicts among cooperating agents. We designed and implemented a simulator for the preference model called the preference model simulator (PMS). We used distributed scheduling in manufacturing as a case study for testing the preference model characteristics. This chapter describes the software design and implementation of the PMS. We present the case study, along with the results obtained from the simulation study that is described in chapter 7.

6.1 Overview

The objectives for developing the PMS are as follows:

- To test the behaviour of the preference model described in Chapter 4.
- To compare the results obtained from the simulation with the theoretical results presented in Chapter 5.

The Cooperating Knowledge Based System (CKBS) approach, discussed in detail in Chapter 3 is used in our implementation of the simulator. As stated in Chapter 3, the CKBS approach is an engineering paradigm as opposed to the mentalistic paradigm of DAI/MAS. Effectiveness, reliability, performance, and usability are of great importance in CKBS.

Agents in CKBS are autonomous and cooperative. The preference model requires agents to be distributed and cooperative to achieve a common global goal through negotiation. For the aforementioned reasons, the preference model lends itself naturally to the CKBS approach.

During the design process we used the notion of classes and methods adopted by the object-oriented approach [BOOC94]. Thus enabling us to apply abstraction in modelling the various system components and easy transition from the design phase to the implementation phase.

We first present an agent design in section 6.2, then we discuss agent communication in section 6.3, and in section 6.4 we discuss the implementation of the PMS.

6.2 Agent Design

We based our agent design on the CKBS model described in section 3.2.1. While the main concepts of the CKBS agent are retained, some features, which are not necessary for the implementation of the preference model, are omitted. Our agent design provides the default behaviour and structures for all the different types of agents (section 6.2.1).

The agent relations are categorised into internal relations and external relations; this separation of relations into internal and external relations conforms to the schema layering outlined in section 3.4. The internal relations that relate to the agent's internal work are found in the home model while the external relations are the agent's interface to the outside world. The agent structure, shown in Figure 6.1, is divided into four parts: graphical user interface, communication components, public components and home schema of the agent. We discuss these agent parts below.

Interaction between the human user and the agent is performed via the graphical user interface (GUI). The user can utilises the GUI to change certain parameters such as initial budget, initial allocation cost, and reallocation cost, and also to monitor the agent status.

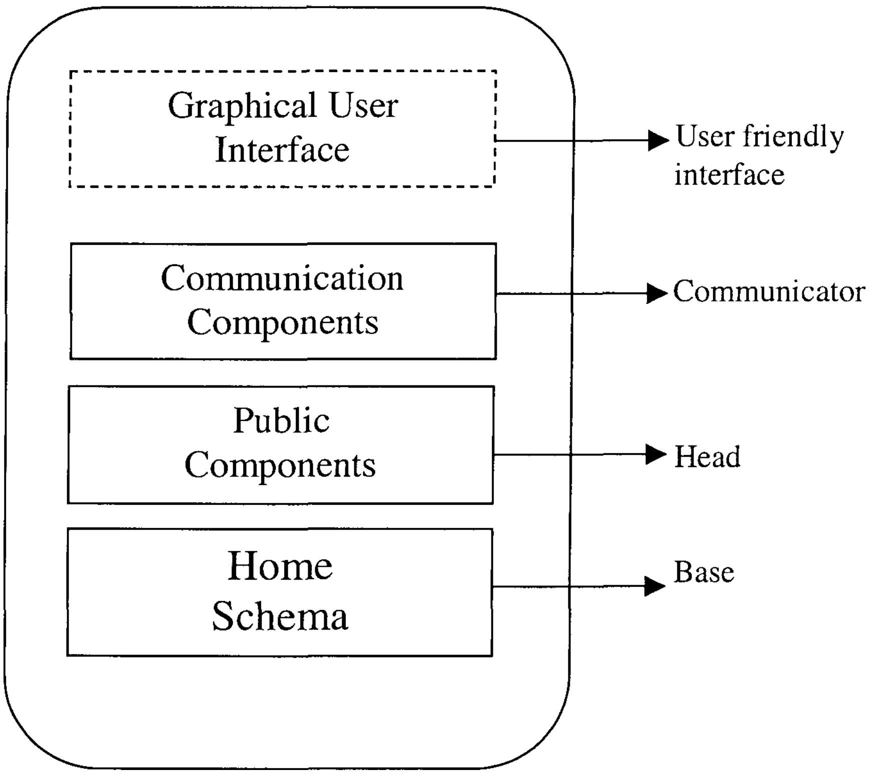


Fig. 6.1 Agent Structure

The communication layer is used by the agent to communicate with the outside world and is discussed in section 6.3.

The home schema includes the internal structures and the intrinsic functions needed for the execution of the subtasks. It forms part of the cohort architecture and is located at the base of the cohort.

The public component includes agent identity, a task schema that includes a dependency schema, and a preference schema. The identity holds the agent basic information and can be represented as follows:

```
Identity (
        ID
        Type
        Address
)
```

The ID attribute is a unique key that represents the agent. The type attribute reflects the agent type (see later) such as coordinator agent, home agent, etc. The address attribute is needed for inter-process communications between the agents. As the agents can be anywhere in the network, the address attribute represents the agent address in the network and its communication details.

We assume that the task is described and decomposed by the system user during the system execution. Each subtask has a set of attributes and can be represented as follows:

```
SubTask {
     Task ID
     Duration
     Predecessors;
     Successors;
     Resources;
     Preferences;
}
```

Task ID uniquely identifies each subtask held by the agent. Precedence schema is stored in the predecessors and successors attributes. The preference schema is stored in the preferences attribute. Subtask duration and required resources are stored in the duration and resources attributes respectively.

6.2.1 Agent Types

Each agent has a different function to perform in the system. In this section we describe each of these agents and the role it carries out. The basic features of each agent are inherited from the generic agent described above.

The Coordinator Agent

The coordinator agent is responsible for scheduling subtasks and cooperating with other coordinators during the allocation process. The coordinator agents use the allocation process of the preference model outlined in chapter 4 to trade preference values among cooperating agents. A coordinator agent is assigned for each task that is decomposed into subtasks.

The Home Agent

The home agent is responsible for executing the subtask. Each home agent can perform more than one skill. In the system there can be more than one home agent with the same skill. The subtask can specify a preference on a home agent to be used when carrying out the task.

The Directory Agent

The directory agent maintains a list of the agent identities that are currently running in the system. Also each agent keeps a local directory of those agents that it contacted recently. This to reduce the number of communication in the system.

6.3 Operational Structure and Agent Communication

The basic system structure consists of multiple agents connected through a communication network (internet or a local network) as shown in Figure 6.2. A Message passing schema is used for agent communication. The communication network must provide the following services [HAMA98]:

- Dispatch and reception of messages among various agents that exist in different locations in the network.
- Buffers should be available for storing incoming and outgoing messages. This is helpful for transmitting the requests in case of failure.
- Errors and failures that occur, due to agent break-down, during transmission should be identified and reported back to the initiating agent.

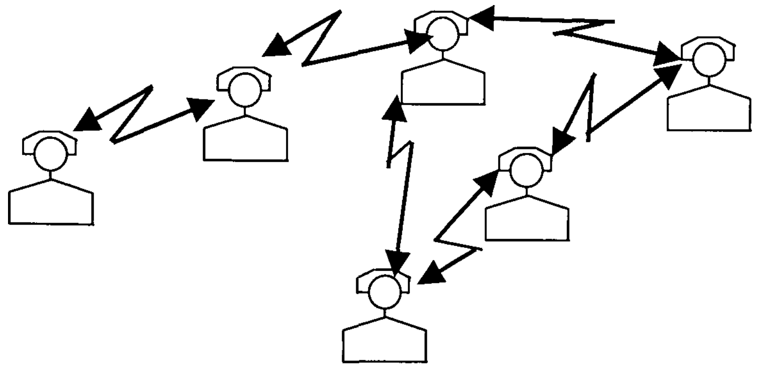


Fig. 6.2 Basic PMS structure

6.3.1 Agent Messages

As mentioned before agent communication is handled by a message passing schema. The general message syntax is as follows:

<message tag><sender identity><receiver identity><message contents>

The message tag identifies the type of the message, the identities of the senders and the receivers are contained in the sender and the receiver parts of the message respectively, while the message contents part carries the actual message. The message is interpreted according to the tag. There are different message types, we classify these types according to the communicating agents interactions as shown below:

- Agent[*] ⇔ Directory agent.
- Coordinator agent ⇔ Coordinator agent.
- Home agent ⇔ Coordinator agent.

The different message types used in the system are shown in Figure 6.3 and outlined below.

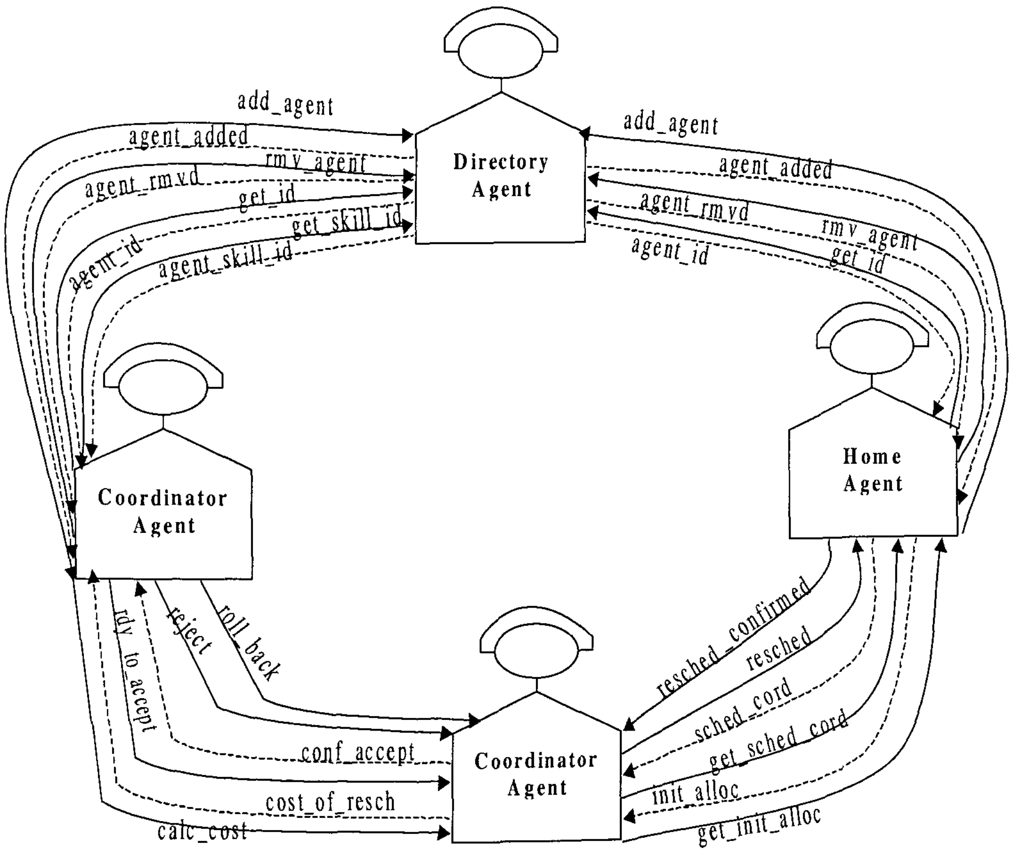


Fig. 6.3 Message types used in the system

[*] By agent here we mean a home agent or a coordinator agent.

Agent and Directory Messages

These messages are exchanged between the coordinator agents or the home agents and the directory agent. These messages are:

add_agent: This message is sent by the agent requesting an identity. This is sent during the loading of the agent.

agent-added: This message is sent by the directory agent to the agent in response to the add-agent request. It informs the agent that its request for identity is granted and assigns a unique identity to the agent.

get_id: This message is sent by the agent to the directory agent requesting the identity of another agent.

agent_id: This message is sent by the directory to the agent in response to the *get_id* request.

get_skill_id: This message is sent by the agent to the directory agent requesting the identity of a home agent with certain types of skills.

agent_skill_id: This message is sent by the directory agent to the agent in response to the *get_skill_id* request. It contains all the home agents that are capable of performing the requested skill.

rmv_agent: This message is sent by the agent to the directory agent requesting removal from the system.

agent_rmvd: This message is sent by the directory agent to the agent in response to the *rmv_agent* request.

Coordinator and Coordinator Messages

These messages are exchanged amongst the coordinator agents. These messages are :

calc_cost: This message is sent as a request to a coordinator to calculate the cost of rescheduling one of its subtasks.

cost_of_resch: This message is sent in response to a *calc_cost* request. It contains the cost of rescheduling a subtask.

rdy_to_accept: This message is sent to inform the coordinator that the proposal for rescheduling is acceptable and requesting confirmation.

reject: This message is sent to inform the coordinator that the proposal for rescheduling is rejected.

conf_accept: This message is sent to confirm acceptance of the proposal.

roll_back: This message is sent asking all coordinators to roll back to the previous state.

Home and Coordinator Messages

These messages are exchanged between the coordinator agents and the home agents. These messages are :

get_init_alloc: This message is sent as a request to a home agent to assign an initial allocation to a subtask.

init_alloc: This message is sent in response to a *get_init_alloc* request. It contains the initial allocation of the subtask.

get_sched_cord: This message is sent requesting a list of coordinators allocated to given slots.

sched_cord: This message is sent in response to a *get_sched_cord* request. It contains the list of coordinators that are allocated to the given slots.

resched: This message is sent asking the home agent to reschedule a subtask.

resched_confirmed: This message is sent by the home agent to confirm rescheduling of the subtask.

6.4 Implementation

At the time we started the development of the PMS we looked at different agent development environments. A non-exhaustive list is presented in Tables 6.1 and 6.2 [UMBC]. These listed in Table 6.1 include academic and research agent development platforms and the list in Table 6.2 includes commercial and industrial agent development platforms.

Table 6.1 Academic and research platforms for agent development

Name	Research Group	Comments
JIAC	Technical University Berlin DAI-Lab	A Java class library for the development of a universal architecture of agent-oriented systems (ALBA99)
MAST	Technical University of Madrid	A general purpose distributed framework for the cooperation of multiple heterogeneous agents (IGLE95)
OAA	SRI AI Centre	A framework for integrating heterogeneous software agents in a distributed environment (MART99)
Zeus	British Telecom Lab ISR Group	A library of software components and tools that facilitate the design, development and deployment of agent systems [NWAN99]

Table 6.2 Commercial platforms for agent development

Name & URL	Company	Comments
Aglets (http://www.trl.ibm.com/aglets/)	IBM Japan	An environment for programming mobile Internet agents in Java
AgentBuilder (http://agentbuilder.com)	Reticular Systems, Inc.	An integrated software development tool to build intelligent agent-based applications
JACK (http://www.agent-software.com.au)	Agent Oriented Software Group	An environment for building, running and integrating JAVA-based multi-agent systems using a component-based approach.

At the time of writing many such platforms exist. For a more comprehensive list of academic agent development platforms see reference [UMBCA], and for commercial platforms see reference [UMBCB].

When considering these and other agent development platforms at the time the simulation was being created, we excluded frameworks that are free for non-commercial or academic use. Therefore, the candidate frameworks fulfilling our constraint are those in Table 6.1.

For evaluation purposes we considered the following characteristics of the frameworks, each characteristic has a weight (from 1-3) indicating its overall importance:

1. Stage of development (maturity): how long has the framework been in development? Are all features available? Is it bug free? - weight: 3.
2. Use of the Java language. This is our primary programming language, this is necessary to speed up the learning curve - weight: 3.
3. Good library documentation and the availability of tutorial material - weight: 3.

4. Support for Agent design and communications, this is necessary to speed up the development of the simulator - weight: 2.
5. Good debugging tools - weight: 2.
6. Community support, important for solving any problems encountered. It is judged on the availability of active mailing lists and discussion boards - weight: 1.

Our evaluation is shown in Table 6.3. The values in the table indicate the presence and quality of the corresponding feature: 0 indicates its absence, 10 indicates that it is very well implemented. This evaluation shows that the JDK appears to be the form most suited for the development of the simulator. We therefore implemented the simulator using Java Development Kit (JDK1.2.2) [FLAN99].

Table 6.3 Feature analysis

Framework	**Maturity**	**Java**	**Documentation**	**Agent Support**	**Debugging Tools**	**Community Support**
JDK	9	10	9	5	7	8
JIAC	6	0	5	7	5	5
MAST	6	0	6	7	4	4
OAA	5	0	6	7	4	4
Zeus	7	10	7	7	5	6

6.4.1 PMS Infrastructure

The basic PMS infrastructure consists of multiple agents connected through a communication network (internet or a local network) as shown in Figure 6.4.

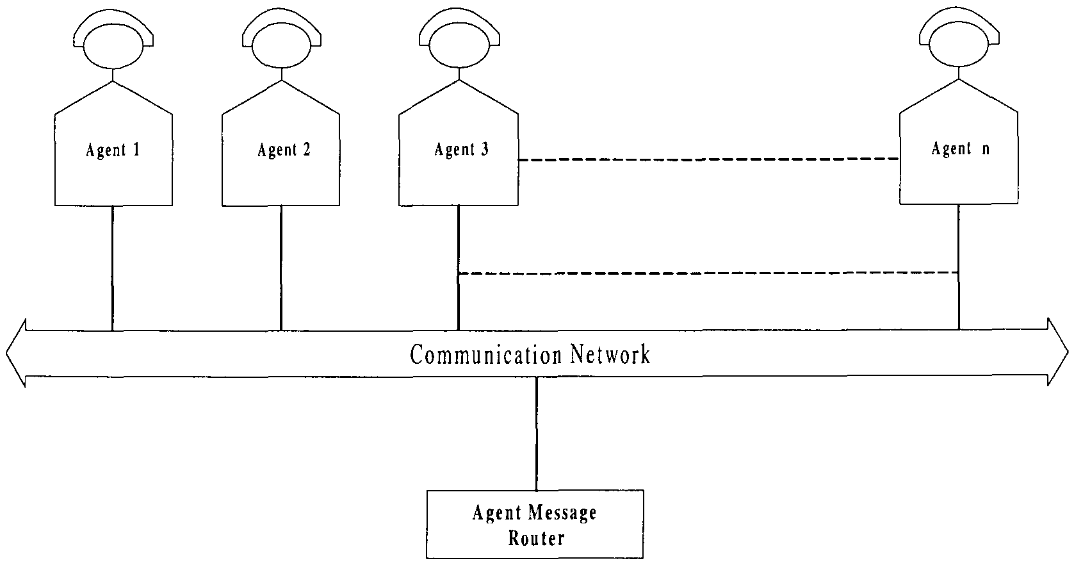


Fig. 6.4 PMS infrastructure

The Agent Message Router (AMR) handles interactions among the agents. Each agent is assigned a unique identity by the AMR. Agents can either broadcast a message to all agents or can send a message to a specific agent or group of agents.

For agent communication we used a modified message-passing scheme as described in [FARL98]. We chose message passing as a communication method as it is relatively simple to implement using the java.io.package, and there are no communication overheads as in the case of CORBA[†] [ORFA97]. This message passing scheme, the basic structure of the AMR, and the agents are described below.

[†] CORBA home page ((Object Management Group) http://www.omg.org

6.4.2 The Message-Passing Scheme

A message is a structured piece of information sent from one agent to another over a communication channel. These messages can be requests made to one agent by another, or can be data or notification sent to another agent [Farl98]. Based on this definition, the following is a representation of a generic message object.

```
public abstract class MessageClass
{
  //Attributes
    protected String id;
    protected Vector argList;
  //Constructor
    public MessageClass(String mid){…}
  //Methods
    public void addArg(String arg){…}
    public String getID(){…}
    public void setId(String mid){…}
    public Vector getArg(){…}
    public abstract boolean Do(){…}
}
```

Messages are treated as a string of tokens, a message is simply a series of tokens followed by an end of message indicator. The first token is the message identifier and the rest are arguments of the message. Each message object has an identifier, *id*, and a list of arguments, *arglist*. The methods *getID()* and *getArg()* are used to get information about the message object. The abstract method *Do()* is used to interpret the message arguments and to perform whatever is required for the message depending on its type. A different implementation of the *Do()* method is used in the subclasses of the *MessageClass* for each type of message defined in our system.

The class *MessageHandler*, is implemented so that an agent can receive and send messages over the established connection, a representation of this class is shown below:

```
public abstract class MessageHandler implements Runnable
{
  //Attributes
     public static MessageHandler current = null;
     Hashtable connections = new Hashtable();
     Hashtable handlers = new Hashtable();
  //Constructor
     public MessageHandler(iputStream i, outputStream
               o) {…}
  //Methods
     synchronized public int nextAgentId(){…}
     synchronized public Vector getAgentIds(){…}
     synchronized public int addAgent(InputStream i,
               OutputStream o){…}
     synchronized public int addAgent(String
               name,InputStream i, OutputStream o){…}
     synchronized public void addAgent(int id,
               InputStream i, OutputStream o){…}
     synchronized public boolean removeAgent (int id) {…}
     synchronized  public int getAgentId(String n){…}
     synchronized protected AgentConnection getAgent(int
               id){…}
     public Message readMsg(int id) throws {…}
     public boolean sendMsg(Message msg, int id) throws
               IOException {…}
     public boolean sendMsg(Message msg) throws
               IOException {…}
     public void run(){…}
     protected Message buildMessage(String msgId) {…}
}
```

Various methods are supplied in this class for adding, removing and getting agent connections. Each connection is associated with an *id* number, such connections are stored in a table *connections*, that is maintained by the MessageHandler. There are two versions of *sendMSg()*: one sends a message to a specific agent and the other broadcasts the message to all the agents. Every time an agent is added, the following events take place:

- A connection is established to hold the details of the InputStream and OutputStream connected to the agent, this is stored in a hashtable using the agent's *id* as the key.
- An *AgentHandler* is created and given the *id* number of the agent.
- A new thread is created for the *AgentHandler.*
- The new thread is started.

The AgentHandler implements a runnable interface and its *run()* method is a loop that continuously attempts to read messages from its agent and then acts on it as shown below:

```
public void run() {
   while (true) {
        try {
        Message m = Msghandler.readMsg(AgentId);
        m.Do();
   }
        catch (IOException e) {}
}
```

Multiple connections are handled by creating a thread for each agent. The thread can asynchronously read messages and act on them. Agents can be added to the MesageHandler at any time. To allow asynchronous agent handling, the *readMsg()*, *sendMesage()* and the methods for adding and removing agents are synchronised.

6.4.3 The Agent Message Router (AMR)

The AMR performs the following tasks:

- Register new agents and provide them with a unique identifier.
- Remove agents.
- Send messages.
- Broadcast messages.

We need to provide an identity for each agent in the system so that transactions among agents can be traced and targeted to individual agents. The Identity class, shown below, is provided to support this function.

```
public class Identity implements Serializable {

  //Attributes
    Hashtable property = new Hashtable();
  //Constructors
    public Identity(int id) {..}
    public Identity(){}
```

```
     //Methods
       public boolean equals(Object o) {..}
       public int getId() {}
       public String getName() {..}
       public void setName(String n) {..}
       public Object getProperty(Object key) {..}
       public void setProperty(Object key, Object val) {..}
}
```

We use two properties to identify each agent. We use the name property as a descriptive property that can be used in a user interface and the integer id is used as an internal identifier to tag each agent. These properties and additional properties that can be used to further define the agent are stored in the *property* list defined in the class. A set of methods is provided for setting and getting these properties. In order for identity objects to pass back and forth between agents on the network, the Identity class is made to implement the *Serializable* interface.

The AMR uses the MessageHandler to route messages back and forth between cooperating agents, a ServerSocket to accept socket connections, and a port number that it listens to for asynchronous connections from agents. A representation of the AMR class is shown below:

```
public class AMR implements Runnable {
  //Attributes
    MessageHandler msgHandler = new MessageHandler();
    ServerSocket socket = null;
    int port = 5009;// Other port number can be uses
    private AMRGUIWindow window;
  //Constructors
    public AMR(int p) {..}
    public AMR(){..}
    public AMR(ServerWindow S) {..}
  //Methods
    protected void initHandler() {..}
    public void run() {
            // Make the server socket.......
               . . .
    }
    public Identity newMember() {..}
    public boolean remove(Identity i) { .. }
    public Vector getMembers() {..}
    public boolean send(Identity to, Identity from,
        String mtag, String s) throws IOException {..}
        public boolean broadcast(Identity from, String
           mtag, String s) throws IOException {..}
    }
```

The *inithandler()* method is called in each constructor. It is used to initialise the *MessageHandler*. The *run()* method in the *AMR* class creates the *ServerSocket* that listens to its designated port for connections requests from agents. Each time a new connection is made, the agent is added to the handler by calling its *addagent()* method. The *AMR* creates a unique *Identity* for the agent by calling *newMember()*, then a message is sent to the agent containing its *Identity*.

To create an *AMR* object, we implemented a *CreateAMR* which has a main method in which the AMR object is created and a GUI (Graphic User Interface) that shows the AMR status and communication messages. The main classes used (the *MessageHandler*, the *Identity* , and the *messageClass*) in AMR, their main methods, and their relations are shown in Figure 6.5.

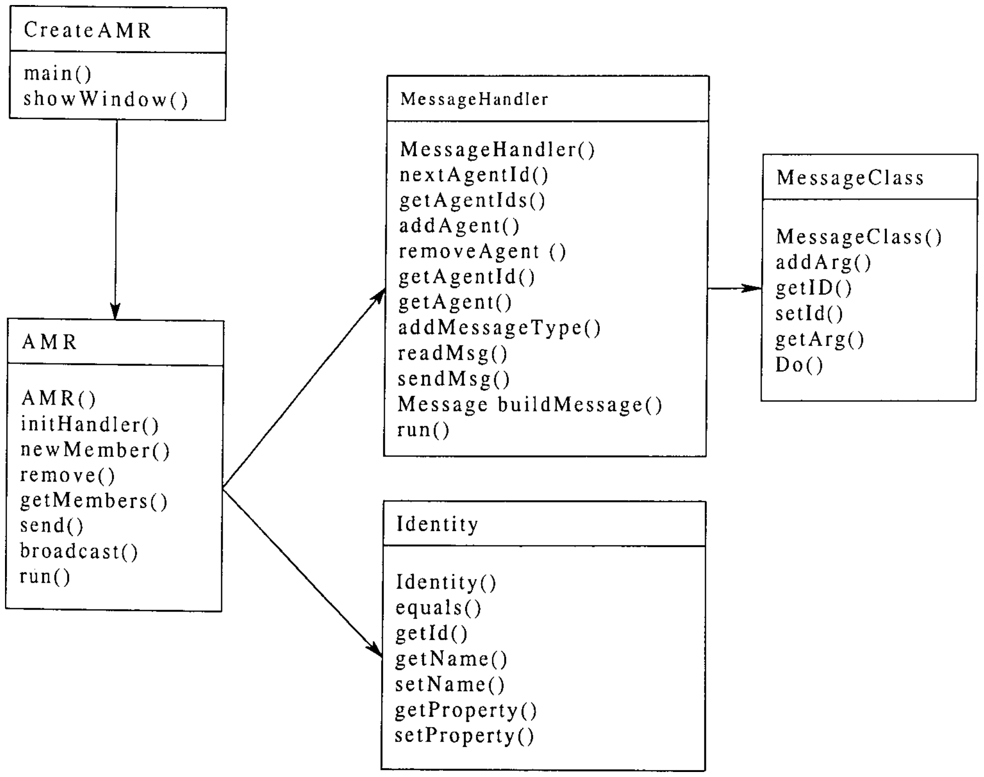
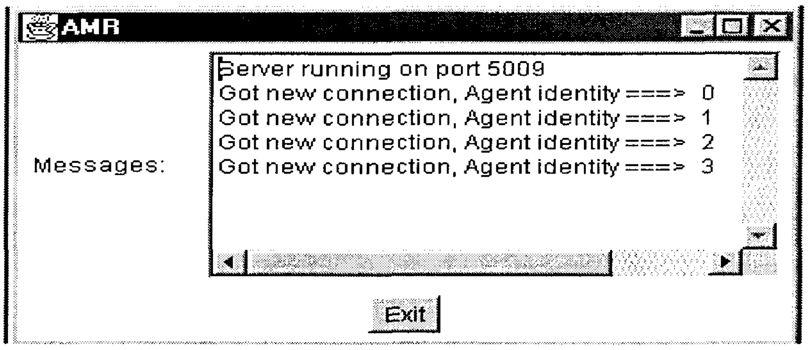


Fig. 6.5 Main classes used in the AMR

We can create an AMR on a given port on a host, then any client can connect to the system by creating the client agent, using the AMR host and port number. The agent can then engage in cooperative tasks with any other agents connected to the host using the appropriate methods. The AMR services each connection made in a separate thread. A screen shot of a running AMR is shown in Figure 6.6.



```
Server running on port 5009
Got new connection, Agent identity ===> 0
Got new connection, Agent identity ===> 1
Got new connection, Agent identity ===> 2
Got new connection, Agent identity ===> 3
```

Fig. 6.6 Screen shot or running AMR

6.4.4 Agent Implementation

As the main concern of this implementation is to demonstrate the working of the preference model and to perform experiments to verify our theoretical formulations, we have a generic agent class (parent class). It is used to build the different types of agents, discussed in section 6.4, and to provide the default behaviour and structures for all agents. In this implementation each agent has the ability to be connected to the *AMR* and to other agents in the system and can engage in communication with them. Also each agent can send messages and be notified of incoming messages. A representation of our agent class is shown below:

```
public class Agent
{
  //Attributes
    MessageHandler handler = new MessageHandler();
    Identity id = null;
    String name;
    private AgentGUIWindow window;
  //Constructors
    public Agent(String host, int port,
             String n) {..}
```

```
        //Methods
          public Identity getIdentity() { .. }
          public boolean connect(Properties p) {..}
          public boolean send(String tag, String msg,
                 Identity dst) throws IOException {..}
          public boolean broadcast(String tag, String msg)
                 throws IOException {..}
          public boolean notify(String tag, String msg,
                  Identity src) throws IOException {..}
    }
```

This class provides the constructor with a name along with the host and port number of the AMR to which it will connect. The constructors initialise the *Messagehandler*, establish the appropriate connection to the AMR, and create a GUI to show agent status and communication messages. Each agent has a unique identity defined by the *Identity* class discussed in the previous section. The *send()* method sends a message to a particular agent, while the *broadcast()* method broadcast the message to all agents in the system. The notify() method checks the message type or body and reacts accordingly.

The main classes used in the Agent class, that is *MessageHandler* and *Identity*, their main methods, and where these classes are used are shown in Figure 6.7.

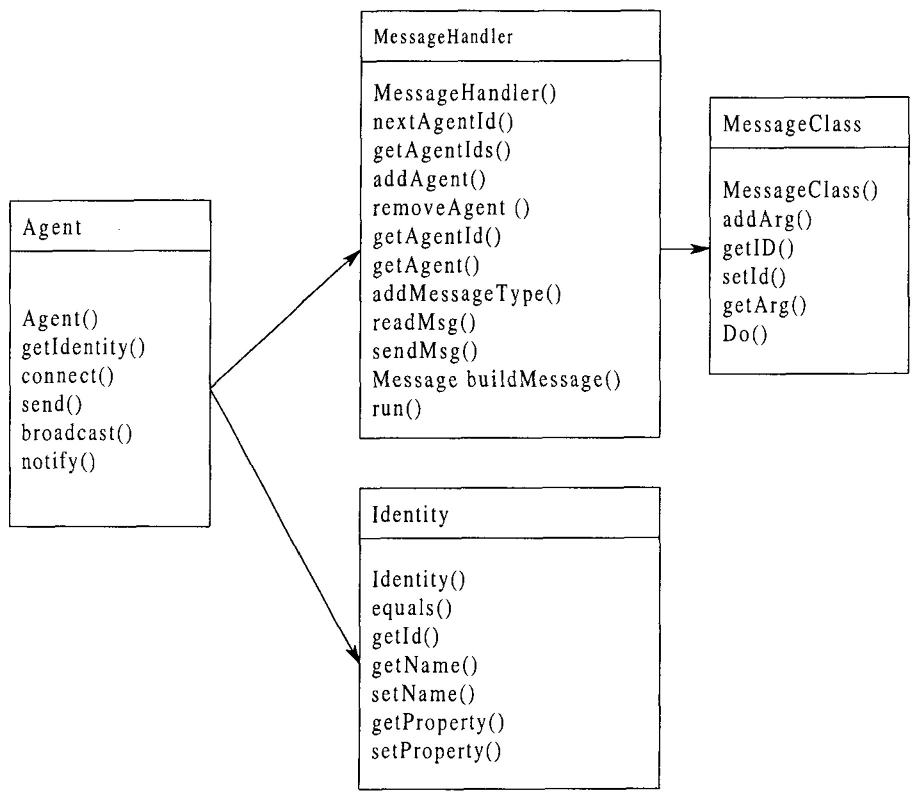


Fig. 6.7 Main classes used in the Agent class

6.4.5 Implementation of Agent Types

As stated before the focus of our implementation is scheduling in a manufacturing environment. In this environment, we need an agent for task management, which we call the *coordinator agent* and an agent for machine management, which we call the *home agent*. We also use a *directory agent*. Each agent has a different function to perform in the system. In this section we describe each of these agents and the role it carries out. The basic features of each agent are inherited from the basic *Agent* class described in section 6.4.4.

The Coordinator Agent Implementation

A coordinator agent is assigned for each task. Each task is decomposed into subtasks stored in a hashtable, with an initial budget. A presentation of the task is shown below:

```
public class Task {
    //Attributes
      public String taskName;
      public Hashtable subTasks;
      public int initailBudget;
      public int currentBudget;
   //Constructor
     public Task(String name,Hashtable
              subtasks,int intbud){..}
}
```

Each subtask is presented using the following class:

```
public class subTask {
   // Attributes
        public String subTaskname;
        public int duration;
        public Vector predecessors;
        public Vector successors;
        public Hashtable resources ;
        public Hashtable preferences;
   //Constructor
        public TaskClass(String name,int duration,
                 Vector preds, Vector suc,
                 Hashtable res, Hashtable prefs{..}
}
```

Preferences for each subtask is stored in a hashtable, and each preference is presented using the following class:

```
public class Preference {
        //Attributes
          public String prefName;
          public String target;
          public int value;
          public int offer;
```

```
              //Constructor
                public Preference (String name, String
                       tget, int val, int offer){..}
      }
```

Figure 6.8 shows the relationship between the *coordinator* agent, the *task* , the *subTask,* and the *preference* classes.

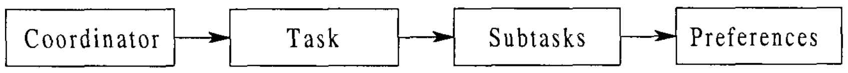
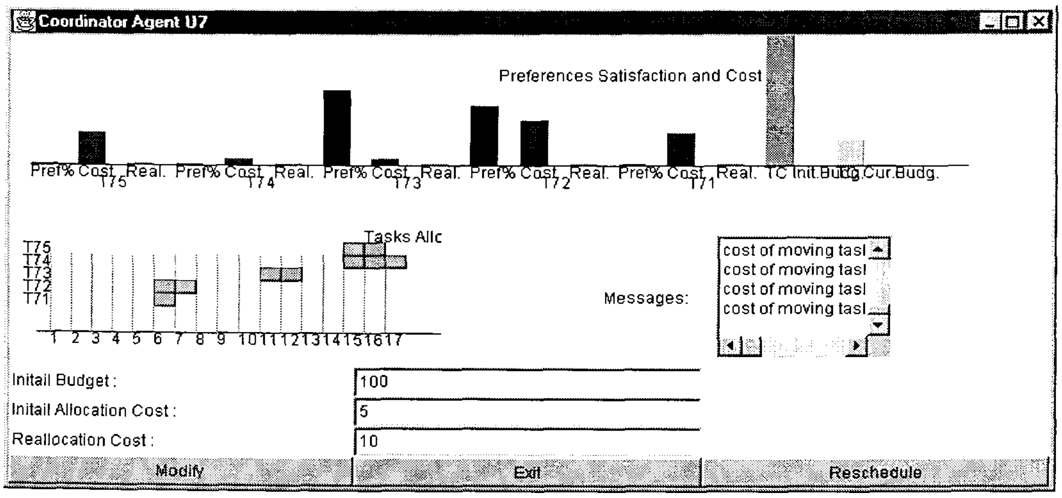


Fig. 6.8 Main classes used in the coordinator

We provided a GUI that enables the user to change certain parameters (initial budget, initial allocation cost, and reallocation cost), and show the cost , remaining preferences and the messages received and sent by the coordinator. A screen shot of the coordinator GUI is shown in Figure 6.9.


Coordinator Agent U7
Preferences Satisfaction and Cost
Pref% Cost Real. T75 Pref% Cost Real. T74 Pref% Cost Real. T73 Pref% Cost Real. T72 Pref% Cost Real. T71 TC Init.Budg. Cur.Budg.
Tasks Allc
T75 T74 T73 T72 T71
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
Modify
Exit
Reschedule

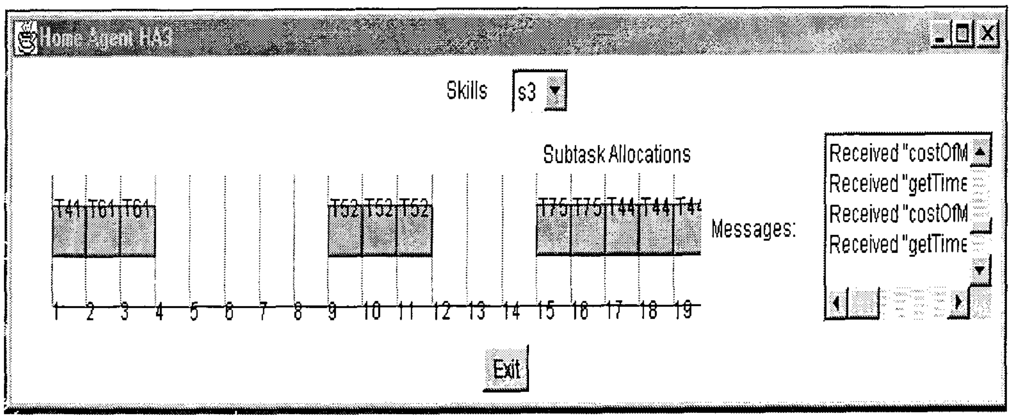
Messages:
cost of moving tasl
cost of moving tasl
cost of moving tasl
cost of moving tasl

Initail Budget: 100
Initail Allocation Cost: 5
Reallocation Cost: 10

Fig. 6.9 A screen shot of the coordinator GUI

The Home Agent Implementation

During the task allocation process, outlined in chapter 4, the home agent is consulted regarding the task allocation. According to this consultation the task allocation is initiated. Once the task allocation process has ended all affected coordinator agents send the appropriate home agents messages informing them of the new allocations. Subsequently the home agents update their schedules. A graphical interface that enables the user to see the current status of tasks scheduled and the messages received and sent by the home agent is provided. A screen shot of this graphical interface is shown in Figure 6.10.



Skills s3

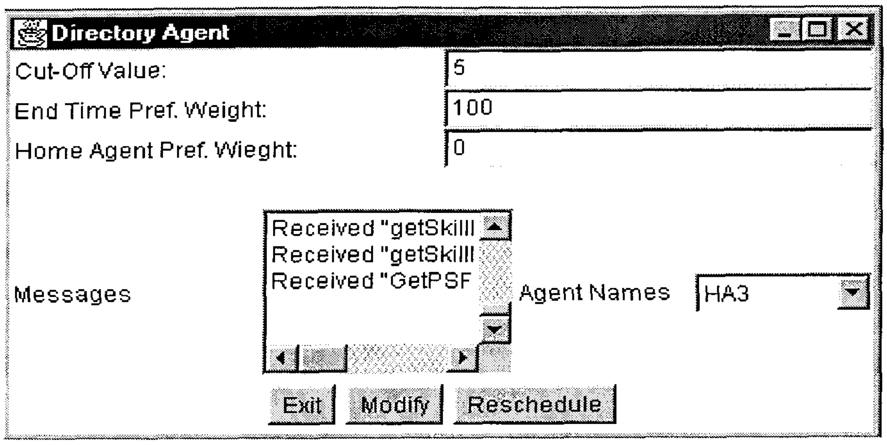
Messages:
Received "costOfM
Received "getTime
Received "costOfM
Received "getTime

Fig. 6.10 A screen shot of the home agent graphic interface

The Directory Agent Implementation

Upon creation the agent registers with the directory and subsequently it receives a unique identity from the directory agent. This identity is used while communicating with other agents. On leaving the system the agent is removed from the list and the details of this removal are cascaded to other agents to update local directories. We should note that the directory agent should be created first. Also the agent class maintains a default address for the directory.

A GUI is provided to enables the user to change certain parameters (preference cut-off and the weights for the different preferences), and to show the messages received and sent by the directory. A screen shot of the directory GUI is shown in Figure 6.11.


Directory Agent

Cut-Off Value: 5

End Time Pref. Weight: 100

Home Agent Pref. Wieght: 0

Messages

Received "getSkillI
Received "getSkillI
Received "GetPSF

Agent Names HA3

Fig. 6.11 A screen shot of the directory GUI

6.5 Running PMS

To simplify the implementation and to emphasise the main concepts of the preference model, we have limited the number of preference types to be handled to one type, namely the end-time preference. Also we have limited the resource type to continuous resource (see section 4.2.1). These limitations do not constrain the PMS from demonstrating the basic concepts of the preference model, which is the main focus of the PMS as stated at the beginning of this chapter.

In this implementation we run the AMR first, then the directory agent is loaded and then the coordinators and the home agents can be loaded. The number of coordinators and home agents and their attributes are read from a disk file and these can be changed from one run to another. Though the user can remove a coordinator or a home agent during the run, the cascading effect of this is not dealt with in this implementation. We thought that this complicates the implementation and leaving it out does not prevent us achieving the objectives of the PMS.

The coordinator agents load the subtasks and their attributes from disk storage and starts the allocation process outlined in Chapter 4. The intermediate results and the end results are stored on disk to be analysed later. These results are discussed in the next chapter. The main events that take place during run time are summarised in Figure 6.12.

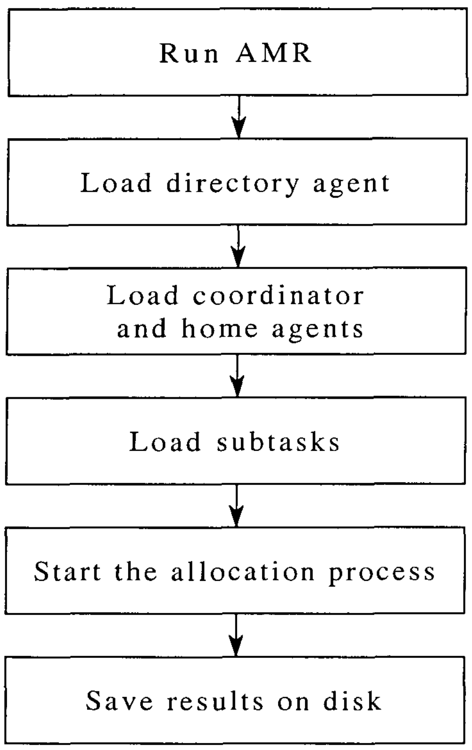

Run AMR
Load directory agent
Load coordinator and home agents
Load subtasks
Start the allocation process
Save results on disk

Fig. 6.12 The main events during run time

6.6 Summary

In this chapter we discussed the design and implementation of the PMS. We designed the PMS to demonstrate and verify the working of the preference model. We built an infrastructure for the message passing system to be used by the various agents in the system. We used an abstract class for the agent from which all agents in the system inherit their properties. Message types and structures used in the demonstrator were presented. The different type of agents and their roles in the implementation were discussed. We discussed the assumption made in the implementation and the main events that take place during the running of the system. In the next chapter we shall present and analyse the results obtained from running a case study on the implemented simulator.

Chapter 7

Exploratory Experiments

This research is mainly driven experimentally. Therefore the operational functionality of the preference model was developed incrementally using the several hundred experiments we conducted using the PMS presented in chapter 6. Apart from the initial numerous test runs to verify the programme correctness, several hundred experiments have been conducted with many permutations and combinations, first to develop an insight and then to determine what is or is not significant. These experiments have been carried out over a period of three years. Due to the non-linearity of the process, initially we were not even sure that the processes would converge, and that the final preference value would at all be stable or that a simple theoretical model would be possible. It was always the experimental results that guided us to our conclusions. The objectives of the experiments presented in this chapter are as follows:

1. To show that the preference model converges to a stable value, regardless of the number of tasks and their configuration.
2. To study the characteristics of this convergence and to determine the effect and significance of the different parameters in the preference model on the convergence to the final solution.
3. To investigate if the preference model behaviour can be predicted using mathematical formulae, at least qualitatively.

A group of experiments were planned to help us in achieving these objectives. A case study, presented in section 7.1 is used throughout these experiments. The experiments, presented in this chapter, are divided into three groups. The objective of the first group is to demonstrate that preference model converges, for the second group it is to investigate the effect of the different parameters on the convergence characteristics, and for the third group it is to investigate the preference losses discussed in section 4.2.4. These experiments are presented in sections 7.2, 7.3, and 7.4 respectively. We summarise our findings in section 7.5.

7.1 The Simulated Case Study

For the purpose of the experiments presented in this chapter we have considered the problem of scheduling a number of tasks T^i. We have used a set of coordinators U^1, U^2, .. U^n, each U^i is responsible for scheduling a (global) task T^i, each task T^i being further subdivided into subtasks: T^i_j | {j = 1, 2, .. m}, one or more subtasks being allocated to a home agent A_k. But in reality we have used up to n = 14, that is, up to 14 coordinators (and hence 14), up to six subtasks (m= 6) in each task, and three home agents, each sharing many subtasks of different coordinators. The subtasks have precedent subtasks and their allocation to the target agents can be considered as the allocation to Assembler agents in manufacturing. The preferred resource type used in our simulation is the end-time slot of a subtask. Apart from resource conflicts, precedent constraints can also disallow the allocation of some resources. We have used the algorithm outlined in section 4.3 to find a schedule that satisfies as much preference values as possible. According to this algorithm, each subtask is initially allocated the earliest possible slot that satisfies the precedence constraints. As explained in detail in Chapter 4, the coordinator will accept a negotiation for an exchange if its offer price $O \geq$ the cost C for the expected preference gain g. After the negotiation, an actual exchange with preference gain $g' \mid g' \leq g$ is proposed. The coordinator will accept it if $g' \geq \varphi$, but pay pro-rata to $C*g'/g$. If the exchange is unsuccessful, the negotiation will continue for another possible exchange.

In the case study we used in our simulation, each subtask T^i_j has a preference value V_i and an offer price O_i that the task is willing to pay for preference satisfaction. Also the costs C_i and C_r, are set by the task agent, and indicate the price that the task should pay for a preferred allocation (see section 4.2.2 for the explanation of those parameters). The preference values and offer prices are assigned randomly using the *random* method available in the Java Math package. The user-defined Preference Reduction Function, ρ, was used to allocate reduction on preference gain for slots away from either side of the preferred end time slot (section 4.2.1). Observe that subtasks have precedent constraints. The diagram in Figure 7.1 shows the coordinators, tasks, subtasks, and tasks attributes, for n=6. To obtain more general results we have experimented with three different values of n and with different values for the different subtask attributes (i.e. duration, resource agent, preferred end time, preference value, offer price).

In order to generalise the results and not to be biased towards a specific configuration of values and subtasks we used a variety of test cases. Each coordinator is assigned a unique set of subtasks. This is done so that the only knowledge about other agents is acquired through negotiation. The main features that were included in the test cases are as follows:

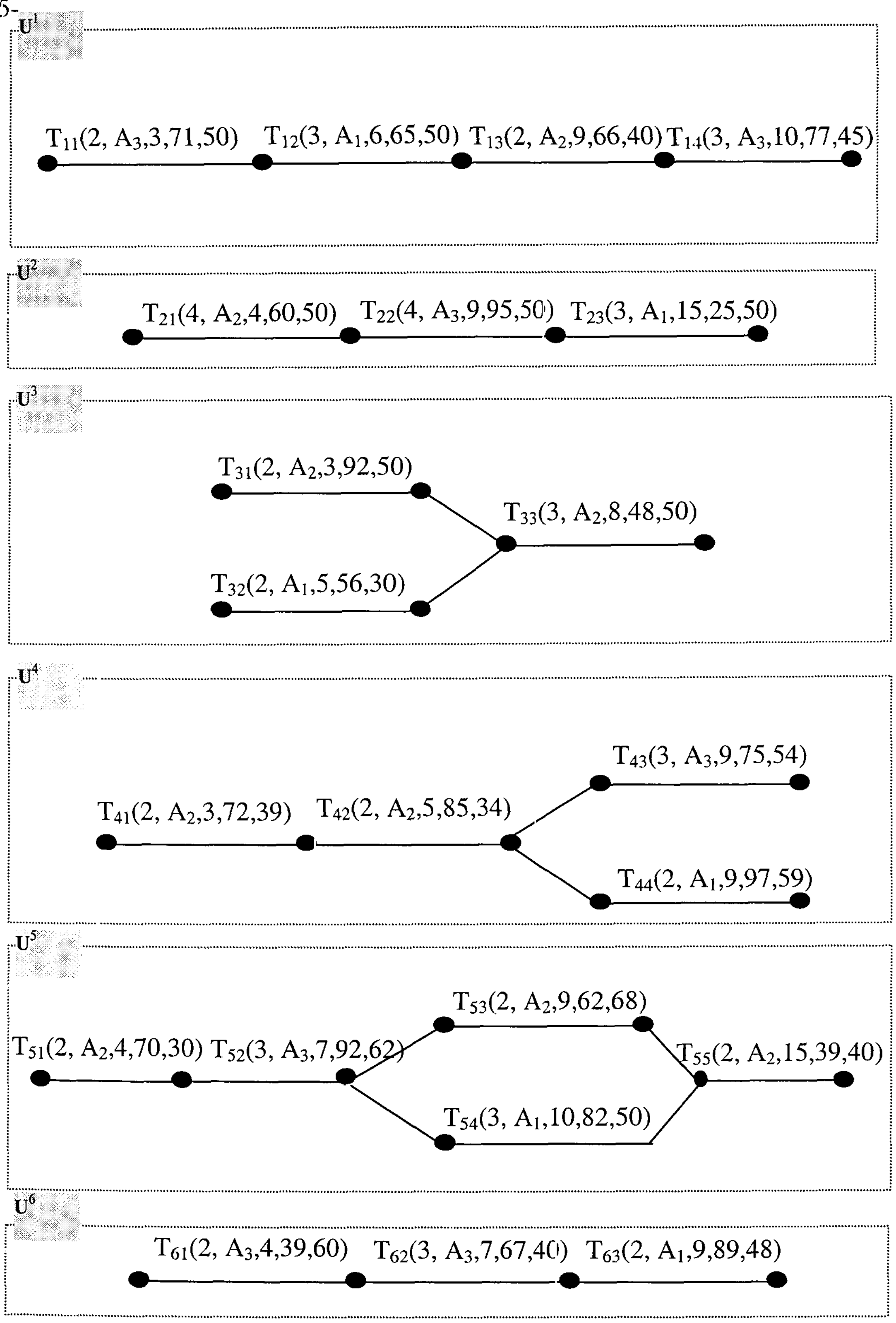
1- Variable numbers of subtasks for each coordinator. As can be from Figure 7.1, different numbers of subtasks were assigned for the different coordinators.

2- Variable numbers of predecessor and successor subtasks, i.e. subtasks can have one or more predecessor and successors.

3- Interdependency of subtasks so that one subtask that belongs to one coordinator is dependent on the results of a subtask that belongs to another coordinator.

4- Though the preference values and offer prices are assigned randomly, as indicated above, a constraint is made so that they are close to each other in order to ensure contention for resources.

5-


U^1
T_{11}(2, A_3,3,71,50)
T_{12}(3, A_1,6,65,50)
T_{13}(2, A_2,9,66,40)
T_{14}(3, A_3,10,77,45)
U^2
T_{21}(4, A_2,4,60,50)
T_{22}(4, A_3,9,95,50)
T_{23}(3, A_1,15,25,50)
U^3
T_{31}(2, A_2,3,92,50)
T_{33}(3, A_2,8,48,50)
T_{32}(2, A_1,5,56,30)
U^4
T_{43}(3, A_3,9,75,54)
T_{41}(2, A_2,3,72,39)
T_{42}(2, A_2,5,85,34)
T_{44}(2, A_1,9,97,59)
U^5
T_{53}(2, A_2,9,62,68)
T_{51}(2, A_2,4,70,30)
T_{52}(3, A_3,7,92,62)
T_{55}(2, A_2,15,39,40)
T_{54}(3, A_1,10,82,50)
U^6
T_{61}(2, A_3,4,39,60)
T_{62}(3, A_3,7,67,40)
T_{63}(2, A_1,9,89,48)

Parameters: (Task Name (duration, resource agent, preferred end time, preference value, offer price))

Fig. 7.1 Task Parameters.

7.2 Investigation of Convergence

As stated before, the main objective of this group of experiments is to investigate if the preference model converges to a stable solution, and to see if this stable solution remains the same regardless of the order of initial allocation. We start by investigating the gain in preference values when only one task (including its subtasks) is reallocated, using two scenarios. In the first scenario the preferences of the reallocated tasks are not considered, while these are taken into consideration in the second scenario. Then we change the order of the initial allocation to investigate whether such variations affect the gain in preference values. We expect that a gain in preference values should be achieved in both cases, although the gain might be less in the second case as other task preference values are considered. Also, we expect the cost to be affected at the points where a gain in preference value is achieved. The order of task allocation should have less effect on the final gain in preference value, although the intermediate gain values might be different.

7.2.1 Individual Task Reallocation (without considering other tasks preferences)

In these experiments all tasks are initially allocated to the first available slot. Tasks are allocated according to the order T_1, T_2 ... T_6, without considering preferences. Then one task is subsequently reallocated to satisfy its preferences. We expect a gain in preference value for this task, while other tasks might lose some of their preference values. The tasks that gained preference values pay those disadvantaged tasks. As a result of this reallocation some tasks can be reallocated. These tasks are reallocated to the first available slot without taking their preferences into account.

Figures 7.2 shows the result of conducting this experiment for task T_3 when tasks are initially allocated according to the order T_1, T_2 ... T_6. Figure 7.3 shows the result of this experiment for task T_3 when tasks are initially allocated according to the order T_6, T_5 ... T_1. These figures show how the total cost (C^3) and gain in preference values (G^3) of T_3 vary during the reallocation process. Iteration points are indicated by square symbols for the gain in preference value, and by diamond symbols for the total cost. With respect to the G^3 graph, the Y-axis values indicate the percentage preference gain while in the case of the (C^3) graph, they indicate the total cost units.

The G^3 graph shows that during the initial allocation (iteration 0) around 40% of the preference value is satisfied when tasks are initially allocated according to the order T_1, T_2 ... T_6, while 44% of the preference value is satisfied when tasks are initially allocated according to the order T_6, T_5 ... T_1. A preference gain occurs at iteration 1 in both cases and remains constant throughout the subsequent iterations, around 93% in both cases.

The C^1 graph shows an increase in cost at iteration 1, 160 cost units in the first case and 90 cost units in the second case. We think the difference between these cost values (70 cost units) is due to the fact that the negotiating tasks involved are not the same and therefore the exchanged preferences can be different. This is expected as each gain in preference value is associated with an increase in cost and the results conform to our expectations.

Since only one task is competing to gain preference value, we might expect that all preference values should be satisfied, but, as the results show, this is not the case. This is due to the cost constraint. The cost the task is willing to pay is less than the actual cost, and therefore not all the preference value is satisfied.

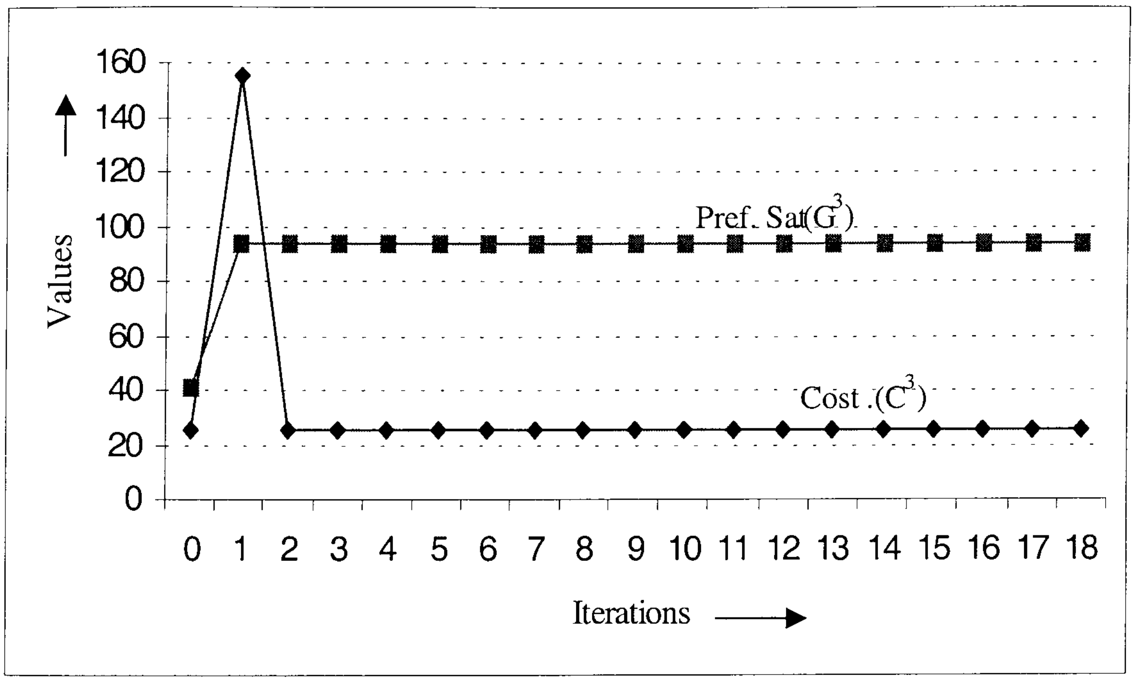


Fig. 7.2 T^3 preference satisfactions & cost variation (without considering other tasks preferences and initial allocation order 1,2,3,4,5,6)

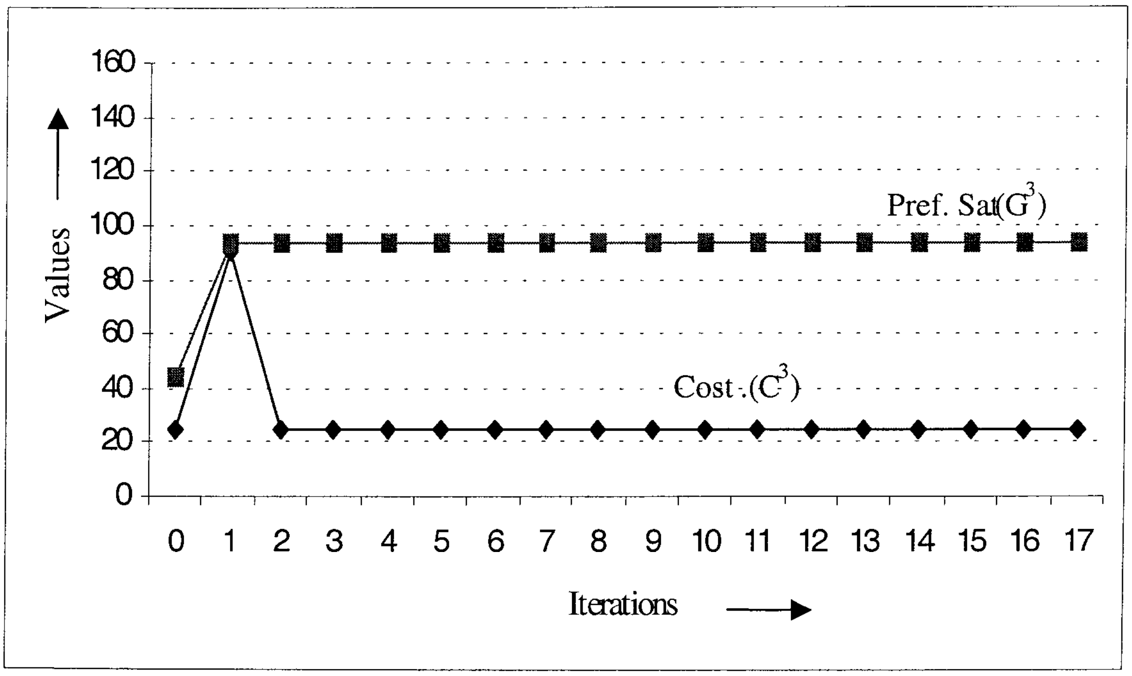


Fig. 7.3 T^3 preference satisfactions & cost variation (without considering other tasks preferences and initial reverse allocation order 6,5,4,3,2,1)

Appendix A shows graph of results obtained when this experiment was conducted using the other tasks and with different order of initial allocation. The results are similar to the one above and conform to our expectations.

7.2.2 Individual Task Reallocation (considering other tasks preferences)

These experiments are the same as in the previous section except that preferences of the affected tasks are taken into account during their reallocation. We expect the gain to be less and that it might require more iterations to converge to the final value. This is because the other tasks are now also competing to gain preference values, in contrast to the previous experiment where only one task is allowed to compete for gain in preference values.

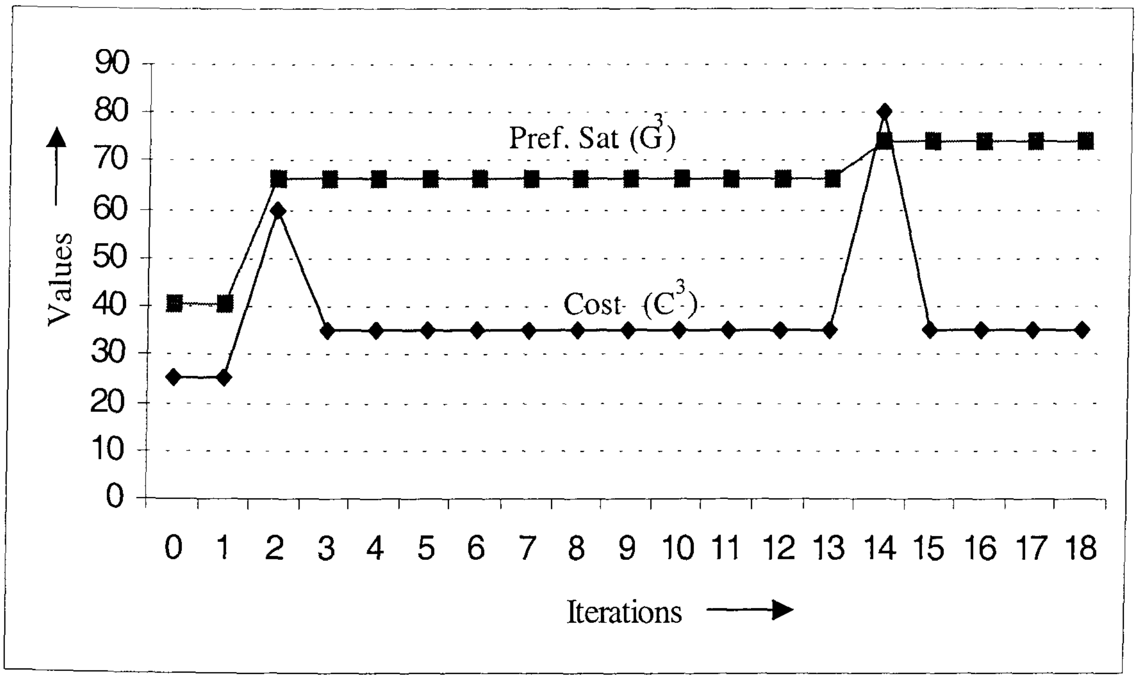


Fig. 7.4 T^3 preference satisfactions & cost variation (considering other tasks preferences)

The result of this experiment for task T_3 is shown in Figure 7.4. The results for the other tasks are shown in Appendix B. Figure 7.4 shows how the total cost (C^3) and gain in preference values (G^3) of T_3 varies during the reallocation process. Iteration points are indicated by square symbols for the preference value gain, and by diamond symbols for the total cost. It can be seen that the graphs in Figure 7.4 are similar to the ones obtained in the previous experiment but differ in that the preference value in the G^3 graph converges (to approximately 74%) after iteration 14. The preference value is less than the value reached in the previous experiment, and it took more iterations to reach this value. This is to be expected as in this case there is contention on resources and other tasks are negotiating for the same slot. Therefore T^3 has to trade some of its preference value with other tasks and this causes the drop in the preference value (T^2, T^4, and T^6) as shown in Figure 7.5.

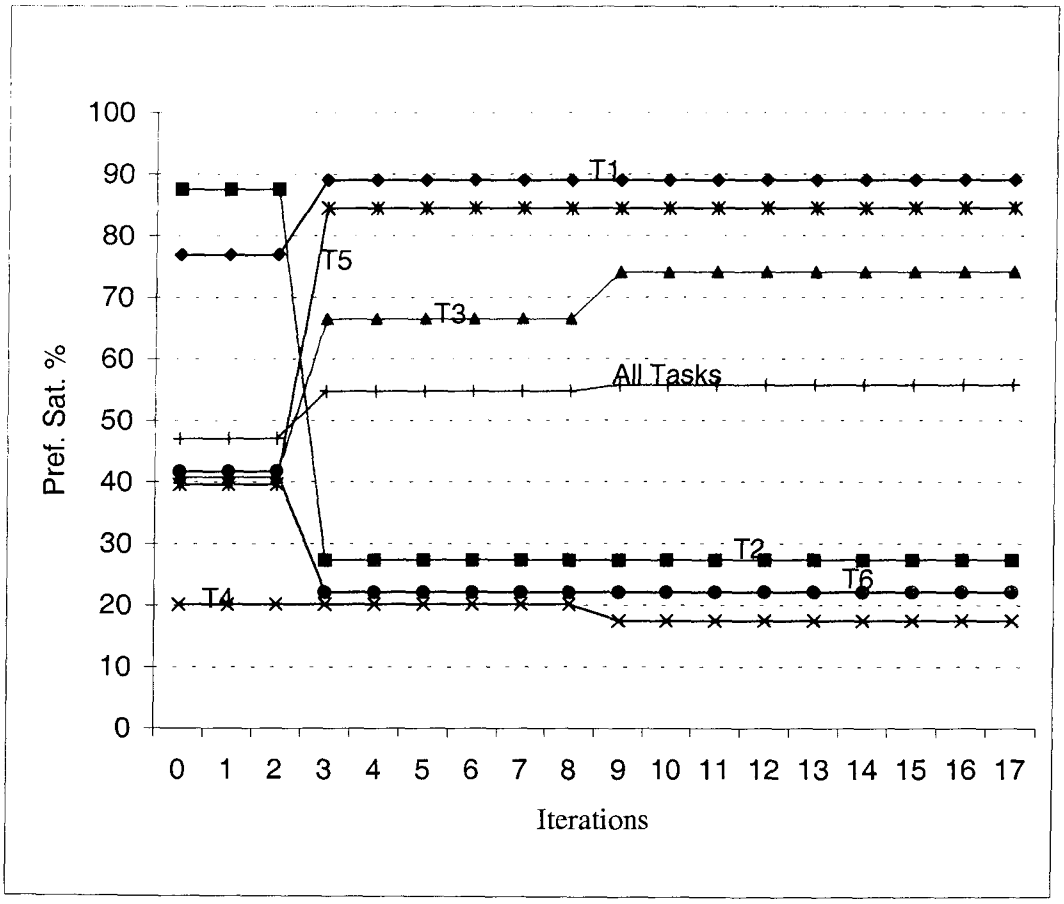


Fig. 7.5 Preference satisfactions for all the tasks when reallocating T^3 only and considering other tasks preferences

7.2.3 Reallocation of all Tasks

In this group of experiments we demonstrate the overall gain in preference value when all tasks are reallocated. The procedure is the same as in the previous two groups of experiments, but instead of considering the reallocation one task at a time, all tasks are allocated concurrently. We expect this to affect the overall gain in preference value for all tasks. This value should be higher than the value obtained when reallocation was performed on individual basis as, in this experiment, all the tasks are cooperating to get a better overall preference value. Also, the gain in preference values for each individual task might be different to the one obtained during the individual reallocation.

Initially all tasks are allocated to the first available slot, tasks being allocated according to the order T_1, T_2 ... T_6, without considering preferences. Then all the tasks are reallocated concurrently. The result of this experiment is shown in Figure 7.5.

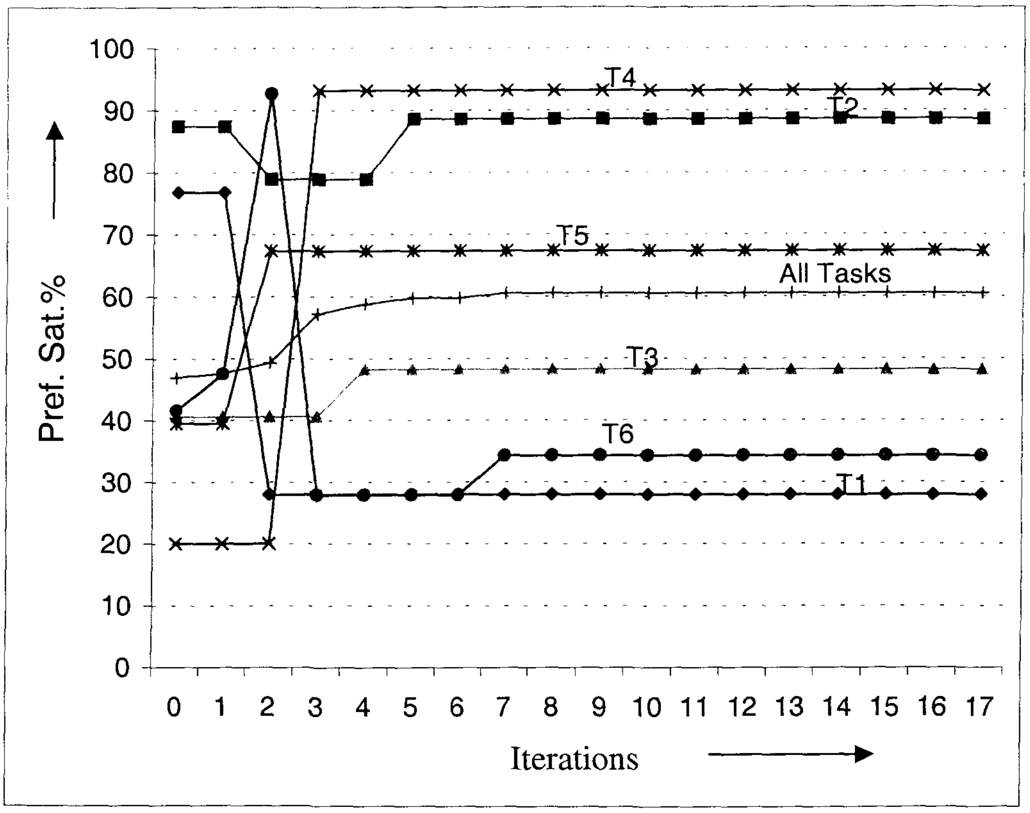


Fig. 7.6 Preference satisfactions for all the tasks when reallocating all the tasks

Figure 7.6 shows how the preference value (G^i) for each task T_i varies during the reallocation process and it also shows the variation of the overall preference value. The graphs in this figure follow the same pattern as the ones in the previous experiments, though the final satisfied preference values are different. In this case there are more exchanges of preference values among the tasks. Compared with the initial satisfied preference value, the final satisfied preference values for all the tasks, apart from T_1 and T_6, have increased. This experiment shows that the preference model converges to a solution that satisfies as many preference values as possible.

7.3 Investigation of Convergence Characteristics

To demonstrate that the preference model also works satisfactorily for different configurations of tasks and subtasks, we repeated this experiment for different numbers of tasks and coordinators as well as different cut-off values and different initial allocation ordering. Sample presentations of the results of these experiments are shown in Figure 7.6, Figure 7.7, and Figure 7.8, the rest are shown in Appendix C.

7.3.1 The Effect of Initial Order

For this experiment we used 6 coordinators and 24 subtasks. The tasks are allocated first using initial order $T_1, T_2, \ldots, T_6$. The experiment is then repeated using a reverse initial order $T_6, T_5, \ldots, T_1$. This is also repeated for another two random initial orders. Figure 7.7 shows the results of these experiments. The results show that the convergence is stable as the final preference value in the investigated cases converges to similar values (61%, 60, and 59%). This is expected as the tasks negotiating are not the same and it takes a different number of iterations, depending on the initial situation, to reach the same final preference value. This also affects the number of iterations needed to reach convergence. Figure 7.7 shows that the iteration cycle at which this convergence is achieved differs from one case to another. For example, for the order of allocation T_1, T_2, . . . , T_6 convergence occurs at iteration 8, for the reverse order of allocation convergence occurs at iteration 6, and for random order of initial allocation convergence occurs at iteration 12.

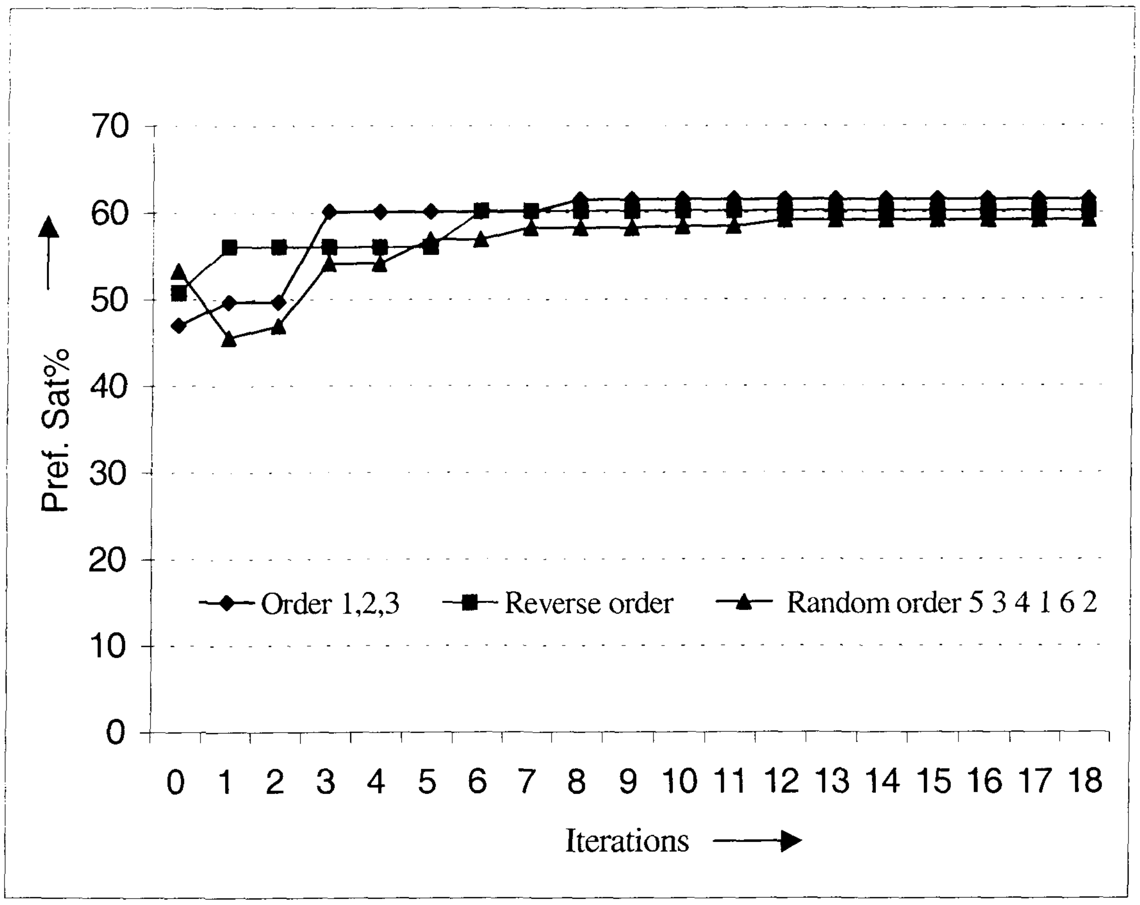


Fig. 7.7 Effect of initial allocation order on convergence characteristics

7.3.2 The Effect of Varying the Number of Coordinators and Subtasks

To investigate the effect of the number of tasks and coordinators on the convergence characteristics, we investigated five different cases: A, B, C, D, and E. Cases A and B use 6 coordinators and 24 subtasks, C uses 7 coordinators and 29 subtasks, while D and E use 14 coordinators and 54 subtasks. Although some cases share the same number of coordinators and subtasks, the subtask attributes (preference value, offer, and so on) are different. The results are shown in Figure 7.8. This demonstrates that the gain in preference value follows the same pattern in all cases. In all these cases the gained preference value converges to a value larger than the initial value, though this gain might be small, as in case D. These results motivated us to try to find a mathematical model that would describe these curves and to search for the parameters that affect their characteristics. This led us to formulate the theoretical distribution model discussed in chapter 5, and to perform more experiments, presented in chapter 8, to validate the model and to investigate the effect of the different parameters of the preference model.

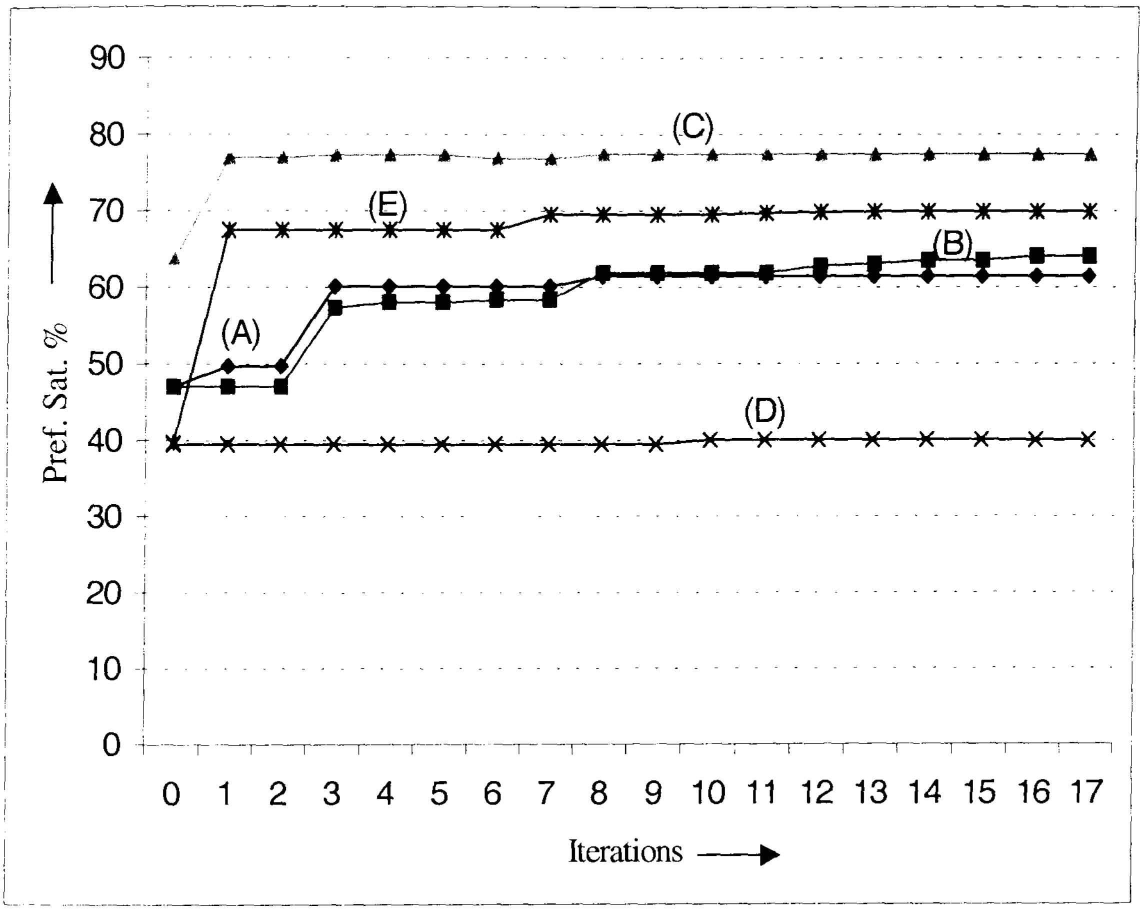


Fig 7.8 Preference satisfaction for different cases

7.3.3 The Effect of Varying Cut-off Value

To investigate the effect of varying the cut-off value (φ) on the convergence characteristics. Initially we used 10 coordinators and 45 subtasks and used different values for φ (5, 15, 25, and 35). The results are shown in Figure 7.9. We also conducted the same experiment using different numbers of coordinators and subtasks, the results of which are shown in Appendix C. These diagrams shows that φ has little effect on the final preference value (± 1%) but that it did affect the number of iterations needed to reach this final value. From the results of these experiments we can say that the lower the value of φ the fewer iterations were needed to reach convergence. Initially we thought the effect of φ might be much larger than this. To investigate this we undertook the experiments in section 7.4 so that we can identify the sources of losses.

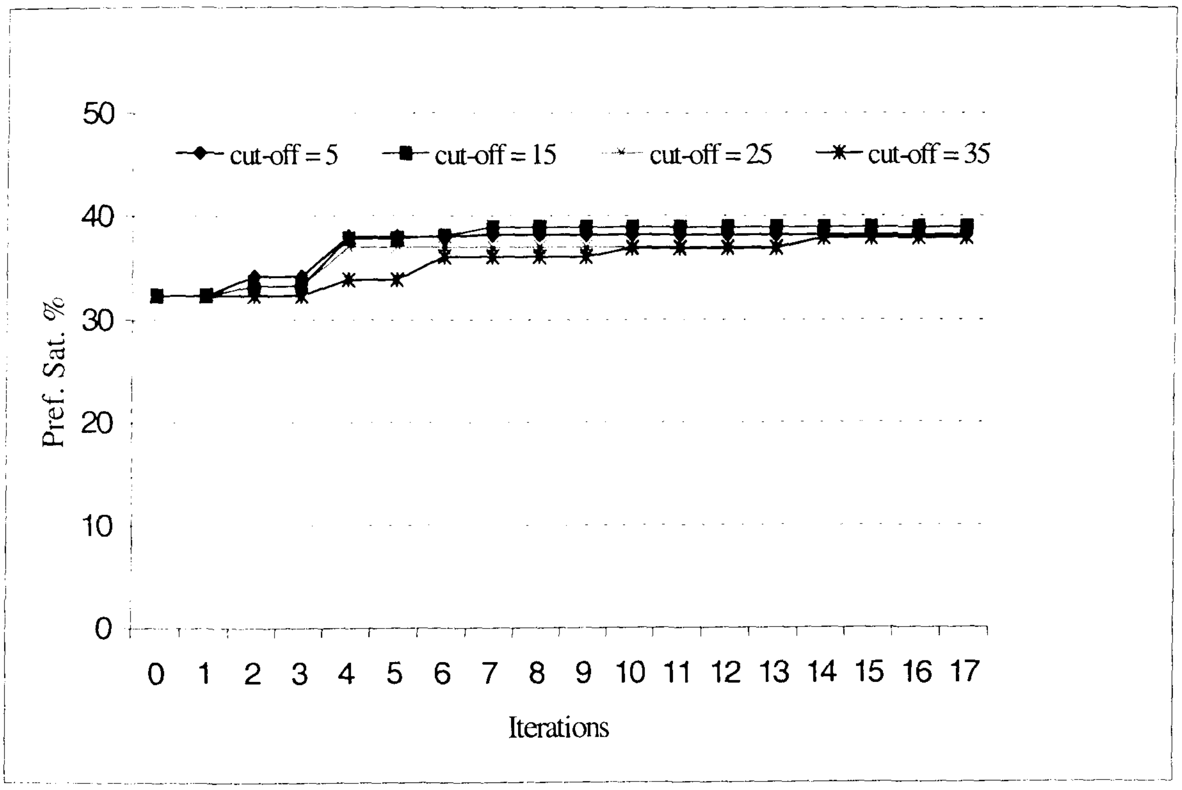


Fig 7.9 Varying the cut-off value (φ)

7.4 Sources of Preference Loss

Five sources of preference loss during the allocation process were identified in section 4.3.3. To recall, these sources are:

(a) Bid failure (see step 7 in the allocation process)

(b) The cost cut-off (see step 8 in the allocation process)

(c) Preference cut-off φ (see step 11 (a) in the allocation process)

(d) Delayed cut-off φ (see step 11(b1) in the allocation process)

(e) The π loss (see step 11(b2) in the allocation process)

The bid failure in (a) is a result of severe conflicts on resources, so no movement is possible. The loss due to preference cut-off (c and d) implies that the conflicts are still severe, but not as bad as in (a). In (e), also due to resource conflict, the preference gain exceeds the cut-off, but not as much as expected. The conflicts on resources are better than in (c) and (d). The cost cut-off loss is due to lack of funds, either because exchanges are too expensive as a means, or because the fund has run out.

We conducted several experiments to monitor the above losses and see if we can derive a form of relation which would predict and control these losses

Table 7.1 shows a typical result from one of these experiments in which we used 6 coordinators, 24 subtasks, the preference cut-off value (φ) is set to 30%, and the preference loss rate (ρ), see section 4.2.1, is set to 5%. Results of other cases are shown in Appendix D. For the purpose of these experiment we treated the preference cut-off and the delayed cut-off (c and d) as one item, referred to as the average preference cut-off loss.

Examining the results of these experiments we find it difficult to predict which exchanges would be considered too expensive and when the funds would actually run-out, particularly since the coordinators both gain and lose funds from each other like money in a market during the processing. These preference losses are not independent. The failure of a subtask to gain preference value due to any one of the above mentioned four reasons might have enabled it to make a much better gain later and this is difficult for us to know or predict. Therefore, monitoring the preference losses from these four sources at each iteration would not guide us to control or predict the solution during the allocation process.

Table 7.1. Sources of preference loss (24 tasks)

Subtask	Average pref. cut-off loss	Cost cut-Off loss	Average Pi loss
T11	7.3	5.0	0.0
T12	10.5	29.7	0.0
T13	8.3	0.0	0.0
T14	0.0	5.0	56.4
T21	0.0	6.6	10.0
T22	11.3	58.0	24.0
T23	8.0	0.0	27.7
T31	0.0	0.0	0.0
T32	11.8	35.4	53.1
T33	10.4	9.7	64.1
T34	7.2	9.1	71.3
T35	5.0	5.0	64.4
T41	7.3	0.0	11.1
T42	9.0	0.0	10.6
T43	0.0	0.0	3.5
T44	8.7	0.0	17.0
T45	10.0	15.2	9.8
T51	6.0	6.0	5.3
T52	9.0	0.0	7.5
T53	9.3	0.0	20.7
T54	10.0	27.0	35.1
T61	7.0	0.0	10.5
T62	10.3	10.4	15.5
T63	6.3	32.8	25.8
Total	8.6	18.2	22.6

Table 7.1: This Table shows the different sources of preference loss during the allocation process for 24 tasks and 6 coordinators. φ=30% and ρ =5%.

7.5 Summary

We presented the simulation results of the initial experiments conducted on the simulator described in Chapter 6. We used scheduling of different numbers of distributed tasks as a case study in our simulation.

We used the first set of experiments to demonstrate that preference model converges and to show that there is a common pattern to this convergence. These experiments demonstrated that φ has little effect on the final preference value (± 1%) but that it did affect the number of iterations needed to reach this final value. We concluded that the lower the value of φ the fewer iterations were needed to reach convergence.

In the second group of experiments we investigated the different sources of preference losses. We found it difficult to predict which exchanges would be considered too expensive and when the funds would actually run-out as the coordinators both gain and lose funds from each other like money in a market during the processing. Thus, we came to the conclusion that monitoring the preference losses from the different four sources at each iteration would not guide us to any way of controlling or predicting the solution during the allocation process.

Although we can conclude from these results that the preference model provides a convergence to a "satisfactory" global solution, these experiments do not show the effect of task parameters on the convergence to a solution. Initially we thought we could predict the solution by monitoring preference losses, in order to predict the solution. The experiments in section 7.3 proved us wrong. Therefore we conducted further experiments using different combinations and permutation of tasks and coordinators. These experiments, which are presented in the next chapter, gave us deeper insight into the working of the preference model.

To concentrate on the fundamental characteristics of the preference model, the experiments presented in this chapter dealt with a single preference value. In our discussion of the preference model in section 4.2 we made the assumption that preferred resource types of a subtask are orthogonal to one another. Thus, to calculate the overall gain in preference value we need to sum the individual gain in each preference value for each source type, this involves extra programming and extra processing. This would only have affected the number of iterations needed to reach convergence and the gain in preference value. The general characteristics would remain the same. Therefore, we decided to concentrate on implementing a single preference value and defer the idea of using multiple preferences for future work.

The experiments, presented in this chapter, were performed during the initial stages of the research and led us to the mathematical formulation presented in chapter 5. Also, these experiments motivated us to explore the effect of the distribution of preference values and other parameters on the functionality of the preference model. In the next chapter we present further experiments that cast more light on the behaviour of the preference model.

Chapter 8

Simulation Results - Further Study

The results in the previous chapter motivated us to conduct further experiments to explore the working behaviour of the preference model. In this chapter we aim to present results obtained from these extended experiments. The experiments in this chapter use the same simulation study for scheduling in a manufacturing environment as was used in the previous chapter. The experiments that are presented in this chapter are divided into the following two sets:

I. The experiments that are presented in the first set are those experiments that were conducted to illustrate some of the basic desirable properties of the preference model, namely: that the solution converges to a stable final preference value independent of the initial order of subtask processing and that its value is also stable against reasonable variations in the preference cut-off value φ and the offer price.

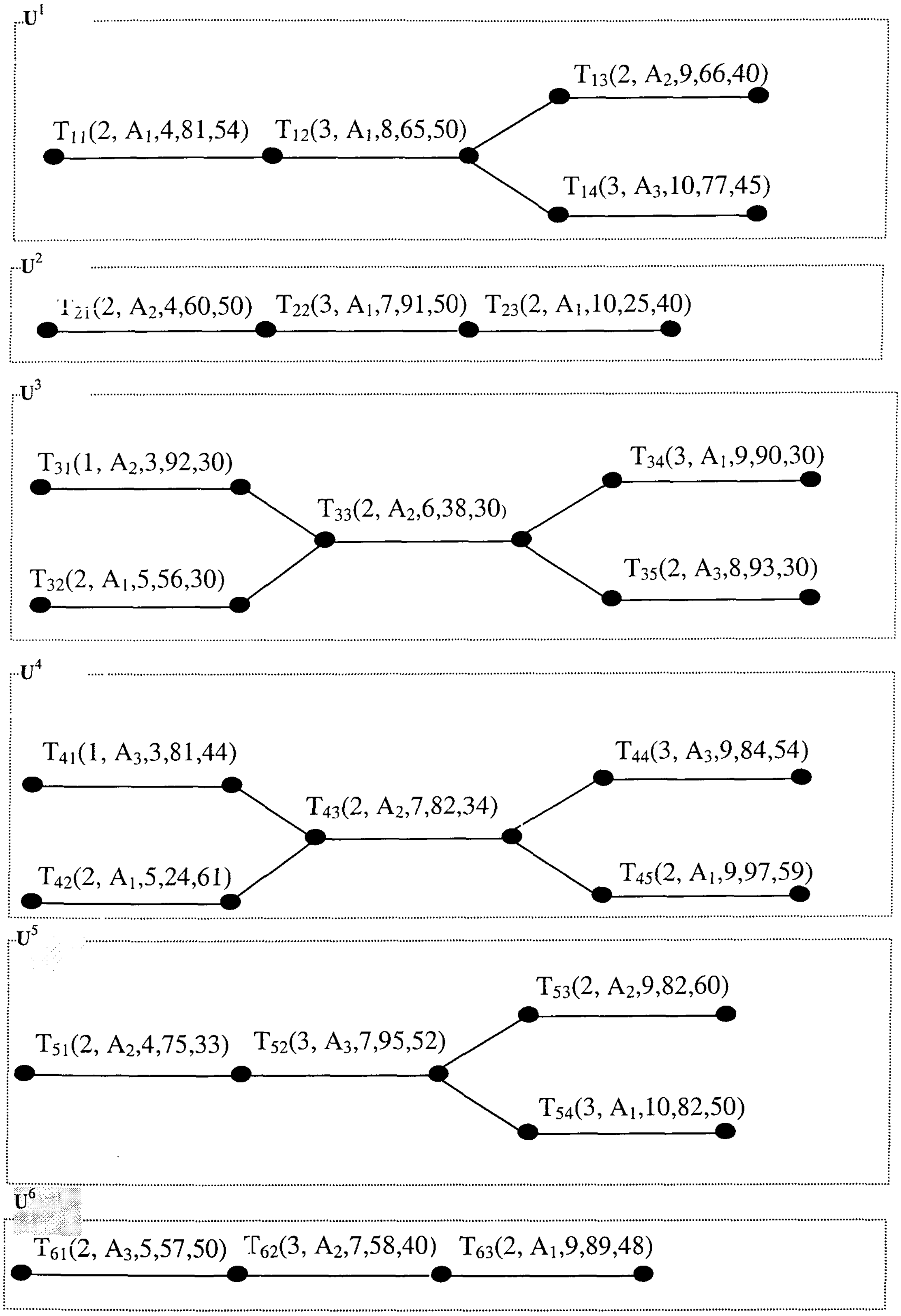
II. The experiments that are presented in the second set are those experiments that were conducted to show that the results conform with the predicted results calculated by using the formulae from the theoretical model developed in Chapter 5.

We used the PMS implementation described in Chapter 6 for the purpose of these experiments. For the simulation study we have used a set of coordinators, each responsible for a global task that is being further subdivided into subtasks. Our objective is to find a schedule that satisfies as many preference values as possible, for which we have used the algorithm outlined in Chapter 4. In section 8.1 we outline the case study and we present our results in the subsequent sections.

8.1 The Case Study

As the experiments in this chapter are an extension of the ones described in the previous chapter, we have used the same types of configuration. A detailed presentation of the case study was given in section 7.1, for the sake of clarity we give a brief summary of the case study and show a presentation of the case study in Figure 8.1 with different parameters and configuration. This figure shows the coordinators, tasks, subtasks, and their attributes, for 6 coordinators. In this case study we used a set of coordinators U^1, U^2, .. U^n, each U^i with a (global) task T^i, each task T^i being further subdivided into subtasks: $T^i_j \mid \{j = 1, 2, .. m\}$, one or more subtasks being allocated to a home agent A_k. We used up to n = 14, up to six subtasks (m= 6) in each task, and three home agents (k=3). Also we used the algorithm outlined in section 4.3 to find a schedule that satisfies as many preference values as possible. We used a varying number of coordinators and tasks with different attributes to obtain a large number of results.

So that our results would not be biased we adopted the same strategy in choosing the test cases as the one described in section 7.1.


U^1
T_{13}(2, A_2,9,66,40)
T_{11}(2, A_1,4,81,54)
T_{12}(3, A_1,8,65,50)
T_{14}(3, A_3,10,77,45)
U^2
T_{21}(2, A_2,4,60,50)
T_{22}(3, A_1,7,91,50)
T_{23}(2, A_1,10,25,40)
U^3
T_{31}(1, A_2,3,92,30)
T_{34}(3, A_1,9,90,30)
T_{33}(2, A_2,6,38,30)
T_{32}(2, A_1,5,56,30)
T_{35}(2, A_3,8,93,30)
U^4
T_{41}(1, A_3,3,81,44)
T_{44}(3, A_3,9,84,54)
T_{43}(2, A_2,7,82,34)
T_{42}(2, A_1,5,24,61)
T_{45}(2, A_1,9,97,59)
U^5
T_{53}(2, A_2,9,82,60)
T_{51}(2, A_2,4,75,33)
T_{52}(3, A_3,7,95,52)
T_{54}(3, A_1,10,82,50)
U^6
T_{61}(2, A_3,5,57,50)
T_{62}(3, A_2,7,58,40)
T_{63}(2, A_1,9,89,48)

Parameters: (Task Name (duration, resource agent, preferred end time, preference value, offer price))

Fig. 8.1 A presentation of the case study for n = 6

We classify the experiments into two sets. One set concerns those experiments that were conducted to confirm some basic desirable properties, while in the second are those experiments that were conducted to verify our theoretical model. In the next sections we present the results of these experiments and then we discuss their findings.

8.2 Verification of Basic Properties (First Set)

The objectives of these experiments are to show that:

III. The solution converges to the same final preference value independent of the initial order of subtask processing.

IV. That value is also stable against reasonable variations in the preference cut-off value φ.

V. That value is stable against higher offer prices by some coordinators.

In the following subsection we present the experiments we have conducted to achieve the above objectives.

8.2.1 Variation in Subtasks Processing Order

In section 7.3.1 we presented an experiment that investigated the effect of changing the order of the initial allocation on the convergence to the final value. We repeated that experiment but this time the specified end-time preference value was set to be the same as the one achieved in the initial allocation. So that if these subtasks are allocated in the arrival order (which was the end time order) over three target agents without paying any attention to their preferences, 100% preference values will be automatically achieved. In order to show that the solution converges to the same final value independent of the initial order of subtask processing, we carried out an experiment with 24 subtasks of all mixes but with non-conflicting end times slots (preferred resource). We then allocated these tasks in the reverse order without taking any preference into account. This yielded 30% preference gain. On this distribution we applied our model and re-allocated the subtasks, this time (iteration 1) taking preferences into account. This first iteration achieved 100% gain. We repeated this experiment with different initial ordering, and each case 100% gain was achieved at the first iteration. In these experiments the value of preference cut-off, f, was kept fixed at 5%.

We show the results of this experiment, which were performed using 6 coordinators and 24 subtasks in Figure 8.2. This figure shows subtask allocations after the initial allocation and for the subsequent two iterations. The order of the initial allocation was in random order. As can be seen from the figure, the maximum gain in preference value was achieved after the first iteration. There are no changes on the allocation after the first iteration.

The results of this experiment and the experiments presented in 7.3.1 confirms that our model behaves as we expected, and that it does lead to convergence. A significant point is that this model produces results which are independent of initial allocations (i.e the order of subtask processing), this is difficult to achieve using traditional machine scheduling.

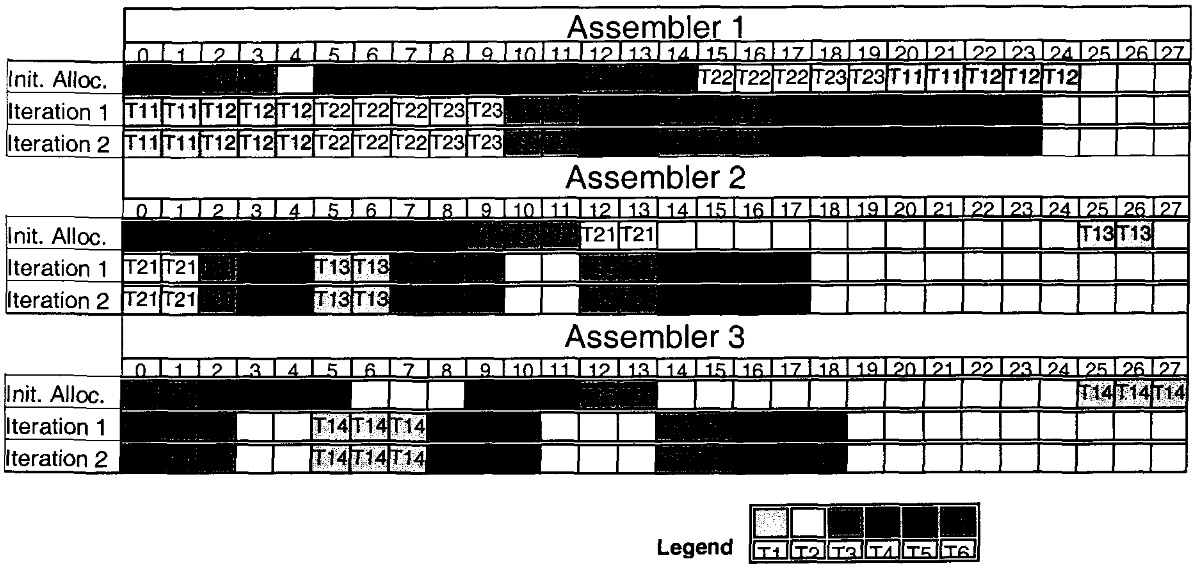


Fig.8.2. The results of allocation of subtasks with non-conflicting end times

8.2.2 Variation in Cut-Off Value φ

Our reason for using the preference cut-off φ was to reduce unnecessary processing and to avoid cycles. The effect of a cut-off value is complex and hence needed investigation. We were interested in answering the following questions:

I. What should be the correct value of φ?

II. Is it sensitive to some values?

In order to answer these questions we carried out experiments that varied the preference cut-off values from 1 percent to 100 percent of each subtask preference value. Figure 8.3 shows the results of these experiments. We can see from the graph in this figure that the resultant preference gain is largely flat up to 20 percent cut-off values, and then tapers gently downward for higher cut-off values. This result gives us the confidence that the final preference gain is not sensitive to any reasonable cut-off value, which is likely to be between 1 and 10 percent.

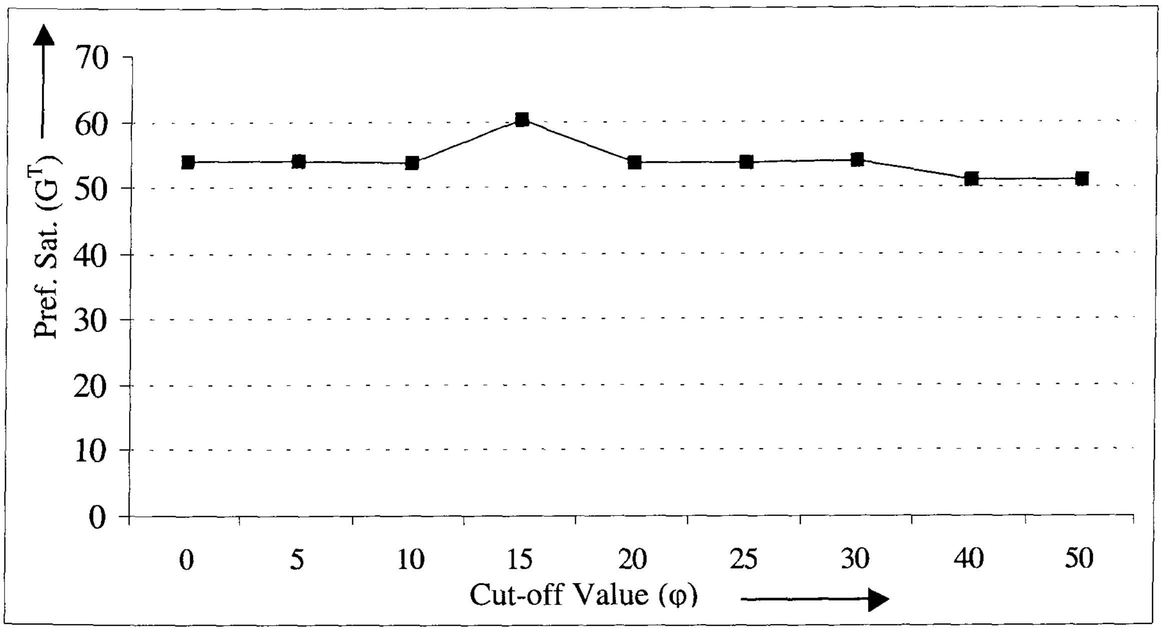


Fig 8.3. Effect of varying cut-off value (φ) on preference satisfaction (G^T).

8.2.3 Variation in Offer Price

We investigated whether higher offer prices by some coordinators can distort the results significantly. We used six coordinators, each task T^i of the coordinator U^i having *m* number of subtasks, *m* varying from 3 to 6. Initially all subtasks were allocated on the first available (time) slots in the order T^1, T^2, ..., T^6, at a given offer price (taken from Figure 8.1) and preference cut-off value (φ), but without considering preferences.

Then a series of iterations was carried out for the same φ. At each iteration, the offer price of one of the coordinators was raised by 5%, and its subtasks reallocated, taking only its preferences into consideration. The results presented in Figure 8.4 shows the changes in the preferences satisfied during the allocation of each task (T^1, T^2, ..., T^6) in which only preferences of that task were taken into account. So our conclusion is that higher offer prices do not make any drastic change, and therefore our model produces a stable preference gain.

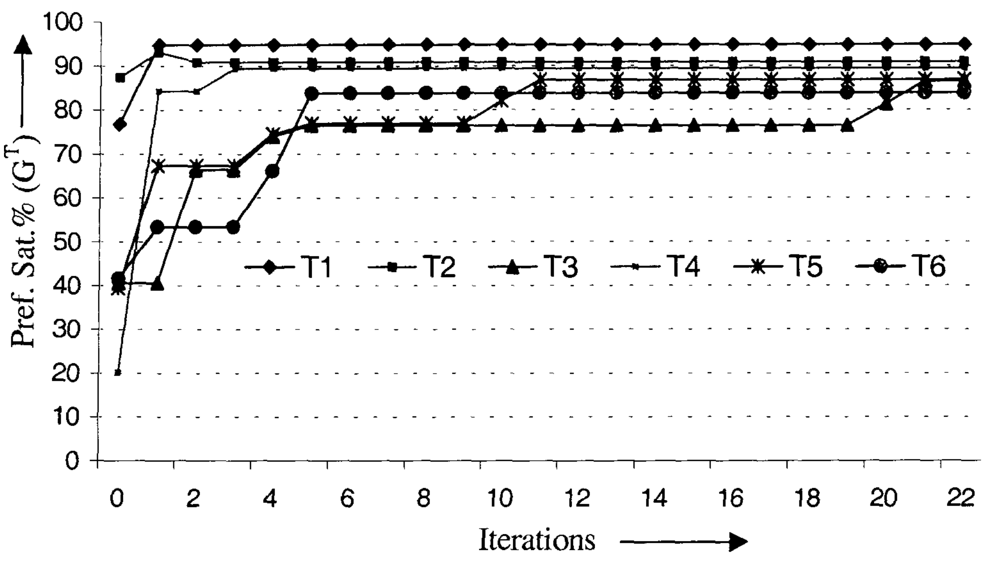


Fig. 8.4. Preference satisfaction (G^T) over iterations.

Next we examined whether the preference gain and the cost of one task is affected by the increasing offer prices of the other coordinators. We have conducted this experiment for each task T^i, but selected arbitrarily to show it for T^3 in Figure 8.5, which was typical. It shows the preference gain and cost for T^3, against the accumulated offer prices of the other coordinators. Evidently the increase in their offer prices did not affect the preference gain of T^3s in any significant way. This result is typical for all T^i's.

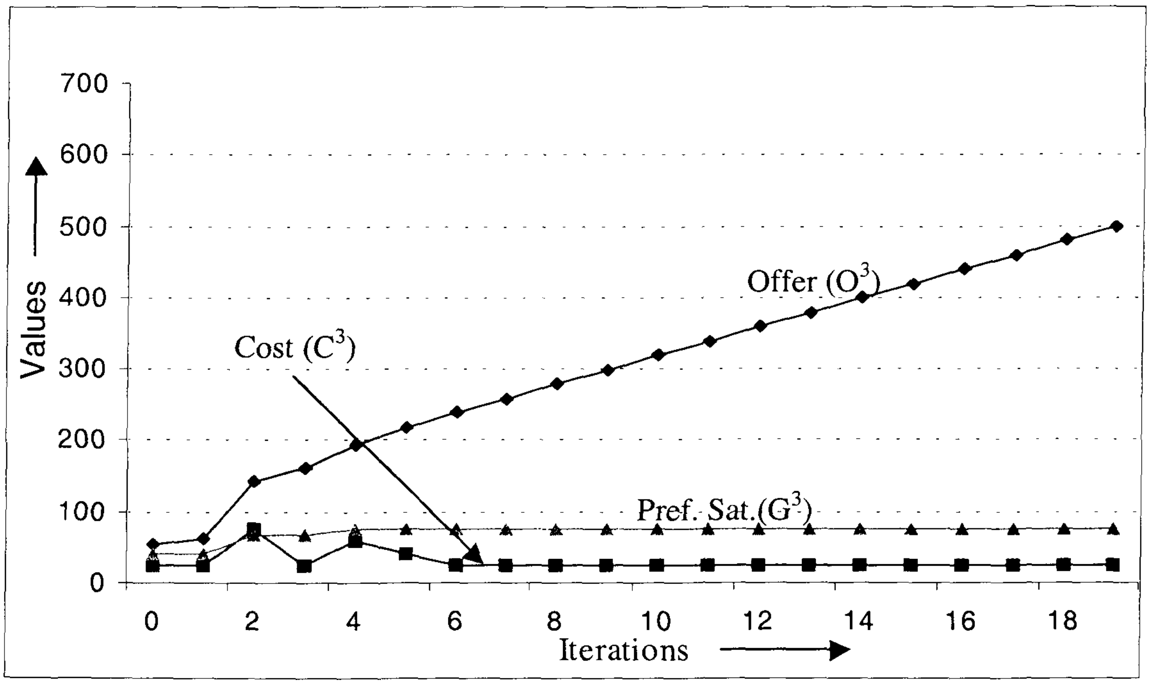


Fig. 8.5. T^3 preference satisfactions, offer & cost variation.

8.3 Verification of the theoretical model (Second Set)

Our aim from this set of experiments was to verify that the preference gain over iterations obeys the theoretical model discussed in Chapter 5. In these experiments we used different distributions of subtask preference values. Distributions from six experiments on preference-based task allocations by three target agents are shown in the Figures 8.6(a) to 8.6(g). Preferences were assigned on end-times for 29 subtasks (7 coordinators) in Figure

8.6(b), for 12 subtasks (3 coordinators) in Figure 8.6(c), and for 54 subtasks (14 coordinators) in the other figures. The preference cut-off value (φ) and the preference loss rate (ρ), see section 4.2.1, were sometimes varied as shown in the tables later. The distributions in the last four figures are skewed (see section 5.3). We shall use these distributions in our analyses presented in the following subsections

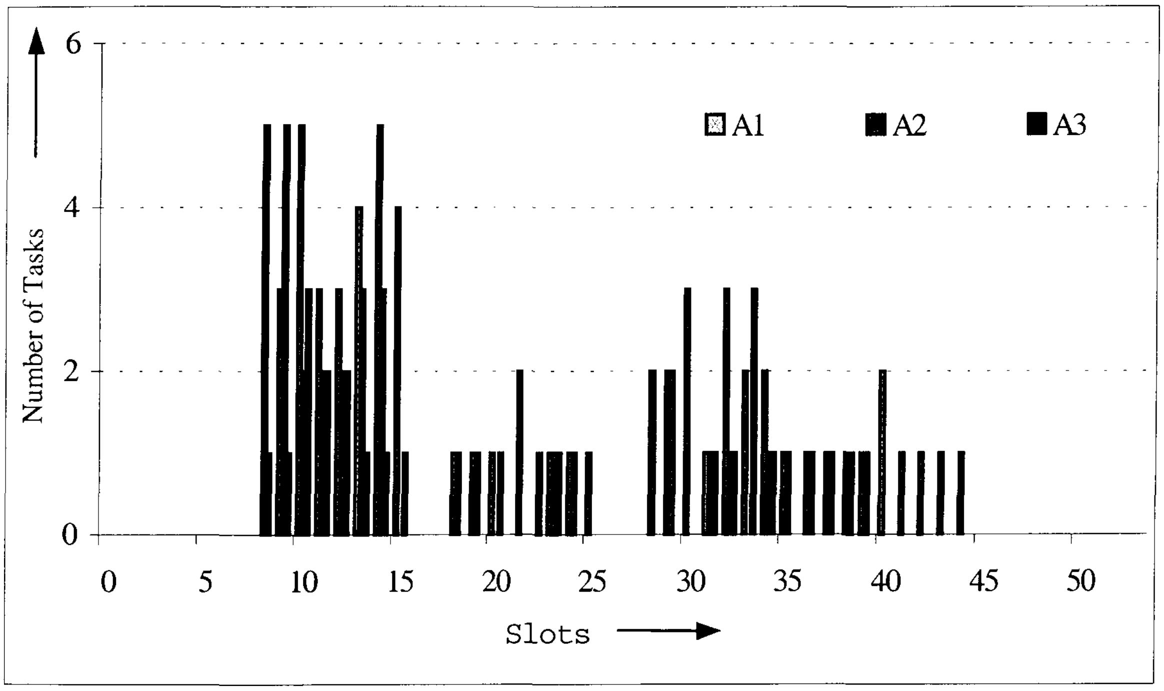


Fig. 8.6(a). Skew-free distribution (54 subtasks).

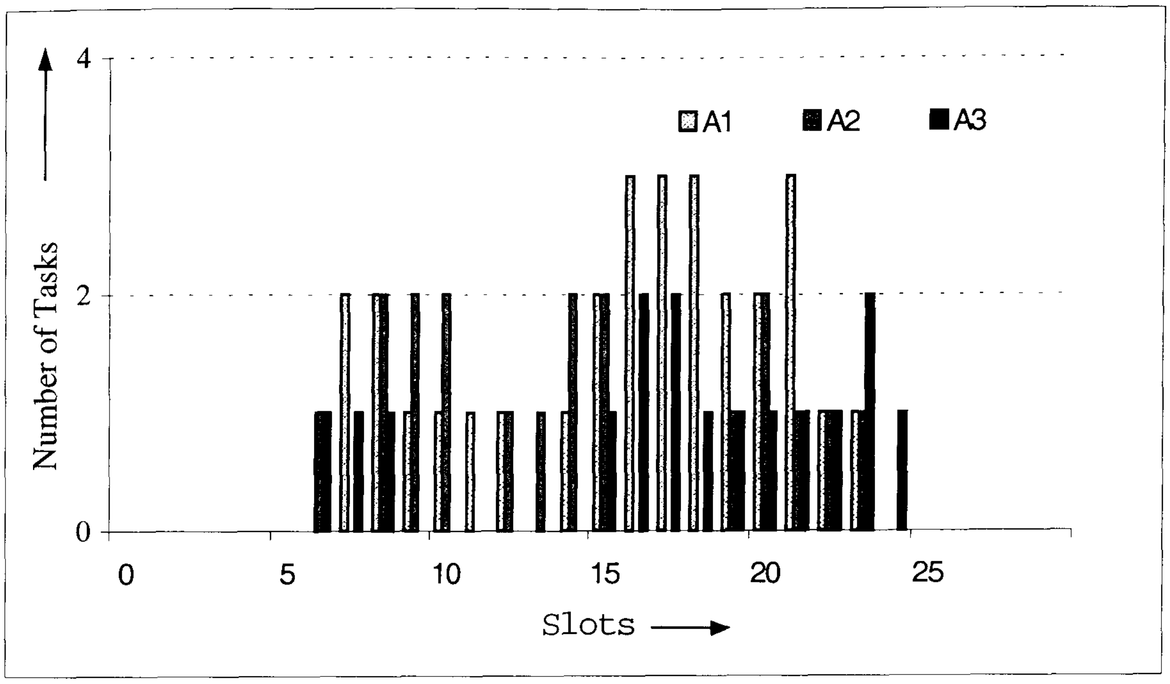


Fig. 8.6(b). Skew-free Distribution (29 subtasks).

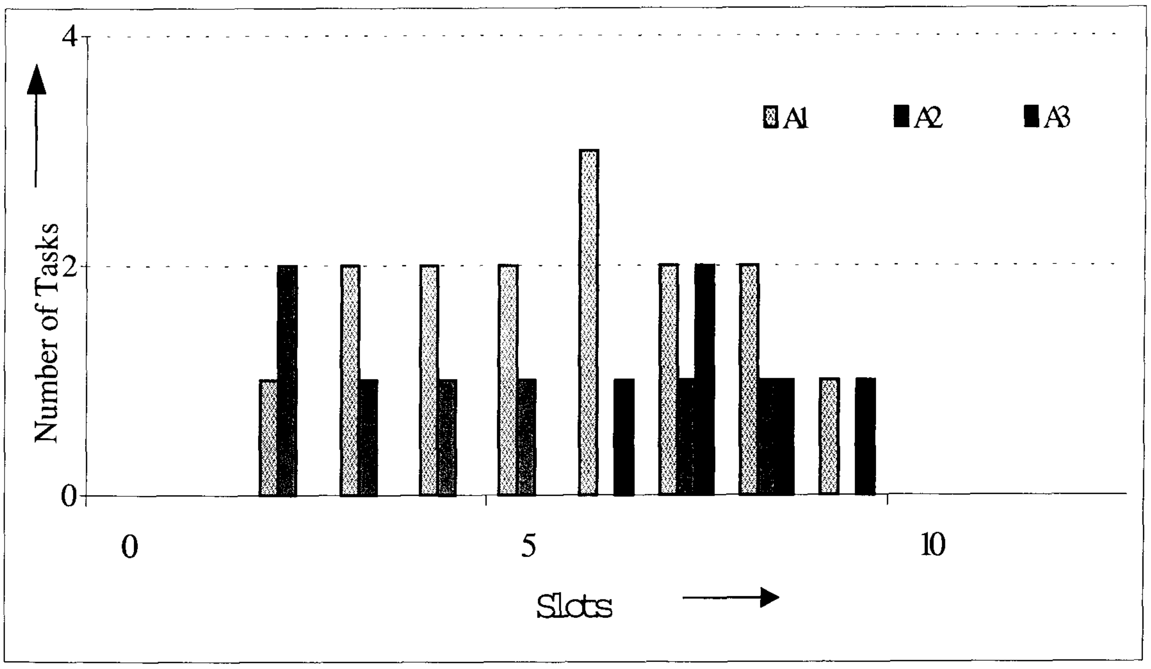


Fig. 8.6(c). Skew-free distribution (12 subtasks).

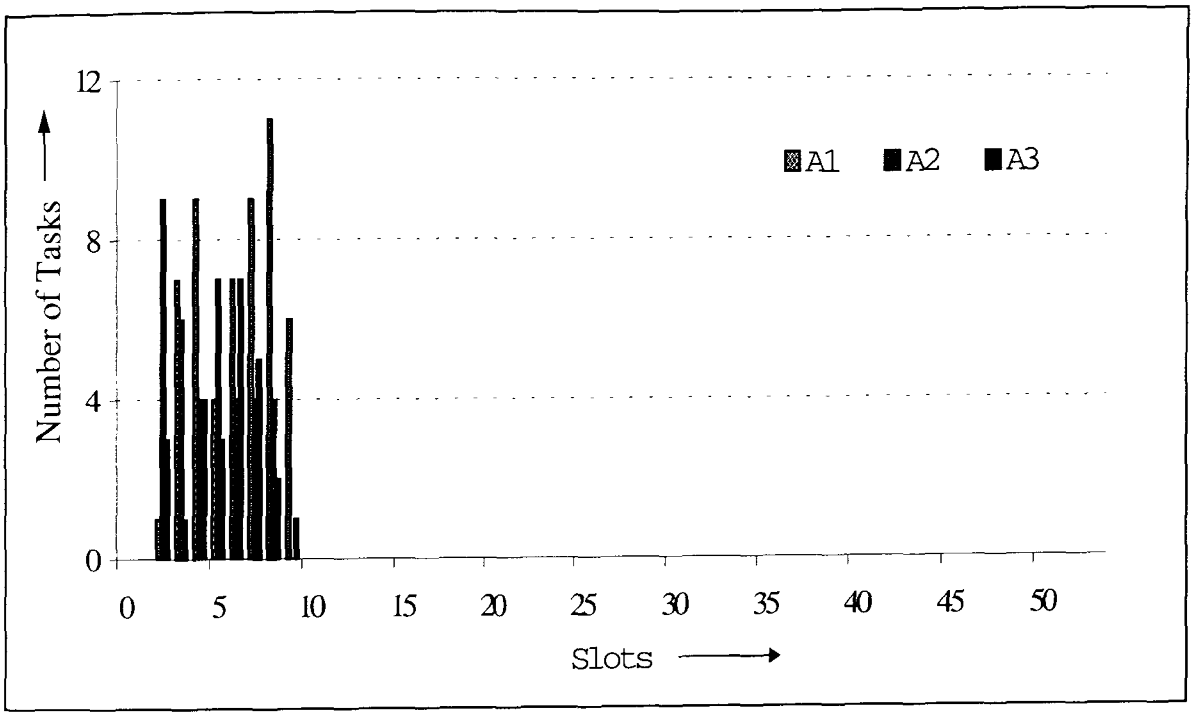


Fig. 8.6(d). Skewed distribution (54 subtasks).

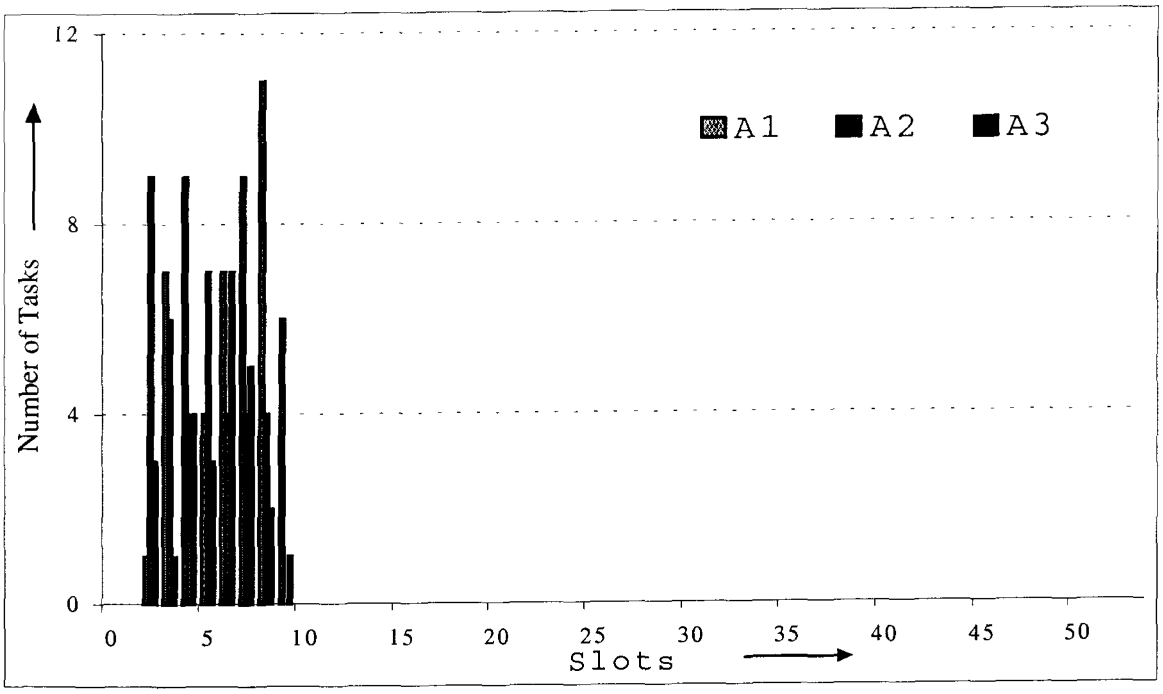


Fig. 8.6(e). Skewed distribution (54 subtasks).

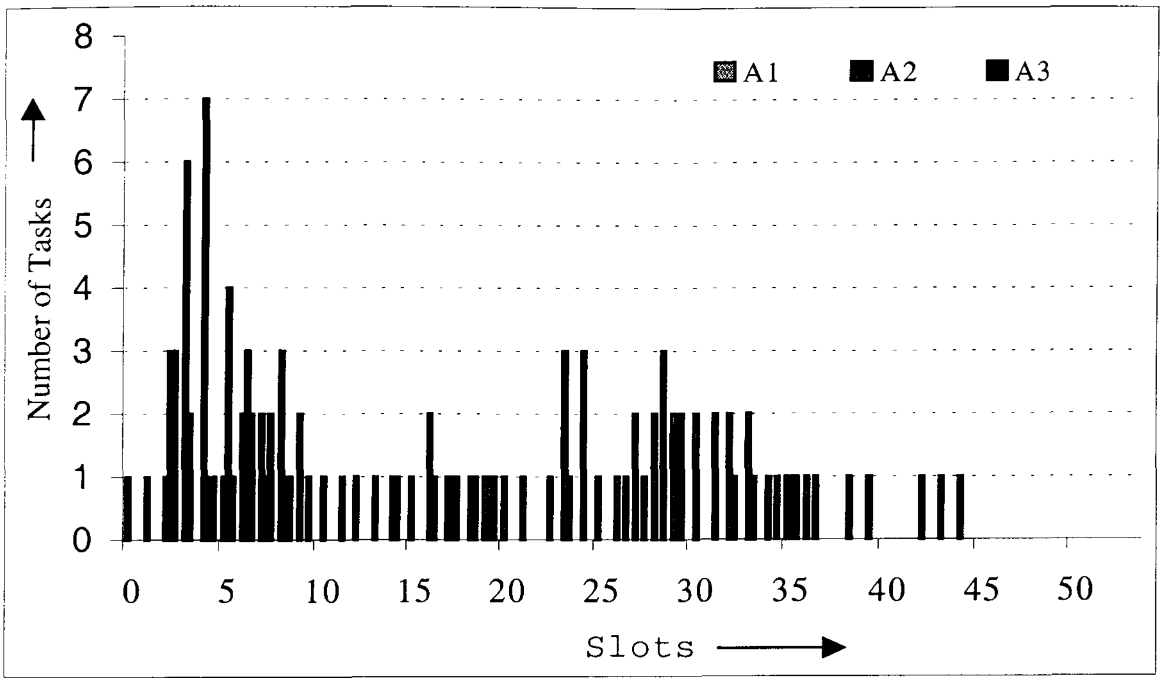


Fig. 8.6(f). Skewed Distribution (54 subtasks)

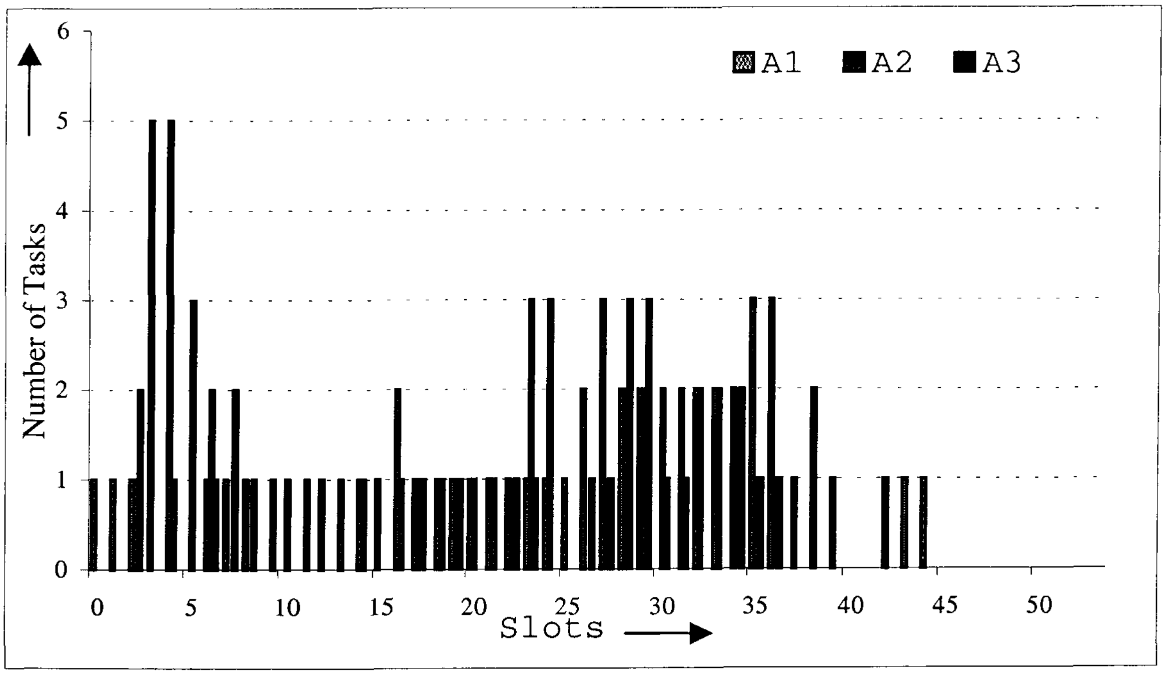


Fig. 8.6(g). Skewed distribution (54 subtasks)

We categorise the results of the experiments under the following three categories:

1. The effect of variable distribution and fixed φ value on the exponential function of the predicted remaining preference values, R^T.
2. The effect of variable φ value and fixed distribution on R^T.
3. The effect of variables φ and ρ on R^T.

We present the results for the experiments that fall under the first category in subsection 8.3.1, the results for the second category in section 8.3.2, and the result for the third category in subsection 8.3.3.

8.3.1 Fixed φ Value and varied Distribution.

In this study we used data from Figures 8.6(d), 8.6(e), 8.6(f), and 8.6(g), with the preference cut-off value φ fixed at 5% to demonstrate that the distribution of R^T over the iterations matches the exponential pattern predicted by the theory. We show the exponential fit on the results in Fig 8.7. The graph shows that all the cases obey the predicted exponential curve. The final value of R^T is different for each case shown in the figure.

We can conclude that this variation in R^T is due to the distribution effect as this is the only factor that changed during these experiments. Examining the distribution graphs in Fig. 8.6 we can observe that the more dispersed the distribution of preference values the less is the value of R^T. For example for the sparse distribution of preference values shown in Fig 8.6(g) the value of R^T is around 25%, while for the more congested distribution of preference values shown in Fig 8.6(d) the value of R^T is around 60%. This behaviour confirms to the formulation of the theoretical model discussed in section 5.2.

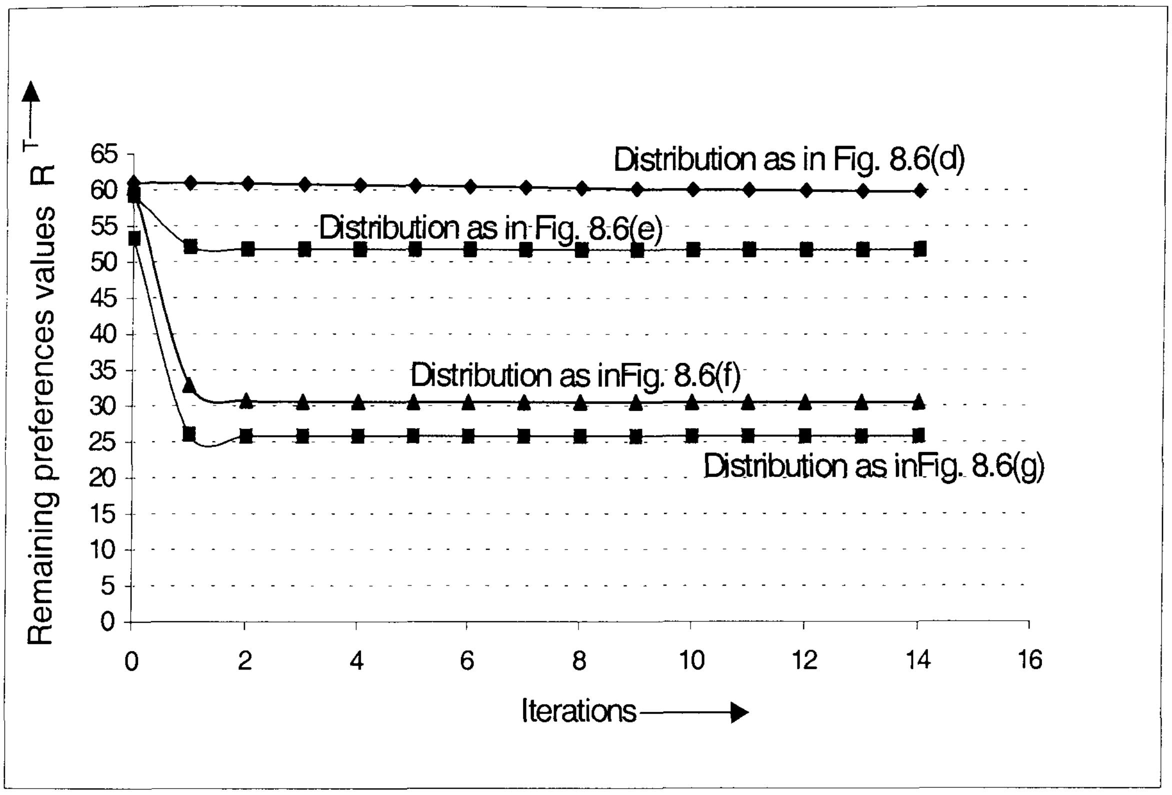


Fig. 8.7. Confirmation of the Exponential Pattern for R^T for Fixed φ.

8.3.2 Variable φ value.

As mentioned, φ is used to avoid cycles that lead to non-convergence. To demonstrate the effect of φ values on preference gain (inverse of the remaining preference value R^T), we have used the distribution of preference values from Figure 8.6(a), with $\varphi = 2$, 5, and 8, as displayed in Figure 8.8. For clarity of presentation the plot for φ is 8 is not shown, as its result is similar to the others.

As in the previous experiments the plots in Figure 8.8 fit our theoretical exponential distribution. The two horizontal straight lines show the predicted minimum and maximum remaining preference values R^T in %. Variation of φ value has little effect on R^T, as can be seen from the graphs in Figure 8.8. The greater the value of φ the more gain in the preference value. Using high values of φ could lead to higher gain in preference values but at the expense of more processing time. We did not investigate the effect of varying φ values on the processing time during the course of this research as processing time was not one of our major concerns. We hope to investigate this in future work.

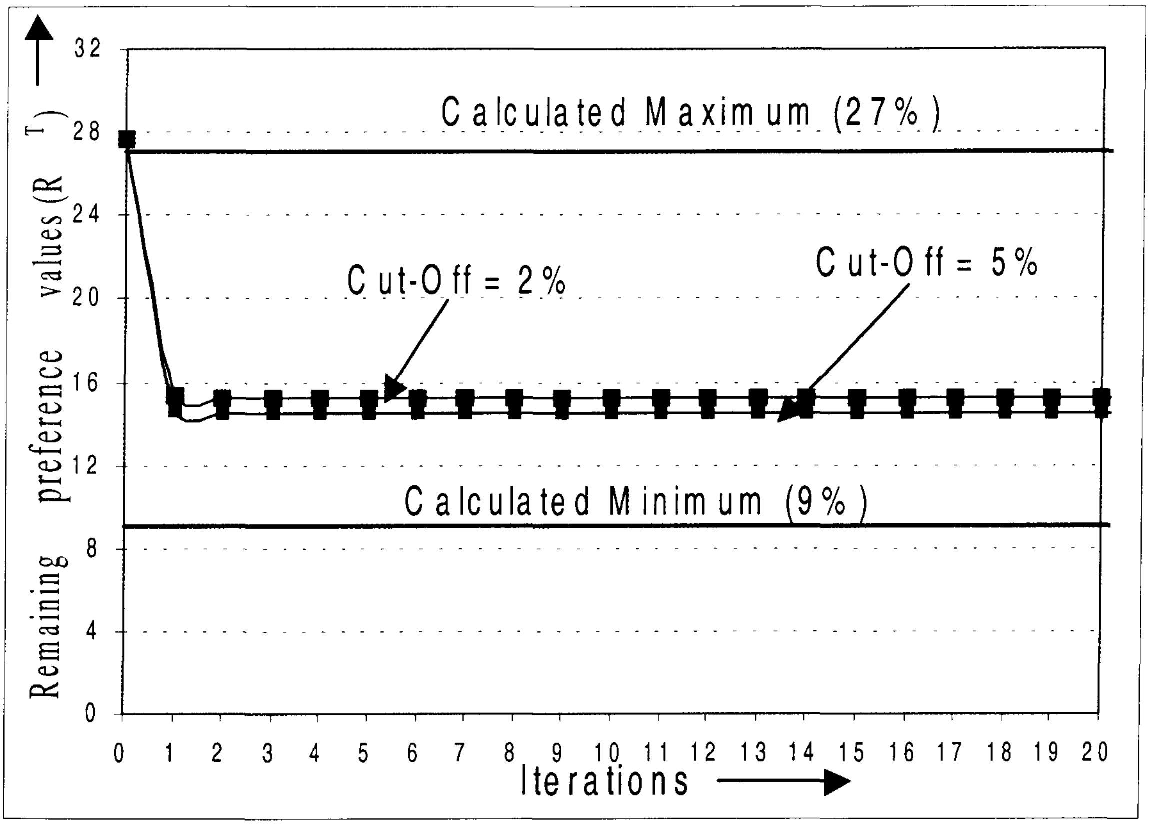


Fig. 8.8. Confirmation of the Exponential Pattern for R^T for Variable φ.

8.3.3 Predicted Remaining Preference Values

The objective of this set of experiments is to compare R^T, the predicted remaining preference values (inverse of the preference gain) after the final iteration, with the actual values obtained from the simulation experiments, based on the subtask distributions given in Figures 8.6(a) to 8.6(g). The actual values are expected to lie between the upper and lower bounds of the predicted values for both skewed and non-skewed distributions with variable values of φ and ρ, as shown in the four tables below. Tables 8.1, 8.2, and 8.3, show results for unskewed distribution for 54, 29 and 12 subtasks respectively. Table 8.4 shows results for skewed distribution for 54 subtasks.

Table 8.1. Unskewed Distribution (Figure 8.6(a))

Case	No. of Subtasks	Pref. Loss Rate (ρ)	Cut-off Value (φ)	Remaining Pref. Values (R^T)		
				Predicted		Actual
				Lower	Upper	
1	54	5	2	23	32.5	27
2	54	5	5	23	32.5	25
3	54	5	8	23	32.5	24
4	54	2	2	9	27	15
5	54	2	5	9	27	14

Table 8.1: This Table shows data from Figure 8.6(a) for 54 subtasks, with five pairs of different φ and ρ values, referred to as the five Cases, numbered 1 to 5. All simulation values lie within the predicted bounds.

Table 8.2. Unskewed Distribution (Figure 8.6(b))

Case	No. of Subtasks	Pref. Loss Rate (ρ)	Cut-off Value (φ)	Remaining Pref. Values (R^T)		
				Predicted		Actual
				Lower	Upper	
6	29	5	5	9.9	23	23
7	29	5	10	9.9	23	23
9	29	7	5	13.9	32	26
9	29	10	5	19.8	46.2	37
10	29	10	10	19.8	46.2	37

Table 8.2: This Table shows data from Figure 8.6(b) for 29 subtasks, with five pairs of different φ and ρ values, referred to as the five Cases, numbered 6 to 10. All simulation values lies within the predicted bounds.

Table 8.3. Unskewed Distribution (Figure 8.6(c))

Case	No. of Subtasks	Pref. Loss Rate (ρ)	Cut-off Value (φ)	Remaining Pref. Values (R^T)		
				Predicted		Actual
				Lower	Upper	
11	12	5	5	5	12	11
12	12	5	7	5	12	9
13	12	5	12	5	12	8
14	12	10	5	10	24	22
15	12	10	7	10	24	21
16	12	10	12	10	24	15

Table 8.3: This table shows data from Figure 8(c) for 12 subtasks, with six different pairs of φ and ρ values, referred to as Cases 11 to 16. Again the simulation values lie within the bounds of the predicted values.

Table 8.4. Skewed Distribution (results summary)

Distribution Source	Case	No. of Subtasks	Remaining Pref. values %		
			Predicted		Actual
			Lower	Upper	
Figure 8.6 (d)	16	54	57	66	59
Figure 8.6 (e)	17	54	42	51	50
Figure 8.6 (f)	18	54	22	32	30
Figure 8.6 (g)	19	54	22	32	25

Table 8.4: This Table shows data from the remaining four figures for 54 subtasks with fixed values of φ at 5% and that of ρ also at 5%. The entries are numbered as Cases 16 to 19. Again the simulation result obeys the predictions.

From the results presented in the above tables we can see that all the experimental results lie between the two boundaries predicted by the theoretical model, therefore we can conclude that the simulation experiments confirm our theoretical model.

8.4 Summary

We presented the simulation results of the experiments conducted on the simulator presented in Chapter 6. We used distributed scheduling in manufacturing as a case study in our simulation. The experiments were classified into two categories. In the first category are those experiments conducted to confirm some basic desirable properties of the preference model, while in the second category are those experiments that were conducted to verify the formulae in the theoretical model.

We can conclude the following from the results of the experiments:

1. The preference model provides convergence to a “satisfactory” global solution.
2. The initial order of allocation does not affect the final results.
3. The values calculated by the theoretical model are a good estimate of the best preference values that can be achieved by a given set of tasks and resources.

Chapter 9

Evaluation and Concluding Remarks

In this thesis we have presented an agent based cooperative preference model that can be used to solve problems in distributed systems that are often characterised by multiple valid solutions, a solution being considered to be valid if it meets all the constraints. In our approach, the different choices are specified in terms of user preferences on the different desirable aspects (i.e. resources) of the solution. Using this model, we have shown how to derive a preference-based solution in the presence of contention on resources. In that event, the best solution that can be achieved is the one that meets as many preferences as possible.

One of the most important achievements of this research is the formulation of a theoretical performance model. Using this formulation, the user can describe the quantitative behaviour of the preference model and estimates the boundaries of the solution, i.e. the best preference values that can be achieved (see Chapter 5). Our approach in deriving this formulation followed the classical scientific tradition of observation-modelling-hypothesis-experiments. This model was developed after conducting trial-and-error experimentation on a simulator designed for experimentation and testing purposes. The results from the initial experiments conducted on the simulator (see Chapter 7) not only showed that the preference model would converge to a solution, but also showed that there is a pattern to this convergence. This motivated us to explore farther and investigate to see if we could model this pattern mathematically. As a result of this exploration, and after conducting more experiments, we formulated a theoretical performance model that describes the quantitative behaviour of the preference model and estimates the best preference values that can be achieved (see Chapter 5). The results of the experiments that were conducted after developing this model (see Chapter 8) are consistent with our mathematical formulation. We have submitted this model for publication in [DEEN03B], and our initial work was published in [DEEN02].

We have developed a multi-agent approach for solving distributed problems where cooperative autonomous agents work together to solve a joint task. Each autonomous agent (task agent), under the supervision of a relevant task coordinator, solves its part of the subtask in cooperation with other task agents. To resolve contention for the same resources, a market-based payment scheme has been applied that allows the preferences to be bought and sold by the contending task agents, through the medium of their coordinators. The best solution is achieved for a task when further iterations do not increase the task total preference value, at that point convergence is achieved.

Thus, the general preference model, which has been proposed in this thesis (see Chapter 4) includes a preference specification strategy, a preference processing technique, and a theoretical performance model. The preference model can produce a solution, regarded as the best solution, for trading and satisfying preferences and enforces timely termination in a "fairly competitive" market for cooperative agent-based systems. That solution is independent of the order of task agent requests for the resources. Section 9.1 of this chapter summarises the achievements of this work. We present an evaluation for this work in section 9.2.

For the simulation study we have used the scenario of distributed scheduling in manufacturing, where agents, (coordinators), resolve a set of global tasks into subtasks that have precedent subtasks. The coordinators perform the allocation of subtasks to the target agents (assemblers) through cooperation and negotiation, in which preferred resources are exchanged with payments. Agent-based systems support distributed scheduling, contrary to traditional centralised scheduling for manufacturing systems. We based our simulation for the multi-agent system on Cooperating Knowledge-Based Systems (CKBS) (see Chapter 3). To simplify the implementation of the simulator we imposed some limitations, these limitations are discussed in section 9.3.

We have implemented the simulator using Java Development Kit (JDK1.2.2). Java is popular, widely used, well supported, available freely, and comes with well-defined interfaces to system functions, multithreading capabilities, communication protocols, graphical tools, and is the author's primary programming language. At the time of writing, different agent development platforms, such as JADE [JADE], FIPA-OS[FIPAOS], and JACK[JACK], have become available. We discuss the potential use of these development platforms in section 9.4.

Briefly, we can conclude that the results of the study show that our agent based strategy reached convergence on the final preference value for the whole system in a form that could be estimated using our mathematical formulation. We also emphasise the fact that this estimated value is also independent of the initial order of subtask allocation. Although we used scheduling in distributed manufacturing systems to illustrate this, the potential application areas for our approach are diverse and are not restricted solely to scheduling. We discuss these application areas in section 9.4.

9.1 Achievements

In general, the work described in this thesis contributes to the field of distributed problem solving and multi-agent systems through the introduction of a preference model that can derive a solution that satisfies as many preferences as possible. The agent-based preference model presented in this thesis has been developed after several years of work in agent-based systems and in the HMS project. The specific contributions of this research are as follows:

- A preference model that can be used by cooperative autonomous agents that are working together to solve a joint task.
- A quantitative mathematical formula that can describe the quantitative behaviour of the preference model and estimate the boundaries of the solution, i.e. the best preference values that can be achieved form an estimate of the minimum remaining preference value that can be achieved for a given problem.
- Using a cost-based negotiation approach, the preference model presented provides techniques to specify user preferences, and an algorithm that ensures that the system can derive a preference-based solution in the presence of contention on resources. In that event, the best solution that can be achieved is the one that meets as many preferences as possible.
- The effects of uniform and non-uniform clustering of requests for the same resource instances were studied and the effects of the non-uniform distribution of these clusters (skewed distribution) on the formula were shown.

9.2 Evaluation

We would have liked to evaluate our preference model by comparing the results obtained by our implementation with those of other manufacturing scheduling systems. Unfortunately, this has proved to be infeasible as most of the current manufacturing scheduling systems concentrate mainly on resolving constraint conflicts and give only little consideration to solving preferential conflicts. Thus, to evaluate our work, we have expressed the objectives of this work (see section 1.3) in the form of a number of questions and our evaluation of the model is expressed in the form of answers to these questions.

In view of the presence of severe non-linearity we were mainly concerned with the following questions:

- *Has the proposed model been successfully implemented?*

  We have run several experiments to test that the simulator behaves according to the model rules. The results, presented in chapters 7 and 8, demonstrated that the preference model simulator implements the proposed model successfully.

- *Can the technique provide convergence to a global preference value?*

  The experiments demonstrated the effectiveness of the preference model and showed that convergence can be achieved using our preference model.

- *Does the initial order of allocation affect the final result?*

  When using the agent based preference-processing approach presented in this thesis the initial order of allocation has no effect on the final result.

- *Is a simplified theoretical performance model feasible?*

A theoretical performance model that models the behaviour of the system was developed. The simulation results verify the validity of this theoretical performance model.

- *Do the global values achieved in practice agree with the values calculated by the theoretical model?*

  The values calculated by the theoretical performance model are a good estimate of the best preference values that can be achieved by a given set of tasks and resources.

As can be seen from the answers given to the questions above, all of the objectives set forth for this thesis have been met by this work.

9.3 Current Limitations

The present preference model and the simulator have some present limitations, as noted below:

- Although the preference model, presented in this thesis, produces a solution that satisfies as many preferences as possible, it does not guarantee an optimal solution. To consider all alternative solutions is infeasible in a large set of subtasks, as the number of possible solutions grows exponentially with the number of subtasks.
- We assumed the resources types to be orthogonal so that a preference on one type does not affect that on another resource type. This assumption of orthogonality allowed us to restrict ourselves to considering only a single resource for the purpose of the preference model formulation and implementation. In some cases there could be resource type dependency. For example if a task has a preference

on a machine and also on end time, then the end time will be dependent on a machine. In this case we can linearise it by pairing the machine with their available time slot as the instances of a single resource type. In cases where linearisation is not possible, the evaluation process will have to consider all combinations, as discussed in section 4.2. Thus resource type dependency will involve extra processing and this should not affect our algorithm.

- In the experiments conducted we only dealt with a single preference value. Our intention was to make the implementation as simple as possible and to concentrate on the fundamental characteristics of the preference model. As observed above, simulating the effect of multiple preference values would only affect the execution time, as there will be more solutions to search for. Therefore, the issue of dealing with multiple preference values has been left for further research. For this purpose we designed the simulator in a way that it can be adapted to deal with multiple preferences by including dummy routines that deal with the issue of using multiple preferences.
- At present the simulator works in a batch mode, where tasks are specified in external files before running the system. The results are also stored in external files before they are migrated to other software for processing and graph drawing. We would like to extend the GUI part of the simulator so the specification of task parameters and result analysis are all integrated within the same package.

9.4 Value to Industry and Further Work

We see the potential of applying the preference model in many application areas, not only scheduling, but also in distributed processing where initial plans need to be merged into a global plan, such as for distributed project management and concurrent engineering. This research could have impact on the industry by:

- Improving schedules.
- Enhancing resource utilisation.
- Supporting cooperative relationships among the different project partners.

This research provides a foundation to develop a distributed scheduling algorithm that can help to produce a schedule that will satisfy as many participants' preferences as possible. It can even predict the boundaries of the solution.

We are however, aware that industrial applications are a lot more complex, and therefore to verify the real usefulness of the model to industries, an industrial trial will be necessary, an eventuality we would welcome.

For future work we hope to extend the implementation to include multiple preferences so that we can investigate further how this affects the model. We hope to make use of the agent development tools available today, such as JADE (Java Agent Development Environment) that is regarded as a robust and efficient environment for distributed multi-agent systems. Such tools were not available during the development stage, we think such tools can have a positive impact on the development time.

We would also like to investigate how the model behaves in practical situations. We hope to implement the model in such real-life applications as distributed project management, university timetabling, and meeting scheduling, and to compare our results with the results produced by current software packages designed specifically for use in such application domains.

Our approach can be used in applications where there is a need to combine different schedules, subject to dependencies and shared resources with preferences on such resources, into one single schedule. This problem is an example of a resource allocation problem, as there is a need to allocate the shared resources effectively among involved tasks, information about such resources and dependencies is distributed among tasks. Using a central resource controller has serious drawbacks, including lack of robustness; if this controller fails the whole system fails, and its computation and communication demands on a single bottleneck process. Our approach can overcome such limitations, with decision making being distributed among the processes controlling the separate resources. Resource allocation is performed by agents which have the ability to make decisions regarding the allocation of resources. Such agents need to cooperate together to find an effective common schedule to accomplish the task set by the project.

References

[AARIA] URL: http://www.aaria.uc.edu/

[AKKI98] R. Akkiraju, R. Goodwin, P. Keskinocak, S. Murthy, F. Wu: "A New Decision Support System for Paper Manufacturing", Proceedings of the 6th International Workshop on Project Management and Scheduling, Istanbul, Turkey, 1998.

[ALBA99] S. Albayrak and D.Wieczorek: "JIAC - a Toolkit for Telecommunication Applications", proceedings of the third International Workshop on Intelligent Agents for Telecommunication Applications (IATA'99), Stockholm, Sweden, Springer-Verlag, Berlin, Germany, pp.1-18, 1999.

[AROR98] S. Arora, "The Approximability of NP-hard Problems", this survey was a plenary lecture at ACM Symposium on Theory of Computing, 1998.

[BAKE98] A. D. Baker: "A Survey of Factory Control Algorithms that can be Implemented in a Multi-Agent Heterarchy: Dispatching, Scheduling, and Pull", Journal of Manufacturing Systems, Volume 17, No. 4, pp. 297-320, 1998.

[BAKE99] A.D. Baker, H. Van Dyke Parunak, K. Erol: "Agents and the Internet: Infrastructure: Mass customisation", in IEEE Internet Computing, Volume 3, Issue 5, Sept.-Oct. pp. 62-69,1999.

[BargainFinder] URL: http://bf.cstar.ac.com

[BASU98] C. Basu, H. Hirsh, W. Cohen: "Recommendation as Classification: Using Social and Content-Based Information in Recommendation", Proceedings of the fifteenth National Conference on Artificial Intelligence, pp.714-720, 1998.

[BECK94] JC. Beck and MS. Fox: "Supply Chain Coordination via Mediated Constraint Relaxation", Proceedings of the First Canadian Workshop on Distributed Artificial Intelligence, Banff, AB, 1994.

[BECK94A] JC. Beck: "A schema for constraint relaxation with instantiations for partial constraint satisfaction and schedule optimisation", Computer Science, University of Toronto, MSc Thesis, 1994.

[BOGA94] N. R. Bogan: " Economic Allocation of Computation Time with Computational Markets", Master's thesis, Department of Electrical Engineering and Computer Science, MIT, 1994.

[BOOC94] G. Booch: "Object-Oriented Analysis and Design", published by Benjamin Cummings, 1994.

[BONG98] L. Bongaerts: "Integration of Scheduling and Control in Holonic Manufacturing Systems", Ph.D. thesis KU Leuven, 1998.

[BRAT88] M. E. Bratman, et al: "Plans and Resource Bounded Practical Reasoning", Computational Intelligence, Vol. (4:4), pp349-355, Nov 1988.

[BREN02] R.W. Brennan, X. Zhang, Y. Xu, and D.H. Norrie: "A reconfigurable concurrent function block model and its implementation in real-time Java", Integrated Computer-Aided Engineering, 9(3), pp. 263-279, 2002.

[BROW95] D.E Brown et.al.: "A survey of intelligent scheduling systems", Intelligent Scheduling Systems edited by D.E. Brown and W.T. Scherer, 1995.

[BRUS98] H.V Brussel, J. Wyns, P. Valckenaers, L. Bongaerts, and P. Peeters: "Reference architecture for holonic manufacturing systems: PROSA", Computer in Industrial, Vol. 37, pp.255-274, 1998.

[BURK01] K. Burke and K. Aytes: "Preference for Procedural Ordering in Distributed Groups: How Do Media and Repeated Interaction Affect Perceptions and Procedural Structuring?", proceedings of the thirty-fourth Hawaii International Conference on System Sciences (HICSS-34), IEEE Computer Press, 2001.

[CHEN98] J.Q. CHENG and M.P. WELLMAN: "The WALRAS Algorithm: A Convergent Distributed Implementation of General Equilibrium Outcomes ", Computational Economics 12, pp 1–24, 1998.

[CHRI94] J.H. Christensen, "Holonic Manufacturing Systems: Initial Architecture And Standards Directions", presented at first European Conference on Holonic Manufacturing Systems, Hannover, Germany, 1994.

[CHRI98] J.Christensen: "Holonic Manufacturing Systems", technical report, ftp://hms@ifwpd7.ifw.uni-hannover.de/hms-wps/wp1-seng/html/t1/d11f.htm, Rockwell Automation, 1998.

[CONI03] V. Conitzer and T. Sandholm: "Universal Voting Protocol Tweaks to Make Manipulation Hard", In Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03), Acapulco, Mexico, 2003.

[DAM94] M. Dam, M. Zachariasen: "Tabu Search on the Geometric Travelling Salesman Problem", Thesis for a MSc. degree at the department of computer science at the University of Copenhagen (DIKU), 1994.

[DARR94] Timothy P. Darr and William P. Birmingham: "An Attribute-Space Representation and Algorithm for Concurrent Engineering", The University of Michigan, Department of Electrical Engineering and Computer Science CSE-TR-221-94, October 1994.

[DARR94A] T.P. Darr and W.P. Birmingham: "An Attribute-Space Representation and Algorithm for Concurrent Engineering", Artificial Intelligence for Engineering Design, Analysis, and Manufacturing, AI EDAM 10(1) 1996.

[DAST01] M. Dastani, N. Jacobs, C.M. Jonker, and J. Treur: "Modelling user preferences and mediating agents in electronic commerce", Lecture Notes in Computer Science, also available at: www. citeeer.nj.nec.com/dastani99modelling.html, (2001).

[DEDEMAS] URL: http://dedemas.ifw.uni-hannover.de.

[DEEN96] S. M. Deen: "An architectural Framework for CKBS Applications", IEEE Transactions on Knowledge and Data Engineering, Vol. (8:4), pp 663-671, 1996.

[DEEN97] S. M. Deen: "A Database Perspective to a Cooperation Environment", CIA'97 Proceedings, edited by P. Kanzia and M. Klusch, Springer, pp 19-41, 1997.

[DEEN97A] S. M. Deen, R. Jayousi, B. Ndovie and B. Taha: "An agent-Based Approach to Dynamic Scheduling in Traffic Control", technical report, Department of Computer Science, University of Keele, England, 1997.

[DEEN98] S. M. Deen: "A fault tolerant cooperative distributed system", IEEE DEXA Work-shop, R Wagner, ed., Vienna, 1998.

[DEEN99] S.M. Deen, A computational model for a cooperating agent system, technical report, DAKE Group, Computer Science Department, Keele University, 1999.

[DEEN99A] S.M. Deen and C.A. Johnson: "Towards a Theoretical Foundation for Cooperating Knowledge Based Systems", proceedings of the 11th International Symposium, Methodologies for Intelligent Systems, 1999.

[DEEN00] S.M. Deen :"An investigation into a Computational Model for Holonic Manufacturing Systems", technical report, DAKE Group, Computer Science Department, Keele University, 2000.

[DEEN02] S.M. Deen and R Jayousi: "Preference Based Task Allocation in Holonic Manufacturing", the 13th International Conference on Database and Expert Systems Applications (DEXA'02), published by the IEEE Computer Society pp 573-577,2002.

[DEEN03] S.M. Deen and C.A. Johnson: "Formalizing an Engineering Approach to Cooperating Knowledge Based Systems", IEEE Transactions on Knowledge and Data Engineering, vol. 15, NO. 1, pp103-117, 2003.

[DEEN03A] "Agent Based Manufacturing - Advances in the Holonic Approach", edited by S.M. Deen, Springer-Verlag (Heidelberg, Germany), 2003.

[DEEN03B] S.M. Deen and R Jayousi: "A Preference Processing Model For Cooperative Agents", submitted for publication to JIIS, 2003.

[DOYL94] J. Doyle: “A Reasoning Economy for Planning and Replanning”, In Technical Papers of the ARPA Planning Initiative Workshop, 1994.

[EPHR94] E. Ephrati, G. Zlotkin , J. S. Rosenschein, "Meet your destiny: a non-manipulable meeting scheduler", Proceedings of the 1994 ACM conference on Computer supported cooperative work, p.359-371, October 22-26, 1994.

[EPHR96] E. Ephrati, and J.S. Rosenschein: "Deriving consensus in multi-agent systems", Artificial Intelligence, Elsevier Science BV, Amsterdam, vol. 87 (1-2), pp. 21-74, 1996.

[FARL98] J. Farley: "Java Distributed Computing", O'Reilly & Associates, ISBN: 1-56592-206-9, 1998.

[FIEL00] A. Field, P. Hartel, and W. Mooij:."Personal DJ: An architecture for personalised content delivery", in Proceedings of the tenth international conference on World Wide Web, pages 1–7, also available from: http://www10.org/cdrom/papers/384/, 2001.

[FINI93] T. Finin and R. Fritszon: "KQML - A language for protocol and information exchange", Proceedings of 13th international conference on Distributed Artificial Intelligence, edited by M. Singh and published by Maryland University, USA, 1993.

[FIPA] FIPA: Foundation of Intelligent Physical Agents, http://www.fipa.org.

[FIPAOS] http://www.nortelnetworks.com/products/announcements/fipa.

[FISH94] M.Fisher: "Specifying and Executing Protocols for Cooperative Action", Proc. Second Int'l Working conf. Cooperating Knowledge-Based Systems, pp.295-306, 1994.

[FLAN99] D. Flanagan: "Java in a Nutshell", 3rd. Edition, O'Reilly & Associates, ISBN: 1-56592-487-8, 1999.

[FLET97] M. Fletcher: " An Agent-based approach to dynamic network management" PhD thesis, Department of Computer Science, University of Keele 1997.

[FLET98] M Fletcher: A Critique of Holonic Manufacturing Systems: Architectural requirements and standards, technical report, DAKE Group, Computer Science Department, Keele University, 1998.

[FLET98A] M Fletcher, ed.: "Holonic Systems Architecture, technical report, DAKE Group, Computer Science Department, Keele University, 1998.

[FLET00] M. Fletcher, S.M. Deen, and P. G: "An Evaluation of Rescheduling Techniques and Architectures in Holonic Manufacturing Systems", Technical

Report, Department of Computer Science, University of Keele, England, May 2000.

[GARR96] L. Garrido and K. Sycara: "Multi-agent meeting scheduling: preliminary results", 1996 International Conference on Multi-Agent Systems (ICMAS '96)", pp. 95 - 102, 1996.

[GENE92] M.R. Genesereth and R.E. Fikes: "Knowledge Interchange Format Reference Manual", Computer Science Department, Stanford University, USA, June 1992.

[GEYI99] F. Geyik and I.H. Cedimoglu: “A Review of the Production Scheduling Approaches Based-on Artificial Intelligence and the Integration of Process Planning and Scheduling”, Proceedings on Swiss Conference of CAD/CAM'99, A. Belhi, P.J. Erard and A. Bouras (Ed.), Neuchatel University, Switzerland, pp.167-174, 1999.

[GOLD94] D. E. Goldberg: "Genetic and Evolutionary Algorithms Come of Age", Communications of the ACM, Vol. 37, No. 3, pp. 113-119, March 1994.

[HAMA98] M. Hamad: "Design and Implementation of a Cordinator Agent for Cooperative Task Processing", Mphil Thesis , Department of Computer Science, University of Keele, England, 1998.

[HAYN97] T. Haynes, S. Sen, N. Arora, and R. Nadella: "An Automated Meeting Scheduling System that Utilizes User Preferences", in Proceedings of The First International Conference on Autonomous Agents, February, 1997.

[HEIK01] T. Heikkila, M. Kollingbaum, P. Vackenaers, G.-J. Bluemink, "An agent architecture fo manufacturing control: manAge", Computers in Industry 46, pp 315-331, 2001.

[HERT95] A. Hertz, E. Taillard, D. de Werra, "A Tutorial on Tabu Search", Proc. of Giornate di Lavoro AIRO'95, (Entreprise Systems: Management of Technological and Organizational Changes), 13-24, 1995

[IBM] http://www.research.ibm.com/pdtr/paper.html

[IGLE 95] C.A Iglesias, J.C Gonzälez, and J.R Velasco: "MIX:A general purposemultiagent architecture" Proceedings of the IJCAI'95 Workshop on Agent Theories, Architectures and Languages,Montréal,Canada,ACM Press, pp. 216-224, 1995.

[JACK] http://www.agent-software.com/shared/demosNdocs/JACK_Manual_WEB.

[JADE] http://sharon.cselt.it/projects/jade.

[JOHN89] D. S. Johnson, C. R. Aragon, L. A. McGeoch, et al: "Optimization By Simulated Annealing: An Experimental Evaluation, Part I (Graph Partitioning)", Operations Research, Vol. 37, No. 6, pp. 865-892, 1989.

[JOO00] K.H. Joo, T. Kinoshita and N. Shiratori: "Agent-based Grocery Shopping System Based on User's Preference", Proceedings of the Seventh International Conference on Parallel and Distributed Systems (ICPADS'00 Workshop: Flexible Networking and Cooperative Distributed Agents, pp.499-505, IEEE, 2000.

[KEEN93] R. L. Keeney and H. Raiffa: "Decisions with Multiple Objectives: Preferences & Value Tradeoffs", Cambridge University Press, ISBN: 0-521-43883-7, 1993.

[KEIN97] T. Keinonen: "Expected Usability and Product Preference", Proceedings of DIS'97 Conference Designing Interactive Systems, August,1997 Amsterdam. ACM. pp 197-204.

[KIM01] K. Kim: “Distributed Coordination of Project Schedule Changes: An Agent-Based Compensatory Negotiation Approach”, CIFE Technical Report #130, Stanford University, Dec., 2001.

[KOLLOO] M. Kollingbaum, T. Heikkila, P. Peeters, J. Matson, P. Valckenaers, D. McFarlane, G. J. Bluemink, "Emergent flow shop control based on MASCADA agents". In Proceedings of MIM 2000, July 2000.

[KOCJ00] W, Kocjan: "Dynamic scheduling. State of the art report", technical report T2002 -28, Computer Science Laboratory, Mälardalen University, available from: http://citeseer.nj.nec.com/kocjan02dynamic.html, 2002.

[LEIT01] P. Leitão, J. Barata, L.M. Camarinha-Matos and R. Boissier: "Trends in Agile and Co-operative Manufacturing", Proceedings of Low Cost Automation Symposium, Berlin, http://www.ipb.pt/~pleitao/papers/rm2001.pdf, 2001.

[LIU98] T. H. Liu, A. Goel, C. E. Martin, K. S. Barber: “Classification and Representation of Conflict in Multi-Agents Systems”, Technical Report TR98-UT-LIPS-AGENTS-01, The University of Texas at Austin, 1998.

[LOCHOO] Loch, C. H., and C. Terwiesch: "Product Development and Concurrent Engineering", in Swamidass, P. M. (ed.): Encyclopaedia of Production and Manufacturing Management, Dordrecht: Kluwer Academic Publishing 2000, 567 - 575. Reprinted in: Swamidass, P. M. (ed.): Innovations in Competitive Manufacturing, Dordrecht: Kluwer , pp. 263 - 274, 2000.

[LUX97] A. Lux and D. D. Steiner: "Understanding Cooperation—An Agent’s Perspectives", Readings in Agents, M. N. Huhns and M. P. Singh, eds., pp 471-480, Morgan Kaufmann, 1997.

[MACADA] http://www.mech.kuleuven.ac.be/pma/mascada/welcome.html.

[MART99] D.L Martin, S.J Cheyer, and D.B Moran: "Open Agent Architecture: A framework for building distributed software systems", Applied Artificial Intelligence, Taylor & Francis Ltd, London, 13(1-2), pp. 91-128, 1999.

[MATU96] F. Maturana and D.H. Norrie: "Multi-Agent Mediator Architecture for Distributed Manufacturing", Journal of Intelligent Manufacturing, Vol. 7, pp. 257-270, 1996.

[MCCA98] J. F. McCarthy and T. D. Anagnost: "MUSICFX: An Arbiter of Group Preferences for Computer Supported Collaborative Workouts", Proceedings of the ACM 1998 Conference on Computer Supported Cooperative Work (CSCW '98), pp. 363-372, 1998.

[METAMORPH2] http://imsg.enme.ucalgary.ca/research.htm#Metamorph2

[MovieFinder] URL:http://www.moviefinder.com

[MUDG00] Mudgal and J. Vassileva: "Bilateral negotiation with incomplete and uncertain information: a decision-theoretic approach using a model of the opponent", in Cooperative Information Agents IV: The Future of Information Agents in Cyberspace, eds. M. Klusch and L. Kerschberg, LNAI 1860, Springer, pp.105-118, http://citeseer.nj.nec.com/mudgal00bilateral.html, 2000.

[MUKH01] R. Mukherjee, G. Jónsdóttir, S. Sen, and P. Sarathi: "MOVIES2GO: an online voting based movie recommender system", Agents 2001, pp. 114-115, 2001.

[MUEL96] J.P.Mueller: "The Design of Intelligent Agents", LNAI volume 1177, Springer, 1996.

[MURT97] S. Murthy, J. Rachlin, R. Akkiraju, and F.Wu, "Agent-Based Cooperative Scheduling", Proceedings of AAAI Workshop on Constraint and Agents, 1997.

[NDOV94] Baird Ndovie: " Simulation of a conflict management system for Air Traffic Control", Proceedings of the second International Working Conference on CKBS, Keele university. Published by the DAKE centre, pp 235-254, 1999.

[NEUM44] J. Von Neuwn and O. Morgenstern: "Theory of Games and Economic Behavior", Princeton University Press, Princeton,N.J., 1944.

[NWAN99] H.S Nwana, D.T Ndumu, L.C Lee, and J.C Collis, "ZEUS: A toolkit forbuilding distributed multiagent systems", Applied Artificial Intelligence, Taylor & Francis, London, 13(1-2), pp. 129-85, 1999.

[ORFA97] R. Orfali and D. Harkey, "Client/server programming with Java and CORBA", John Wiley and Sons, ISBN: 0471-24578-X, 1997.

[PARK98] K. Park, M. Sitharam, and S. Chen, "Quality of Service Provision in Noncooperative Networks: Heterogenous Preferences, Multi-Dimensional QoS Vectors, and Burstiness", Proceedings of the ACM International Conference on Information and Computation Economies, pp. 111-127, 1998.

[PARU91] H. V. D. PARUNAK: "Characterising the Manufacturing Scheduling Problem", Journal of Manufacturing Systems, vol. 10, no. 3, pp. 241-258, 1991.

[PARU97] H. Van Dyke Parunak, A.D. Baker, and S.J. Clark, "The AARIA Agent Architecture: An Example of Requirements-Driven Agent-Based System Design", Agents '97 Marina del Rey, ACM, 1997.

[PENN00] David M Pennock, Eric Horvitz, and C. Lee Giles. Social choice theory and recommender systems: Analysis of the axiomatic foundations of collaborative filtering. AAAI, pp. 729–734, Austin, TX, 2000.

[PROSA] http://www.mech.kuleuven.ac.be/pma/project/goa/prosa.htm

[RABE93] L. Rabelo, S. Toure, and T. Velasco: "Artificial Neural Networks for Flexible Manufacturing Systems Scheduling", Computers and Industrial Engineering, Vol. 25, No. 1-4, pp. 385-388, 1993.

[RABE94] R. Rabelo, and L.M.Camarinha-Matos: "Multi-Agent based Dynamic Scheduling", International Journal on Robotics and Computer Integrated Manufacturing, Vol. II, No.4, pp 303-310, 1994.

[RABE99] R. J. Rabelo, L.M. Camarinha-Matos, H. Afsarmanesh, "Multi-agent-based agile scheduling". Robotics and Autonomous Systems 27, pp. 15-28, 1999.

[RAO91] A. S. Rao and M. P. Georgeff: "Modeling Agents within a BDI architecture", KR'91, edited by R. Fikes and E. Sandewall, pp. 473-484, published by Morgan Kaufmann, 19991.

[ROGE99] S. Rogers, C. Fiechter, and P. Langley: "An Adaptive Interactive Agent For Route Advice", Proceedings of the Third International Conference on Autonomous Agents, pp. 198-205, Seattle, ACM, 1999.

[SAND99] Tuomas W. Sandholm: "Distributed rational decision making", Multi-agent Systems.A Modern Approach to Distributed Artificial Intelligenc", Gerhard Weiss, editor, Chapter 7, pp. 201–258, MIT Press, 1999.

[SCHA95] A.Schaerf: "A Survey of Automated Timetabling", A Technical Report, National Research Institute for Mathematics and Computer Science, 1995.

[SEAR69] J. R. Searle: "Speech Act", Cambridge University Press, 1969.

[SEN97] S. Sen, T. Haynes and N. Arora: "Satisfying user preferences while negotiating meetings" International Journal of Human-Computer Studies, Academic Press, London, Vol. 47, 407-427, 1997.

[SEN98] S. Sen and E.H. Durfee: "A formal study of distributed meeting scheduling", Group Decision and Negotiation, Kluwer Academic Publishers, Dordrecht, the Netherlands, 7, pp. 265-289, 1998.

[SEN99] S. SEN and G. Weiss: "Learning in Multiagent Systems", Multi-agent Systems, A modern Approach to DAI, edited by G. Weiss, MIT Press, ISBN: 0-262-23203-0, pp 259-330, 1999.

[SEO00] Y. W. Seo and B. T. Zhang: "A Reinforcement Learning Agent for Personalized Information Filtering", Proceedings of the International Conference on Intelligent User Interface, 2000 (IUI-2000), pp. 248-251.

[SHEN99] W. Shen, D.H. Norrie, "Agent-Based Systems for Intelligent Manufacturing: A State-of-the-Art Survey", An extended HTML version of the paper published in Knowledge and Information Systems (KAIS), vol. 1, no. 2, pp. 129-156, available from http://imsg.enme.ucalgary.ca/publication/abm.htm, 1999.

[SHEN02] W.Shen, Distributed Manufacturing Scheduling Using Intelligent Agents, IEEE Intelligent Systems, pp. 88-94, Jan./Feb 2002.

[SHIN00] T. Shintani, T. Ito, and K Sycara: "Multiple Negotiations among Agents for a Distributed Meeting Scheduler", Proceedings of the Fourth International Conference on Multi Agent Systems (ICMAS'2000), poster.

[SMIT80] R.G. Smith: "The contract net protocol: High-level communication and control in a distributed problem solver", IEEE transactions on computers, vol. 29, no. 12, pp. 1104-113, 1980.

[SOUS99] P. Sousa, C. Ramos, "A distributed architecture and negotiation protocol for scheduling in manufacturing systems", Computers in Industry, vol. 38, no. 2, pp. 103-113, 1999.

[SOUSA99] P. Sousa, C. Ramos, and J. Neves: "Contracting Tasks between Autonomous Resources - an Application to Scheduling of Manufacturing Orders, the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM-99), London, UK, 1999.

[SYCA89] K. Sycara: "Multi-Agent Compromise via Negotiation", Distributed Artificial Intelligence (Vol. 2), Gasser, L. and Huhns, M., ed., Morgan Kaufmann, Los Altos, CA, 1989.

[SYCA89A] Sycara, K. "Argumentation: Planning Other Agents' Plans", Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89), Detroit, Mich., 1989.

[SYCA90] Sycara, K. "Persuasive Argumentation in Negotiation", Theory and Decision, vol. 28, no. 3, pp. 203-242, 1990.

[SYCA91] K. Sycara, S. Roth, N. Sadeh, and M.S Fox: "Distributed Constrained Heuristic Search", IEEE Transactions on Systems, Man and Cybernetics, vol. 21, no. 6, pp. 1446-1461, 1991.

[TOENOO] H.K. Tonshoff, I. Seilonen, G. Teunis, and P. Leitao, "A Mediator-Based Approach for Decentralised Production Planning, Scheduling, and Monitoring", ICME, 2000.

[UMBC] http://agents.umbc.edu/Applications_and Software/Software/.

[UMBCA] http://agents.umbc.edu/Applications_and Software/Software/Academic/.

[UMBCB] http://agents.umbc.edu/Applications_and_Software/Software/Commercial/index

[WAL1874] L.Walras: “Elements of Pure Economics”, 1874. English translation by W. Jaffe, published by Allen and Unwin, 1954.

[WANG01] J. Wang: “Ranking Engineering Design Concepts Using a Fuzzy Outranking Preference Model”, Fuzzy Sets and Systems 119, pp 161-170, 2001.

[WATS94] L. Watson, I, and F. Marir: "Case-Based Reasoning: A Review", The Knowledge Engineering Review, vol. 9, no. 4, pp. 355-381, 1994.

[WELL93] M.P. Wellman "A Market-Oriented Programming Environment and its Application to Distributed Multicommodity Flow Problems", Journal of Artificial Intelligence Reasearch, Morgan Kaufmann publishers, San Francisco, Vol. 1, pp1-23, 1993.

[WELL96] M.P. Wellman: "Market-oriented programming: Some early lessons", In S. Market-Based Control: A Paradigm for Distributed Resource Allocation. Clearwater (ed.), World Scientific, 1996.

[WERK92] K.J. Werkman: "Multiple Agent Cooperative Design Evaluation Using Negotiation", Artificial Intelligence in Design", Kluwer Academic Publishers, 1992.

[WONG94] S. T. C. WONG: "Preference-Based Decision Making for Cooperative Knowledge-Based Systems", ACM Transactions on Information Systems, Vol. 12, No. 4, pp 407-435, 1994.

[WOOL99] M. Wooldridge: "Intelligent Agents", Multi-agent Systems, A modern Approach to DAI, edited by G. Weiss, MIT Press, ISBN: 0-262-23203-0, pp 27-77, 1999.

[WOOL01] M.J. Wooldridge: "An Introduction to Multiagent Systems", Willey, ISBN 047149691X, 2001.

[WREN99] A. Wren and R. Kwan: "Installing an urban transport scheduling system", Journal of Scheduling, Vol. 2, pp 3-17,1999.

[WYNS99] J. Wyns: "Reference Architecture For Holonic Manufacturing Systems the key to support evolution and reconfiguration", Ph.D. Thesis, K.U.Leuven, ISBN 90-5682-164-4, 1999.

Appendix A

In this appendix we present the results of the experiments that are similar to the one discussed in section 7.2.1. This group of experiments investigates the convergence of the gain in preference values and cost when only one task, including its subtasks, is reallocated without considering the preferences of the reallocated tasks. Also, we reverse the order of the initial allocation to investigate whether such variations affect the gain in preference values and cost.

In these experiments all tasks are initially allocated on the first available slot, tasks are allocated according to the different order without considering preferences. Then one task is subsequently reallocated to satisfy its preferences. As a result of this reallocation some tasks can be reallocated. These tasks are reallocated to the first available slot without taking their preferences into account.

Figures in this appendix show how the total cost (C^i) and gain in preference values (G^i) of T_i vary during the reallocation process. Iteration points are indicated by square symbols for the gain in preference value, and by diamond symbols for the total cost. With respect to the G^1 graph, the Y-axis values indicate the percentage preference gain while in the case of the (C^1) graph, they indicate the total cost units.

Figure A.1 shows the result of conducting this experiment for task T_1 when tasks are initially allocated according to the order T_1, T_2 ... T_6. Figure A.2 shows the result of this experiment for task T_1 when tasks are initially allocated according to the order T_6, T_5 ... T_1.

The G^1 graph shows that during the initial allocation (iteration 0) around 80% of the preference value is satisfied when tasks are initially allocated according to the order T_1, T_2 ... T_6, while 0% of the preference value is satisfied when tasks are initially allocated according to the order T_6, T_5 ... T_1. This is expected, as T_1 was last to be allocated and has initially less chance of its preferences to be satisfied in the second case. A preference gain, around 95%, occurred at iteration 1 in both cases and remained constant throughout the subsequent iterations.

The C^1 graph shows an increase in cost at iterations 1, 120 cost units in the first case and 180 cost units in the second case. We think the difference between these cost values (60 cost units) is due to the fact that negotiating tasks are not the same and therefore the exchanged preferences can be different. This behaviour is similar to the experiment discussed in section 7.2.1.

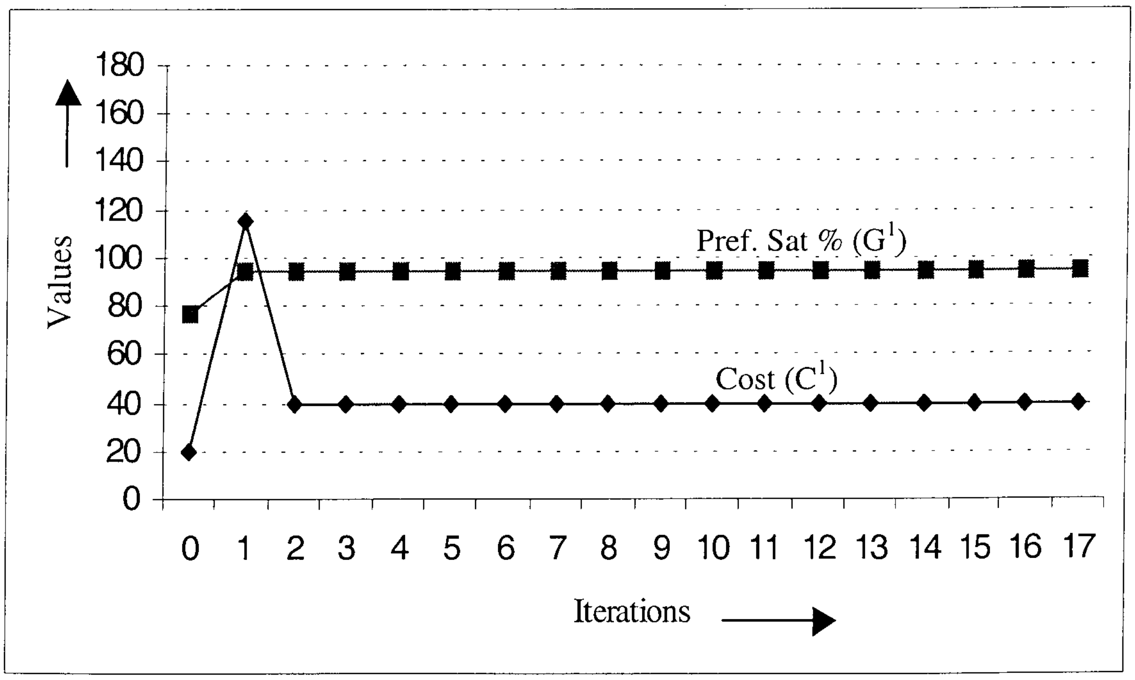


Fig. A.1 T^1 preference satisfactions & cost variation (without considering other tasks preferences and initial allocation order 1,2,3,4,5,6)

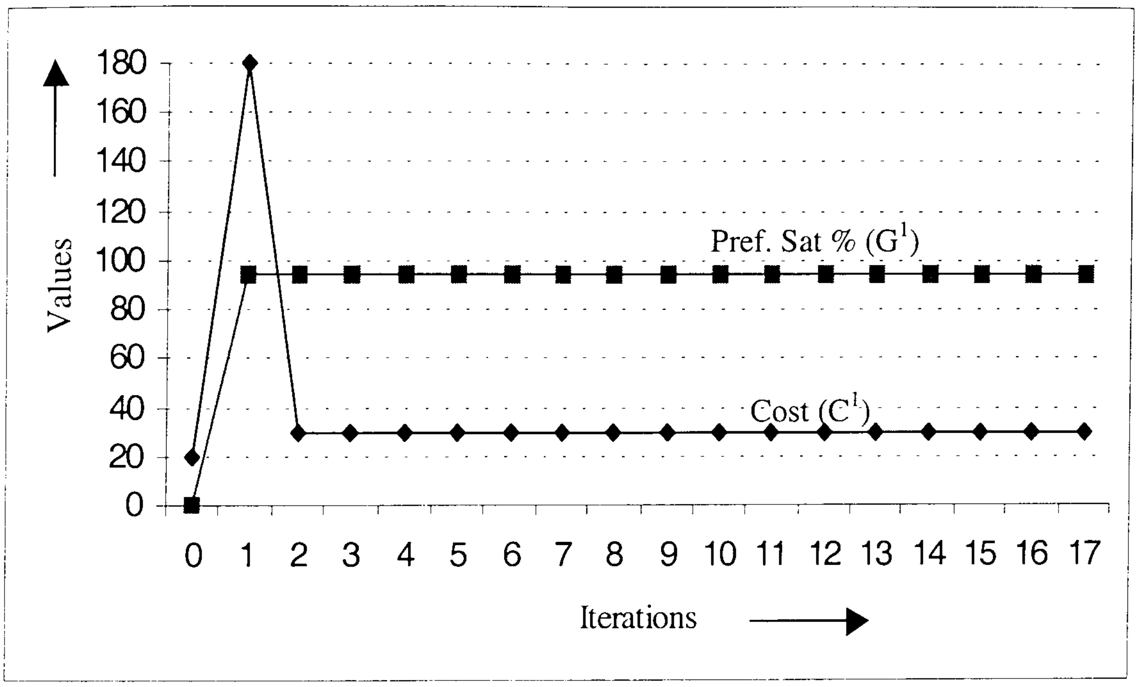


Fig. A.2 T^1 preference satisfactions & cost variation (without considering other tasks preferences and initial reverse allocation order 6,5,4,3,2,1)

The remaining figures in this appendix show the results of the above experiment when repeated for tasks order T_2, T_4, T_5, and T_6. Figures A.3 and A.4 show the results of conducting the experiment for task T_2. Figures A.5 and A.6 show the results of conducting the experiment for task T_4. Figures A.7 and A.8 show the results of conducting the above experiment for task T_5. Figures A.9 and A.10 show the results of conducting the above experiment for task T_6. The results of all these experiments are similar and confirm to the behaviour of the preference model

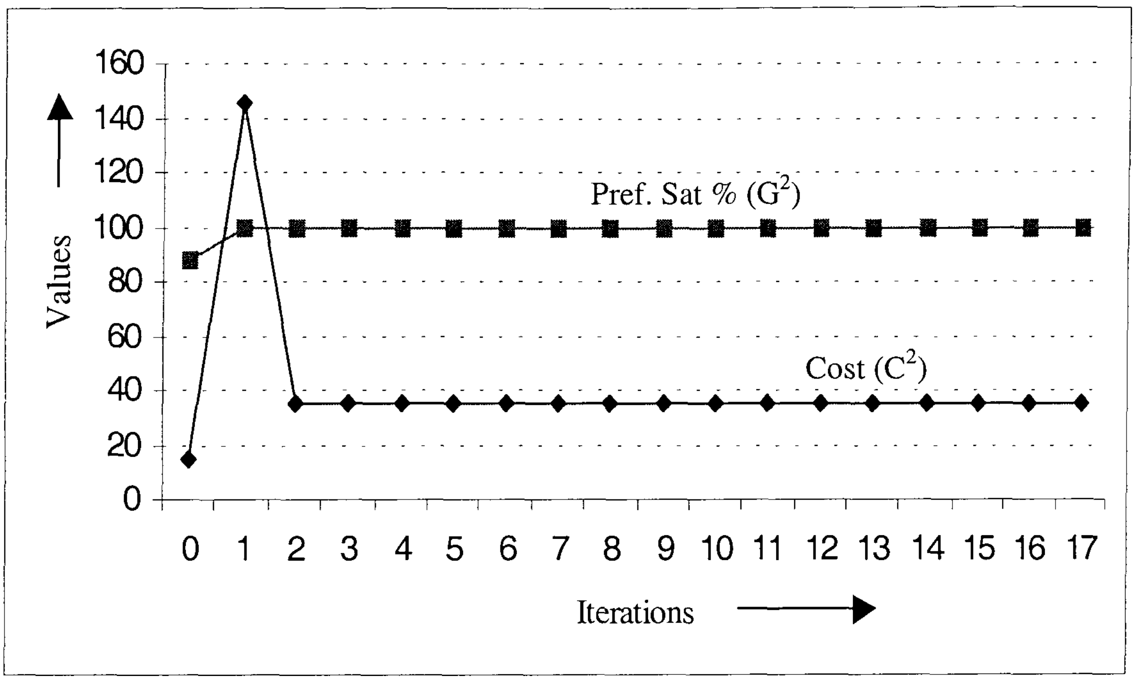


Fig. A.3 T^2 preference satisfactions & cost variation (without considering other tasks preferences and initial allocation order 1,2,3,4,5,6)

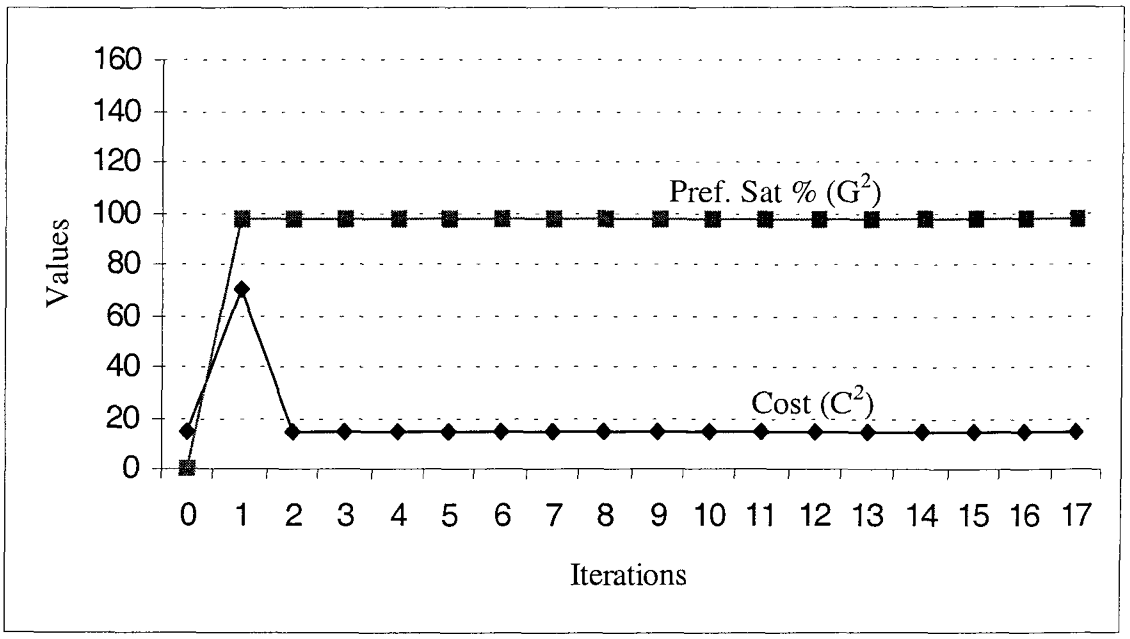


Fig. A.4 T^2 preference satisfactions & cost variation (without considering other tasks preferences and initial reverse allocation order 6,5,4,3,2,1)

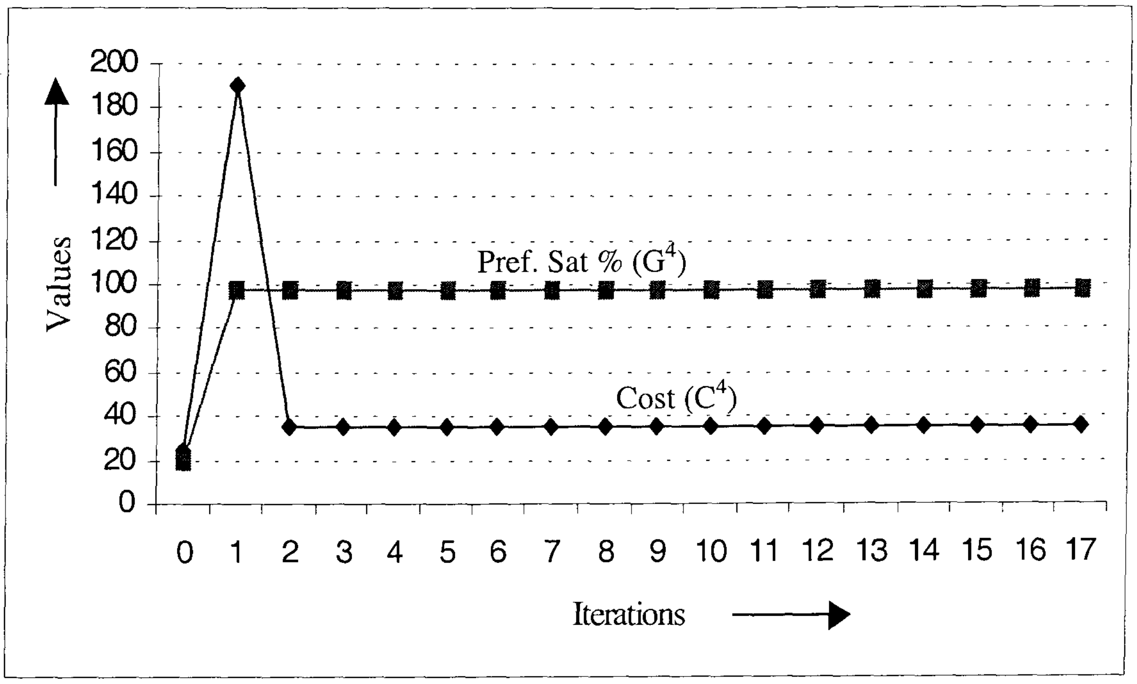


Fig. A.5 T^4 preference satisfactions & cost variation (without considering other tasks preferences and initial allocation order 1,2,3,4,5,6)

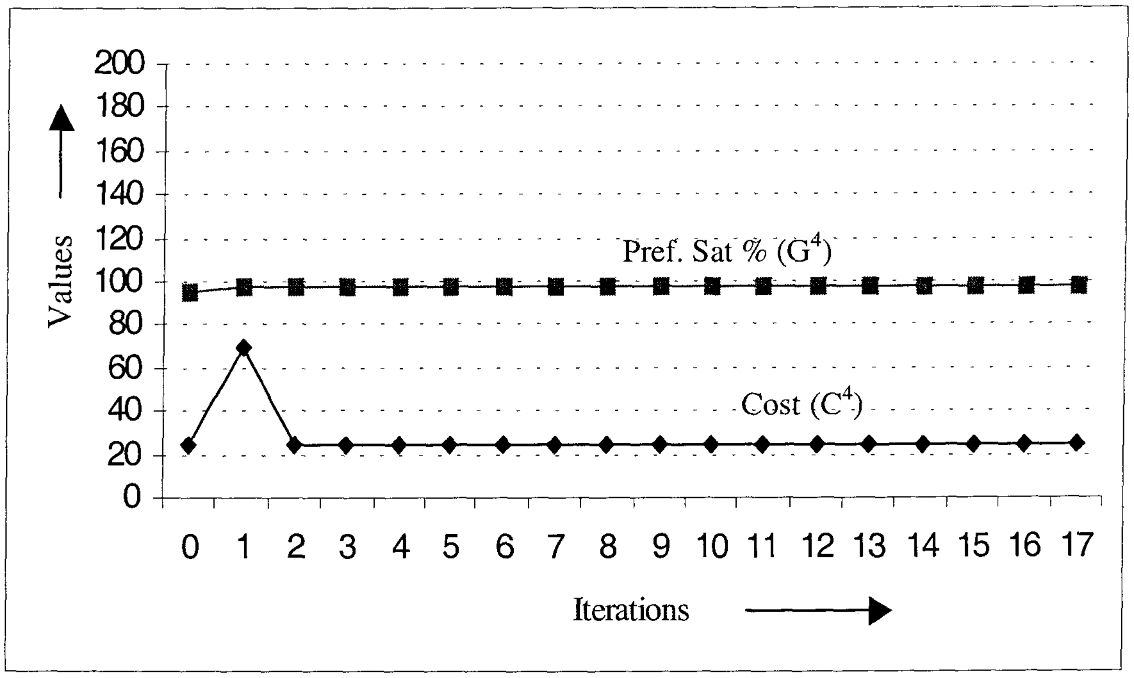


Fig. A.6 T^4 preference satisfactions & cost variation (without considering other tasks preferences and initial allocation order 6,5,4,3,2,1)

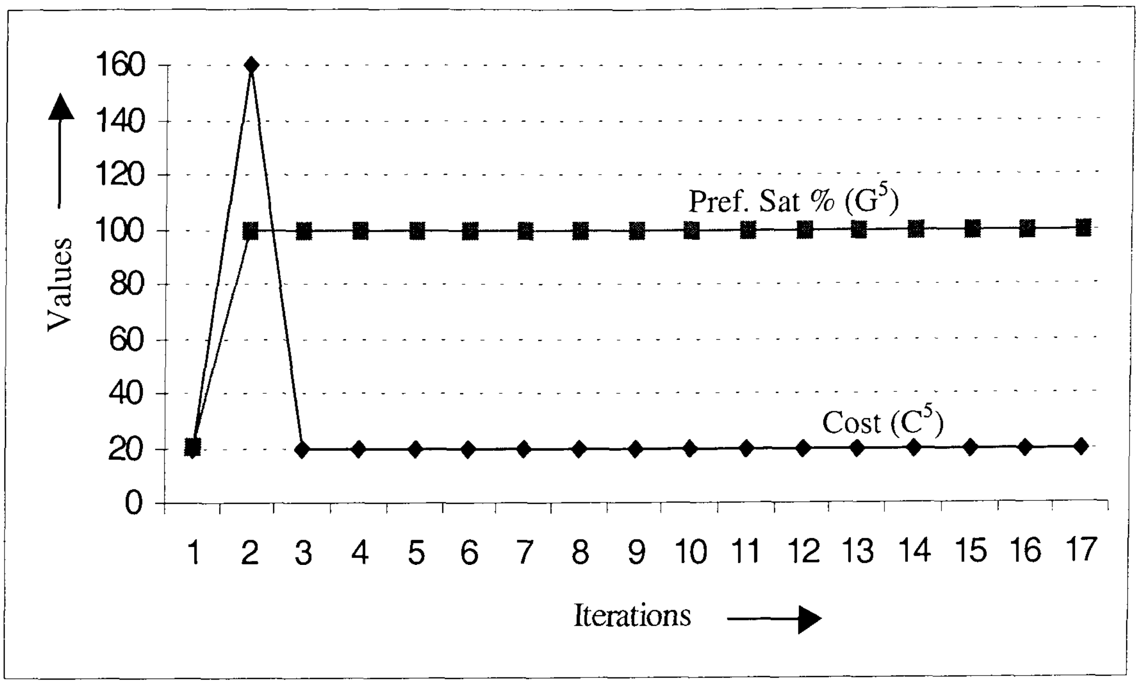


Fig. A.7 T^5 preference satisfactions & cost variation (without considering other tasks preferences and initial allocation order 1,2,3,4,5,6)

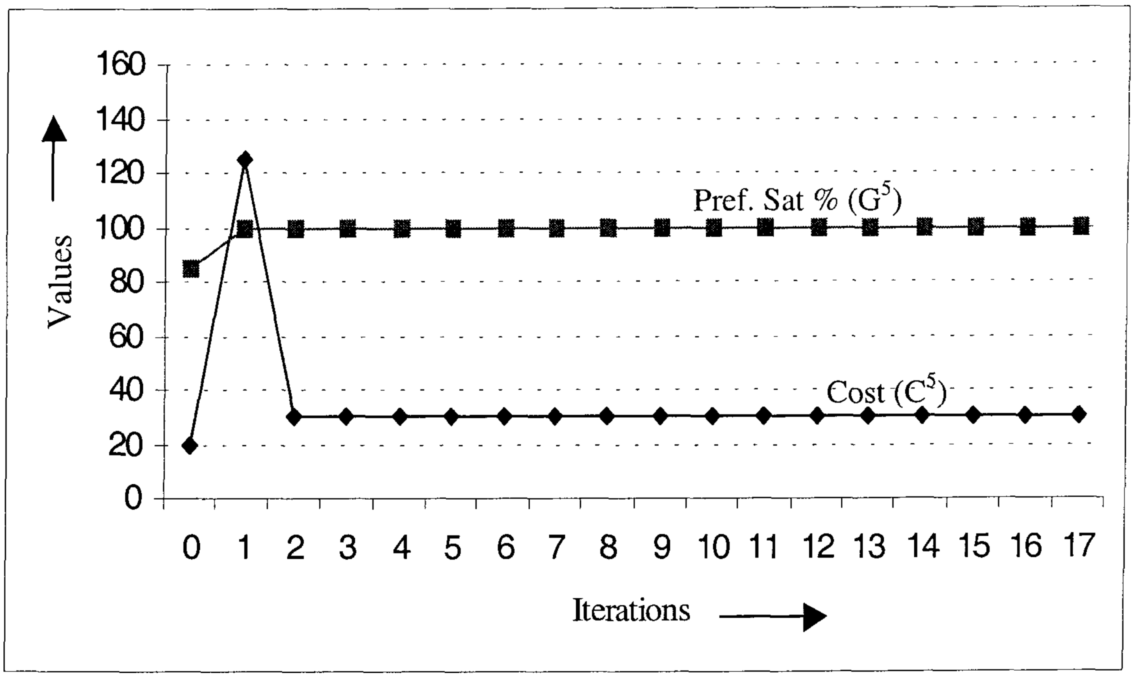


Fig. A.8 T^5 preference satisfactions & cost variation (without considering other tasks preferences and initial allocation order 6,5,4,3,2,1)

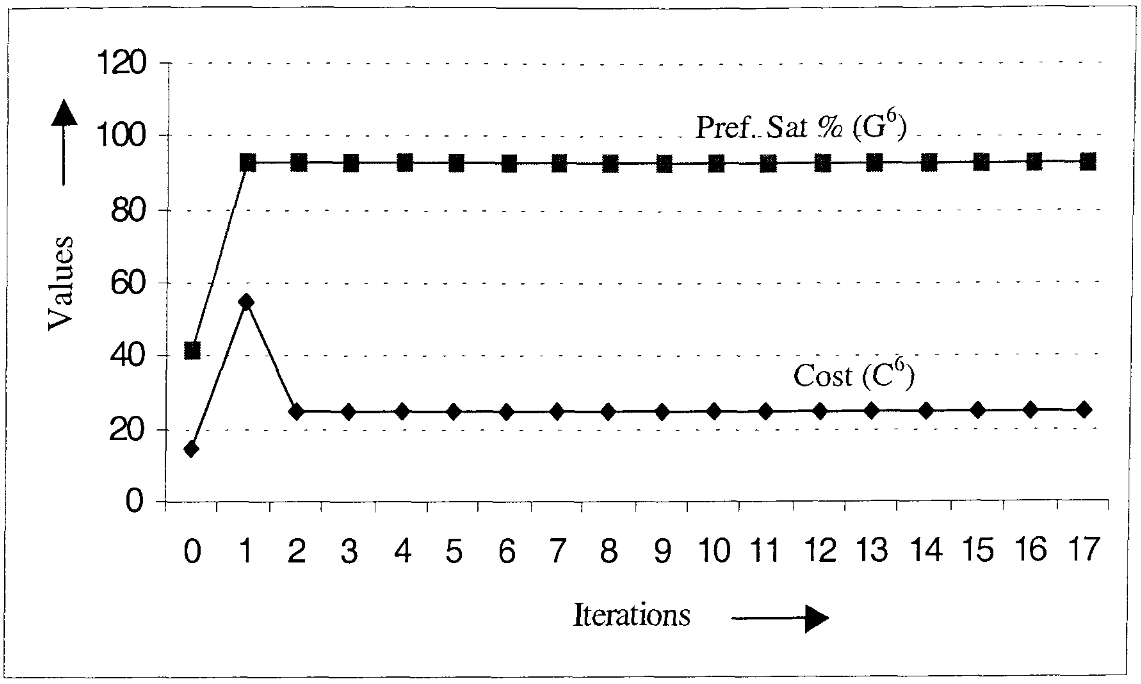


Fig. A.9 T^6 preference satisfactions & cost variation (without considering other tasks preferences and initial allocation order 1,2,3,4,5,6)

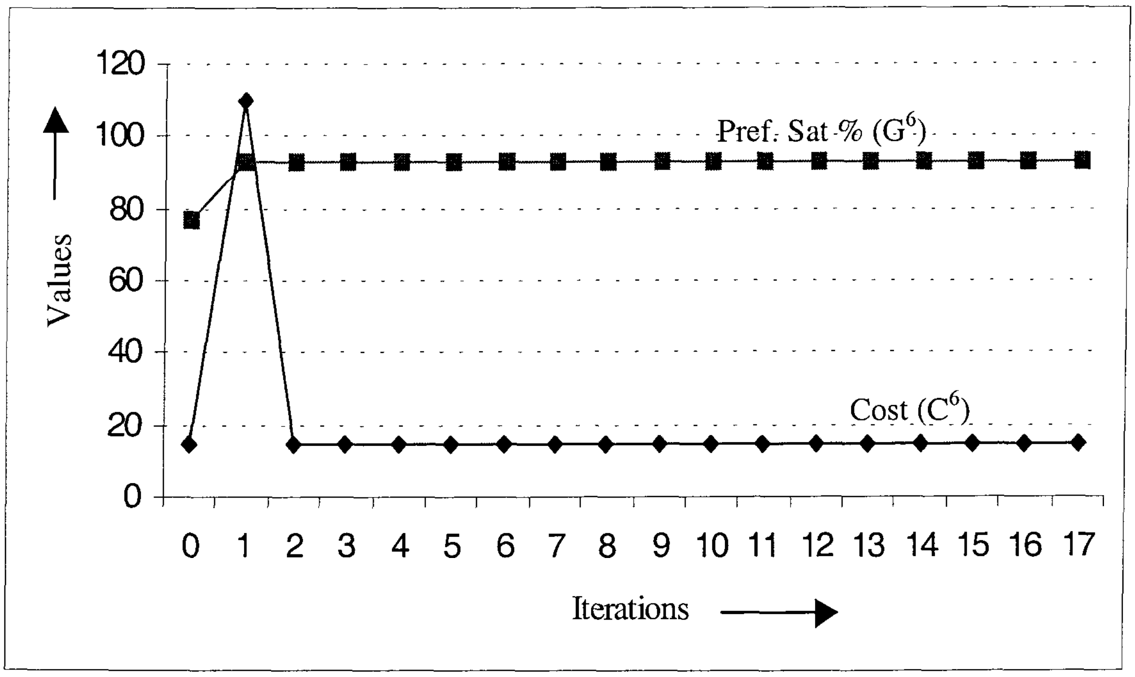


Fig. A.10 T^6 preference satisfactions & cost variation (without considering other tasks preferences and initial allocation order 6,5,4,3,2,1)

Appendix B

In this appendix we present the results of the experiments that are similar to the one discussed in section 7.2.2. This group of experiments investigates the convergence of the gain in preference values and cost when only one task, including its subtasks, is reallocated considering the preferences of the reallocated tasks. Also, we reverse the order of the initial allocation in one experiment to investigate whether such variations affect the gain in preference values and cost.

In these experiments all tasks are initially allocated on the first available slot, tasks are allocated according to the different order without considering preferences. Then one task is subsequently reallocated to satisfy its preferences. As a result of this reallocation some tasks can be reallocated. These tasks are reallocated to the first available slot taking their preferences into account.

Figures in this appendix show how the total cost (C^i) and gain in preference values (G^i) of T_i vary during the reallocation process. Iteration points are indicated by square symbols for the gain in preference value, and by diamond symbols for the total cost. With respect to the G^1 graph, the Y-axis values indicate the percentage preference gain while in the case of the (C^1) graph, they indicate the total cost units.

Figure B.1 shows the result of conducting this experiment for task T_1 when tasks are initially allocated according to the order T_1, T_2 ... T_6. Figure B.2 shows the result of this experiment for task T_1 when tasks are initially allocated according to the order T_6, T_5 ... T_1. Figure B.3 shows the result of conducting this experiment for task T_4 when tasks are initially allocated according to the order T_1, T_2 ... T_6. Figure B.4 shows the result of conducting this experiment for task T_5 when tasks are initially allocated according to the order T_1, T_2 ... T_6.

These graphs are similar to the graphs in Appendix A except that the final gain in preference value is reduced. For example in the case of T_5 it converged to 100% in A.7, while in B.4 it converged to 92% (8% reduction). This is expected, as explained in 7.2.2 is due to the fact that other tasks are also competing to gain preference values.

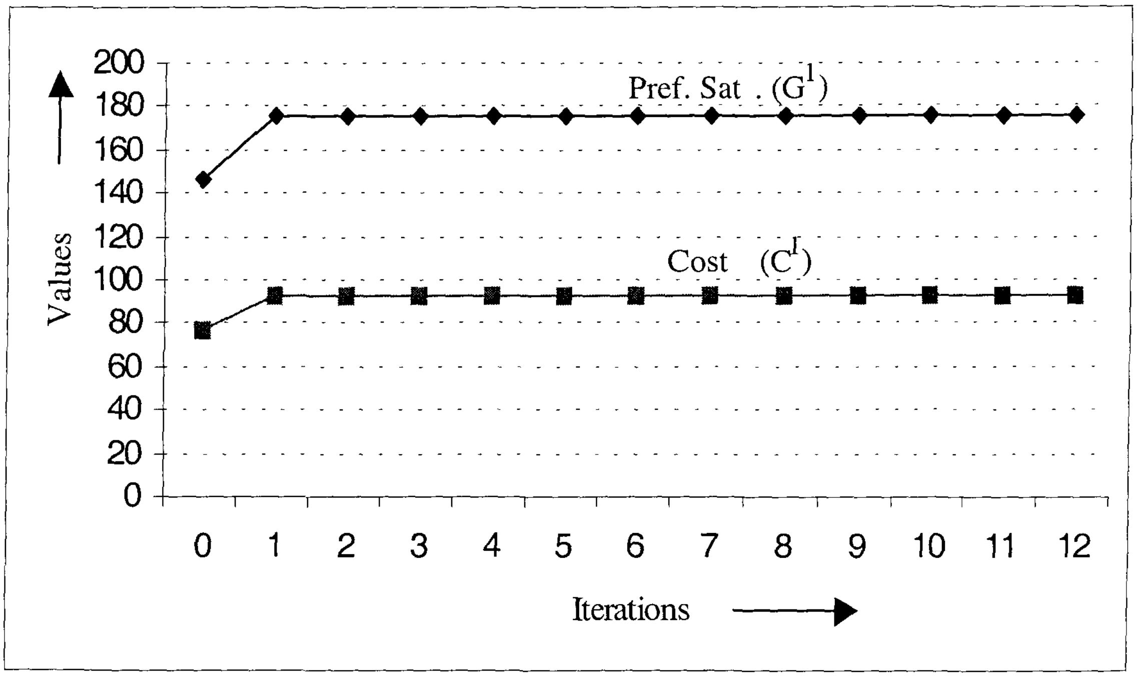


Fig. B.1 T^1 preference satisfactions & cost variation
(considering other tasks preferences and initial allocation order 1,2,3,4,5,6)

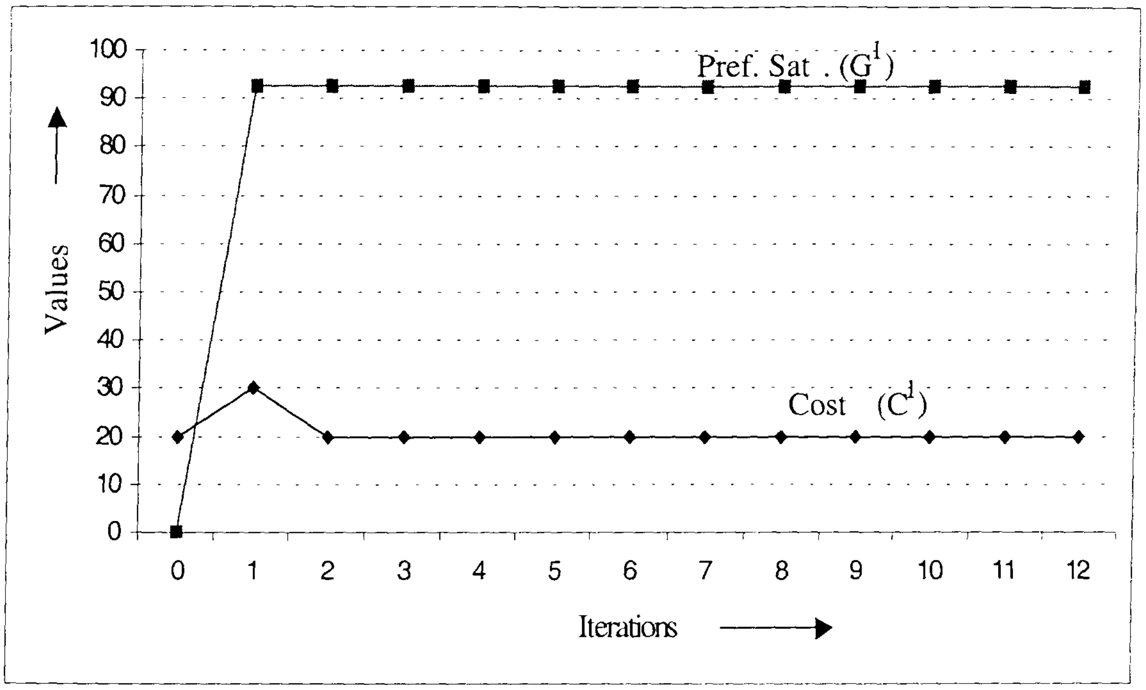


Fig. B.2 T^1 preference satisfactions & cost variation (considering other tasks preferences and initial reverse allocation order 6,5,4,3,2,1)

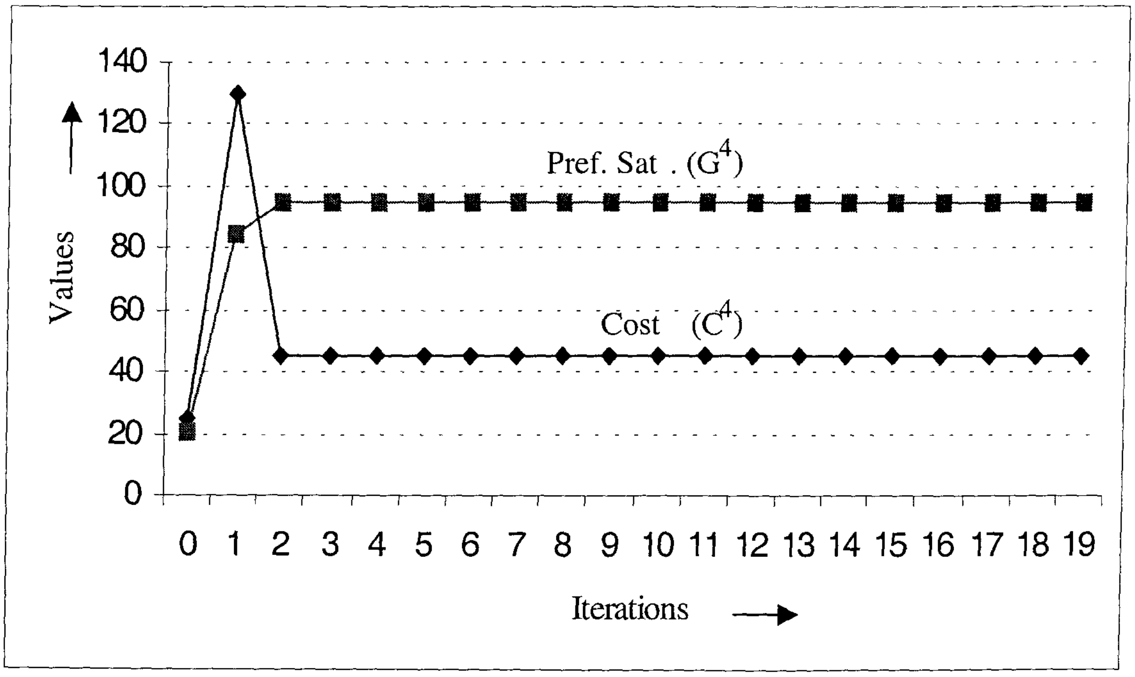


Fig. B.3 T^4 preference satisfactions & cost variation (considering other tasks preferences and initil allocation order 1,2,3,4,5,6)

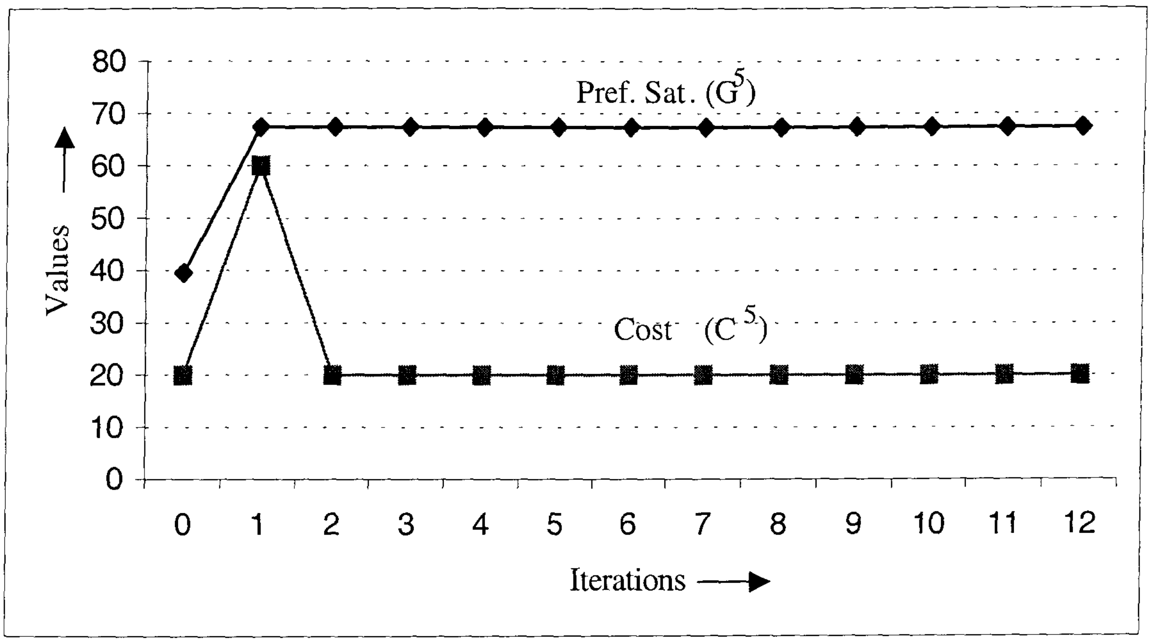


Fig. B.4 T^5 preference satisfactions & cost variation
(considering other tasks preferences and initil allocation order 1,2,3,4,5,6)

Appendix C

In this appendix we present the results of the experiments that were conducted to investigate the impact of varying the cut-off value (φ) on the convergence characteristics. These are similar to the experiment presented in 7.3.3. Figure C.1 shows the result for conducting the experiment using 14 coordinators and 54 subtasks. Figure C.2 shows the result for conducting the experiment using 6 coordinators and 24 subtasks. We used different values for φ as shown in the figures. As explained in 7.3.3 and shown in these diagrams, φ has little effect on the final preference value (± 1%). Though, φ did affect the number of iterations needed to reach this final value. As we stated before we can say that the lesser the value of φ the less iterations we need to reach convergence.

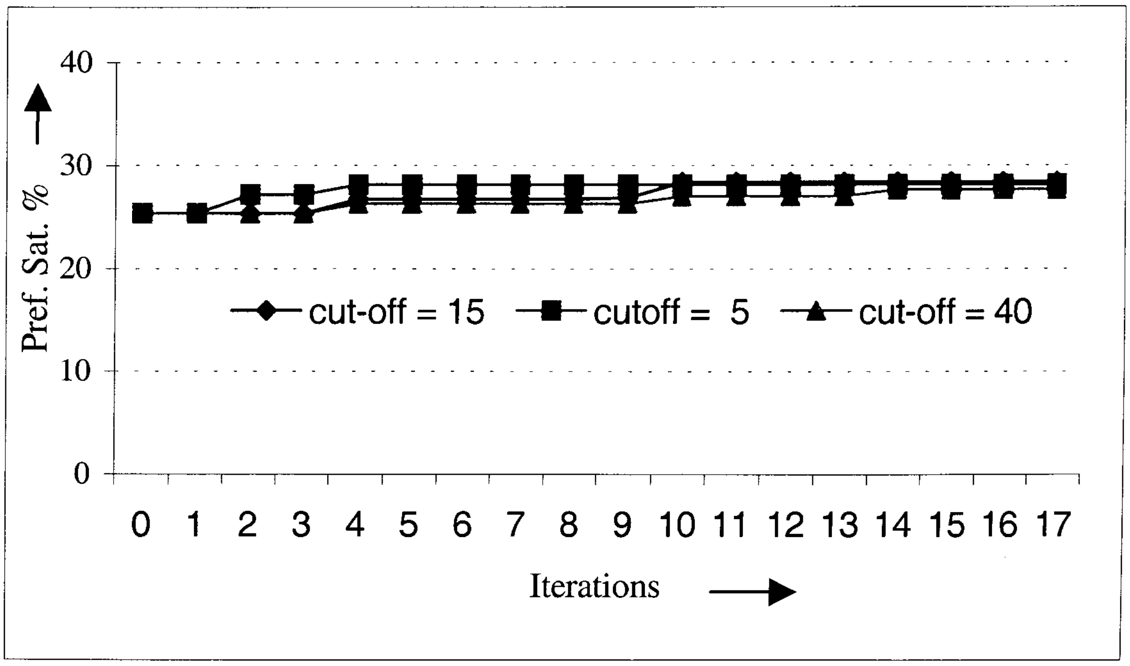


Fig. C.1 Varying the cut-off value (φ) (54 subtasks and 14 coordinators)

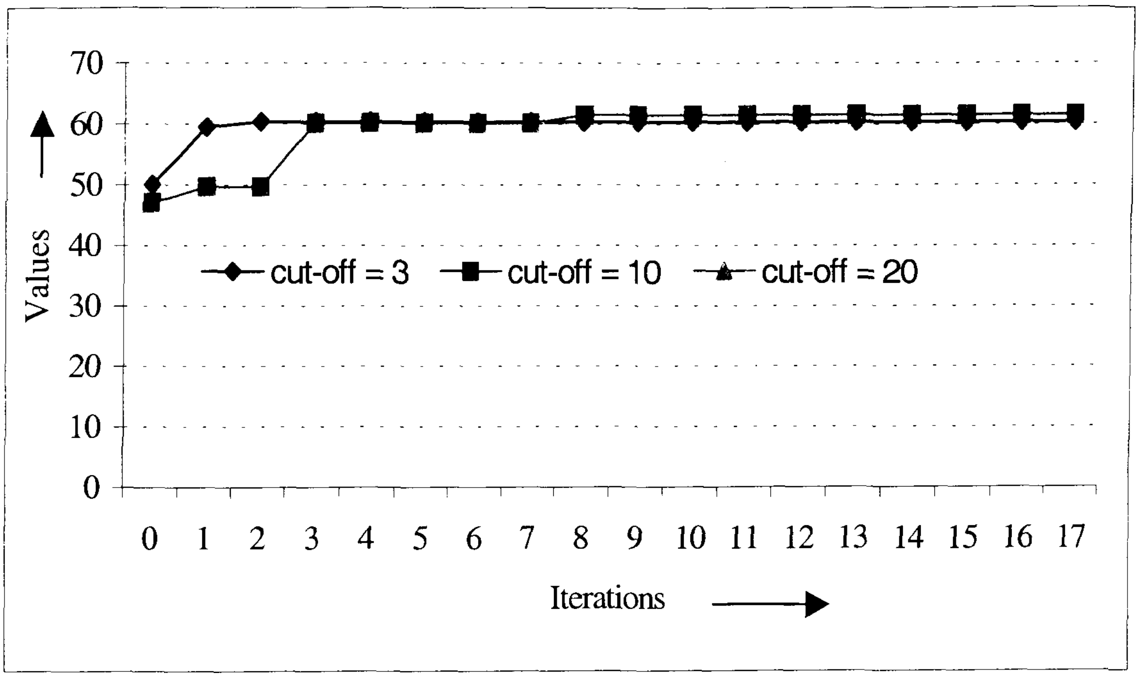


Fig. C.2 Varying the cut-off value (φ) (24 subtasks and 6 coordinators)

Appendix D

In this appendix we present the results of the experiments that are similar to the one discussed in section 7.4. This experiment is conducted to monitor the preference losses discussed in section 4.3.3. In these experiments we used 6 coordinators, 24 subtasks, and varied the preference cut-off value (φ) and the preference loss rate (ρ), see section 4.2.1. The results are shown in the tables D.1, D.2, and D.3. Table D.1 shows the result of the experiment when φ is set to 20% and ρ is set to 5%. Table D.2 shows the result when φ is set to 20%, and ρ is set to 10%. Table D.3 shows the result when φ is set to 10%, and ρ is set to 10%.

As stated in section 7.4, we find it difficult to predict which exchanges would be considered too expensive and when the funds would actually run-out using these results. This due to the fact that coordinators gain and lose funds from each other like money in a market during the processing. These preference losses are not independent. Therefore, monitoring the preference losses from these four sources at each iteration would not guide us to control or predict the solution during the allocation process.

Table D.1. Sources of preference loss

Subtask	Average pref. cut-off loss	Cost cut-Off loss	Average Pi loss
T11	7.3	5.0	11.1
T12	10.3	36.3	29.5
T13	8.3	0.0	35.9
T14	0.0	0.0	32.1
T21	0.0	6.6	10.0
T22	9.7	41.0	36.9
T23	10.0	16.0	41.1
T31	0.0	0.0	0.0
T32	12.1	26.1	39.1
T33	10.9	9.9	49.0
T34	11.5	17.9	70.3
T35	5.0	5.0	49.3
T41	7.3	0.0	11.1
T42	9.0	0.0	14.2
T43	0.0	0.0	9.0
T44	11.0	0.0	27.1
T45	10.0	10.0	16.3
T51	6.0	6.0	5.3
T52	9.0	0.0	8.9
T53	10.9	0.0	24.9
T54	10.9	38.7	59.9
T61	7.0	0.0	10.5
T62	10.3	10.5	15.5
T63	0.0	39.1	47.8
Total	9.3	19.1	27.3

Table D.1: This Table shows the different sources of preference loss during the allocation process for 24 tasks and 6 coordinators. φ=20% and ρ =5%.

Table D.2. Sources of preference loss

Subtask	Average pref. cut-off loss	Cost cut-Off loss	Average Pi loss
T11	0.0	0.0	0.0
T12	11.0	36.2	13.8
T13	9.0	9.0	0.0
T14	0.0	0.0	65.2
T21	0.0	13.2	20.0
T22	14.0	50.6	23.4
T23	12.0	21.3	43.6
T31	0.0	0.0	0.0
T32	14.4	38.5	64.3
T33	0.0	0.0	74.6
T34	13.0	17.5	95.8
T35	10.0	10.0	75.3
T41	0.0	0.0	2.6
T42	13.0	22.7	40.6
T43	0.0	0.0	11.3
T44	11.8	39.0	37.6
T45	11.0	0.0	13.8
T51	11.0	11.0	10.7
T52	0.0	0.0	13.5
T53	10.0	0.0	23.8
T54	14.0	49.1	95.0
T61	0.0	0.0	2.3
T62	11.3	43.8	77.6
T63	13.5	43.9	95.0
Total	11.9	29.0	37.5

Table D.2: This Table shows the different sources of preference loss during the allocation process for 24 tasks and 6 coordinators. φ=20% and ρ =10%.

Table D.3. Sources of preference loss

Subtask	Average pref. cut-off loss	Cost cut-Off loss	Average Pi loss
T11	0.0	0.0	0.8
T12	0.0	0.0	1.9
T13	0.0	0.0	11.3
T14	0.0	0.0	10.8
T21	0.0	0.0	10.4
T22	9.0	43.8	96.7
T23	0.0	0.0	96.3
T31	0.0	0.0	0.0
T32	0.0	43.9	71.4
T33	0.0	0.0	81.6
T34	8.0	13.0	98.6
T35	0.0	0.0	80.6
T41	0.0	0.0	0.8
T42	0.0	13.0	35.6
T43	0.0	0.0	3.5
T44	9.0	39.0	51.9
T45	0.0	0.0	13.8
T51	9.0	12.7	19.7
T52	0.0	0.0	11.6
T53	0.0	0.0	32.7
T54	9.0	28.1	55.6
T61	0.0	0.0	0.8
T62	8.0	46.3	94.1
T63	8.0	43.9	100.0
Total	8.6	31.5	40.8

Table D.3: This Table shows the different sources of preference loss during the allocation process for 24 tasks and 6 coordinators. φ=10% and ρ =10%.