Distributed Constraint Optimization, Resource Allocation and Scheduling in Large Scale Agent Networks

Panagiotis Karagiannis

PhD Thesis

Supervisor: Nikolaos Samaras, Associate Professor

Department of Applied Informatics

University of Macedonia
Thessaloniki
March, 2011

Ευχαριστώ θερμά τους κ. Άγγελο Σιφαλέρα και Βασίλειο Λαζαρίδη για τη βοήθειά τους κατά τη διάρκεια της εκπόνησης της διδακτορικής μου διατριβής. Επίσης ευχαριστώ θερμά τους κ.κ. Κωνσταντίνο Παπαρρίζο και Νικόλαο Σαμαρά για την επιστημονική και ηθική τους στήριξη. Τέλος ευχαριστώ θερμότατα την οικογένειά μου και κυρίως τους γονείς μου στους οποίους και αφιερώνω τη διδακτορική μου διατριβή.

ABSTRACT

The thesis explores new directions pertaining to methods for scheduling and allocating atomic and complex tasks in large-scale networks of homogeneous or heterogeneous cooperative agents. Tasks are requests for resources managed by the agents that populate the network. The proposed methods encapsulate the concepts of searching, task allocation and scheduling seamlessly in decentralized processes. Consequently, there is no need for accumulated or centralized knowledge. Furthermore, centralized coordination is also not necessary. Efficient searching for agent groups that can facilitate the scheduling of tasks is accomplished through the use of a dynamic overlay structure of gateway agents and the exploitation of routing indices. Gateway agents are network nodes that due to specific topological issues have the ability to accumulate limited knowledge relevant to the resources available in their immediate neighbourhood. They issue, keep and constantly update routing indices based on their view on local resources. Routing indices are used for directing requests χfor resources towards parts of the network where the probability of them being served is high. Efficient routing leads to formation of agent teams that try to break down each task into smaller pieces and allocate the appropriate resources to it. The task allocation and the scheduling of complex tasks are accomplished by combining dynamic reorganization of agent groups and distributed constraint optimization methods. After a complex task has been broken down into subtasks, constraints that may exist between the derived subtasks or constraints inherent to the resource allocation procedure are modelled as a distributed constraint optimization problem. The newly formed agent team that has been assigned the problem tries to solve it optimizing the cost due to constraints, at the same time. Upon success the requested resources are allocated accordingly. Otherwise the complex task is forwarded on its entirety so as to another agent team can be formed and try to solve the problem Experiments have demonstrated promising results.

ΠΕΡΙΛΗΨΗ

Η διδακτορική διατριβή παρουσιάζει πρωτότυπες μεθόδους για τον καταμερισμό πόρων και αποτελεσματικό τρόπο κατανομής αυτών σε μεγάλα δίκτυα ομογενών ή ετερογενών πρακτόρων. Οι προτεινόμενες τεχνικές συμπεριλαμβάνουν και υλοποιούν τις έννοιες της αναζήτησης, του καταμερισμού πόρων και της κατανομής πόρων σε ένα ενιαίο σύστημα, πλήρως κατανεμημένο. Δεν είναι απαραίτητο να υπάρχει κεντρικό σύστημα διαχείρισης δεδομένων ή οποιασδήποτε άλλης οντότητας σχετικής με τη διαδικασία. Η αποτελεσματική υλοποίηση του περιβάλλοντος επικοινωνίας μεταξύ ομάδων πρακτόρων και της διαδικασίας αναζήτησης επιτυγχάνεται με τη βοήθεια δικτύων επικάλυψης. Τα δίκτυα αυτά είναι δυναμικά και αποτελούνται από πράκτορες οι οποίοι λόγω τοπολογίας είναι σε θέση να διατηρούν εκτεταμένη γνώση όσον αφορά τους πράκτορες με τους οποίους γειτνιάζουν. Διατηρούν επιπλέον δείκτες αναδρομολόγησης που χρησιμοποιούν για την κατεύθυνση του εργασιακού φόρτου προς περιοχές του δικτύου πρακτόρων οι οποίες κρίνεται ότι έχουν αυξημένες πιθανότητες να ικανοποιήσουν τη ζήτηση σε πόρους. Ο καταμερισμός και η κατανομή των πόρων του συστήματος γίνονται με δυναμική αναδιάρθρωση ομάδων πρακτόρων, κάτι το οποίο οδηγεί σε σχηματισμό προσωρινών ομάδων τα μέλη των οποίων είναι πιθανοί αποδέκτες τμήματος ή τμημάτων των διεργασιών που απαιτούν πόρους προς ίδια χρήση. Οι διεργασίες που ζητούν πόρους και οι περιορισμοί μεταξύ τους ή μεταξύ της διαθεσιμότητας των πόρων του συστήματος μοντελοποιούνται και παίρνουν την μορφή ενός κατανεμημένου προβλήματος βέλτιστης ικανοποίησης περιορισμών. Στη συνέχεια η νεοσυσταθείσα ομάδα προσπαθεί να λύσει το κατανεμημένο πρόβλημα. Σε περίπτωση επιτυχίας δεσμεύονται και οι αντίστοιχοι πόροι του συστήματος. Σε άλλη περίπτωση το πρόβλημα προωθείται έτσι ώστε μια άλλη ομάδα πρακτόρων να συσταθεί και να προσπαθήσει να το λύσει. Πειραματικά αποτελέσματα έχουν δείξει ότι η παραπάνω προσέγγιση επιτυγχάνει άκρως ικανοποιητικά αποτελέσματα.

TABLE OF CONTENTS

CHAPTER 1

1. Introduction

1.1 Distributed Task Allocation and Coordination

The present thesis addresses the problem of distributed task allocation and coordination in large-scale networks of homogeneous or heterogeneous cooperative agents. There are many real life problems that are naturally distributed among a set of agents (sensor networks, flight reservation systems, network congestion control and routing, meeting scheduling coordination, supply chain management etc). Even with centralized systems in mind resource allocation and coordination is a typical but fairly difficult problem. In the case of personal computers, for instance, the operating system is responsible for allocating its resources. In distributed systems, task allocation is NP Complete in the general case (Rothkopf et al., 1995, Modi et al., 2001).

The general problem involves a network of nodes (agents) with certain capabilities (resources). Agents must allocate the resources they control to a set of tasks. In the case of a computer network resource or capability types could be computational power or storage capacity. Tasks that require these resources enter the network in an arbitrary fashion. A set of such tasks is successfully allocated, if the network manages to find the node or nodes that can provide the resources each task has demanded. Moreover, there are cases where a task is temporally constrained. Temporally constrained tasks are more complex because they require certain resources within a bounded time interval. In addition, there are cases where agents must find particular assignments for allocating sets of tasks that either optimize the network's efficiency or minimize their computational effort.

Distributed task allocation problems most frequently arise in peer-to-peer systems, grids and virtual organizations. Along with task allocation, decentralized control of large-scale systems of cooperative agents is a hard problem in the general case as well. The computation of an optimal control policy when each agent possesses an approximate partial view of the state of the environment (which is the case for large-scale distributed systems) and agents' observations are interdependent (i.e. one agent's actions affect the observations of the other) is very hard even if agents' activities are independent (i.e. the state of one agent does not depend on the activities of the other) (Goldman and Zilberstein, 2004). Decentralized control of such a system in cases where agents have to act jointly is even more complex. In the case of joint activity, subsets of agents have to form teams in order to perform tasks subject to constraints. Acting as a team, each group has to be coordinated by scheduling subsidiary tasks with respect to temporal and possibly other constrains, as well as other tasks that team members aim to perform (e.g. as members of other teams).

Coordination in agent systems is primarily focused on the cooperative control of systems that are formed by a collection of homogeneous or heterogeneous decision-making components. Significant problems that may be encountered include limited

processing resources, distributed knowledge or information and poor communication capabilities. Regardless of what has been mentioned it is prominent that the major issue in decentralized coordination is the distributed nature of decisions that have to be made and knowledge thereof. While this is not an issue in centralized architectures, it is clear that not all parts of an agent network can be aware of the all facts concerning the collective computational effort at any specific time instance. Therefore, strategic decisions that affect the system's performance should be categorized and sorted based on estimates of potential impact on the system's performance. Subsequently they should be communicated to all participants. Otherwise, as in our approach, every agent should decide based on limited knowledge repositories (itself and its immediate neighbours). We have studied effectively the ways in which limited knowledge can lead to achieving solutions for the task allocation problem in distributed systems.

Let us for instance consider a scenario (e.g. a terrorist attack) where crisis-management agents (policemen, fire brigades, health professionals, army forces) need to jointly perform numerous tasks within a certain time interval (i.e. agents have a limited time horizon due to the emergency of the situation). Each agent has its own capabilities and they all form an acquaintance network depending on the physical communication means available. While requests for joint activities arrive (e.g. new explosions, civilians requesting for help etc.) agents need to form teams to handle each individual situation. This comprises three interleaved sub-problems: searching for the appropriate agents, allocating tasks to them according to their capabilities, and scheduling these tasks subject to temporal and other constraints. In contrast to previous works where these sub-problems are largely tackled independently, here the three-step process is viewed and solved as a single problem.

This work provides extensive study in the area of effective and efficient searching through the dynamic construction of overlay networks of gateway agents and the exploitation of routing indices. Decisions concerning the formation of the gateway agents network and the construction of routing indices are made by each agent individually and they are based on facts that can be derived by each agent's set of one hop away neighbours. The search for the appropriate agents that will handle each incoming request is done via the "heads" of services (gateway agents). A gateway agent knows best the availability and capabilities of its subsidiaries (exploiting routing indices). However, since heads have to manage numerous incoming requests (e.g. emergent situations in the above scenario), they can propagate such requests to subsidiaries, which act so as to form task-specific teams depending on the requirements of each situation.

A request may need to be propagated several times before a team of agents capable of handling it is gathered. After a team of agents is formed, the members of the team need to jointly schedule their activities taking into account interdependencies, as well as their other scheduled activities. This is in itself a hard problem which our framework has tackled using a combination of dynamic team reorganization and distributed constraint optimization methods. Every agent team that is assigned a request must find within a finite period of time if there is a proper way to serve it. In that case it will search for the optimal assignment of the appropriate resources to the request. Otherwise, the team must decisively reach to the conclusion that either the request can not be ultimately served or the specific agent team has inadequate

resources to do it. In the former case the system drops the request, while should the latter occur the request is propagated one more time and a new team that will try to serve the request is constructed, operating as previously described.

1.2 Related Work

In distributed or decentralized systems, where participants form a network, it is extremely difficult for each member to have an exact view of its location in relevance to the global network configuration. Moreover, the degree of difficulty increases as these networks become larger and issues as coordination or collaboration come to the picture. In relatively large agent networks for instance, gathering precise information pertaining to an agent's environment can be frustrating even if we focus on the physical placement or capabilities of neighboring agents. Furthermore, there are certain events such as an agent loosing capabilities (or gaining new) that would require constant monitoring if another agent must be fully aware of them. These issues are prominent when a team of agents is in the process of working together towards certain goals. In an already formed agent-team, if a member has to leave or cannot respond for a time period, creates problems that are not easy to control.

A notable early work combining resource allocation and coordination in multi-agent systems can be found in (Liu and Sycara, 1998). The authors embrace the notion of tightly coupled agents, giving attention to concepts as coordination and real-time scheduling. The main contribution of this work is that it presents an efficient coordination technique and integrates it with the process of addressing a real-time problem.

In (Modi et al., 2001) there is a formal definition of the resource allocation problem. The authors view distributed resource allocation as a problem in which a group of agents must conform to specific criteria while assigning their resources to a set of requesting tasks. They are motivated by a number of real-world problems such as distributed sensor networks, disaster rescue and timetable scheduling.

The work in (Goldman and Zilberstein, 2004) claims that coordination and control techniques in agent systems is far more complex when it is decentralized and agents must collaborate for to complete mutual efforts. Joint activities of subsets of agents within a broader infrastructure, especially when certain limitations exist, require control structures that are capable of scheduling subsidiary tasks with respect to existing restrictions. Therefore, the authors argue that the control process can be modeled as a decentralized observable Markov decision process (Goldman and Zilberstein, 2004). They also present two algorithms that solve optimally classes of goal-oriented decentralized processes. Decentralized control under temporal constraints is also studied in (Koes et al., 2005). The authors propose an algorithm for centralized coordination of heterogeneous robots with spatial and temporal constraints. Although the authors demonstrate that their method can find schedules for small groups of robots very fast, there is no information on the scalability of their approach.

Extreme teams were deployed in distributed resource allocation by (Scerri et al., 2005) where they introduced the LA-DCOP algorithm. LA-DCOP performs task allocation after consulting dynamically computed capability thresholds. Tasks are

represented by tokens something that minimizes communication. Furthermore, the algorithm is able to create potential tokens to deal with concurrency issues of inter-task constraints. LA-DCOP was further improved in (dos Santos and Bazzan, 2009) where the eXtreme-Ants algorithm where agents in extreme teams also use methods to divide the labor of performing resource allocation.

Task allocation in extreme teams is generally associated with four features: (i) dynamic environments, in which tasks can appear and disappear; (ii) agents perform multiple tasks given their available resources; (iii) agents have overlapping functionality to perform the tasks but with differing levels of capability; and (iv) inter-task constraints can be present, imposing simultaneous execution requirements. In this thesis, we deal with all four key issues, extending the problem along the following dimensions: (a) handling temporal constraints among tasks, (b) dealing with agents that do not have the capabilities to perform every task, and (c) integrating searching with task allocation and scheduling.

Coordination in large-scale networks of heterogeneous agents using token-based approaches demonstrated promising results. The proposed algorithm in (Xu et al., 2005) uses tokens as a means of encapsulating shared components. This way tokens can carry information, tasks or resources, forming a knowledge repository agent teams use to coordinate their actions. Tokens actually work as access control mechanisms. Agents that have a token have exclusive access to the resources it carries and it is up to the agent to use the token or pass it to a teammate. Agents use local decision theoretic models to route tokens. Token-based methods have proved themselves scalable and capable of coordination large-scale systems. Nevertheless, token-based approaches do not inherently deal with scheduling constraints and dynamic settings. In our approach, tokens concerning the availability of resources and capabilities are being used for constructing agents' partial views of the network status using routing indices. Routing is further restricted to the connected dynamic sub-network of gateway agents which manage searching.

In (Mailler and Lesser, 2006b, Mailler, 2006) a protocol was proposed for solving a distributed resource allocation problem while conforming to soft real-time constraints in a dynamic environment. Although this is not the same problem as task allocation, it is worth mentioning that the approach towards solving the resource allocation problem taken in (Mailler and Lesser, 2006b) has certain similarities with our approach. To be precise, resource allocation was solved modeled as an optimization problem, similar to a Partial Constraint Satisfaction Problem. The protocol of (Mailler and Lesser, 2006b) uses constraint satisfaction based pruning techniques to cut down the search space, coupled with a hill climbing procedure.

It has to be noticed that in our effort we do not deal with communication decisions for optimizing information sharing/exchange as done in (Goldman and Zilberstein, 2003), or for proactively exchanging information (Zhang et al., 2004) This is orthogonal to our research which may further increase the efficiency of the proposed method. However, we point that this can not be done in any way such that agents share a global view of the environment state (Pynadath and Tambe, 2002, Xuan et al., 2001, Yen et al., 2001).

The (C_TÆMS) representation (M. Boddy and Maheswaran, 2006) is a general language (based on the original TÆMS language (Davin and Modi, 2006)) widely used for distributed planning and scheduling in multiagent systems. In order to deal with uncertainties, C_TÆMS tasks have probabilistic utilities and durations. Agent coordination and scheduling in dynamic and uncertain environments using C_TÆMS has been addressed by various approaches within the DARPA Coordinators program (Maheswaran et al., 2008, Musliner and Goldman, 2006, Smith et al., 2007). These approaches are more general than ours in the type of tasks they consider, since for the purposes of this thesis we limit ourselves to tasks with fixed duration. Musliner et al. use distributed Markov Decision Processes (MDPs) as the underlying formalism to capture uncertainty (Musliner and Goldman, 2006). Smith et al. use Simple Temporal Networks (STNs) to infer feasible start times for the tasks allocated to agents in the system (Smith et al., 2007). Once dynamic changes occur, agents heuristically determine task insertions in their schedule and change the STN constraints in order to make such insertions feasible. Finally, (Maheswaran et al., 2008) introduced the Predictability and Criticality Metrics (PCM) approach in which dynamic changes are handled by making schedule modifications, chosen heuristically through simple metrics from within a set of possible modifications that can increase the team utility.

In a more relevant note to our work, in (Sultanik et al., 2007) the authors solve a class of multi-agent task scheduling problems by mapping specifications expressed in a subset of C_TÆMS to distributed constraint optimization problems (DCOPs) and hence allowing the use of algorithms such as Adopt (Modi et al., 2005). Apart from the mapping, this work focuses on the use of constraint propagation to prune the domains of the variables in the resulting DCOPs and hence achieve efficient solving. We should note that this work and all the C_TÆMS ones mentioned above, focus on how to handle distributed planning and scheduling. In contrast, the approach we present here views the whole process of task allocation and scheduling as tightly interconnected problem and to this extend we propose a combination of methods, concerning all of its aspects.

Finally, we need to point out that the approach presented in this work extends the work proposed in (Theocharopoulou et al., 2007). Indeed, here we presents and evaluates configurations of the overall method using new methods for task allocation and scheduling. Experimental results demonstrate that the new methods outperform the ones proposed in (Theocharopoulou et al., 2007).

1.3 Contributions

Being interested in the development of decentralized methods for efficient task allocation and coordination in large multi-agent systems, this work had to contribute advances in diverse problems. Normally, problems of such complexity are broken down into subproblems. Subsequently research efforts focus on addressing each part of the whole picture independently. One major contribution of this thesis is that the distributed task allocation and coordination issue is addressed by building a framework that integrates the two separate procedures seamlessly into a coherent infrastructure. Moreover, our approach is resilient, flexible and highly consistent in producing promising results.

Although experiments used parameters that stressed all possible system configurations to their limits, quantitative measures were highly promising. In all experimental sets the combined demand of the tasks that had to be served was equal to the agent network's resource capacity. Additionally, all tasks consisted of smaller subtasks with randomly different requirements. This also added complexity to the solution efforts. Each different subtask being part of a complex task entering the agent network came with a set of constraints that ought to be satisfied. Nevertheless, and despite these difficulties our approach achieved astonishing results raising the percentage of successfully allocated tasks as high as more than 95%.

Another major advance is that the approach introduced in this thesis proved to be reliable in low-density networks. In our experiments, no agent network ever exceeded an inarguably low 3% as an edge density ratio. Notwithstanding, experiments proved that the proposed task allocation scheme displays consistently robust performance capabilities not matter how sparse the agent network is. As far as the agent network graph remained, connected edges could be left out without affecting significantly the system's performance. For instance with density ranging between 0.5% and 2.5% the difference in the percentage of tasks that were successfully served was less than 1.5% when the best case was compared to the worst. Besides, what little was gained in performance was lost in communication costs. In cases with networks of higher densities, improvements in performance were possible but the amount of exchanged messages between agents increased rapidly. An obvious advantage of this attribute is that our scheme can be deployed in real time distributed systems, where connections between agents being geographically scattered display similar characteristics.

The advantages of complete search methods for distributed task allocation problems over incomplete methods have already been indicated in previous works. Here we further extend these efforts by building a dynamic framework for testing various alternative aspects. In research endeavors up to date, experimental sets were compared based solely on static parameter alterations. Although this is done here as well by deploying dynamic reorganization tactics, we created a completely new experimental concept. Consider the case where an algorithm is tested towards a set of predefined benchmark problems. Keeping all parameters set to certain values every system snapshot during a single solution procedure is identical through different solution efforts provided they are taken at the same step of the whole process. This is entirely not the case in our approach. Let us illustrate our claim.

It is feasible to produce the exact same agent network between two different experiments. It is also possible to have the same task set that will require the exact same agent resources and will bear the exact same constraints between any pair of subtasks. We can also provide guarantees that every task will enter the network at the same node. We will also use the same algorithm for distributed task allocation. In case the agents in our network did not have the ability of communicating, they would process each task and decide to provide resources to it or drop it thus producing the same solution in each different run. In our scheme, there is not only communication but decision-making based on it as well. Even if all tasks appear were they are expected at fixed times we end up with different solutions every time. Our agents communicate asynchronously which means that the message delivery sequence is completely unpredictable. While this could be considered commonplace, the case becomes different should we consider our agent coordination scheme where routing

and agent team formation is bound to be different even with the same message delivery sequence. One of the many reasons for such behavior could be caused by a quantum of difference in agent team formation. Thus, the decision sequence in each agent is completely unpredictable as well, making all steps in the solution procedure unique at each run. We managed to demonstrate that even in environments of extreme unpredictability our framework displays high efficiency rates through effective problem modeling and constant adaptability. A subsidiary benefit was also the fact that algorithms for distributed task allocation were tested through a novel testing framework. Further contributions as far as quality assessment issues are concerned entail steps toward new directions to self-organization approaches for ad-hoc networks, token-based approaches for coordination in large-scale systems and distributed constraint satisfaction/optimization. In summary, qualitative contribution aspects are categorized as follows:

- A generic method is proposed for task allocation and scheduling that combines dynamic reorganization of agent teams with distributed constraint optimization. Several versions of this method, which may differ either in the way task allocation or/and scheduling is performed, are implemented and compared experimentally.
- Careful consideration of the experimental results demonstrates the efficiency of the proposed overall method and different variations of it. Two different configurations with distinct distributed constraint optimization methods are compared and evaluated the efficiency of the task allocation and scheduling mechanism. Results show that the task allocation methods presented coupled with the complete constraint optimization algorithm Adopt achieve quite promising results.

1.4 Outline

The thesis is structured as follows:

Chapter 2 summarizes related work in the area of agents and multi-agent systems.
Chapter 3 presents an extensive review on centralized and distributed Constraint Satisfaction Problems.
Chapter 4 formalizes the concept of the agent networks which were used, how tasks were modeled and the methods in which these tasks were arranged in order to challenge the agent network's resource allocation capability.
Chapter 5 describes how we organized our agents into overlay networks and the way routing indices for task distribution were formed.
Chapter 6 provides Distributed Constraint Optimization methods for modeling and scheduling complex tasks. Furthermore, it discusses Local Search Methods for solving Distributed Constraint Optimization Problems. Finally, it presents Adopt a complete algorithm for Distributed Constraint Optimization.
Chapter 7 presents a seamless framework for searching, tasks allocation and scheduling.
Chapter 8 comprises the system implementation, the experimental methods and results thereof.
Finally, Chapter 9 concludes the thesis providing insightful comments for future directions.

This page is intentionally left blank

CHAPTER 2

2. Software Agents and Distributed Artificial Intelligence

2.1 Agents and Software Agents

Agent technology originated after intrinsic research requirements stemming from the field of artificial intelligence. An agent is anything that can perceive its environment through sensors and act upon that environment through actuators (Russell and Norvig, 2010). Autonomy is the most essential feature, which differentiates an agent from other simple software entities (Jennings and Wooldridge, 1998). A software agent is an entity that functions continuously and autonomously in a particular environment, often inhabited by other agents and processes (Shoham, 1993).

Continuity and autonomy as defined by Shoham (Shoham, 1993) imply that an agent should be able to draw initiatives and carry out activities toward its goals in a flexible and intelligent manner. Thus, we can infer that agents are components that have goals, which should be ultimately reached. Agents also possess intelligence, which allows them to behave accordingly. Intelligence is highly connected with responsiveness. Agents should be responsive to changes that may happen in their environment or their mental state as a result of the information they sense or receive. All these activities should form a seamless process without requiring any form of either guidance or intervention, be it human or software driven. Therefore, agents should be able to operate autonomously. As already mentioned an agent living and acting continuously in an environment over a certain period of time should be able to adjust its behavior accordingly. Such an attribute involves deciding after evaluating and considering any existing source of information, and potential learning abilities. This may also require a certain degree of socialization with other agents (Jennings and Wooldridge, 1995). It is therefore expected that any agent inhabiting an environment or domain where other agents exist and function, should to be able to communicate and cooperate with them. In case, the requirement for mobility is also present it must do so while emigrating (Bradshaw, 1997).

Software agents nowadays emerge as an important computing paradigm and provide a new development paradigm for software engineering in many aspects, for example, towards implementing intelligent user interfaces (Lieberman, 1997), electronic commerce applications (Nwana et al., 1997), business process management activities (Bernuy and Joyanes, 2008), and digital libraries (Yang et al., 2002).

2.2 Foundation for Intelligent Physical Agents

The Foundation for Intelligent Physical Agents (FIPA) is an Institute of Electrical and Electronics Engineers (IEEE) Computer Society standards organization. Its purpose is to promote agent-based technology and interoperability. The core mission of FIPA is to assist the interoperation between agents across heterogeneous technologies.

While originally formed to produce software standards specifications for interacting agents and agent-based systems FIPA has since developed and has been working on numerous specification standards. These include agent platform architectures to support communicating agents, agent communication languages, content languages for expressing messages and interaction protocols that expand the scope from single messages to complete transactions (FIPA, 2005). FIPA Abstract Architecture provides abstract architecture for multi-agent systems.

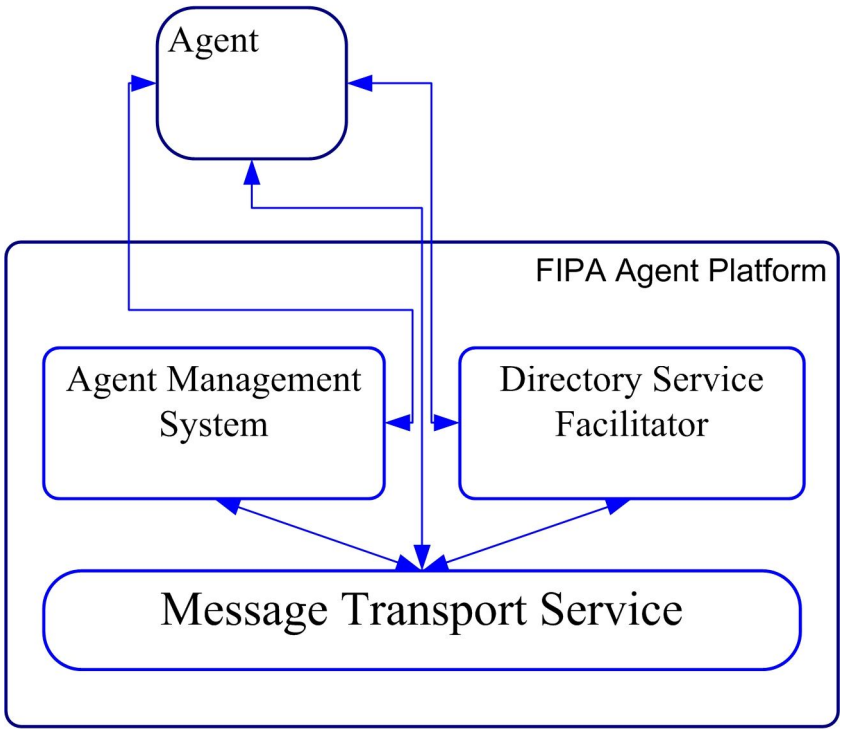


Figure 2.1: The FIPA Agent Platform.

Amongst the technology-oriented features of its abstract architecture there are entities for platform management (white and yellow pages), message transport and agent communication protocols. An overview of the FIPA agent platform is shown in Figure 2.1 (FIPA, 2005).

2.3 Agent Based Computing

Agent Based Computing emerged after combining two existing technologies. Those were Artificial Intelligence (AI) and Object- Oriented Distributed Computing. Agent Based Systems' primary objective is to empower the field of Artificial Intelligence with as much computational effectiveness as possible (Bradshaw, 1997). Agents are intelligent, autonomous, software components capable of interacting with other agents within an application. Their intention is to achieve a goal or sometimes, a common set of goals, thus contributing individually to the resolution of any given problem (Genesereth and Ketchpel, 1994). Agents nowadays are able to inter-operate within modern applications (e-commerce, data mining, information retrieval etc.).

Generally, an agent is an assistant that works on behalf of others (agents or humans). In its simplest form an agent represents an actor and is thereby a personification of its actions (Jennings and Wooldridge, 1996). In AI, an agent is a software entity that lives inside a computational environment, operates is a continuous and autonomous fashion and is capable of cooperating with software entities sharing similar characteristics. An agent can also be associated with its mental state. An agent's mental state can be composed of components like belief, capabilities, choices, and commitments. Wooldridge and Jennings (Jennings and Wooldridge, 1995) have introduced weak and strong notions of agencies. They have used the stronger notion of an agency to imply "a computer system that, in addition to having some basic properties, can be either conceptualized or implemented using concepts that are more usually applied to humans." The interactive nature of multi-agent systems calls for consensus on agent interfaces in order to support interoperability among agents coming from various sources (Lieberman, 1997). Foundation of Intelligent Physical Agents (FIPA) has developed standards (FIPA, 1998a; FIPA, 1998b; FIPA, 1998c; FIPA, 1998d, FIPA, 2003) for building interoperable agent-based systems.

Characteristics are essential physical properties of agents. The following are a set of the most commonly used agent characteristics. There are mainly two agency models: the weak agency model and the strong agency model .The weak agency model, which is defined by Wooldridge and Jennings, is widely accepted. The first set describes the weak notion of an agency (Jennings and Wooldridge, 1995). These are:

- Autonomy**:** Agents act on their or another entity's behalf without further guidance being necessary. Agents operate without the direct intervention of human or others, and have some kind of control over their actions and internal state.
- Social Ability**:** Agents interact with other agents (and possibly humans) using the assistance of an agent communication language.
- Reactivity: Agents perceive their environment (which may be the physical world, a user via a graphical user interface, a collection of other agents, the Internet, or perhaps all of these combined) and respond in a timely fashion to changes that occur in it.
- Pro-activeness: Agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative.

Additionally there are researchers who tend to believe that there should be a stronger more specific meaning when referring to the term agency. These work in addition to the characteristics already mentioned. There is no definitive set of such characteristics. Nevertheless, bibliography most commonly includes:

- Communication: Agents communicate with other agents working toward a common goal by exchanging a sequence of messages.
- Mobility: An agent can migrate from an agent system to another in a pre-determined fashion or at their own discretion. Accordingly, agents can be static or mobile.
- Learning: Agents may have the ability to learn new information about the environment they operate in. In such way they are able to enhance their

performance and dynamically improve their own behavior by achieving capabilities that require high intelligence standards.
- Cooperation: Agents collaborate and cooperate with other agents during execution to minimize redundancy and solve common problems.
- Veracity: An agent must be only communicating what it believes is true.
- Benevolence: Agents are assumed to do only what is asked by other agents or humans.
- Rationality: Agents should act in suitable or even optimal ways in order to achieve their goals.

Autonomy and communication are considered to be the most basic agent characteristics. On the other hand, communication and mobility contribute toward another model of distributed computing (also known as agent-oriented programming).

2.4 Programming Agents

Once an agent-system has been adequately designed, it needs to be implemented. Yoav Shoham (Shoham, 1993) used the term Agent Oriented Programming (AOP) to describe this new programming concept. Generally, Agent Oriented Programming is a programming framework for building applications using autonomous components. Shoham also introduced Agent-0 a prototype agent-oriented programming language. In Agent-0 an agent's life-cycle is presented in Figure 2.2.

Agent Oriented Programming can be viewed as a specialization of Object Oriented Programming, where basic computation consists of agents and their states. An agent's state comprises Beliefs, Decisions, Capabilities and Obligations. Therefore agents can inform, request, offer, accept, reject, compete, and assist each other based on their own beliefs, desires, and intentions (Kim, 2005). The agents, their mental states and their interactions would have to be programmed to build an application that accomplishes its goal.

Since (Shoham, 1993), have been proposed many specialized programming languages and platforms for agent programming. Two examples of agent oriented languages that make extensive use of object oriented languages are JACK (Bordini et al., 2005) and Jadex (Pokahr et al., 2005). They both use Java as the underlying language.

On the other hand, there is the theoretically driven approach. This aims at designing a stand-alone agent programming languages from scratch, using mental attitudes and knowledge representation as primary concepts. Examples of the theoretically driven approach are languages like 3APL (Hindriks et al., 1999), AgentSpeak (Rao, 1996), GOAL (Hindriks et al., 2001), or IMPACT (Subrahmanian et al., 2000).

Other frameworks for Multi-agent Systems also exist. An early approach , which uses roles, organizational units and links is MOISE (Hannoun et al., 2000). These concepts allow MOISE to regulate social exchanges between agents and build organizational structures for fulfilling various tasks. MOISE was later extended to MOISE+ (Hübner et al., 2002). In MOISE+ roles units and links are considered as the structure of an organizational model. Apart from structure the authors state that an organization

should incorporate two additional and distinguishable concepts. These are functioning (global plans, tasks, etc) and deontic (norms, laws, etc).

OperA (Dignum, 2004) is a model acting as an abstract protocol that indicates how agents should act according to existing social requirements. For interacting with each other agents use contracts, which consist of formal expressions. Generally, the OperA framework allows interaction between heterogeneous agents and differentiates organizational characteristics from agents' goals. Specifically, agents are organized in a structure of roles and interactions. Moreover, there is a social model which defines which agents have specific roles and an interaction model which defines how agents interact.

CArtAgO (Ricci et al., 2007) is another general framework for creating and deploying virtual environments for multi-agent systems. The framework is based on the Agents & Artifacts (A&A) meta-model for modeling and designing multi-agent systems. The Agents & Artifacts model uses concepts drawn from human working environments. Agents perform goal-oriented activities. In assistance to agents, there are artifacts the agents use as resources in order to construct tools they use to achieve their goals.

Further worth mentioning methods for modeling multi-agent systems include OMNI (Dignum et al., 2004), E-Institutions (Esteva et al., 2001), GaiaExOA (Zambonelli et al., 2003) and Tropos (Kolp et al., 2006).

The majority of these agent oriented programming frameworks have been based on declarative features. They provide well defined semantics and support powerful knowledge representation techniques. Their primitive component is reactive rules. Programs are defined as unstructured sets of reactive rules. Nevertheless, the absence of program structure, leads to disadvantages such as difficulties in code re-use, one of the most important features of modern programming languages.

2.5 Agent Communication Language

Agent communication, also known as the agent-based messaging concept (Finin et al., 2000), provides a universal messaging language with a consistent speech-act-based, uniform messaging interface for exchanging information, statically or dynamically, among software entities. Agent communication has the following advantages over the traditional client - server (RPC) based communication:

- Communication takes the form of de-centralized, peer-to-peer message exchange sequence, as opposed to the traditional client-server roles.
- While there can be synchronous message exchange, communicating agents derive great functionality from the fact that they are able to exchange messages asynchronously.
- There can be a universal message based language with speech-act-based interface.
- There is a single method invocation for all types of message exchanges.

Agent communication through message exchange works with at least two participating parties. There is the sending agent that generates the information and

transmits it and at least one receiving agent that receives the message and uses the information contained. The information that is exchanged between the communicating parties is usually formally coded into a universally understood Agent Communication Language (ACL). Communication is established when the sending agent wishes to convey information relevant to its own mental state or view. In such case the sending agent produces a relevant string containing the aforementioned data.

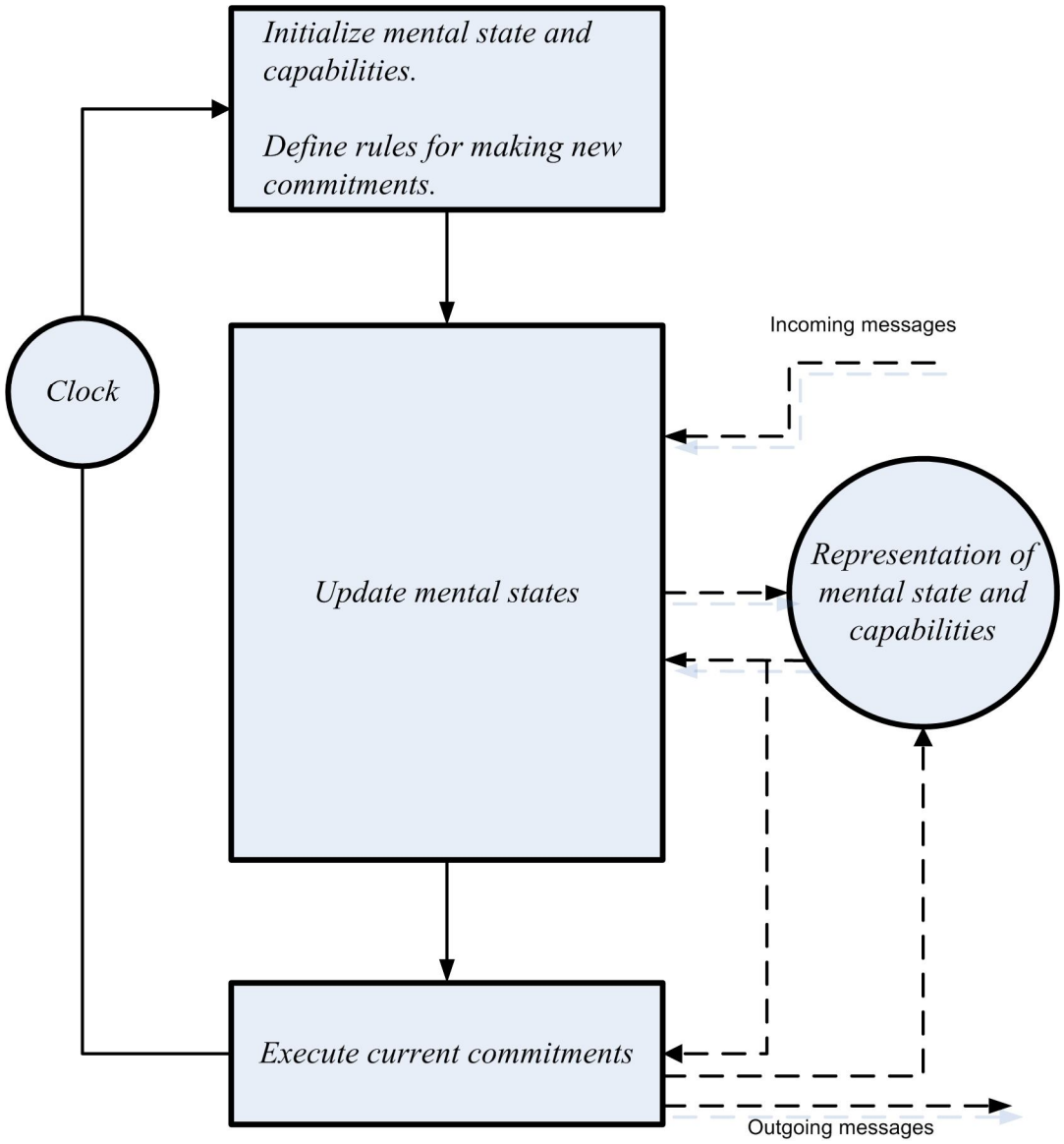


Figure 2.2: Agent-0 flow diagram (Shoham, 1993)

Thereafter and according to the FIPA framework, this ACL coded message string is passed to potential recipients after invoking the appropriate remote method. That means that the sending agent invokes the message method of the recipient agent(s) and passes the string through it. This procedure is carried out through agent proxies. It is the agency's (agent platform) obligation to provide appropriate listing of such proxies, which are also followed by a service that helps sending agents to locate the recipient agent(s) proxies. Though such a procedure may seem complex in static agent societies, it is critical in mobile agent systems. The receiving agent, on receiving this

message, decodes the information and then performs the necessary actions. In case of a bi-directional communication, the recipient agent(s) may communicate the result back to the sender by using the same process.

2.6 The Agent Communication Model

For Agent Communication Languages, a common valid model (Agent Communication Model), should be constructed considering certain requirements. An ACL must be declarative, syntactically simple, and readable by people (Cost et al., 2001). Its form must also be concise, easy to parse and generate. The semantics should be based on solid theoretical grounds and should be unambiguous. There are also the concepts of ontology, content language, and the actual agent communication language.

Ontology defines and enumerates the terms of a particular application domain. The content language is used to combine ontology terms into meaningful. It is solely represents an agent's mental state. This is often perceived as the agent's view of the world. Ontology and content language are often perceived as an agent's representation. In a particular domain, however, where agents exist and strive for their goals there is the need of a communication language in order to model any existing knowledge or the dissemination thereof.

The agent communication language acts as a medium for exchanging dialogs among agents, containing sentences of the content language and the ontology of any given application's particular domain. Moreover, the agent communication language should be unaware of the content language of choice and the inherent ontology, acting as a wrapper of the information and knowledge flow. It should provide the outer encoding layer, which determines the type of agent interaction, identifies the network protocol with which to deliver the message. It should also supply the speech act which can be communicative or performative.

ACLs range from some form of primitive communication to elaborated standards. Two of the most widely used ACLs are KQML (Knowledge Query Manipulation Language), and FIPA ACL (Mayfield et al., 1995), (FIPA, 1998a). Knowledge interchange format (KIF) is often used as a content language with KQML. Likewise, semantic languages (SL) are often used to represent the application domain, even though the FIPA ACL specification document does not make any commitment to a particular content language.

2.7 Multi-agent Systems (MAS)

The most commonly used terms that pertain to the agent-oriented model are agents and multi-agent systems (MAS). Agents can be defined as autonomous, problem-solving computational entities perceiving their environment through sensors and acting upon it. They are also capable of operating in dynamic and open environments (Singh & Huhns, 2005; Wooldridge, 2002). However, they could be constructed using the foundations of object-oriented technology. In addition to being autonomous an agent should be also capable of acting based on its beliefs or goals and interacting with similar entities, forming agent societies. Therefore, agents should be operating in

suitable environments which can facilitate interaction and cooperation with other agents. Such environments are known as multi-agent systems.

Multi-agent systems (Wooldridge M., 2002, Biswas P. and Hong L. (ed), 2007) are composed of a set of agents, often sharing common goals. They are useful for modeling and developing distributed software applications with synchronous or asynchronous component interactions. Multi-agent systems differ from non-agent based systems because agents are intended to be autonomous software units. They frequently use intelligent functionalities and they interact through high-level protocols and languages.

MAS are based on the idea of cooperative working environments. In Computer Science, cooperative working environments are viewed as groups of interacting software components bearing attributes that assist them to deal with computational problems in which centralized approaches find hard to solve (Jennings, 2000). MAS follow a distributed model that contains a community of social software components, which can act on behalf of their owners (Wooldridge, 2000).

Multi-agent systems have been developed in order to assist a variety of application domains, including electronic commerce, air traffic control, workflow management, transportation systems, and Web applications, among others. In such areas of expertise multi-agent systems aim to provide solutions for highly dynamic issues in the construction part of complex systems, as well as mechanisms for coordinating any group of independent agents' behaviors. Multi-agent systems support the scenario that one problem is divided into multiple sub-problems. These sub-problems may be further broken down to the level which can be handled and solved by one or more agents after they adjust their interests and goals accordingly. Moreover, the next-generation agent computing concept calls for the development of multi-agent systems, possessing the key characteristics of self-managing systems, namely self-healing, self-protecting, self-adapting, and self-optimizing (Kephart & Chess, 2003).

Multi-agent systems offer certain advantages over any alternatives that may rival their effectiveness. First of all, there are problems that are distributed in nature. Thus, they can be solved by a framework that fundamentally reflects their structure. Since multiple agents can work in parallel, multi-agent systems can gain performance improvement from parallelizing the work of different agents. As there is no central control in multi-agent systems and tasks are executed by multiple agents simultaneously, there is no single critical point of failure in such systems. Furthermore, multi-agent systems are modular which means that it is easy to scale the any multi-agent system by including more agents.

2.8 Agent Organizations

Multi-Agent Systems (MAS) have been defined as organizations or societies of agents, that interact together to coordinate their behavior and cooperate to achieve common goals. Organization in MAS could be viewed as the process of organizing a set of agents. However, organizations in MAS are entities with requirements and objectives.

In a traditional multi-agent system, agents have limited resources, there is no centralized control and applications are asynchronous. Furthermore, social issues, as behavior, reactiveness and cooperation become critical facts while building agents. On the other hand agent organizations (Figure 2.3) are entities where agents interact within a structured environment in order to achieve a global purpose. Organizations in MAS comprise agents that act bearing in mind concepts such as roles, groups, tasks and interaction protocols. The primary objective of every agent within an organization is to play effectively any role that will contribute at achieving the organization's goals.

In agent organizations, agents are active, communicative entities that play roles within groups. An agent can have multiple roles and be a member of many groups. A role is an abstract representation of the functional characteristics an agent has in a group. Groups finally, are sets of agents sharing common characteristics.

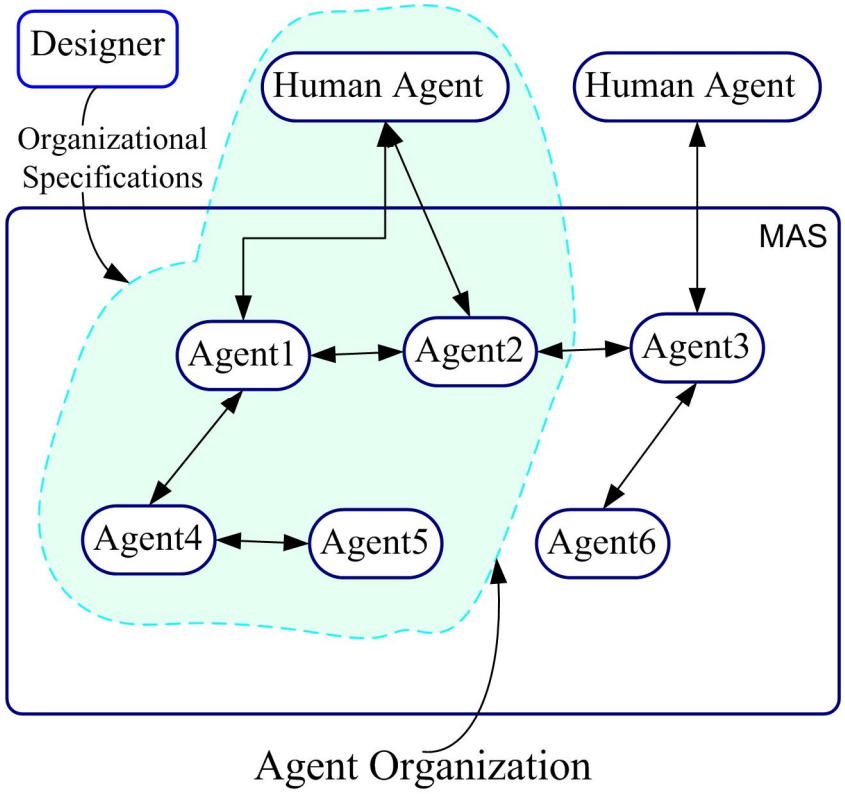


Figure 2.3: An Agent Organization within the scope of a multi-agent system.

2.9 The Framework behind Multi-Agent Systems

All theoretical concepts considered there is still the requirement of efficient implementation of a multi-agent system. Primarily, there is the need of an intermediate layer of abstraction. This layer must include all fundamental components capable of providing all necessary means to manage the primitive resources of an agent. The development of a multi-agent system, as in all agent-based applications, would be highly simplified should the underlying infrastructure supports the basic attributes of an agent. As stated, these are the agent's identity, autonomy, co-existence, communication, mobility if any, security, and lifecycle management.

A straightforward requirement for such an infrastructure is the necessity to provide the runtime environment for the domain-dependent roles of the agents. Wooldridge and Jennings define agents by their characteristics. Any software entity that can display autonomy, social ability, reactivity and pro-activity, is an agent in weak notion. Moreover, if further characteristics such as adaptability, mobility, veracity, and rationality are present then our software entity is an agent in strong notion. Any infrastructure that aims to facilitate software agents must provide the aforementioned traits.

The architectural prototypes for agent-based applications involve a combination of functional and quality requirements that outline their behavior. These form a set of core requirements for agent-based systems. Most MAS frameworks meet some or all of the following functional and quality requirements.

2.9.1 Functional Requirements

- Autonomy: Agent-based systems should be automated and self-managed without any form of outside intervention.
- Interoperability: Agent-based applications must be able to interoperate with each other. This must hold for application running either on the same or on a different platform. Agents should be able to communicate with other agents even if they happen to be part of different organizations or live on different environments. Another critical issue is migration which should be feasible when necessary.
- Reusability: Agent-based systems should be developed in a way that allows agent reconfiguration and use in applications that have different requirements and serve diverse purposes.
- Cooperative task processing: Agents must cooperate and collaborate with each other to solve common problems or obtain common goals.
- Agent communication: Agents can interact with other agents. A sound agent-based systems should provide this capability
- Agent negotiation. In extend to agent communication, agent negotiation is a process by which a number of agents can reach a mutually acceptable agreement.
- Agent coordination. Manages interactions and dependencies among agent activities. It is the formal framework where under which meaningful interactions between software agents can take place.
- Survivability: In their majority large-scale systems consist of many components, each of which has certain degree of reliability. Therefore, such systems have to detect and deal with the possibility of component failures. Agent-based systems must provide means of avoiding serious impacts from situations such as agent failure.
- Scalability: Agents are autonomous. Therefore, cooperation among them must be independent of the number of individual agents that participate in any joint effort.

2.9.2 Quality requirements

- Usability: Agent-based systems interface options that the agents will find easy to comprehend and train themselves to use.
- Extendibility: Agents are quite often programmed in order to demonstrate different skills or capabilities. An agent-based system must posses the technology to facilitate any traits that an agent is supposed to posses.
- Modifiability: Agent-based systems should allow agents to be modifiable without changing the existing functionality of the system.
- Reliability: Agent-based systems should be reliable and fail proof as much as possible. Even in cases of total system failure it is very critical for the underlying infrastructure to provide methods to minimize data loss and provide necessary functionalities for sufficient system recovery.
- Adaptability: Agents may be programmed to carry different task-specific processes, which may extend the functionality the system has so far.
- Security: Agent-based systems should prevent unauthorized data access from untrusted entities.
- Performance: Agent-based systems should accomplish their tasks in reasonable time and in the least disruptive manner.

2.10 Self-* properties of Multi-Agent Systems

The term self* properties for multi-agent systems integrates concepts such as self-configuration, self-localization, self-optimization, self-stabilization, self-adaptation and self-organization. The possibilities of finding new concepts under the self-* prototype are actually very broad. Here, we briefly explain the most common properties, found in the related literature:

- Self-configuration means that a given system is able to be aware of the arrangement of its parts by itself at any given moment.
- Self-localization is the concept that distributed nodes can find out, without external help, their relative position regarding to their neighbor's position.
- Self-optimization indicates that a distributed system can automatically monitor its resources and control their availability following certain predefined or dynamic plans.
- Self-stabilization is the idea that a system can follow certain procedures to recover from an undesirable state within a finite period, without external help.
- Self-adaptation is the concept that systems should be able to be aware of any conditions that could generate problems in reaching their goals. Moreover, they should be able to either avoid these problems or diagnose possible occurrences and make the necessary changes to prevent undesirable effect.
- Last, we have to mention self-organization; an aspect which this thesis has tackled as far as large-scale agent systems are concerned. Self-organization is a great way for building scalable systems, where the main concerns are coordination and collaboration towards a set of goals. Self-organization in such environments involves the interaction and co-operation of all components (homogeneous or heterogeneous) that comprise a unified system towards specific objectives. Self-organization is especially important in many today's

applications such as ad hoc networks. In ad-hoc networks where many heterogeneous components have to work together, a system's ability to manage itself without external intervention is critical for its overall performance.

2.11 Overlay Networks and Multi-Agent Systems

An overlay network is a virtual network of nodes and logical links between them. Most commonly, an overlay network would be a subset of the nodes of one large network that have agreed to be mutually accessible in order to achieve common goals. Overlay networks are literally built on top of existing networks in order to add another layer of services.

In (Harchol-Balter et al., 1999) the authors introduced the Resource Discovery Problem. Their aim was to address the case that in large distributed networks of computers, there is high probability that a subset of machines face the need to cooperate in order to perform a common task or achieve a common goal. In (Harchol-Balter et al., 1999) at the beginning, there is a weakly connected knowledge graph of nodes or processes. Prior to commencing collaboration, the nodes first discover the identities of all the other nodes. The goal is to construct the complete knowledge graph from an arbitrary weakly connected graph. The Name-Dropper algorithm introduced, achieves that goal and according to the authors, its performance is near optimal with respect to time complexity and network communication.

In (Garcia-Molina and Crespo, 2003) the authors propose that overlay networks should be form by taking into account semantics. Thus, the efforts concentrate on how to discover semantically similar nodes and use them to form a cluster. For deciding which nodes are semantically similar, all nodes store documents with classification hierarchies. Classifications techniques include text matching, Bayesian networks and clustering algorithms.

The work in (Angluin et al., 2005) addresses overlay network formation by assuming that communication between nodes primarily relies on network address discovery. Any node can send messages to any other node once it knows the other node's address. Again, the initial state consists of a number of nodes organized into a weakly connected graph. The algorithm re-organizes the initial graph as a low-depth binary search tree, using randomized pairing and distributed merging.

Management Overlay Network (MONs) aim to help management issues in large distributed applications. In (Liang et al., 2005) MONs are instances built on-demand and involve overlay structures that allow users to execute management commands, such as application status querying. In order to form overlay networks the authors follow two approaches. The first uses random trees. To form a random tree each node selects its children randomly by taking into account its own view of nearby nodes. The node then sends messages to the nodes it has chosen. If a node receives such a message for the first time, agrees to become a child and refuses further similar requests until released. The second approach involves Directed Acyclic Graphs. There nodes are divided to levels and nodes are picked from each level with specific attention given to avoid cycles.

2.12 Coordination and Searching in Large-Scale Multi-Agent Systems

Coordination in distributed systems is generally considered a very complex matter. In one of the early approaches (Findler and Elder, 1995) the authors present the question: *"How can a group of geographically distributed agents properly allocate a set of tasks among themselves while satisfying different types of constraints? "* Their solution was to handle the problem by extending the contract net protocol. Agents evaluate themselves and exchange their evaluations. The agent that seems to have the best capabilities is assigned to perform the associated tasks. This method allows the agents to act as a group and determine task assignments, without external intervention.

In a large-scale system with decentralized control it is very hard for agents to possess accurate partial views of the environment, and it is even harder for agents to possess a global view of the environment. Furthermore, the agents' observations can not be independent, as one agent's actions can affect the observations of the others: for instance, when one agent leaves the system, then this may affect those agents that are waiting for a response; or when an agent schedules a task, then this must be made known to its team-mates who need to schedule temporally dependent tasks appropriately.

Even if agents' activities are independent, (i.e. the state of one agent does not depend on the activities of the other) (Goldman and Zilberstein, 2004) the overall complexity is exceedingly higher compared to centralized systems. Moreover, decentralized control of cooperative agent systems in cases where agents have to act jointly adds an even higher amount of complexity. In the case of joint activity, subsets of agents have to form teams in order to perform tasks, subject to ordering constraints. Acting as a team, each group has to be coordinated by scheduling subsidiary tasks with respect to temporal and possibly other constrains, as well as other tasks that team members aim to perform (e.g. as members of other teams).

The control process can be modeled as a decentralized partially-observable Markov decision process (Goldman and Zilberstein, 2004). The computation of an optimal control policy in this case is simple given that global states can be factored, the probability of transitions and observations are independent, the observations combined determine the global state of the system and the reward function can be easily defined as the sum of local reward functions.

Decentralized control in systems where agents have to act jointly in the presence of temporal constraints, is a challenge. This challenge has been recognized in (Koes et al., 2005), where authors propose an anytime algorithm for centralized coordination of heterogeneous robots with spatial and temporal constraints, integrating task assignment, scheduling and path planning. In addition to providing a centralized solution, they do not deal with ordering constraints between tasks. Although the authors demonstrate that their method can find schedules for up to 20 robots within seconds, it is unclear how it would perform in networks with hundreds of agents. On the other hand, it has to be pointed that their algorithm, being efficient for small groups and providing error bounds, may be combined with our approach, given small

groups of potential teammates. However, this is an issue for further research since the inclusion of ordering constraints will affect the solution complexity.

Extreme teams is a term coined in (Scerri et al., 2005), emphasizing on four key constraints of task allocation: (a) domain dynamics may cause tasks to appear and disappear, (b) agents may perform multiple tasks within resource limits, (c) many agents have overlapping functionality to perform each task, but with differing levels of capability, and (d) inter-task constraints may be present.

Token-based approaches are promising for scaling coordination to large-scale systems effectively. In (Moyaux et al., 2003) the authors idea is that a supply chain can be view as a multi-agent system. Moreover, they use tokens for coordinating the multi-agent system that controls the procedure of order placement in supply chains. Their concept is built around the notion that tokens can be deployed in a lower level coordination mechanisms based on communication.

The algorithm proposed in (Xu et al., 2005) focuses on routing models and builds on individual token-based algorithms. Additionally, in (Xu et al., 2005) authors provide a mathematical framework for routing tokens, providing a three-level approximation to solving the original problem. Token-based approaches do not inherently deal with scheduling constraints and dynamic settings.

The method presented in (Vouros, 2007) performs information searching and sharing in dynamic and large scale networks using a combination of routing indices with token-based methods. The approach in this paper involves 'tuning' a network of agents where each has a specific expertise. 'Tuning' refers to the process of sharing and gathering knowledge so that agents can propagate requests to the appropriate acquaintances. This concept allows efficient and effective information searching and sharing, without altering the agent networks topology. Furthermore, it reduces substantially the searching effort, thus making the system become more efficient.

(Vouros, 2008) also deals with searching and sharing in dynamic and large scale networks extending the work in (Vouros, 2007). Specifically, the paper studies the decentralized control problem, information searching and sharing in large-scale systems of cooperative heterogeneous agents. Furthermore, the author incorporates logical shortcuts as content providers and recommendation links, which is a way of entailing overlay networks in the presented method. The paper also integrates the idea of agents shifting expertise as knowledge can be not only from local sources but from content providers and recommenders as well.

Finally, the work that closer than any other to this thesis is (Theocharopoulou et al., 2007). In (Theocharopoulou et al., 2007)], a method is proposed that allocates temporally interdependent tasks to homogeneous or heterogeneous cooperative agents in dynamic large-scale networks. In this paper searching, task allocation and scheduling are viewed as an integrated problem. The presented method carries out searching through the dynamic assignment of gateway roles to agents. Gateway agents maintain routing indices for finding potential teammates on the process of forming overlay networks. This scheme of dynamically re-organizing agents combined with searching, uses distributed constraint satisfaction techniques and handle the efficient allocation of complex tasks with interdependent subtasks.

CHAPTER 3

3. Constraint Programming

3.1 Constraint Satisfaction

Artificial Intelligence (AI) is primarily focused on solving real life problems. Such problems are prone to common restrictions. Such restrictions may pose as limited resources, temporal constraints, finite variable domains or even hierarchical barriers. Solving problems of that category inevitably leads to searches through possible solutions. Furthermore, this process inevitably leads to having to take into account a huge volume of possibilities. Constraint Satisfaction, a fast paced evolving research area within the auspices of Artificial Intelligence, has largely addressed the majority of such issues. Over the years Constraint Satisfaction theory and applications have dealt with the majority of known problems under specific constraints.

The essence of constraint problems was introduced in AI as early as in the seventies (Apt, 2003). It was then when today's well-known terms like local consistency, back-tracking and branch and bound were defined or introduced. Later in the eighties, Constraint Programming (CP) became an extension of logic programming known as constraint logic programming. The nineties were the breakthrough decade for constraint programming. New problems and new types of constraints were identified. This led to new algorithms and experimentation with solving techniques coming from diverse research areas such as operations research and mathematical logic. Thus, constraint programming became a rapidly evolving research area experiencing substantial growth.

In Artificial Intelligence community a widely accepted standard definition is used for defining search problems. These problems are commonly called Constraint Satisfaction Problems (CSPs) an area within CP's research interests. An introduction to CSP and techniques can be found in (Dechter and Pearl, 1985, Marriott and Stuckey, 1998, Barták(i), 1999, Barták(ii), 1999). Moreover, comprehensive details for CSPs are given in (Apt, 2003, Fruhwirth and Abdennadher, 2003, Rossi et al., 2006).

Formally a CSP is defined by the triple $P = \{X,D,C\}$ (Rossi et al., 2006). In P, X is an n-tuple of variables and D is an n-tuple of domains. X can be of the form

$$X = \{X_1, X_2, X_3, \ldots, X_n\}.$$

Each variable in X has a finite domain of values. For instance for X_m there is a domain D_m of the following structure:

$$D_m = \{D_{m1}, D_{m2}, D_{m3}, \ldots, D_{mk}\}.$$

D comprises domain sets from each variable. Hence:

$$D = \{D_1, D_2, D_3, \ldots, D_n\}.$$

C is an p-tuple of constraints of the form:

$$C = \{C_1, C_2, C_3, \ldots, C_p\}.$$

Each constraint in C is of the form:

$$C_m = \{Rs_m, S_m\}.$$

Rs_m is a relation between two or more variables in S_m = scope(C_m). If we consider the variable domains, a relation Rs_m is a subset of the Cartesian Product of the domains of the variables in S_m (Example 3.1). S_m by itself is a subset of X and denotes the variables involved in the constraint. A relation can be defined either explicitly by giving its subset or implicitly through a mathematical operator, a predicate, or a function. For the CSPs addressed in this thesis it suffices to describe relations implicitly as operations belonging to the set $\{\geq,\leq,>,<,=,\neq\}$.Typically, there are one or two variables in S_m denoting a unary or a binary constraint, respectively.

A solution to P is a n-tuple A where:

$$A = \{A_1, A_2, A_3, \ldots, A_n\}$$

with A_m and D_m. Alternatively we can say that A renders each C_m in C satisfiable.

Example 3.1. A possible CSP instance of the form $P = \{X,D,\}$ can have the following description:

$$X_= \{X_1, X_2, X_3\} \text{ and } D_= \{D_1, D_2, D_3\}$$

The domain set consists of the following individual variable domains:

$$D_1 = \{0,1,3\},\ D_2 = \{0,1,2,4\},\ D_3 = \{1,3\}$$

We can also assume $C = \{C_1, C_2, C_3\}$ where $C_1 = \{Rs_1, S_1\}$, $C_2 = \{Rs_2, S_2\}$, $C_3 = \{Rs_3, S_3\}$. The relations in each constraint are:

$$Rs_1 = \{\leq\},\ Rs_2 = \{\geq\},\ Rs_3 = \{>2\}$$

As seen from the scopes in each constraint C_1 and C_2 are binary constraints and C_3 is unary:

$$S_{1=} \{X_1, X_2\},\ S_2 = \{X_2, X_3\},\ S_3 = \{X_3\}$$

Therefore, the derived constraints can be represented as:

$$X_1 \leq X_2, X_2 \geq X_3 \text{ and } X_3 > 2$$

For the variables X_1 and X_2 the Cartesian Product of the values found in their domains is:

$$X_1 \times X_2 = \{(0,0), (0,1), (0,2), (0,4), (1,0), (1,1), (1,2), (1,4), (3,0), (3,1), (3,2), (3,4)\}$$

After applying the constraint between X_1 and X_2 we end up with the subset Rs_1 :

$$Rs_1 = \{(0,0), (0,1), (0,2), (0,4), (1,1), (1,2), (1,4), (3,4)\}$$

Following the same procedure:

$Rs_2 = \{(1,1), (2,1), (4,1), (4,3)\}$ and $Rs_3 = \{(3)\}$

A solution to the CSP in Example 3.1 is:

$A = \{1,4,3\}$ where $X_1 = 1$, $X_2 = 4$, $X_3 = 3$.

As stated solving a CSP means an extensive search through all possible assignments until a solution is found. Existing search methods are divided into two broad categories. At first, let us assume that Ω is the combination of all possible assignments in D. The categories under reference are *Inference* or *Deduction* and *Search*. Specifically:

- Inference or deduction. Inference methods aim to exclude large subsets of Ω estimating that there is no solution therein.
- Search. These methods search Ω systematically, eliminating subsets of Ω after failures in finding a solution.

3.1.1 Inference Methods

Inference methods try to eliminate subsets of the initial search space Ω to Ω' so that $sol(\Omega) = sol(\Omega')$. Generally, inference methods create equivalent problems using reformulation. This process may add new variables, create a certain variable ordering or add new constraints, but it will ultimately reduce the initial search space. Thus, a simpler problem is created which can usually be solved using simple search algorithms. A straightforward advantage of this category is that by producing a smaller search space, inference methods tend to be faster.

Example 3.2. Let us consider a simple CSP *(P)* with three variables, namely X_1, X_2, X_3. Thus:

$$X = \{X_1, X_2, X_3\}$$

Suppose that all three variables have similar domain values:

$Dx_1 = Dx_2 = Dx_3 = D$ and $D = \{0,1,2,3\}$

Finally, the set of constraints is

$$C = \{(X_1 = X_2), (X_2 = X_3), (X_1 \neq X_3)\}$$

From the constraint set it can be inferred that $X_1 = X_3$. Hence, C van be transformed to:

$$C' = C = \{(X_1 = X_2), (X_2 = X_3), (X_1 = X_3), (X_1 \neq X_3)\}$$

This constraint set clearly makes the problem inconsistent, since the search space is now equivalent to the empty set, and there is no need to search between value assignments. Let us also consider another constraint set:

$$C_1 = \{(X_1 = X_2), (X_2 = X_3)\}$$

By inferring the same extra constraint, we now have

$$C_1' = C = \{(X_1 = X_2), (X_2 = X_3), (X_1 = X_3)\}$$

Search space now includes only sets of identical values for each variable ((0,0,0), (1,1,1) etc.). All constraint sets of Example 3.2 are depicted in Figure 3.1. As mentioned our search space is again diminished and all there is to be done is assign the same value to all variables and check if all constraints are satisfied. If no assignment of the new search space is consistent then the problem cannot be solved. Note that in both examples $sol(\Omega) = sol(\Omega')$.

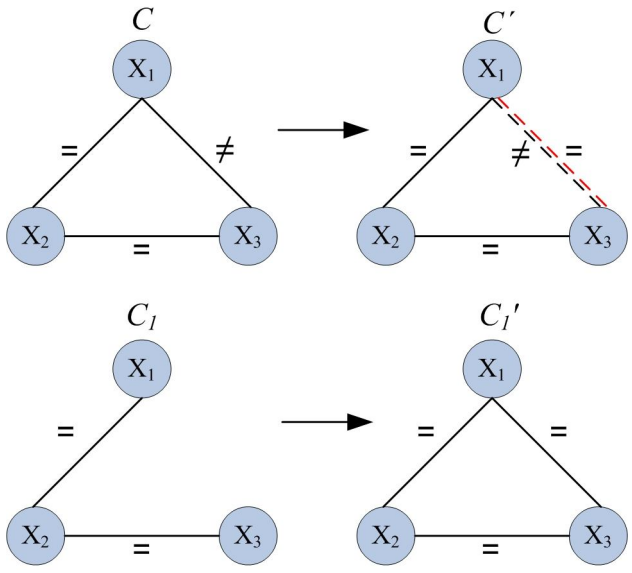


Figure 3.1: Constraint sets of Example 3.2 before and after constraint inference and transformation.

Variable ordering is a basic and very important inference technique (Golomb and Baumert, 1965, Bacchus and van Run, 1995, Gent et al., 1996, Bain et al., 2005, Katsirelos and Bacchus, 2005). A good variable order is critical for solving CSPs. In case we can identify the most difficult part of the problem and solved it first, further search is substantially reduced. Therefore, the ordering in which variables are

instantiated can play a decisive role on the performance of a search. Variable ordering differentiates itself from the traditional sequential variable instantiation, by attempting to explore the search space in a clever way. Such attempts follow standard methodology for each variation of the method. The ordering may be either static or dynamic. For static ordering, the order of variables instantiation is specified before the search begins, and remains unaltered thereafter. In dynamic ordering, the choice of next variable depends on the current state of the search.

A simple example of variable ordering can be found in (Golomb and Baumert, 1965), where the variable with the fewest values in its domain is the next variable to be chosen. Another, more sophisticated example can be found in (Bacchus and van Run, 1995) where the MVR heuristic is applied to a standard search tree and chooses good variable orderings by instantiating next the variable that has the fewest consistent values with the previous instantiation. Thus, by constantly choosing the variables that are most likely to be inconsistent, the MVR heuristic tries to detect and solve difficult the parts of the problem first. This way there is a gradual reduction on the problem's complexity parameters.

Another inference approach in solving CSPs is value ordering (Haralick and Elliott, 1979, Dechter and Pearl, 1987, Vernooy and Havens, 1999, Kask et al., 2004). This method category studies the effect that the order in which values are applied to variables can have on CSPs. In an early study in (Haralick and Elliott, 1979) the proposed value ordering technique was merely choosing the least constraining value as the next value. In (Dechter and Pearl, 1987) a tree relaxation for the CSP problem is created. The result is a tree structured CSP, where the procedure of counting all solutions is polynomial. The tree relaxation technique though only gives us an approximation of the initial problem. Therefore, the computed solutions end up being potential solutions to the initial CSP. For this reason the probability of each one as a real solution is computed and they are checked accordingly. Later works (Frost and Dechter, 1995, Geelen, 1992, Vernooy and Havens, 1999, Kask et al., 2004) provide more sophisticated and competent inference techniques as they further advance the efficiency of the value ordering heuristic.

Perhaps, the most fundamental part of any complete study on CSPs is the one referring to consistency algorithms. Works in this particular area can be traced back in (Waltz, 1972, Gaschnig, 1974, Montanari, 1974). Nevertheless, the groundbreaking paper came from Alan Mackworth in 1977, a work which is one of the most cited papers in Artificial Intelligence (Mackworth, 1977a). Further improvements on non-binary constraints (constraints involving more than two variables) and complexity issues came in (Mackworth, 1977b, Mackworth and Freuder, 1985), as well as in (Mackworth et al., 1985).

Consistency inference algorithms rely heavily on pre-processing the variable domains in order to propagate the repercussions a constraint imposes on a variable to other variables. More specifically, arc consistency ensures that all values in a variable domain are consistent with every constraint before proceeding to the next variable.

Example 3.3. Let *P* be a CSP involving three variables:

$$X = \{X_1, X_2, X_3\}$$

If all three variables have similar domain values, then:

$$Dx_1 = Dx_2 = Dx_3 = D \text{ and } D = \{0, 1, 2, 3\}$$

We also have the following constraint set:

$$C = \{(X_1 < X_2), (X_2 < X_3)\}$$

After checking the first constraint we have to eliminate $X_2 = 0$, as it would never be consistent in $X_1 < X_2$. For $X_1 = 0$ and in order to satisfy the second constraint in C, $X_3 = 0$ and $X_3 = 1$ have to be eliminated as well. The same holds for $X_2 = 3$. We cannot also allow the assignments $X_1 = 2$ and $X_1 = 3$ as with these values for X_1 there is no consistent set of values for *P*. Therefore, our initial search space has been pruned to the Cartesian Product of the following domain sets (Figure 3.2):

$$Dx_1 = \{0, 1\}, Dx_2 = \{1, 2\}, Dx_3 = \{2, 3\}$$

Arc consistency as defined by Mackworth (Mackworth, 1977a), describes consistency in a network of variable domains and constraints (constraint network), in the following way:

Let $Arc(X_i, X_j)$ be an arc representing a constraint between variables X_i and X_j. A constraint between X_i and X_j would be represented as $C_{ij} = \{R_{ij}(X_i, X_j), S_{ij}\}$. Dx_i and Dx_j are the variable domains of X_i and X_j respectively.

A value $\alpha \in Dx_i$ is arc-consistent relative to X_j iff there is a value β (Dxj such that for Xi = α and Xj = β, Rij(Xi, Xj) holds in its scope (Sij).

Accordingly, a variable Xi is arc-consistent relative to Xj iff all values in Dxi are arc-consistent. In this case Arc(Xi, Xj) is also arc-consistent. A CSP is arc-consistent iff all its variables are arc-consistent.

The notion of consistency can be extended so that all constraints related to variables X_i and X_j are satisfied. This concept is known as 2-consistency. A value $\alpha \in Dx_i$ is 2-consistent relative to X_j iff there is a value $\beta \in Dx_j$ such that for $X_i = \alpha$ and $X_j = \beta$, $\forall k$ $R_{ij}^{\,k}(Xi, Xj)$ holds in its scope $(S_{ij}^{\,k})$. Similarly, a variable X_i is 2-consistent relative to X_j iff all values in Dx_i are 2-consistent. Finally a CSP is 2-consistent iff all its variables are 2-consistent.

There is often confusion between arc-consistency and 2-consistency when it comes to normalized CSPs. In normalized CSPs no two different constraints involve exactly the same variables. In binary (constraints involving two variables) normalized CSPs arc-consistency and 2-consistency can be easily confused.

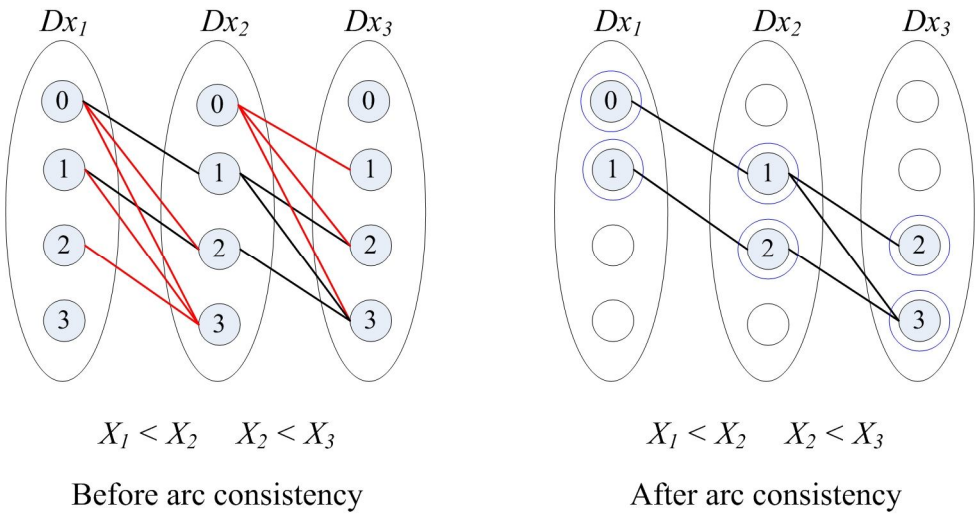


Figure 3.2: Variable domains, constraints and valid variable values for Example 3.3 before and after arc consistency. Arcs depicting constraints have been removed for clarification reasons.

Another consistency concept, that of 3-consistency or path consistency is attributed to Montanari (Montanari, 1974). A CSP is path-consistent if the consistency of any pair of variables, between which there is a constraint, can always be extended to a third variable, bearing a constraint with one of the two initial variables.

Following Mackworth and Montanari, Freuder generalized arc-consistency and defined k-consistency (Freuder, 1978). A CSP is k-consistent if for any k-1 consistent variables there is at least one value in every k^{th} variable, so that all k variables are consistent.

There are several algorithms that were proposed for enforcing arc-consistency. AC1, AC2 and AC3 (Mackworth, 1977a) are such algorithms. In (Mohr and Henderson, 1986) another algorithm that tries to enforce arc-consistency was introduced, namely AC4. In addition, research efforts on algorithms that try to impose arc-consistency, have over the years proposed numerous alternatives. Between those algorithms, worth mentioning are: AC5 (Perlin, 1992), AC6 (Bessière and Cordier, 1993), AC7 (Bessière et al., 1999) and AC8 (Chmeiss and Jégou, 1998).

3.1.2 Search Methods

Search methods in CSPs are designed to look through the search space meticulously, guessing the next move based on heuristics. The core concept starts with an arbitrary variable instantiation which can be easily proved consistent or otherwise. Afterwards, the process continues with major or minor adjustments on the initial instantiation, trying to find consistent values for all the variables in the CSP. In case the selected operation is not a step closer to the solution, it can be retracted or another operation selection mechanism can be used. The algorithm that most certainly describes search methods in CSP is backtracking.

Search methods usually employ randomization techniques so as to avoid dead-ends or stand still nodes during the solution process. Such a subcategory, often referred to as one of the most promising ones, in search methods for CSPs is stochastic local search

(SLS) (Hoos and Stutzle, 2004). Well known SLS algorithms are: simulated annealing, stochastic hill-climbing, tabu search, evolutionary algorithms, ant colony optimization and dynamic local search. Local search algorithms are usually easier to implement than inference algorithms.

Backtracking is a systematic search method for solving CSPs. Its solution procedure is based on depth first traversals of search trees. Backtracking is a complete algorithm (guarantees to find a solution if at least one exists, or prove the problem unsolvable). In general, backtracking chooses values for one variable at a time. If no next variable has a consistent value then the algorithm undoes the current assignment of the last variable and starts again. The search tree is formed gradually as the algorithm proceeds, and each node is added based on various techniques. These are called branching strategies (enumeration, binary choice points and domain splitting being the most popular). After adding a node certain parts of the search space can be subsequently eliminated. If a node cannot lead to a solution it is called dead-end node. The foundations of backtracking can be found in (Davis et al., 1962) and (Golomb and Baumert, 1965). The DPLL algorithm, a backtracking algorithm for SAT problems, is presented in (Davis and Putnam, 1960, Davis et al., 1962).

In (Golomb and Baumert, 1965) the FC (Forward Checking) algorithm is proposed, an improvement to simple backtracking. FC maintains arc consistency on constraints with one uninstantiated variable. A variation of FC algorithm is the MAC algorithm. MAC was introduced in (Sabin and Freuder, 1994) and it maintains arc consistency on constraints with at least one uninstantiated variable. FC is extended in (Van Hentenryck, 1989) for non-binary constraints. Gaschnig in (Gaschnig, 1978) presented the BJ (backjumping) algorithm. This algorithm abstains from backtracking to the last consistent value in the search tree. In this way it avoids producing already discovered dead-ends. Supplementary research works for backtracking are: CBJ (Prosser, 1993), DBT (Ginsberg, 1993), FC-DBT and MAC-DBT (Jussien et al., 2000) as well as PBT (Ginsberg and McAllester, 1994).

Hill-climbing search comprises yet another known family of local search algorithms. It is sometimes referred to as iterative improvement or iterative search (Yokoo, 1997). Generally algorithms of this class try to improve their current state step by step. Hill-climbing algorithms commit variable value assignments and then use an evaluation method for appraising the states that are can be reached from the current state. Based on this appraisal the algorithm proceeds to assume the best state. A disadvantage for hill-climbing methods is their susceptibility to be trapped in a local minimum (all neighboring states are worse than the current state).

Arguably, the most widely used hill-climbing algorithm is the Min-Conflicts Heuristic (Minton et al., 1990, Minton et al., 1994). It starts with random variable assignments and tries to minimize inconsistencies by assigning new values to randomly selected inconsistent variables. The chosen value is the one that minimizes conflicts. If there is more than one such variable, choice is random between this set. In case of a local minimum entrapment the algorithm restarts itself with a new variable assignment.

Another important member in the hill-climbing family is tabu search (Glover, 1996, Glover and Laguna, 1997). Tabu search uses memory to avoid areas of the search space that may lead to local minima. It simply stores a finite number of already visited

variables and simply ignores them when deciding in which variable (or state) it will move next. Thus it avoids local minima for a fixed number of steps.

Simulated Annealing (Kirkpatrick et al., 1983, Johnson et al., 1989, Aarts and Lenstra, 1997) is a hill-climbing algorithm inspired by physics and statistical mechanics. Here, there is a cost function and its value is computed each time the algorithm moves to a new variable assignment. If the difference between two successive values (δ) of the cost function is greater or equal to zero then the transition is valid. In any other case we choose the variable with probability $e^{-\delta/T}$. T is a control parameter called temperature and it if it reduced gradually then the algorithm converges to a solution.

Evolutionary algorithms have also been used extensively for solving CSPs (Tsang and Warwick, 1990a, Paredis, 1993, Warwick and Tsang, 1994, Hao and Dorne, 1994, Rojas, 1996). They can roughly be divided into two categories. The first is evolutionary algorithms using adaptive fitness functions. A comparison of algorithms in this category is given in (Eiben et al., 1998). The other category is evolutionary algorithms using heuristics. A good study on comparing evolutionary algorithms can be found in (Craenen et al., 2003).

An important group of evolutionary search methods is genetic algorithms for CSPs. They borrow terminology from evolutionary algorithms such as fitness, recombination, mating cross-over, population and mutation. In these algorithms, the new states that are reached come after mating two parent states. Specifically, there is a function, often named as mating function, which derives a new state by recombining the parent states. The parent states are chosen among various states, called the population. The population is a finite set of randomly generated states. New states may be subject to small random changes (mutation). Cross-over points are parameters used to choose new states. Information about genetic algorithms can be found in the works of (Goldberg, 1989, Tsang and Warwick, 1990b, Eiben and Ruttkay, 1997, Xinghuo Yu and Yao, 1998, Tsang et al., 2004).

Dynamic local search is a collective term for a number of approaches that try to escape local optima by iteratively modifying the evaluation function value of solutions. Dynamic local search algorithms guide the local search by a dynamic evaluation function. The evaluation function *e(s)* is composed of a cost function *c(s)* and a penalty function. The penalty function is adapted at computation time to guide the local search. A general introduction to dynamic local search can be found in (Hoos and Stutzle, 2004). Various state of the art dynamic local search algorithms can be found in (Hutter et al., 2002, Tompkins and Hoos, 2003, Pullan and Zhao, 2004, Thornton et al., 2004).

3.2 Distributed Constraint Satisfaction

Distributed constraint satisfaction addresses the matter of solving problems that are distributed by nature. In naturally distributed problems knowledge is scattered amongst multiple entities. This also holds for all the variables and constraints inherent in a distributed constraint satisfaction problems (DisCSPs). The reasons which render a problem distributed by nature could be limited capacity, dispersed computational power, transmission costs and privacy reasons. Nevertheless, distributed constraint

satisfaction is primarily concerned with how to reach a solution from a given distributed situation. Furthermore, there are areas in the field of distributed constraint satisfaction in which reaching a potential solution for a distributed problem is not quite enough. This is due to the fact that there is a very strong trend in distributed constraint satisfaction research striving to find optimal solutions. Problems as such are described as distributed constraint optimization problems (DisCOPs). Notwithstanding, unlike parallel problem solving, distributed constraint satisfaction's main concern is not automatically linked to optimal solutions. Despite distributed constraint satisfaction's relative short existence there are numerous achievements to display. Moreover, there are challenges that require great effort in order to be successfully addressed.

A DisCSP is a CSP in which the variables and constraints are distributed among automated agents (Yokoo et al., 1998). In a DisCSP agents communicate by sending messages. Agents can exchange messages only iff the sending agent knows the addresses of the recipient agents. Message delay is finite but random. During a message exchange sequence between any pair of agents, it is usually assumed that messages are received in the order in which they were sent. This does not necessarily hold for two messages send by different agents to the same recipient.

As mentioned each agent has some variables and tries to determine their values. However, interagent constraints may exist and the value assignment must satisfy these interagent constraints. Formally, there exist m agents 1, 2, … , m. Each variable X_j belongs to one agent i (represented as $belongs(X_j, i)$). Constraints are also distributed among agents. The fact that an agent i knows a constraint predicate p_k is represented as $known(p_k, i)$.

A DisCSP is solved iff the following conditions are satisfied:

$\forall i, \forall X_j$ where $belongs(X_j, i)$, a value d_j can be assigned to X_j, and $\forall l, \forall X_j$ where $known(p_k, l)$, p_k is true under the assignment $X_j = d_j$.

Without loss of generality, we can make the following assumptions since many algorithms are formalized under them. Furthermore, relaxing these assumptions to general cases is easy:

- Each agent has exactly one variable.
- All constraints are binary.
- Each agent knows all constraint predicates relevant to its variable.

Various applications have been modeled as DisCSPs, such as distributed resource allocation (Wellman, 1996, Walsh and Wellman, 1998, Petcu and Faltings, 2004, Ostwald et al., 2005, Anussornnitisarn et al., 2005, Endriss et al., 2006), distributed scheduling (Silaghi et al., 2000, Modi and Veloso, 2004, Petcu and Faltings, 2005a, Wallace and Freuder, 2005), distributed planning (Ephrati and Rosenschein, 1997, Sandholm, 2000, Parkes and Shneidman, 2004) and distributed sensor networks (Zhang et al., 2005, Béjar et al., 2005, Petcu and Faltings, 2007).

3.3 Algorithms for Distributed Constraint Satisfaction Problems

Arc consistency algorithms were amongst the first categories of algorithms for classic DSPs that were studied for potential extension to DisCSPs. The first idea was to incorporate parallel procedures for exchanging data and structures by using shared memory. In (Samal and Henderson, 1987) there was an attempt to construct parallel versions of AC1, AC3 and AC4 for shared memory parallel computers. A similar effort, slightly worse in complexity though due to communication overheads, can be found in (Cooper and Swain, 1994). Complexity issues were further improved at a theoretical level in (Kasif, 1990). In (Zhang and Mackworth, 1991, Zhang and Mackworth, 1992) there is a study for the parallelization of consistency CSP algorithms over dual networks. A distributed arc consistency algorithm (DisAC-4) was proposed in (Nguyen and Deville, 1998). This is a coarse-grained parallel algorithm for distributed computer memory, derived from the AC-4 algorithm which is optimal in time complexity ($O(n^2d^2)$ where n is the number of variables, and d is the size of the largest domain). Communication is being carried out through message passing. In the paper there are also termination and correctness proofs as well as theoretical time complexity calculation ($O(n^2d^2/p)$ where p is the number of processors). The same communication scheme is also used in the DisAC-9 algorithm (Hamadi, 2002b) which is optimal in message passing operations ($O(n^2d)$). Other arc consistency algorithms for the CSP were introduced later such as (Silaghi and Faltings, 2005) for search with asynchronous aggregation and (Ringwelski, 2005) for consistency in dynamic CSPs.

Synchronous Distributed Backtracking (DBT) is a set of distributed algorithms that stemmed from backtracking algorithms for the centralized CSP. An early work on DBT is (Yokoo et al., 1998). This algorithm is a simple and complete search algorithm for distributed constraint satisfaction problems and is based on simple backtracking. Agents agree on a fixed variable order and they pass to each other a growing partial assignment until a consistent assignment is reached. When an agent receives a partial assignment it assigns a value to the variable it controls. In DBT for simplification, as in many DisCSPs, an agent is considered to handle only one variable. If this is successful the new partial assignment is passed onto the next agent. If no variable value is consistent with the partial assignment, the agent that had sent the partial assignment is notified and commences backtracking. This algorithm being synchronous cannot take advantage of the parallelism its distributed nature suggests. It is basically a centralized algorithm the execution of which is passed between different processes or execution threads. Only one agent is actively processing value assignments at a given time. Moreover, there is a pre-determined fixed variable order which cannot be altered at execution time. Due to these deficiencies distributed algorithms that came as extensions to centralized backtracking algorithms had relatively poor performance. Workarounds to parallelism deficiencies for DBT algorithms have been sought in (Meisels and Zivan, 2003). Advances were merely restricted to communication privacy issues. Another approach presented in (Nguyen et al., 2004) outperformed (Meisels and Zivan, 2003) by two orders of magnitude. Nevertheless, backtracking algorithms can be asynchronous and thus used effectively in DisCSPs. Asynchronous backtracking has received a great deal of research efforts focused primarily on DisCSPs.

The Asynchronous Distributed Backtracking algorithm (ABT) was developed by Makoto Yokoo in (Yokoo et al., 1992). In ABT opposite to DBT, the agents process their variables concurrently and asynchronously. The agents initialize by instantiating their variables concurrently and send messages to all agents that have a constraint with their variable. This is their way of checking whether their assignment is consistent. If this value is not consistent, the recipient agent will send a message to the value sending agent, stating that the assignment is inconsistent (nogood message). Messages in ABT are asynchronous. The sending agent does not have to wait for an answer. Therefore execution can be continued. The ABT algorithm also utilizes the agent view perspective. Each agent keeps a data structure which holds the latest known assignment of all the values it has received. These values are assignments to variables controlled by other agents that participate in the same constraints with the variables of the holding agent. These data structures are used by every agent to check if his own variable assignment is consistent with the assignments of others. We have also priority values for variables, thus creating a total (static) order, under the purpose of avoiding infinite processing loops. In all constraints, the agent that has the lower priority is the recipient and the agent that has the higher priority is the sending agent. ABT is a complete algorithm. Actually, the unbounded recording of nogoods ensures the algorithm's completeness. It has exponential time complexity but no exponential space complexity. ABT displays a weakness similar to the ones described for DBT. The pre-determined static variable priority can produce exponential expansion to the search space. A significant improvement to the ABT algorithm by (Yokoo et al., 1992) is M-ABT in (Hamadi and Ringwelski, 2010).

Concurrent dynamic backtracking algorithms follow a slightly different approach for solving DisCSPs. They achieve concurrency by running multiple search processes at the same time. In distributed search, distinct subproblems are spread on several processing units and search is performed by the way of collaboration. Search processes are run simultaneously but each one in disjoint parts of the search space. In parallel or concurrent search, processes concurrently perform searches in disjoint parts of a state-space tree (Hamadi, 2002a). The key fact is that each search space includes all the variables but not all of their values. The concept of distributed dynamic backtracking was studied in the works of (Bessière et al., 2001). The proposed algorithm (DisDB) is asynchronous and complete. It performs dynamic jumps over the set of conflicting agents and it uses the distributed static variable order method. DisDB performs nogood recording and is shown in experiments to be marginally better than ABT.

Asynchronous Forward-Checking (AFC) is another family of distributed search algorithms on DisCSPs. In AFC search is depicted in a data structure called Current Partial Assignment (CPA). This set is empty before the search is started. The first initializing agent contributes its values to it and sends it to the next agent. Thus a single and synchronous data assignment structure is maintained throughout the whole procedure. Each agent contributes its values only if they are consistent with the values assigned so far. Otherwise, the algorithm backtracks to a former agent. After each value assignment a copy of the updated CPA is sent to all agents and they perform forward checking after receiving it. A significant early contribution for the family of distributed forward checking algorithms was presented in (Brito and Meseguer, 2003), where the proposed methods performs as good as the ABT algorithm. Another substantial contribution toward asynchronous forward checking algorithms is the one

in (Meisels and Zivan, 2007). This work uses dynamic variable ordering and it dynamically orders variables during search. Experiments showed that the AFC algorithm of (Meisels and Zivan, 2007) outperforms the ABT algorithm and in many cases the difference can span up to an order of magnitude. As in (Brito and Meseguer, 2003) the algorithm is complete.

The Asynchronous Aggregation Search (AAS) algorithm was developed by (Silaghi et al., 2000) and is the first known algorithm that separates public and private information. Generally in DisCSP algorithms, constraints are public (agents may share knowledge pertaining to a specific constraint) and variables are private (only the agent that controls a variable has complete knowledge of it). Furthermore, in AAS the search technique differs in that the solution procedure involves working with sets of partial solutions. Exchanged messages consist of aggregated valuations for variable combination sets.

In (Hamadi et al., 1998) there is the description of the Distributed Intelligent BackTracking algorithm (DIBT). It is an asynchronous distributed backtracking algorithm, based on BT an algorithm introduced in (Golomb and Baumert, 1965). DIBT performs exhaustive search space exploration. During failures conflict directed backjumping or graph based backjumping (CBJ) (Dechter, 1990) provides resolution. Hamadi et al. also implement a heuristic for static variable ordering, before the initial assignment. In contrast to ABT, DIBT does not record nogoods. Experiments showed that DIBT outperforms DBT in issues as the number of constraints checks, the number of messages and average time until solution is reached. The static variable ordering is considered a weakness of the algorithm. This also stands for the fact that it is incomplete (Bessière et al., 2001), contrary to what it is argued in (Hamadi et al., 1998).

Further work on intelligent backtracking resulted in the IDIBT (Interleaved Distributed Intelligent BackTracking) an algorithm that improves DIBT (Hamadi, 2001, Hamadi, 2002a). IDIBT mixes parallel and distributed search. It uses concurrency for exploring different parts of the search space, performing depth-first backtrack searches in parallel. Search subspaces are interleaved within every agent. Thus they can explore distinct subspaces at the same time. Nevertheless, procedures related to different subspaces are distinct and asynchronous as in all concurrent search algorithms (Lynch, 1996). Other concurrent distributed algorithms such as ConcBT and ConcDB can be found in (Zivan and Meisels, 2004a) and (Zivan and Meisels, 2004b, Zivan and Meisels, 2006) respectively. ConcDB is considered one of the best algorithms in concurrent distributed search. The same holds for M-IDIBT which is claimed to greatly improve IDIBT (Hamadi and Ringwelski, 2010).

The Asynchronous Weak Commitment Search algorithm (AWCS) was contributed by Makoto Yokoo (Yokoo, 1995). It is primarily based on the grounds of the weak commitment search algorithm for CSPs (Yokoo, 1994). Nevertheless, in related literature AWCS is mostly viewed as a modification of the ABT algorithm where the min-conflict heuristic (Minton et al., 1994) is used for changing the variable priorities dynamically and partial solutions are completely abandoned after one failure. In AWCS the use of the min-conflict value ordering heuristic implies that each agent always chooses from the consistent values with regards to higher priority agents. This value must also have the least number of constraint violations with regards to lower

priority agents. Dynamic variable priority change prevents exhaustive search. When a lower priority agent cannot find a consistent value, the priority order is changed and the lower priority agent gets the highest priority amongst the non-consistent agents. This means that when an agent has no way of assigning a consistent value to one of its variables, it will simply change its priority assuming the highest possible value. This is a way of inferring that previously selected values by other agents, although consistent were not as good decisions as they initially presented themselves. Thus the conflicting agent raises its priority, chooses the value that minimizes its conflicts with agents with lower priority and takes no further action. A consequence of such an action is that within a single failure's span, the partial solution constructed so far is completely abandoned. A failure to expand a partial solution (nogood) is always stored and used as a new constraint. This will prevent a potential reoccurrence and guarantees the completeness of the algorithm. A weakness of the AWCS algorithm is that it cannot discard recorded nogoods. Therefore, it has exponential space complexity.

A problem of noteworthy importance in all local search algorithms, centralized or distributed is how to escape local minima. In (Yokoo and Hirayama, 1996) Makoto Yokoo and Katsutoshi Hirayama provided a novel alternative to this. The proposed Distributed Breakout (DisBO) algorithm despite being derived from the Breakout algorithm for centralized CSPs (Morris, 1993) proved at first to be very competent for difficult instances of the distributed graph coloring problem. Notwithstanding, the major innovation of the Distributed Breakout Algorithm was that instead of detecting that a group of agents is caught in a local minimum, each agent has a way of determining if itself is in a quasi-local-minimum. During the algorithm's execution each flawed assignment (containing constraint violation) is assigned a weight factor. An agent *Xi* is in a quasi-local-minimum if it is violating some constraint and neither it nor any of its neighbors can make a change that results in lower cost for all. Thus a quasi local minimum is a weaker condition than a local minimum. Of course when an agent reaches a local minimum is in a quasi local minimum as well. But the opposite does not apply. Therefore, when caught in a quasi local minimum an agent changes the weight of constraint violations to escape the situation. Due to the fact that the Distributed Breakout algorithm can be caught in local optima, it is not complete. Further improvement methods and variations of the initial approach are found in (Hirayama and Yokoo, 2005).

In distributed constraint satisfaction problems general tree orderings have a single agent placed at the root. Each node-agent has zero or more children. All agents except the root have a parent. Nevertheless, there is a group of algorithms that uses a different ordering scheme, the pseudo-tree (Freuder and Quinn, 1985). A pseudo-tree for DisCSPs is a tree ordering that connects the agents participating in a constraint. If two agents share a constraint, then one of these agents is an ancestor of the other. Given a node in the tree the set of its own assignment and the assignments of all its ancestors, constitute a partial solution. Therefore, all paths rooted at this specific node form individual subproblems and can be solved independently. This is possible because there are no constraints between subproblems.

Constraints (edges) can only exist between a node and its descendants or ancestors. The Distributed Pseudo-tree Optimization algorithm (DPOP) makes extensive use of pseudo-trees (Petcu and Faltings, 2005b). The algorithm is formulated for

optimization problems (DisCOPs) but can be used for DisCSPs as well. DPOP was ultimately inspired by the sum-product algorithm (Kschischang et al., 2001) and it was extended to support arbitrary topologies. After the algorithms first step (pseudo-tree formation) is complete, the second step involves messages (util-message) from the leaves to the root of the pseudo-tree. Each agent receiving a util-message uses it for assigning its own value on the grounds that it must cause minimal cost. Then the agent forwards the message after it has added its own value to it. When this step is complete the root agent commences the next and final step by sending its optimal value to its descendants (value-message). This value is again used in each child node for computing its own optimal value. The obvious drawback is that message length can grow exponentially. Indeed DPOP is time and space exponential in the induced width of the problem. Moreover, agents cannot process their values unless they have received certain assignments from ancestors or descendants, depending on the execution step. This makes DPOP synchronous but as stated in (Petcu and Faltings, 2005b) complete.

In (Petcu and Faltings, 2007) the authors are dealing with message length for DPOP. This resulted in a new algorithm, namely MB-DPOP (Memory Bounded DPOP). Here, there is a controlled parameter that defines the amount of available memory. At each node the available memory must be greater than d^k, where d is the domain size. Sub-graphs with width higher than k are identified and then they are reduced so that their width is k. The remaining nodes are called cycle-cut nodes. In each such area all combination values of cycle-cut nodes are explored. Results are stored and integrated into messages that are propagated through the pseudo-tree, as in DPOP.

3.4 Distributed Constraints and Optimization Methods

As we have already mentioned a constraint satisfaction problem is considered a constraint optimization problem if we can define a cost function in which constraints participate under predefined weights. Our goal is to find a solution, which minimizes the cost while satisfying all constraints. As opposed to Distributed Constraint Satisfaction, Distributed Constraint Optimization does not search for a particular solution where the only prerequisite is that all constraints must be satisfied. Optimizing constrained problems means that some solutions are considered more desirable than others and it is upon the way the cost function is formed to lead the solution procedure to the most desirable solutions. Efforts to optimize methods for distributed constraint satisfaction have resulted in various algorithms for Distributed Constraint Optimization.

The majority of the first efforts towards distributed optimization has emerged after adapting centralized variations of backtrack search. These methods for optimizing constraint problems were transformed so as to work in distributed environments. Branch and bound techniques stemming from the centralized optimization scheme were among the first to be applied for finding optimal solutions for distributed constraint problems. Specifically, in (Land and Doig, 1960) proposed the Branch and Bound algorithm for finding optimal solutions to a variety of problems in discrete and combinatorial logic. In (Hirayama and Yokoo, 1997) the SyncBB algorithm was introduced a distributed version of the Branch and Bound algorithm. The Branch and Bound philosophy is to gradually improve the quality of a first complete instantiation. After the first complete instantiation, which holds as the best solution so far, its cost is

stored as an upper bound on the cost that the algorithm tolerates. During future searches and before a possible variable assignment, we compute the variable's partial cost and the cost the new assignment is bound to produce. If this cost is larger than the current upper bound, the search backtracks. Whenever there is new complete assignment with lower cost, we update the best current solution and the upper bound as well. In this work, Hirayama and Yokoo also introduce the Iterative Distributed Breakout algorithm (IDB). Although SyncBB performs significantly better than IDB the authors note that IDB should be used if we choose to find a nearly optimal solution fast.

In (Gershman et al., 2006) the authors use Branch and Bound methods for distributed constraint optimization. They present the Asynchronous Forward Bound (AFB) search algorithm a new algorithm for solving distributed constraint optimization problems (DisCOPs). AFB keeps one consistent partial assignment at all times. Every agent upon assigning a consistent value propagates the bounds on the cost of solutions all unassigned agents. To do so the propagating messages contain copies of the agent's partial assignment. Agents assign variables sequentially and propagate their assignments asynchronously. It is shown that AFB outperforms SyncBB.

DPOP (Petcu and Faltings, 2005b) as stated in Section 3.3 is another algorithm for DisCOPs. DPOP uses pseudo-trees and parents accumulate costs of all combinations of partial solutions from their children. Then parents calculate and generate all the possible partial solutions, which include the partial solutions it received from its children and its own assignments and sends the resulting combinations up the pseudo-tree. Once the root agent has received all the information its children have generated, it produces the optimal solution and propagates it down the pseudo-tree to so that every agent becomes aware of it.

Another approach for achieving optimality for DisCSPs is the Optimal Asynchronous Partial Overlay (OptAPO) algorithm (Mailler and Lesser, 2004). OptAPO is an optimization version of the Asynchronous Partial Overlay (APO) algorithm originally introduced in (Mailler and Lesser, 2003) and later refined and extended in (Mailler and Lesser, 2006a). In OptAPO, conflicting agents choose a mediator. There they transfer their data and the mediator solves the partial problem. To do this, the agents that take the role of the mediator, compute the optimal value cost for the agents under conflict and attempt to change the assignments of the variables in question in order to achieve this optimal value. Thus, the mediator centralizes a relevant portion of the problem, and conducts a Branch and Bound search on it. If this cannot be achieved without causing cost for agents not participating in a particular conflict, the mediator links with those agents assuming that their cost has increased by its decision.. Therefore, in a future step these agents appear as conflicting agents as well. This way the mistake of increasing the cost for these agents will not be repeated.

Due to the fact that APO and OptAPO completeness was challenged Grinshpoun and Meisels, presented formal proofs of soundness and completeness in (Grinshpoun and Meisels, 2008). Moreover, Grinshpoun and Meisels have identified the problematic parts of APO and made the necessary modifications so as to resolve these problems. Therefore, they introduced CompAPO and CompOptAPO which are complete versions of APO and OptAPO respectively. CompAPO was subsequently compared to Asynchronous Backtracking (ABT) and Asynchronous Weak Commitment (AWC).

CompAPO performed worse than ABT and AWC in problems of sparse and medium density. Only in problems off high density, CompAPO outperformed ABT and AWC.

Arguably, one of the most prominent algorithms for distributed constraint optimization is Adopt (Modi et al., 2005). In Adopt, every agent checks its best assignment according to the context it holds. Context refers to an agent's view of other agents' assignments. The agent chooses the assignment with the lowest cost, according to its current context and propagates the new assignment. Thus, all the agents that belong to an agent's view must adjust their assignments accordingly. Extensive details pertaining to the algorithm are given in Section 6.4.

Adopt is considered to be in the group of the most efficient algorithms in distributed constraint optimization. Nevertheless, in (Gershman et al., 2006) the authors argue that for some problem instances AFB outperforms Adopt. Also in (Mailler and Lesser, 2006a) is stated that OptAPO performs better than Adopt as far as computation and communication is concerned when it faces the graph coloring problem. Another work (Gershman et al., 2008), gives a more comprehensive picture. There, for random Max-DisCSPs of low constraint density and low constraint tightness Adopt and AFB perform well. SynchBB is close but DPOP shows poor performance. For random Max-DisCSPs of high constraint density and low constraint tightness, Adopt, AFB and SynchBB perform very well. Again DPOP's performance is much worse. This situation changes in cases of low constraint density and high constraint tightness where DPOP along with AFB show good results. Here Adopt stays behind. For high constraint density and high constraint tightness Adopt and SynchBB show significant decrease in terms of efficiency. On the other hand DPOP's performance remains on high standards seeming to ignore the increase in terms of the problem's tightness. AFB is the best performing algorithm in this case. It outperforms DPOP by two orders of magnitude. Concluding the authors state that Adopt's main issue is that its performance deteriorates significantly if it has to face tight problems. Also in (Gershman et al., 2008) state that Adopt performs better than CompOptAPO in sparse and dense problems as long as the constraints are relatively loose. This condition shifts for problems with tight constraints. There CompOptAPO performs better than Adopt. Surprisingly, in both dense and tight constraint problems AFB performed better than CompOptAPO, managing to find solutions faster.

Another algorithm that is considered state of the art is No-Commitment Brach and Bound (NCBB) (Chechetka and Sycara, 2006). Like Adopt, NCBB utilizes Depth-First-Search trees (DFS trees) in which there can be nodes with a single parent but multiple children. Only nodes with parent child relationship can share constraints. During initialization, agents choose values so as to minimize their cost. Then ancestors propagate values to their descendants which re-arrange their assignments if necessary. NCBB needs polynomial space and authors in (Chechetka and Sycara, 2006) claim that on random graph coloring problems NCBB performs better than ADOPT another well-known polynomial space algorithm. Although NCBB needs polynomial space, when we are not concerned about communication costs, NCBB outperforms DPOP an algorithm that has exponential space requirements.

Nevertheless, variations of Adopt such as BnB-ADOPT (Yeoh et al., 2008) have improved the initial Adopt algorithm. In (Yeoh et al., 2008) BnB-ADOPT is measured to have increased the performance of the initial Adopt algorithm by an order of

magnitude. The authors have experimented on random graph coloring problems, scheduling meetings and allocating targets to sensors to support their claim. Additionally, BnB-ADOPT outperforms NCBB in all cases with the exception of coloring dense graphs with fast communication where NCBB is faster. Moreover, BnB-ADOPT and NCBB are equally fast at coloring dense graphs with slow communication, coloring sparse graphs and allocating targets to sensors with fast communication.

3.5 Challenges in Distributed Constraint Satisfaction and Optimization

A large number of DisCSPs are situated in dynamic contexts. This implies that in these problems variables, values and constraints are changed in a continuous time fashion. Changes comprise adding, removing or even editing one or more of the above parameters in a quantum time unit. There is a lot work to be done in order to tackle dynamic context problems in a satisfactory way. Until now the majority of works in the area of distributed problem solving is closely related to problems within static contexts.

In DisCSPs information is naturally distributed and the computing nodes are often from different organizations and have heterogeneous data formats, representations and semantics. Information representation and exchange under these conditions is difficult. A major research challenge is to develop schemes that can overcome these difficulties.

For the DisCSP community the ability to handle large-scale problems is viewed as a key factor. Distributed systematic search algorithms have great difficulties to solve problems, as the number of variables increases. In occasions where a problem involves multiple variables, certain issues arise. There is the ability to communicate and process high numbers of messages, proper execution coordination, knowledge management and dissemination and efficient communication guaranties when necessary. However, the greatest weakness is that of incompleteness. Not only it is hard to find a complete distributed algorithm for solving constrained problems but also proving a distributed algorithm to be complete, can be a very strenuous task.

DisCSP algorithms, by definition utilize distributed information and data storage. During the solution procedure, combinations of consistent and inconsistent value assignments take place. The latter are referred to as nogoods. If the search takes long enough, nogoods can occupy considerable storage space. In large-scale systems even if storage is of no issue, identifying re-occurrence of a nogood or avoidance thereof is a very perplexing task. On the other side, agents can never tell if obtained domain information of another variable is complete or up to date. Furthermore, while constraints are never directly revealed a nogood can be perceived as summarized information of a higher level. All distributed algorithms find it critical to avoid previously explored nogoods. This may involve storing all nogoods as they occur. Nevertheless, distributed complete search algorithms exhibit intelligent ways to explore the search space, which makes the inferring of such knowledge (previously explored nogoods) easy.

Distributed constraint satisfaction problems are very often solved by several processors. In all such cases, synchronous execution leads to communication overheads, idle processor periods that could be avoided, possible deadlocks as in operating systems theory even execution failures, due to infinite waiting periods of messages or signals for example that for a number of reasons may have been lost. Moreover, in distributed problems all related parameters are scattered over a network. This network's nodes operate in a certain environment which can have diverse parts. Solver nodes in such cases may be partially unavailable or inaccessible. This can cause long network delay times or message loss. Furthermore, messages can arrive in a different order than they were sent. The use of asynchronous protocols is a major forward step in this direction. Great assistance for asynchronous protocols was provided by the area of multi-agent systems. Agent platforms that offer asynchronous communication capabilities have driven distributed problem solving research to use more efficient methods in problem solving. When an agent does not respond or his messages are lost, the remaining agents can continue to operate. In synchronous algorithms, when a message is lost, algorithm execution halts on the waiting agent's side. This eliminates a node in the network and its consequences are often dire as far as finding proper execution of any algorithm concerned.

This page is intentionally left blank

CHAPTER 4

4. Problem Specification

This chapter addresses the formal definition of problems that involve recourse allocation and scheduling in large-scale agent networks. Furthermore, it clarifies various assumptions that have been made, and gives an overview of the proposed approach. At first a description is given in relevance to the nature of the agent network. Thereafter, the chapter provides an analysis of the type of task requests under consideration. The latter are followed by a detailed specification of the problems studied. An overview of the proposed approach concludes the chapter.

4.1 The Agent Network

In this work we have been consistently using large-scale networks of collaborating agents that are geographically distributed. Therefore, the network's connectivity factor varies between different geographical regions. We have modeled agent networks (also called acquaintance networks) as connected graphs of the form $G=(N,E)$, where N is the set of agents and E is a set of bidirectional edges denoted as non-ordered pairs (A_i, A_j). The (immediate) neighborhood of an agent A_i includes all the one-hop away agents (i.e. each agent A_j such that $(A_i, A_j) \in E$. The set of neighbors (or acquaintances) of A_i is denoted by $N(A_i)$.

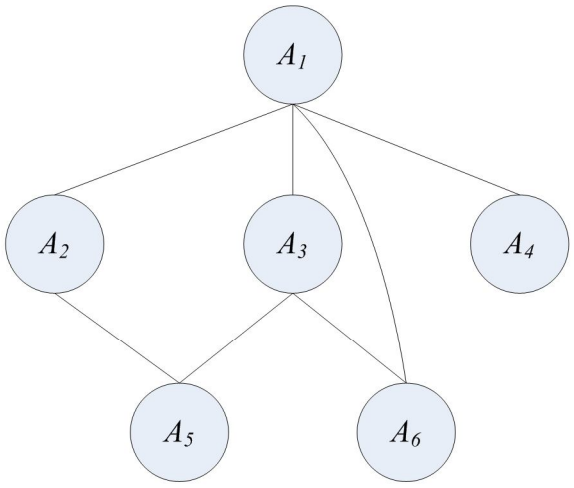
Agents are considered to be time-bounded with specific discrete capabilities. We assume that each agent has a single type of resource and at each time point at most one of the requested tasks can use any such resource. Under these assumptions the problem of allocating tasks to the resources becomes equivalent to the problem of finding available time intervals in the agents' schedule to allocate to the various tasks. Therefore we can consider that the only resource that an agent manages is a set of time units. These assumptions are not restrictive as the method can be extended to other resources, which can be treated mostly as we treat the availability of time units or agents' capabilities.

Consequently, each agent A_i is being attributed with a set of capability types $CapA_i=\{capAi_1, capAi_2, ..., capAi_m\}$, a set of time-units $RA_i=\{rAi_1, rAi_2,...,rAi_n\}$, totally ordered with a relation "consecutive" denoted by "<", and a priority Pi which is a positive integer. The function *earliest(R)* (resp., *latest(R)*) denotes the time point rAi_k in $R, (R \subseteq RA_i)$ for which there is not any other point r in R with $r< r_{ik}$ (resp. $r_{ik} <r$).

The priority is a unique identifier that is assigned to each agent when it enters the multi agent system (Carle and Simplot-Ryl, 2004). This is necessary for the computation of the overlay network (Crespo and Garcia-Molina, 2002) the agents use to keep track of available resources in their immediate neighborhood. Priorities are also crucial for the non-cyclic update of routing indices. These indices are different

for every agent and are used as an indicator for finding the route for forwarding unallocated tasks towards directions where the probability of being allocated is high. On real world systems priorities may depend on several factors such as the battery life of the agent, its availability of resources, its capabilities etc.

The initial state of an example network is shown in Figure 4.1. In this network $G= (N,E)$ as depicted in the figure, each agent has 10 consecutive time-units, which are available for allocation, and a specific set of capabilities.



$G = (N, E)$

$N = \{A_1, A_2, A_3, A_4, A_5, A_6\}$

$E = \{(A_1, A_2), (A_1, A_3), (A_1, A_4), (A_1, A_6), (A_2, A_5), (A_3, A_5), (A_3, A_6)\}$

Figure 4.1: A network of acquaintances.

Specifically, for all agents in Figure 4.1 $earliest(R_i) = 0$ and $latest(R_i) = 10$ ($\forall Ai,\ i = 1,2,...,6$). Capabilities and priority values for this example are defined for every agent in Table 4.1:

A_1: $capA_1 = \{1,3\}$, $P_1 = 1$.
A_2: $capA_2 = \{1,2,3\}$, $P_2 = 2$.
A_3: $capA_3 = \{1\}$, $P_3 = 3$.
A_4: $capA_4 = \{2,3\}$, $P_4 = 4$.
A_5: $capA_5 = \{3\}$, $P_5 = 5$.
A_6: $capA_6 = \{1,2\}$, $P_6 = 6$.

Table 4.1: Capability types and priority values for the acquaintance network of Figure 4.1.

In the example above we assumed that $capA_i = \{1,2,3\}$. Accordingly, we can infer that each agent can have at least one capability type (of type 1 or 2 or 3) and at most three (of type 1 and 2 and 3). Capability types can be easily linked to real systems. For example for agents handling hotel reservations the set of capabilities could be $capHotelAgent_i$ = *{single room reservation, double room reservation, hotel suite*

reservation}. For simplicity reasons, hereafter we represent capabilities as consecutive non-negative integers.

4.2 Task Modeling

For agent networks of the form $G = (N,E)$ we assume that there is a set of tasks which in order to be served require resources the agents handle and therefore can provide. Each demand for agent resources can present itself arbitrarily in any agent (A_i) participating in G. In case Ai has adequate resources and the appropriate capabilities to allocate an incoming task it does so and re-evaluates the resources (time units) that are currently available. If Ai does not have the appropriate time units or its capabilities are inadequate it must form a coalition of neighboring agents, break the task into smaller subtasks if possible and try to allocate the resources the task has requested. Otherwise, A_i must forward the unallocated task to another agent (A_j) that will initiate a similar process in its immediate neighborhood.

Let us consider a set of requests concerning k independent tasks $T = \{t_1,.., t_k\}$. Each task *ti* can be either atomic or complex. In the former case, atomic tasks require the commitment of a single agent. As far as the latter incident is concerned, a complex task may require the joint participation of a set of agents to handlw the atomic subtasks $ti = \{g_{i1}, g_{i2} ,..., g_{ik}\}$ contained in the complex tasks. Each atomic task t_i (or subtask gi_m) is a tuple $<a_i, start_i ,end_i ,Cap_i>$, where α_i, $start_i$ *and* end_i, are all non-negative integers. The first (a_i) represents the maximum number of time units that the task needs in order to be completed. Integer $start_i$ denotes the earliest time point t_i should start to be executed. Respectively, end_i denotes the latest time point *ti* should finish executing. Finally, Cap_i is the set of agent capabilities that are required for the successful execution of the task.

For each complex task t_i consisting of a set of sub-tasks $\{g_{i1} ,..., g_{ik}\}$ there is also a set of constraints C_i corresponding to the interdependencies between the subtasks in *ti*. We consider binary precedence constraints in the general case, specifying temporal distances between the executions of subtasks. For example, the constraint $start(g_{im})+3 \leq start(g_{in})$ means that the execution of subtask g_{in} must start at 3 time units after the execution of subtask g_{im} begins. Accordingly, start (g_{ik}) > 5 means that for the subtask g_{ik} execution must start not earlier than in the 6th time unit of an agent.

Consider a set of requests concerning tasks $T = \{t_1, t_2\}$ that are submitted to the agent network presented in Figure 4.1. Task t_1 comes with 3 individual subtasks $t_1 = \{g_{11}, g_{12}, g_{13}\}$. For the individual sub-tasks we can assume that following (Table 4.2):

$g_{11} = < a_{11} = 1, start_{11} \geq 0, end_{11} < 5, Cap_{11}=3 >$
$g_{12} = < a_{12} = 2, start_{12} \geq end_{11}, end_{12} < 8, Cap_{12} = 2 >$
$g_{13} = < a_{13} = 2, start_{13} \geq end_{12}, end_{13} < 10, Cap_{13} = 1 >$

Table 4.2: Time units in demand, start and end constraints and requested agent capabilities for complex task $t_1 = \{g_{11}, g_{12}, g_{13}\}$ in $T = \{t_1, t_2\}$.

Similarly, task t_2 consists of 2 individual subtasks $t_2 = \{g_{21}, g_{22}\}$. Subtasks g_{21} *and* g_{22} can be of the form (Table 4.3):

$g_{21} = < a_{21} = 1, start_{21} \geq 0, end_{21} < 5, Cap_{11} = 2 >$
$g_{22} = < a_{22} = 2, start_{22} \geq end_{21}, end_{22} < 8, Cap_{1\,2} = 1 >$

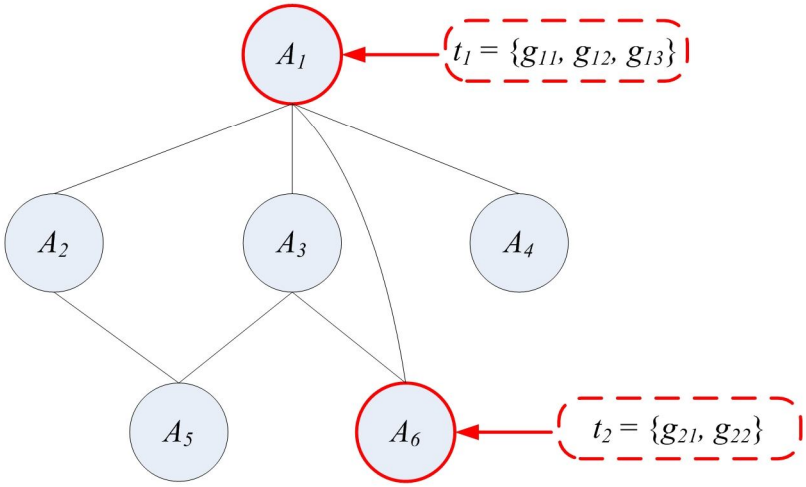
Table 4.3: Time units in demand, start and end constraints and requested agent capabilities for complex task $t_2 = \{g_{21}, g_{22}\}$ in $T = \{t_1, t_2\}$.

If t_1 enters the agent network in agent A_1 and t_2 uses A_6 as an entry point the following situation occurs (Figure 4.2).

$$G = (N, E)$$

$$N = \{A_1, A_2, A_3, A_4, A_5, A_6\}$$

$$E = \{(A_1, A_2), (A_1, A_3), (A_1, A_4), (A_1, A_6), (A_2, A_5), (A_3, A_5), (A_3, A_6)\}$$



Agent capabilities and priorities

A_1: $capA_1 = \{1,3\}$, $P_1 = 1$.
A_2: $capA_2 = \{1,2,3\}$, $P_2 = 2$.
A_3: $capA_3 = \{1\}$, $P_3 = 3$.
A_4: $capA_4 = \{2,3\}$, $P_4 = 4$.
A_5: $capA_5 = \{3\}$, $P_5 = 5$.
A_6: $capA_6 = \{1,2\}$, $P_6 = 6$.

Task set $T = \{t_1, t_2\}$

t_1:
$g_{11} = <a_{11}=1, start_{11} \geq 0, end_{11} < 5, Cap_{11}=3>$
$g_{12} = <a_{12}=2, start_{12} \geq end_{11}, end_{12} < 8, Cap_{12}=2>$
$g_{13} = <a_{13}=2, start_{13} \geq end_{12}, end_{13} < 10, Cap_{13}=1>$

t_2:
$g_{21} = <a_{21}=1, start_{21} \geq 0, end_{21} < 5, Cap_{11}=2>$
$g_{22} = <a_{22}=2, start_{22} \geq end_{21}, end_{22} < 8, Cap_{12}=1>$

Figure 4.2: Emergent situation as the task set $\boldsymbol{T = \{t_1, t_2\}}$ surfaces into the agent network G and requests to use its resources.

It is obvious that A_1 cannot handle t_1 all by itself. Therefore A_1 must look at its immediate neighborhood and form an agent group that is capable of serving t_1.

Among agents near A_1 the ones that posses the required capabilities are A_2, A_4 and A_6. Between the three A_1 will most likely choose A_2 first. That is because A_2 has the highest priority compaired to the other two agents. On the contrary, A_6 seems to have the necessary capabilities and time units to accomodate t_2. Please note that in this example we have chosen to assume that the lower the index the higher the priority of an agent.

4.3 Problem Formulation

After careful consideration of what has been described in paragraphs 4.1 and 4.2, the general problem that this thesis deals with is as follows (Table 4.4).

Given a network of agents $G = (N,E)$ and a set of requests for resource allocation modelled as a set of tasks T, we require:

a) for each atomic task $t_i = <a_i, start_i, end_i, Cap_i>$ in T, to find an agent A_j in N, such that $perform(A_j, t_i) = 1$. This holds if A_j has the required capabilities and time resources:
   $\{(Cap_i \subseteq Cap_j) \wedge (\exists\ R \subseteq R_j$ s.t. $|R| \geq a_i \wedge start_i \geq earliest(R) \wedge latest(R) \geq end_i)$, where R is an interval of consecutive time points$\}$. On the contrary, $perform(A_j, t_i)=0$.
b) for each joint (complex) task t_i in T with subsidiary tasks $\{g_{i1},.., g_{ik}\}$ to find a set of agents G that form a network of potential team-mates (PTN) such that
   i. for each sub-task there is an agent in G that can perform it: $\sum_{gik} perform(A_l, g_{ik}) = |\{g_{i1},...,g_{ik}\}|$
   ii. the precedence constraints between the subtasks of t_i are satisfied.

Table 4.4: Problem formulation for allocating available resources in an agent network represented by graph $G = (N.E)$.

For that purpose, we assume that for each constraint $c_{kl} \in C_i$ between two subsidiary tasks g_{ik} and g_{il} there is a *cost function* f_{kl} providing a measure of the constraint's violation. The total satisfaction of the complex task's constraints is defined as an aggregation of these cost functions. In subsequent chapters, it is explained in detail how complex tasks are further modeled to fit into a cost function, how the cost functions are evaluated and how their aggregation is computed.

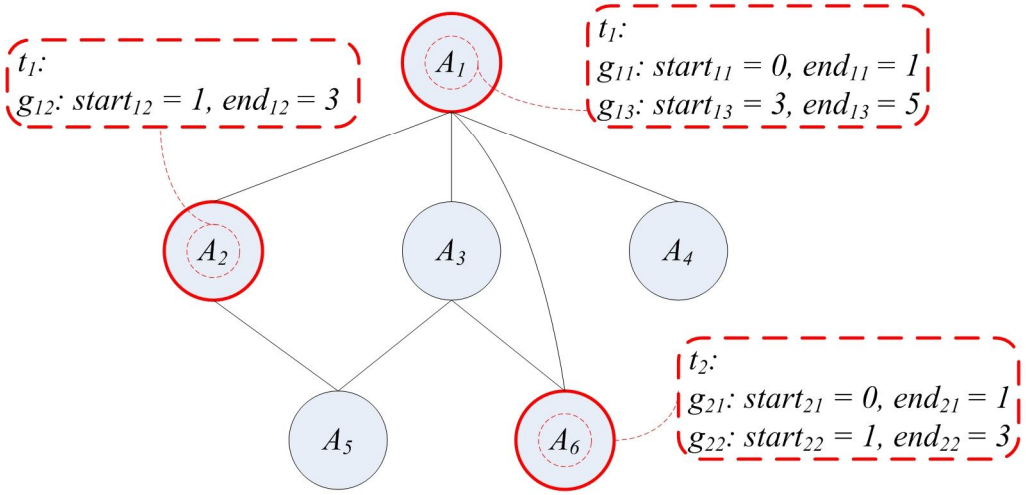
The aim is twofold. First, to increase the benefit of the system, i.e. the ratio of tasks successfully allocated to the total number of requests. As explained above, we consider an atomic task to have been successfully allocated if the system has located an agent that has the available resources and capabilities to serve the requested task. A complex task is considered successfully allocated if a team of agents is formed such that all sub-tasks of the complex task are successfully allocated and all inter-task constraints are satisfied. Our second goal is to increase the message gain, i.e. the ratio of the benefit to the number of messages exchanged.

A possible scenario and its outcome for the problem instance in Figure 4.2 can be as follows. Agent A_1 detects that the subtask g_{12} cannot be served by itself and forms a coalition with agent requesting from A_2 to try to serve subtask g_{12}. Furthermore agent A_1 tries to resolve the constraints inherent to subtasks g_{11} and g_{13}. Moreover, A_1 and A_2 collaborate to resolve the constraints that g_{12} shares with g_{11} and g_{13}. Simultaneously, agent A_6 detects that it is feasible to serve the complex task t_2 and tries to find a solution for the constraints between g_{21} and g_{22}. Figure 4.3 sums up the solution procedure previously described.

$$G = (N, E)$$

$$N = \{A_1, A_2, A_3, A_4, A_5, A_6\}$$

$$E = \{(A_1, A_2), (A_1, A_3), (A_1, A_4), (A_1, A_6), (A_2, A_5), (A_3, A_5), (A_3, A_6)\}$$



Agent capabilities and priorities

A_1: $capA_1 = \{1,3\}$, $P_1 = 1$.
A_2: $capA_2 = \{1,2,3\}$, $P_2 = 2$.
A_3: $capA_3 = \{1\}$, $P_3 = 3$.
A_4: $capA_4 = \{2,3\}$, $P_4 = 4$.
A_5: $capA_5 = \{3\}$, $P_5 = 5$.
A_6: $capA_6 = \{1,2\}$, $P_6 = 6$.

Task set $T = \{t_1, t_2\}$

t_1:
g_{11}=<a_{11}=1, $start_{11} \geq 0$, $end_{11} < 5$, Cap_{11}=3>
g_{12}=<a_{12}=2, $start_{12} \geq end_{11}$, $end_{12} < 8$, Cap_{12}=2>
g_{13}=<a_{13}=2, $start_{13} \geq end_{12}$, $end_{13} < 10$, Cap_{13}=1>

t_2:
g_{21}=<a_{21}=1, $start_{21} \geq 0$, $end_{21} < 5$, Cap_{11}=2>
g_{22}=<a_{22}=2, $start_{22} \geq end_{21}$, $end_{22} < 8$, Cap_{12}=1>

Figure 4.3: A possible outcome of the solution procedure in agent network $G = (N,E)$ after the set $T = \{t_1, t_2\}$ of complex tasks enters the network at agent A_1 for t_1 and agent A_6 for t_2.

To achieve these goals we propose an approach that builds on self-organization approaches for ad-hoc networks, token-based approaches for coordination in large-scale systems, and distributed constraint optimization. Namely, the agents in the network are organized in a dynamic overlay structure consisting of dominating nodes, which act as "heads of service", and non-dominating ones which only provide resources. The dominating nodes are responsible for forwarding task requests through

the network as they are the ones that have some, incomplete, view of the available resources. This is achieved by equipping these nodes with a routing index which is a compact summary of the resources available through itself and its neighbours. Once a complex task request enters the system it is forwarded through the network until a set of agents that have the available resources and capabilities to serve it are located. The way the task "travels" through the network, either broken down to its subtasks or as a whole, is determined by certain heuristics that are explained in Chapter 6. Once an appropriate set of agents is located, they receive the constraints between the subtasks of the complex task; they form a potential team, and try to find a joint schedule for the task so that the interdependencies between its subtasks are satisfied. This is done using distributed constraint optimization techniques. If no satisfying schedule is found then the potential team is reorganized by releasing one or more agents from the team and having other appropriate agents enter it. This process, is repeated until the task has been successfully scheduled or its deadline expires.

This page is intentionally left blank.

CHAPTER 5

5. Agent Organization and Searching

This chapter presents the individual techniques agents use for constructing, organizing and managing collaborative groups. These arrangements are part of the infrastructure that will be used by algorithms that solve Distributed Constraint Optimization Problems (DisCOPs). Specifically, it describes the construction of dynamic overlay networks of gateway agents, and the construction and maintenance of routing indices. The course of action taken in this Chapter follows the work of (Theocharopoulou et al., 2007) as presented in their paper.

5.1 Dynamic Overlay Networks of Gateways

The primary advantage of a dominating set of nodes in a network is that it can be used to "cover" all the nodes in the network. In a DisCSP or a DisCOP, distributed among geographically scattered agents, computing the dominating set of nodes makes the solution procedure much easier. The members of the dominating set (called gateway agents) are aware not only of their own resources and capabilities but also monitor the available resources and capabilities of the agents they cover. In that way gateway agents can form local knowledge clusters and make good decisions, pertaining to the probability of an incoming complex task being served by themselves and the area they cover.

A dominating set of nodes in a network is a connected subset of nodes that preserves and maintains the "coverage" of all the other nodes in the network (Crespo and Garcia-Molina, 2002). In a connected network where each node owns a distinct priority, a node *A* is fully covered by a subset *S* of its neighbours in case the following hold (Carle and Simplot-Ryl, 2004):

- *S* is connected
- Each neighbour (excluding the nodes in *S*) of A is a neighbour of at least one node in *S*
- All nodes in *S* have higher priority than *A*.

A node belongs to the dominating set if no subset of its neighbours fully covers it. Although originally proposed for area coverage and monitoring (F. and J., 2003, Carle and Simplot-Ryl, 2004), nodes may be considered to cover an information space, or the space of capabilities and/or resources required for the execution of tasks. The algorithm of Dai and Wu (F. and J., 2003) for the computation of a dominating set of nodes allows each node to decide about its dominating node status without requiring excessive message exchange and based only in local knowledge: the knowledge of a node's neighbours is sufficient (Carle and Simplot-Ryl, 2004). The algorithm is as shown in Table 5.1:

---

1. Collect information about one-hop neighbours and their priorities

2. Compute the sub-graph of neighbouring nodes that have higher priority
3.**If** (this sub-graph is connected) **and**
((every one-hop neighbour is in this sub-graph) **or**
(it is the neighbour of at least one node in the sub-graph)),
then opt for a non-gateway node.
Else opt for a gateway node.

Table 5.1: Computing the gateway status of an agent

A simple network of 10 agents is depicted in Figure 5.1 for illustration purposes. The form of the presented example network is similar to the ones produced in any of the presented of experiments sets. Here, we consider nodes with low index numbers having the highest priorities (e.g. agent 1 has higher priority than agent 2).

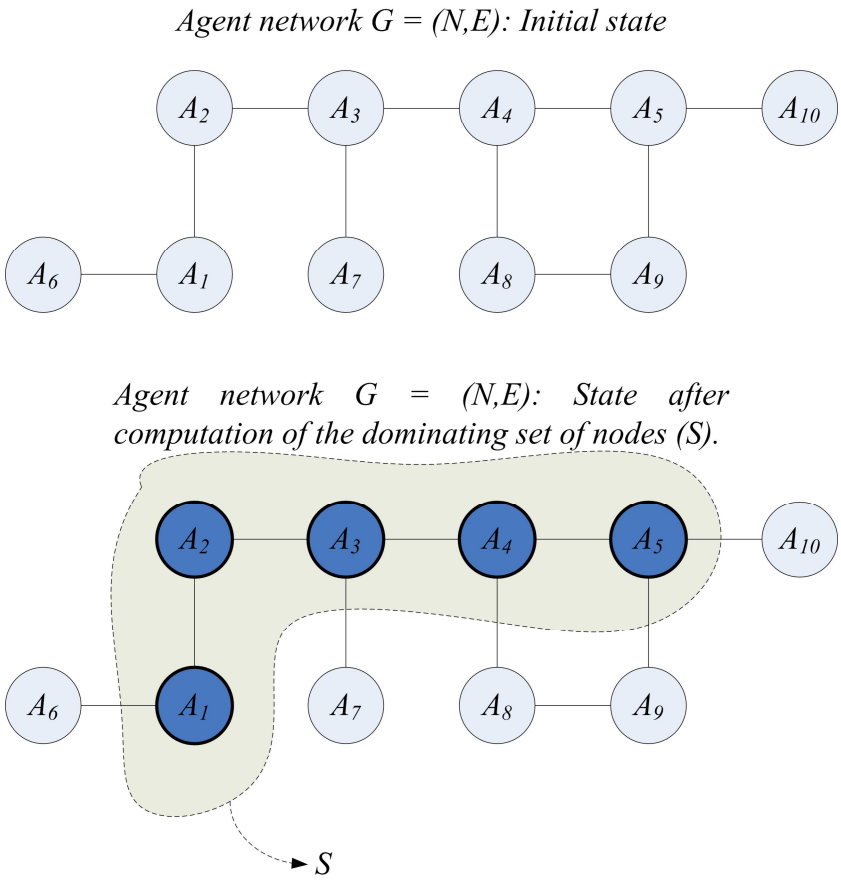


Figure 5.1: Computation of the dominating set of nodes (gateway agents) in an agent network with priority values.

Dominating nodes (gateway agents) are dynamically computed in case the acquaintance network changes. Having computed an overlay network of gateways that constitute the backbone of the system and "cover" all the other (non-dominating or non-gateway) nodes in the network, the propagation of requests and updates of all kinds can be restricted to this set of nodes. Specifically, according to the approach followed throughout the thesis, gateway agents have the responsibility to forward requests to their neighbours and keep a record of their neighbours' availability and

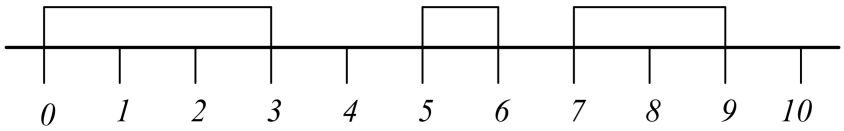
capabilities. Non-gateways forward all requests to gateways and possess only information about their own availability. Therefore, as the percentage of gateways in a network decreases, the searching task is expected to become more efficient, although the maximum workload of the agents is expected to increase.

5.2 Routing Indices

Given a network of agents $G=(N,E)$, and the set of neighbours $N(A)$ of an agent A in N, the routing index (RI) of A (denoted by $RI(A)$) is a collection of $|N(A) \cup \{A\}|$ vectors of resources' and capabilities' availability, each corresponding to a neighbour of A or to A itself. Given an agent A_i in $N(A)$, then the vector in $RI(A)$ that corresponds to A_i is an aggregation of the resources that are available to A *via* A_i. The key idea is that given a request, A will forward this request to A_i if the resources/capabilities available to the network *via* A_i can serve this request.

As it can be understood from the above, the efficiency and effectiveness of *RIs* depend on the way availability is being modelled and on the way this information for a set of agents is aggregated in a single vector (Crespo and Garcia-Molina, 2002).

To compactly capture information about the availability of time units, each agent A_i has a time vector V_i of m tuples $<j,s>$ representing the time-units available to the agent. Each non-negative integer s represents the number of consecutive non-allocated time-units that follow the time point j, $0 \leq j \leq m$. This vector can be depicted as a time line of m points. For example, the time vector $Vi = (<0,3>, <1,2>, <2,1>, <3,0>, <4,0>, <5,1>, <6,0>, <7,2>, <8,1>, <9,0>)$ represents the time line shown in Figure 5.2 with $m = 10$. The vector specifies the existence of 3 available time-units starting from the point 0, 1 available time unit after the point 5, and 2 time units after the time point 7.



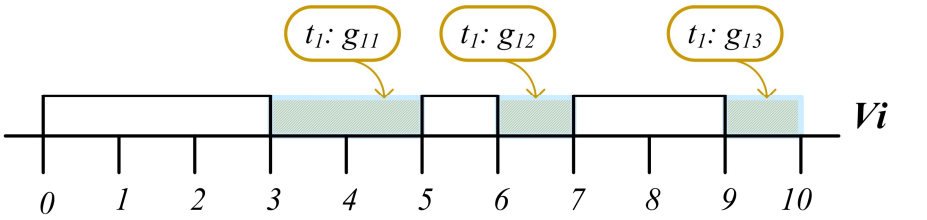
$Vi=(<0,3>,<1,2>,<2,1>,<3,0>,<4,0>,<5,1>,<6,0>,<7,2>,<8,1>,<9,0>)$

Figure 5.2: Vector of time-units availability for agent *Ai*.

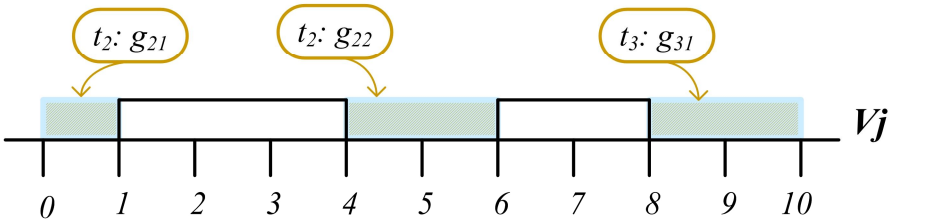
Assuming that m (the total number of time units per agent) is constant for all agents, given two time vectors Vi and Vj, their aggregation, denoted by $agg(Vi,Vj)$, is a vector that comprises elements $<j_{ak},s_{ak}>$, with $0 \leq k \leq m$, such that, given the elements $<j_{ik}, s_{ik}>$ and $<j_{lk}, s_{lk}>$ of Vi and Vj respectively, with $j_{ik} = j_{lk}$, *then* $j_{ak} = j_{ik} = j_{lk}$, and $s_{ak} = max(s_{ik},s_{lk})$. For instance, given the tuples $<j_{i4}, s_{i4}> = <4,4>$ and $<j_{j4}, s_{j4}> = <4,0>$ the corresponding tuple in the aggregation is $<j_{a4}, s_{a4}> = <4, max(4,0)> = <4,4>$. In cases there are consecutive time points with non-descending positive availabilities then all the non-overlapping availabilities are added and applied to the time points in a descending sequence (Figure 5.3). This type of aggregation can generally be applied to any type of resources that can be committed to a single request.

Figure 5.3 shows an example for aggregating the vectors *Vi* and *Vj* of two agents *Ai* and *Aj* respectively with *m=10*. The vectors are depicted as time lines. More precisely, agent *Ai* has committed to allocate the intervals *(3,5), (6,7)* and *(9,10)* to three subtasks of a complex task (t_1). Similarly, the agent *Aj* has committed to allocate the intervals *(0,1),(4,6)* to the two subtasks of complex task t_2 and *(8,9)* to another atomic task (t_3).

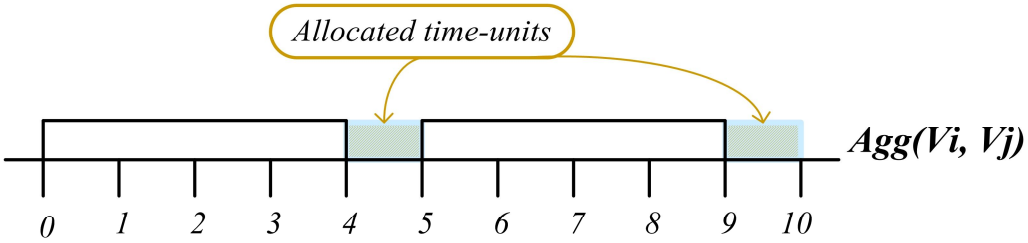
The resulting vector *(agg(Vi,Vj))* shows the availability of the two nodes as a whole, without distinguishing the availability of each node. For instance, if the availability after time unit *2* is calculated, we have $<j_{i2}, s_{i2}> = <2,1>$ and $<j_{j2}, s_{j2}> = <2,2>$. Consequently $<j_{agg2}, s_{agg2}> = <2, max(1,2)> = <2,2>$.



Vi=(<0,3>,<1,2>,<2,1>,<3,0>,<4,0>,<5,1>,<6,0>,<7,2>,<8,1>,<9,0>)



Vj=(<0,0>,<1,3>,<2,2>,<3,1>,<4,0>,<5,0>,<6,2>,<7,1>,<8,0>,<9,0>)



agg(Vi, Vj)=(<0,4>,<1,3>,<2,2>,<3,1>,<4,0>,<5,4>,<6,3>,<7,2>,<8,1>,<9,0>)

Figure 5.3: Vectors Vi , Vj and their aggregation.

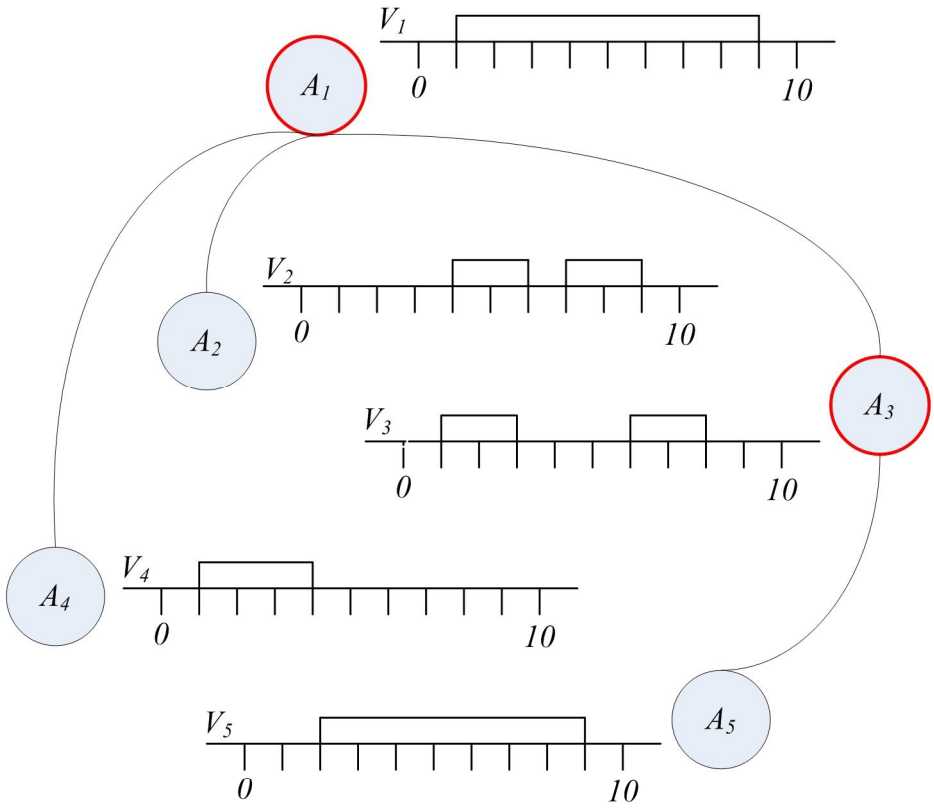
For time point *5* the situation is slightly different. Availability is $<j_{i5}, s_{i5}> = <5,1>$ and $<j_{j5}, s_{j5}> = <5,0>$. Consequently $<j_{agg5}, s_{agg5}> = <5, max(1,0)> = <5,1>$. Similarly, $<j_{agg6}, s_{agg6}> = <6, max(0,2)> = <6,2>$ and $<j_{agg7}, s_{agg7}> = <7, max(2,1)> = <7,2>$. Finally $<j_{agg8}, s_{agg8}> = <8, max(1,0)> = <8,1>$. Availabilities sum up to *1+2+2+1 = 6* and there are *2* overlapping time units. Therefore, we end up with $<j_{agg5}, s_{agg5}> = <5,4>$.

The aggregation of all vectors in the routing index of an agent *A* gives information about the maximum availability of any agent in $N(A) \cup \{A\}$. Specifically, given RI(A),

A updates the indices of its neighbour Ai by sending the aggregation of the vectors of the agents in $N(A) \cup \{A\} \setminus \{Ai\}$ to *Ai*.

Every time the vector that models the availability of resources in a node changes, the node has to compute and send the new vector of aggregations to the appropriate neighbours. Then, its neighbours have to propagate these updates to their neighbours and so on, until they reach nodes whose routing indices are not affected.

In order to see how time-unit availability and propagation work in an agent network we can observe Figure 5.4. There, we have a network of six agents $(A_1, \ldots, A_6)$. For simplification reasons and without loss of generality we can assume that all capabilities (for the tasks and agents) have the same value ($Cap = 1$).



V_1=(<0,0>,<1,8>,<2,7>,<3,6>,<4,5>,<5,4>,<6,3>,<7,2>,<8,1>,<9,0>)

V_2=(<0,0>,<1,0>,<2,0>,<3,0>,<4,2>,<5,1>,<6,0>,<7,2>,<8,1>,<9,0>)

V_3=(<0,0>,<1,2>,<2,1>,<3,0>,<4,0>,<5,0>,<6,2>,<7,1>,<8,0>,<9,0>)

V_4=(<0,0>,<1,3>,<2,2>,<3,1>,<4,0>,<5,0>,<6,0>,<7,0>,<8,0>,<9,0>)

V_5=(<0,0>,<1,0>,<2,7>,<3,6>,<4,5>,<5,4>,<6,3>,<7,2>,<8,1>,<9,0>)

Figure 5.4: A network of acquaintances with a snapshot of the agents' timelines.

Agents A_1 and A_3 are gateway agents. Furthermore, A_1 since it is the highest priority agent, serves as a gateway for gateway A_3 as well. Agent A_3 holds the aggregation

vector of A_3 and A_5 while A_1 hold the aggregation vector of all agents in the network. Therefore A_1 sees that there is a free time slot for every time unit from *0* to *10*. At the same time A_3 considers the time interval from time point *1* to time point *9* free. If complex tasks arrives at A_2 and A_4 and it cannot be served directly (from one agent only) it is forwarded to A_1 where it is broken down to individual subtasks and assigned to the appropriate agents. Now, the subtask recipient agent checks if it can accommodate the assigned subtasks and replies to A_1 accordingly. A similar procedure will take place if A_5 receives a complex task. The only difference is that A_5 forwards tasks to A_3, which upon failure to find a solution itself, it will forward any simple or complex task to A_1. Note that if any agent has forwarded a complex task to a gateway agent, it is not excluded as a candidate recipient of one of the complex task's subtasks.

To capture the availability of time-units in conjunction to the availability of capabilities, we extend *RIs* to multiple routing indices (*MRIs*). An *MRI* for the agent *Ai, MRI(Ai)*, has the generic form {*RIcap1,…, RIcapm*}. Each *RIcapj* represents the available resources that *Ai* can reach through neighbours that own the capability *capj*. *MRIs* not only provide an agent with the information about the temporal availability of its neighbours, but they also capture information about the available agents that own specific capabilities.

Routing indices are rather problematic when the updates of indices propagate in cycles (Crespo and Garcia-Molina, 2002): In the worst case, information about resources availability is misleading, leading to inefficient search mechanisms. Although cycles can be detected, known techniques are not appropriate for open networks where network configurations change frequently. We have managed to deal with cyclic updates of agents' indices by forcing each agent to propagate indices' updates only to neighbouring agents with higher priority. Considering that routing indices are being maintained only by gateway agents, these record the corresponding indices for the non-gateway neighbours and the aggregations of indices for gateway neighbours with lower priority. Updating the indices of gateways by aggregating the indices of their gateway neighbours with lower priority has the following effects:

- Since agents have distinct priorities, indices cannot be updated in a cyclic way, avoiding the distracting affects of cycles to searching.
- Priorities denote the "search and bookkeeping abilities" of agents: Agents with high priorities index their neighbours and guide search.
- Gateways with lower priority do not know about the indices of their gateway neighbours with higher priority. Since requests propagate from high to low priority gateway agents, some of the high priority gateways may function like "traps" since they may not know how to fulfill or propagate requests. Experiments showed that the benefits gained by avoiding cycles outweigh this disadvantage.

As agents schedule tasks, the availability of agents with certain resources and capabilities changes, causing the update of indices. The update of MRIs due to this phenomenon is done dynamically, albeit not instantly, and in parallel to the propagation of requests.

CHAPTER 6

6. Distributed Constraint Optimization

A Distributed Constraint Satisfaction Problem (DisCSP) consists of a set of agents, each of which controls one or more variables, each variable has a (finite) domain, and there is a set of constraints (Yokoo et al., 1998). Each constraint is defined on a subset of the variables and restricts the possible combinations of values that these variables can simultaneously take. A constraint that involves variables controlled by a single agent is called an inter-agent constraint. One that involves constraints of different agents is called an intra-agent constraint.

A Distributed Constraint Optimization Problem (DisCOP) is a DisCSP with an optimization function which the agents must cooperatively optimize. The DisCOP framework has recently emerged as a promising framework for modeling a wide variety of multi-agent coordination problems. Such problems are, for example, distributed planning, distributed scheduling and distributed resource allocation. We now present the way complex tasks are modeled as DisCOPs in our framework and then we describe the constraint solving techniques that have been used for efficient task allocation and scheduling.

6.1 Modeling Complex Tasks as DisCOPs

To efficiently capture interdependencies between the subtasks of a complex task, we model complex tasks as DCOPs. We assume that for each complex task t consisting of a set of sub-tasks $\{g1,,...,gk\}$ and a set of constraints Ct, a set of agents $A=\{A_1,...,A_m\}$, with $m \leq k$, is located using the gateway and routing indices searching infrastructure described in Chapter 5 . Each Aj $\in$ A has the necessary capabilities and resource availability to undertake at least one subtask. The DisCOP model is as follows:

- For each subtask $gi=<ai, starti, endi, Capi>$ there is a variable Xi controlled by an agent $Ai \in A$, corresponding to the start time of gi.
- The domain $D(Xi)$ of each variable Xi is a set $R \subseteq Ri$ of time points that agent Ai can allocate to the start time of gi (Ri is the total set of time units for the agent Ai). $D(Xi)$ includes each time point $r \in Ri$ such that all time units between r and $r+\alpha$ are available on Ai's time line.
- There is a hard intra-agent constraint between any two subtasks that are allocated to an agent Ai, specifying that the execution of the two subtasks cannot overlap. To be precise, for any two subtasks g_i and g_l allocated to Ai, with durations α_i and α_l respectively, there is a hard disjunctive intra-agent constraint $c_{il} (\in Ct) = (X_i + \alpha_i \leq X_l) \vee (X_l + \alpha_l \leq X_i)$. Note that such a constraint exists between any two subtasks (or atomic tasks in general) that are allocated to the same agent even if they belong to different tasks. This occurs when an agent participates in the allocation of more than one complex task.

- A binary precedence constraint $c_{il} \in Ct$ between two variables X_i and X_l is modeled as a soft constraint with a cost function $f_{il} : D(X_i) \times D(X_l) \rightarrow N$ which associates a cost to each pair of value assignments to X_i and X_l (similarly to (Modi et al., 2005)).

Recall that a precedence constraint specifies a temporal distance between the executions of the corresponding subtasks. For example, the constraint $X_i + \alpha_i + 5 \leq X_l$ specifies that the execution of g_l must start at least 5 time units after g_i has finished. Such a constraint may be inter-agent if the two subtasks have been allocated to different agents or intra-agent if the two subtasks have been allocated to the same agent. The cost function f_{il} is defined as follows:

$$f_{il}(d_i, d_l) = \begin{cases} 0, \textit{iff } d_i \ \& d_l \textit{ satisfy } c_{il} \\ M > 0, \textit{otherwise} \end{cases}$$

where $d_i \in D(X_i)$, $d_l \in D(X_l)$ and M is a non negative number. M is a measure of the constraint's violation "degree", defined as the minimum distance that the starting time of one of the subtasks has to be shifted on the time line of the corresponding agent in order for the constraint to be satisfied. Agents try to minimize the cost function associated with such a constraint by repeatedly changing the values of the variables they control. For example let us consider the constraint c_{il}: $X_i + \alpha_i + 5 \leq X_l$ and assume that X_i and X_l take values d_i and d_l respectively such that: $d_i + \alpha_i + 5 > d_l$. Since the constraint is violated, one of the agents that controls the variable X_i or X_l must change the value of its variable at least by M= $d_i + \alpha_i + 5 - d_l$ in order to satisfy the constraint. For example a new value for X_l that satisfies the constraint could be $d_l' = d_l$ + M. Generally, given a constraint $c: Q \leq K$, M is defined as follows:

$$M = \begin{cases} 0, Q \leq K \\ Q - K, Q > K \end{cases}$$

Similarly we can define M for other types of interdependencies. For example, given a constraint $c: Q<K$, M is defined as follows:

$$M = \begin{cases} 0, \ Q < K \\ Q - K + 1, Q \geq K \end{cases}$$

Collectively, the agents in A try to minimize the aggregated function $F(Ct)$ which is a measure of violation for all the constraints of a complex task t :

$$F(C_t) = \sum_{\forall X_i, X_l} f_{il}(d_i, d_l)$$

Note that in the general case no agent in A has complete knowledge of the function $F(Ct)$. Hence, the use of a distributed constraint optimization method by the agents in

A is a necessity. In this thesis, we consider summation as the aggregation operator for the optimization functions, but this is not a requirement.

6.2 Scheduling Complex Tasks

To achieve efficient task allocation and scheduling in dynamic settings, where several requests may enter the system simultaneously, it is essential that a fast, scalable, and dynamically adaptable method is used. Several distributed constraint satisfaction and optimization techniques have been proposed in the literature. For example, there exist simple distributed local search methods, such as the distributed stochastic algorithm (DSA) (Fitzpatrick and Meertens, 2001), and the distributed breakout algorithm (Hirayama and Yokoo, 2005). These are fast and scalable methods but as a downside they are incomplete.

On the other hand, a number of complete distributed CSP and DisCOP algorithms have been proposed, such as Adopt (Modi et al., 2005), DPOP (Petcu and Faltings, 2005b) and their extensions (Davin and Modi, 2006, Atlas and Decker, 2007, Yeoh et al., 2008). These methods guarantee that the computed results are optimal and have been shown to perform satisfactorily in terms of run time for agent networks of up to medium size. However, they cannot yet handle large networks. As will be explained in detail below, in the context of this work we do not consider very large DisCOPs. Although agent networks may consist of many hundreds, or even thousands, of nodes, it is safe to assume that complex tasks that enter the network will consist of relatively few subtasks in most practical cases. Hence, by modeling each complex task as a separate DisCOP, the sizes of the DisCOPs that are generated can be efficiently handled by methods such as Adopt.

The task allocation and scheduling approach we follow in this paper combines distributed constraint optimization with dynamic agent team reorganization. Concerning the scheduling of tasks via constraint optimization, we have compared two different approaches; the first one is an incomplete local search method based on DSA, and the second one is a complete optimization algorithm based on Adopt. Before going into further details in the subsections that follow, let us briefly explain our approach.

Given a complex task t, and assuming that there is a set of agents to which the atomic subsidiary tasks of t have been allocated (i.e. a Potential Teammates' Network – PTN), the agents in the PTN apply a DisCOP algorithm. If a solution is found then t is considered as successfully scheduled and the agents in PTN form a network of teammates. If no solution is found, or a time-out occurs, then PTN self-organizes. That is, one or more of the agents in the PTN exit the PTN, tasks are being allocated to new agents that have the appropriate capabilities, and thus, these agents join the PTN. Then, the agents in the new PTN try again to schedule the subtasks of t using a DisCOP algorithm. This process continues until t is successfully scheduled or a time-out occurs, in which case t is considered as a failed request.

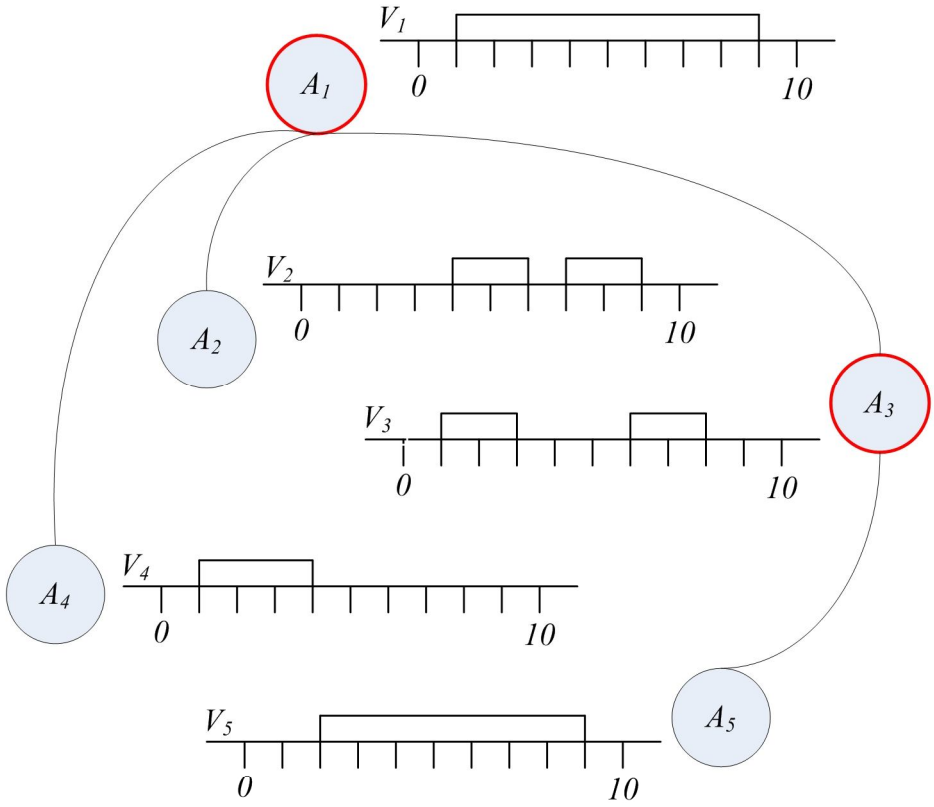
Before describing in detail in the two DisCOP methods that have been implemented in this thesis, let us provide an illustrative example.

Example 6.1. Consider a complex task with 3 individual subtasks *ti = {gi1, gi2, gi3}* that enters the agent network. For the individual sub-tasks we can assume the following:

gi1=<ai1=1, starti1 ≥0, endi1<5,Capi1=1>
gi2=<ai2=2, starti2≥ endi2, endi2<8,Capi2=1>
gi3=<ai3=2, starti3≥ endi3, endi3<10,Capi3=1>

Let us also assume that we have an agent network *G=(N,E)* of 5 agents defined as follows and that t_i uses A_1 to ender the network:

N = {A1, A2, A3, A4, A5}.
E = {(A1, A2), (A1, A3), (A1, A4), (A3, A5)}



V_1=(<0,0>,<1,8>,<2,7>,<3,6>,<4,5>,<5,4>,<6,3>,<7,2>,<8,1>,<9,0>)

V_2=(<0,0>,<1,0>,<2,0>,<3,0>,<4,2>,<5,1>,<6,0>,<7,2>,<8,1>,<9,0>)

V_3=(<0,0>,<1,2>,<2,1>,<3,0>,<4,0>,<5,0>,<6,2>,<7,1>,<8,0>,<9,0>)

V_4=(<0,0>,<1,3>,<2,2>,<3,1>,<4,0>,<5,0>,<6,0>,<7,0>,<8,0>,<9,0>)

V_5=(<0,0>,<1,0>,<2,7>,<3,6>,<4,5>,<5,4>,<6,3>,<7,2>,<8,1>,<9,0>)

Figure 6.1: A network of acquaintances for Example 6.1.

The agent network and the availability of resources for each agent are shown graphically in Figure 6.1. For simplification reasons and without loss of generality we can assume that all capabilities (for the tasks and agents) have the same value (Cap = 1).

In Figure 6.1 there are 2 gateway agents (A_1, A_3) and the priorities between them are set according to their index. In this example higher index values result to higher priority (i.e. A_1 has lower priority than A_3 since its index is lower). This is done in order to demonstrate that priority values do not alter drastically the systems behavior. Consequently only A_3 has a complete view of all the network's resources. A_1 can only record information about the availability of itself and of agents A_2, A_4. Routing indices are not shown as they are simple to compute. The availability of each agent is also shown in vector form.

Assuming that the gateway agent A_1 has received the task request, it performs a search in its immediate neighbourhood in order to find agents that can provide the requested resources for the satisfaction of the task. As already stated, resources belonging to A_3 are completely invisible to A_1. This is straightforward since the priority of the latter is lower than that of the former. Consequently, A_1 will try to allocate the task to itself and/or to agents A_2, A_4. One possible assignment is $\{(A_1, g_{i1}), (A_2, g_{i2}), (A_4, g_{i3})\}$ where each tuple denotes a possible sub-task to agent allocation. Doing so, the three agents A_1, A_2, A_4 form a PTN and will jointly try to schedule the three subtasks.

The next step after the formation of the PTN consists of the construction of a new overlay network according to the inter-agent constraints among the agents in the PTN (let is call this type of overlay networks "c-overlay"). As far as the DisCOP algorithms are concerned, two agents $<Ai, Aj>$ are considered neighbours in the c-overlay network if there is a constraint between their variables. As with the overlay of gateways (with which c-overlay networks should not be confused), the c-overlay network is constructed on-demand and dynamically depending on the existing inter-agent constraints. This is done by any PTN that has been formed in order to schedule a complex task. Such a network is either reformed when the PTN dynamically re-organizes, or it is dissolved when PTN is dissolved. The later happens when the complex task has been either allocated successfully or its allocation has failed.

Since an agent can be a member of more than one PTN that executes a DisCOP algorithm, it can simultaneously belong to many c-overlay networks. But since each DisCOP is for scheduling a different task, messages created for a single c-overlay network cannot travel on edges belonging to another such network.

Returning to Example 6.1, once the three agents in the PTN have received the subtasks allocated to them, they will try to jointly schedule the atomic subsidiary tasks of the complex task by modeling it as DisCOP. In this case, the c-overlay network shown in Figure 6.2 will be formed.

Since there are three subtasks, there are three corresponding variables X_1, X_2, X_3 that are controlled by agents A_1, A_2, A_4, respectively. Constraints are depicted as bi-directional edges between agents. The constraints that involve one variable (unary constraints) are enforced on the domains of the corresponding variables. For example,

value 9 for agent A_1 is not included in the set of potential values for X_1, since this variable must be smaller than 5.

This is the initial state of the DisCOP solving process. Henceforth, each agent will assign values to its variables, send messages to other agents about its state, and react to incoming messages according the DisCOP algorithm used. Message exchange for the DisCOP is carried out exclusively on the c-overlay network constructed due to the constraints in the DisCOP for this specific complex task (e.g. due to the network depicted in Figure 6.1).

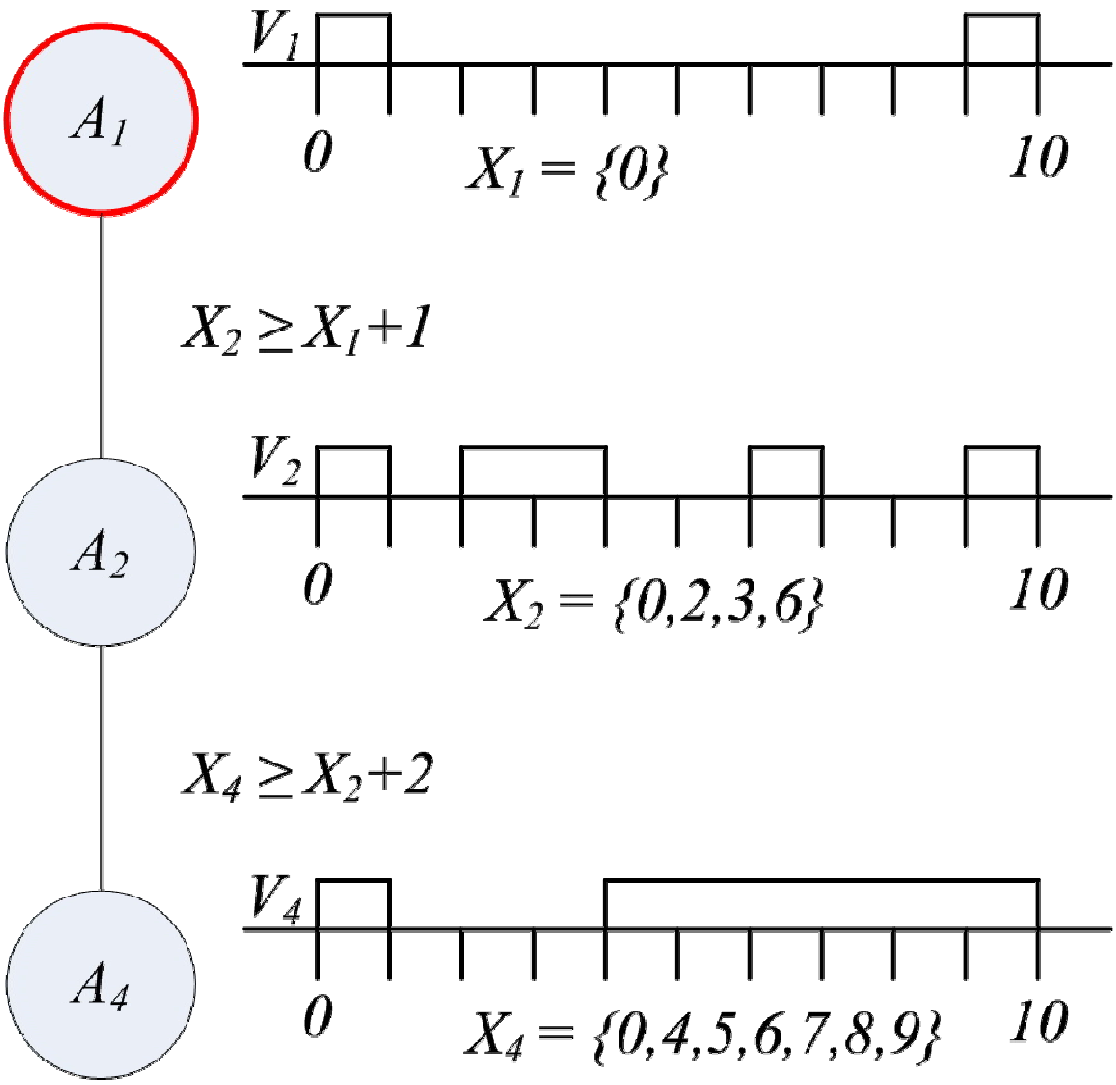


Figure 6.2: A c-overlay network for the network of agents in Figure 6.1.

6.3 Solving DisCOPs using Local Search methods

The first method for solving DisCOPs that we evaluated is a local search procedure. This is an instantiation of the Distributed Stochastic Algorithm (DSA) (Fitzpatrick and Meertens, 2001). In this method each agent may control more than one variable and the optimization criterion takes into account the cost functions of the constraints involved. As demonstrated in (Zhang et al., 2005), DSA displays good performance in

certain constraint optimization problems compared to distributed breakout. In its simplest form the local search method is based on a min conflicts hill-climbing procedure (Minton et al., 1994) without stochastic moves. Table 6.1 depicts this basic algorithm.

Let us now describe the operation of the basic algorithm. Each agent *Aj* initially assigns values to its variables so that the sum of the cost functions is minimized. This can be a dynamic process since subtasks may dynamically allocated to the agent *Aj* during the operation of the system. In other words, as soon as an agent receives a subtask *gi*, it tries to assign a value to *Xi* (i.e. find a start time for *gi*) that minimizes the sum of the cost functions. This may involve changing the assignment of other variables that Aj controls (line 6). If a new assignment of some variable X is made, this is communicated to Aj's neighbours in the corresponding c-overlay network (i.e. the agents that share constraints with Aj involving variable X). All similar assignments made by Aj's neighbours are collected and the constraint violations are recomputed. This may lead to the assignment of new values to Aj's variables (line 7). The choice of value assignments is an important issue of the algorithm. The basic algorithm makes a new assignment only if it can reduce the sum of the cost functions. Note that no single agent has complete knowledge of the cost function F(Ct) of a complex task t (unless all subtasks of t are assigned to a single agent). Therefore, each agent tries to minimize the aggregation of the cost functions for the constraints that it is "aware of". That is, the constraints that involve variables that it controls. In case no assignment that reduces the aggregation of the cost functions can be made, then the agent will not make any change to its variable assignments. In this case, the agent has reached a local optimum and will have to wait for messages from its neighbours in the corresponding c-overlay network.

1: assign values to variables so that the aggregation of cost functions is minimized
2: **while** (termination condition has not been met) **do**
3: **for** each new value assignment of a variable X
4: send the new value to agents that control a variable involved in a constraint with X
5: **end for**
6: collect the neighbours' new values, if any, and compute constraint violations and cost functions
7: choose values for the variables so that the aggregation of cost functions is minimized
8: assign the selected values to the variables
9: **end while**

Table 6.1: Basic Local Search algorithm executed by all agents

This basic algorithm may quickly reach a local optimum as it is not equipped with any technique for escaping such situations (e.g. stochastic moves). In case all agents in the team reach a local optimum, or a termination condition related to the time allowed for constraint solving is met, then, as already explained, the PTN re-reorganizes itself.

Note that local search methods comply with the requirements for speed and dynamicity, but on the other hand, because of their inherent incompleteness, may not

find optimal (or simply feasible) allocations, even if they exist. This means that some complex tasks may not be served, although there may be agents in the system that can cooperatively serve them.

6.4 Solving DisCOPs using Adopt

Alternatively, to the Local Search method we evaluated an alternative configuration of the proposed method using the complete DisCOP algorithm Adopt (Modi et al., 2005). The majority of the existing methods for DisCOP (e.g. local search methods) are not able to provide theoretical guarantees on global solution quality given that agents have to operate asynchronously. Nevertheless, we can overcome this disadvantage by allowing agents to make local decisions based on cost estimates. This approach, introduced in (Modi et al., 2005), results in a polynomial-space algorithm for DisCOPs named Adopt. Adopt guarantees a globally optimal solution. Furthermore, it allows agents to execute asynchronously and in parallel. As noted in (Modi et al., 2005) "The Adopt algorithm consists of three key ideas: a) a novel asynchronous search strategy where solutions may be abandoned before they are proven suboptimal, b) efficient reconstruction of those abandoned solutions, and c) built-in termination detection". A sketch of Adopt's operation is as follows.

6.4.1 Depth First Search trees in Adopt

First, agents form a prioritized tree structure (as in Figure 6.2). The priorities in this structure are decided after considering the constraints between variables inherent in the CSP problem, which has to be solved. In (Modi et al., 2005) the authors prioritize variables using Depth First Search (DFS) trees. Forming DFS trees is done by taking advantage of the relative independence of variables (Freuder and Quinn, 1985). Two or more variables are relatively independent if they share no constraints. Note, that these variables may be related through another variable. Nevertheless, no constraint links them directly. For instance, consider the following constraints:

C_1: $X_1 < X_2$ and C_2: $X_2 < X_3$

Variables X_1 and X_3 are relatively independent though they may be connected through variable X_2. Now, if variables X_1, X_2, X_3 are handled by an agent each, A_1, A_2, A_3 respectively the DFS tree that is formed would have each agent as a node and two edges. The first edge would be between A_1 and A_2 and the second between A_2 and A_3. Since any constraint graph can be ordered to a DFS tree (Lynch, 1996) Adopt assumes that the tree is already computed if necessary. The priority ordering is then used to perform a distributed backtrack search using a best-first search strategy.

The prioritized Depth-First Search (DFS) tree (Figure 6.2) defines parent and child relationships and of course priorities between the agents. Variable value assignments (VALUE messages) are sent down the DFS tree while cost feedback (COST messages) propagate back up the DFS tree. It may be useful to view COST messages as a generalization of NOGOOD messages from DisCSP algorithms. THRESHOLD messages are used to reduce redundant search and sent only from parents to children. A THRESHOLD message contains a single number representing a backtrack threshold, initially zero.

In our framework gateway agents have the responsibility of forming DFS trees (c-overlay networks) and start Adopt. In a DFS tree, all agents are aware of the variables they control and the constraints they carry. Furthermore, if the constraint involves another agent they know who that agent is and if it is a parent or a child agent (i.e. if the other agent has lower or higher priority in the DFS tree). Priorities in a DFS tree are used only during the solution procedure and are different from the priorities defined previously in order to decide which agents are gateways.

6.4.2 Adopt: Initialization Procedure and Backtrack

When an agent is about to start execution of the Adopt algorithm it has to initialize first (INITIALIZE procedure). To do so, the agent chooses the value that minimizes its *LB*. Then the agent backtracks through the BACKTRACK procedure. The INITIALIZE and BACKTRACK procedures that Adopt uses are depicted in Figure 6.3 (a and b, respectively). It is straightforward that after initialization every agent backtracks. Backtracking produces messages send to neighbouring agents. After that, every agent waits for incoming messages and responds to them.

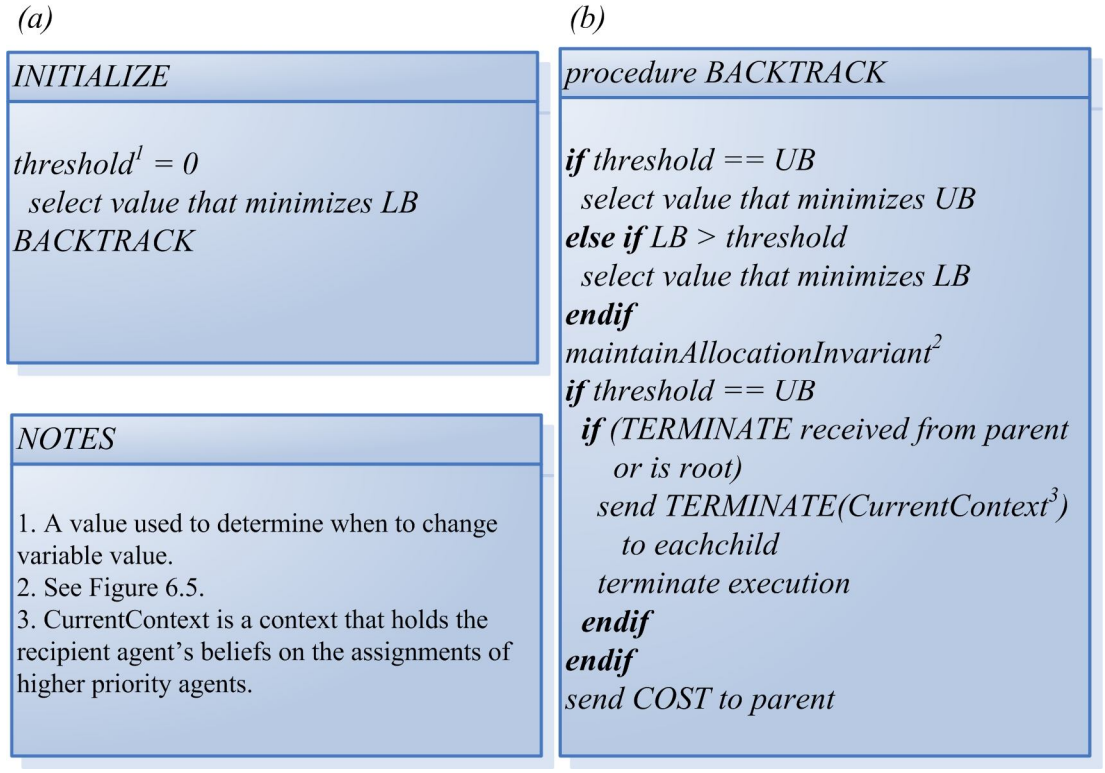


Figure 6.3: Adopt algorithm initialization procedures: (a) INITIALIZE and (b) BACKTRACK.

6.4.3 Adopt: Procedures for incoming messages

There are four kinds of messages in the Adopt algorithm. These are named COST, VALUE, THRESHOLD and TERMINATE. Messages can be sent or received by every agent. When a message is sent, the sender agent must give the message a certain format, in order to be correctly identified by the recipient agent.

VALUE messages are from parents to children, which upon receiving new value assignments to variables that parent agents control, re-evaluate their own assignments.

In case their variable assignments are already consistent there is nothing to be done. Otherwise they choose the values that minimize their cost and sent a COST message to their parents. There is also the termination detection mechanism which detects if the problem is solved or no solution can be found. When the root agent detects that the algorithm should be stopped it sends TERMINATION messages to its children. This message is forwarded all the way down the DFS tree and all agents terminate the algorithm. THRESHOLD messages are send when parent agents detect that due to constraint violation child agent should change their variable assignments.

To be more accurate, during the whole procedure and based on the current available information, each agent keeps on choosing the best value for its variables. To do so, each agent upon receiving a message informing it of a value of a higher priority agent always chooses for its own variable the value that minimizes its *lower bound* (*LB*) as defined in (Modi et al., 2005). When agents change a variable assignment, they inform their children by sending a VALUE message to them. A VALUE message has the form VALUE(*var*, *value*). Therefore if agent A_i want to inform agent A_j that variable x_i has changed its value to *d*, A_i sends to A_j the message VALUE(x_i,*d*).

Each agent in the DFS tree keeps its own view of the values other agents have chosen for their variables. A partial solution in Adopt is called a *context*. A *context* has the form:

$$\{(x_i, d_i),...,(x_j, d_j)\}$$

In a *context,* each pair represents a value assignment to a variable controlled by an agent. Each variable cannot appear more than once in a valid *context*. Two *contexts* are compatible if they contain the same assignment for each variable. The *context* that consists of an agent's view of the assignments the higher priority agents in the DFS tree have made is called the agent's *CurrentContext*.

Every agent computes its *lower bound (LB)* as the sum of its *local cost* and the sum of *lower bounds* its descendants have communicated to it. Specifically, for an agent A_i that controls variable x_i with domain D_i, we name *local cost* the sum:

$$\delta(d_i) = \sum_{x_i, d_j \in CurrentContext} f_{ij}(d_i, d_j)$$

The *local cost* $\delta(d_i)$, is the sum of the values that the *cost function* produces when x_i chooses d_i and the assignments of its ancestors are those of the *CurrentContext*. Agents exchange computed *lower bounds* through COST messages. An agent can send a COST message to its ancestors only. Similarly, agents receive COST message only from their children. A COST message has the form COST(x_j, *d, context*, *lb*, *ub*). If the agent that controls variable x_j (A_j) has sent a COST message to the agent that controls x_i (A_i) then the *context* in the message is the *CurrentContext* that A_j holds.

Additionally, *lb* and *ub* are A_j's *lower bound* and *upper bound* respectively. Now, A_i stores *context* as *context(d,*x_j*) ls* as *lb(d,*x_j*)* and *ub* as *ub(d,*x_j*)*. That means that variable agents A_i and A_j have the same view for the variables their common ancestors have assigned to their variables when A_j sent the COST message to agent A_i. Then, A_i

computes its own *lower* and *upper bound*, which are the *lower* and *upper bound* for the subtree rooted at A_i when $x_i = d$:

$$LB(d) = \delta(d) + \sum_{x_j \in Children} lb(d, x_j)$$

Since $d \in D_i$ there is an *LB* value for every value in D_i. Therefore, *LB* the lower bound for the sub-tree rooted at A_i is:

$$LB = \min_{d \in D_i} LB(d)$$

Similarly:

$$UB(d) = \delta(d) + \sum_{x_j \in Children} ub(d, x_j)$$

$$UB = \min_{d \in D_i} UB(d)$$

Adopt uses its *lower bound* as backtrack threshold for each agent. When an agent knows from previous search experience that *LB* is a *lower bound* for its subtree, it should inform the agents in the subtree not to bother searching for a solution whose cost is less than *LB*. Agents compute *thresholds* in the same manner lower and upper bounds are. For a value $d \in D_i$:

$$threshold = \delta(d) + \sum_{x_j \in Children} t(d, x_j)$$

Every agent initializes its *lower bound* and *threshold* values to *0* (zero) and the *upper bound* value to *inf* (infinite). At any given time, the constraint $LB \leq threshold \leq UB$ must be satisfied for every agent. In case it is violated, the agent changes it variable assignment. Bound intervals track the progress towards the optimal solution. This is the core of the built-in termination detection mechanism. A bound interval consists of both a *lower bound* (*LB)* and an *upper bound* (*UB*) on the optimal solution cost. When the size of the bound interval shrinks to zero (the lower bound equals the upper bound or *LB == UB*) the cost of the optimal solution has been determined and agents can safely terminate when a solution of this cost is obtained. This technique makes the algorithm highly efficient. Furthermore, it requires only polynomial space in the worst case.

Assume a minimal DFS tree of two agents (A_i, A_j) where A_i is the parent and A_j the child. A_i controls variable x_i with domain $D_i = \{1,2,3\}$. A_j controls x_j and its domain is $D_j = \{2,3\}$. Since agents control one variable each we can use an agent's or variable's name interchangeably without any difference. We define the cost function as:

$$f_{ij} = \frac{d_i^2}{d_j}$$

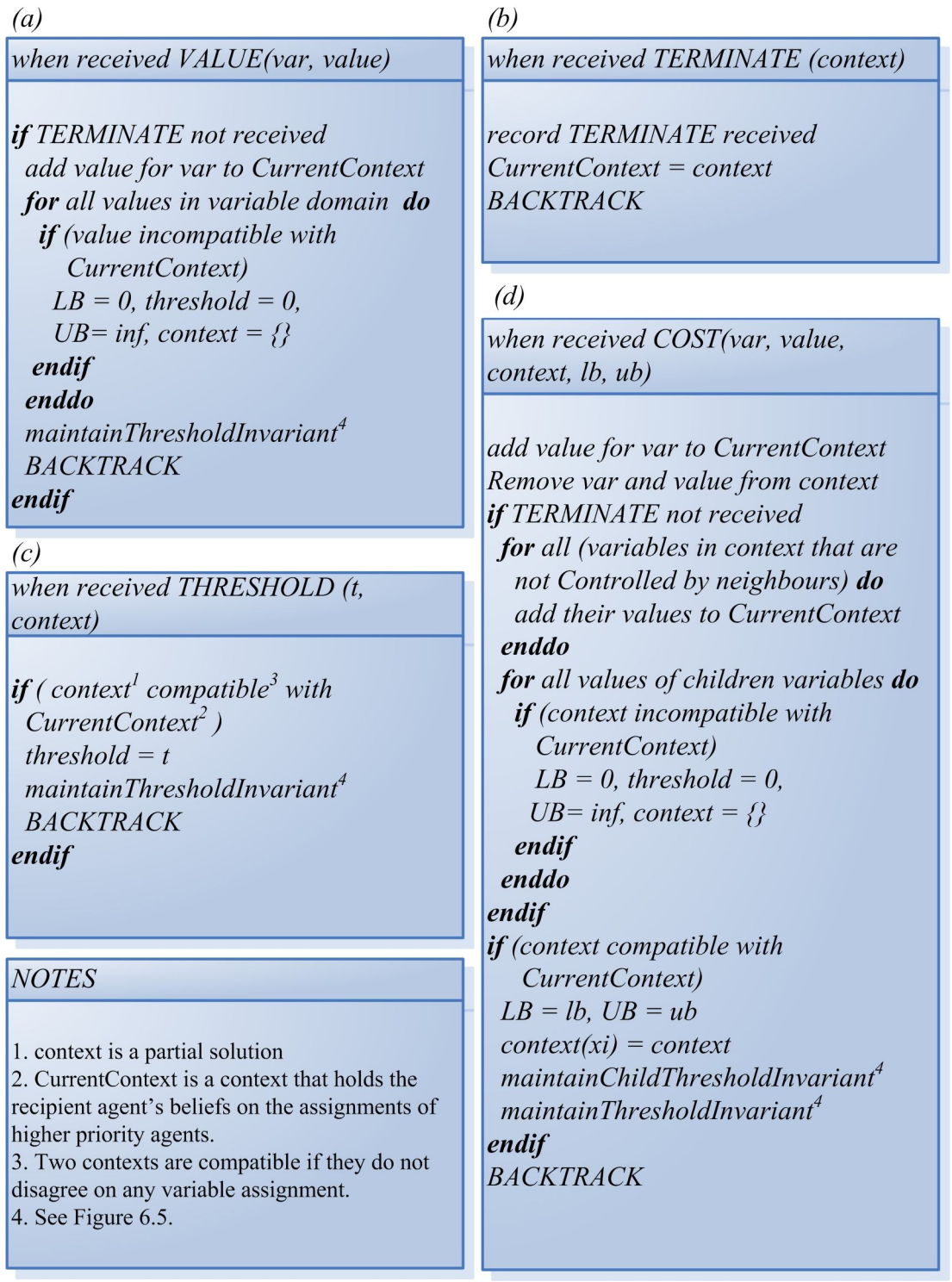


Figure 6.4: Adopt procedures for receiving messages: (a) when received VALUE, (b) when received TERMINATE, (c) when received THRESHOLD and (d) when received COST.

After initialization (Figure 6.3.a) agent A_i has set $lb(d,x_j) = 0$ and $ub(d,x_j) = inf$ (infinite). A_j need not do so since it has no children. Agents A_i and A_j have also set $LB = threshold = \delta(d_i)$, where d_i is the domain value that minimizes LB and $threshold$. They also set $UB = inf$. For illustration purposes, let us start by assuming that A_j receives VALUE(x_i,2). In a real case, A_i after backtracking (BACKTRACK

procedure, Figure 6.3.b) would choose for x_i the value that would minimize its *LB*. After receiving the VALUE message (Figure 6.4.a), A_j after completing the VALUE procedure (Figure 6.4.a) enters the BACKTRACK procedure (Figure 6.3.b) and eventually goes on to re-compute its *lower bound*. Note that there are many possibilities pertaining to the timing of the arrival of the VALUE message. Let us consider that it came right after A_j has completed initialization and has backtracked for the first time.

For $d_j = 2$ we have:

$LB(2) = \delta(2) + \sum lb(2, x_q) x_q \in Children = 4/2 + 0 = 2.$

For $d_j = 3$ we have:

$LB(3) = d(3) + \sum lb(3, x_q) x_q \in Children = 4/3 + 0 = 4/3.$

At that moment for A_j, $LB = min(LB(2),LB(3)) = LB(3) = 4/3$. Since $LB(3) < LB(2)$, A_j chooses to assign $x_j = 3$. At the same time it sets its *threshold* to *4/3*. It computes $UB = UB(3) = 4/3$ and sends to A_i the message $COST(x_j,2,4/3,4/3)$. When A_i receives the COST message it records $context(2,x_j)$, $lb(2,x_j)$ and $ub(2,x_j)$ and re-evaluates its value picking $x_i = 1$ as this value minimizes its *LB* to $LB(1)=1/3$. Also, its *upper bound* is now $UB(1) = 1/3 + 4/3 = 2$. A_i sends this value to A_j. A_j does not change its value but now it has re-computed its bounds and has set $UB=LB=1/3$. Upon receiving $COST(x_j,1,1/3,1/3)$ A_i re-computes its *LB* and *UB*. $LB = 1/3 + 1/3 = 2/3$ and $UB = 1/3 + 1/3 = 2/3$. Now for the root agent $LB=UB$ so A_i terminates and sends a terminate message to A_j.

Each time an agent receives a message it identifies its type and consequently starts the procedure responsible for appropriately handling it. In accordance to the types of messages that can be produced and sent, Adopt uses four procedures to process incoming messages (Figure 6.4). In the previous example, for brevity, we focused on the various calculations that agents must do after receiving messages. All types of messages result in changing variable values, bounded intervals and thresholds and potentially produce new messages. It is obvious that the algorithm is asynchronous and the volume of exchanged messages can be quite large. Nevertheless, space requirements remain relatively low.

6.4.4 Adopt: Procedures for updating Backtrack Thresholds

Adopt uses Backtrack Thresholds to reconstruct abandoned solutions. Efficient reconstruction of abandoned solutions, when needed, is a major advantage of the algorithm. It not only increases its speed but reduces the required space, as well. The basic idea is that a parent agent usually knows the (from previous searches) the lower bound of its subtree. Consequently, it informs its children not to search for a solution with lower cost than *LB*. The threshold value serves as a measure for the solution's optimality. Threshold is always less than or equal to the cost of the optimal solution. Therefore, the optimal solution can always be found.

There are the three procedures Adopt uses to update Backtrack Thresholds. These are described in Figure 6.5.

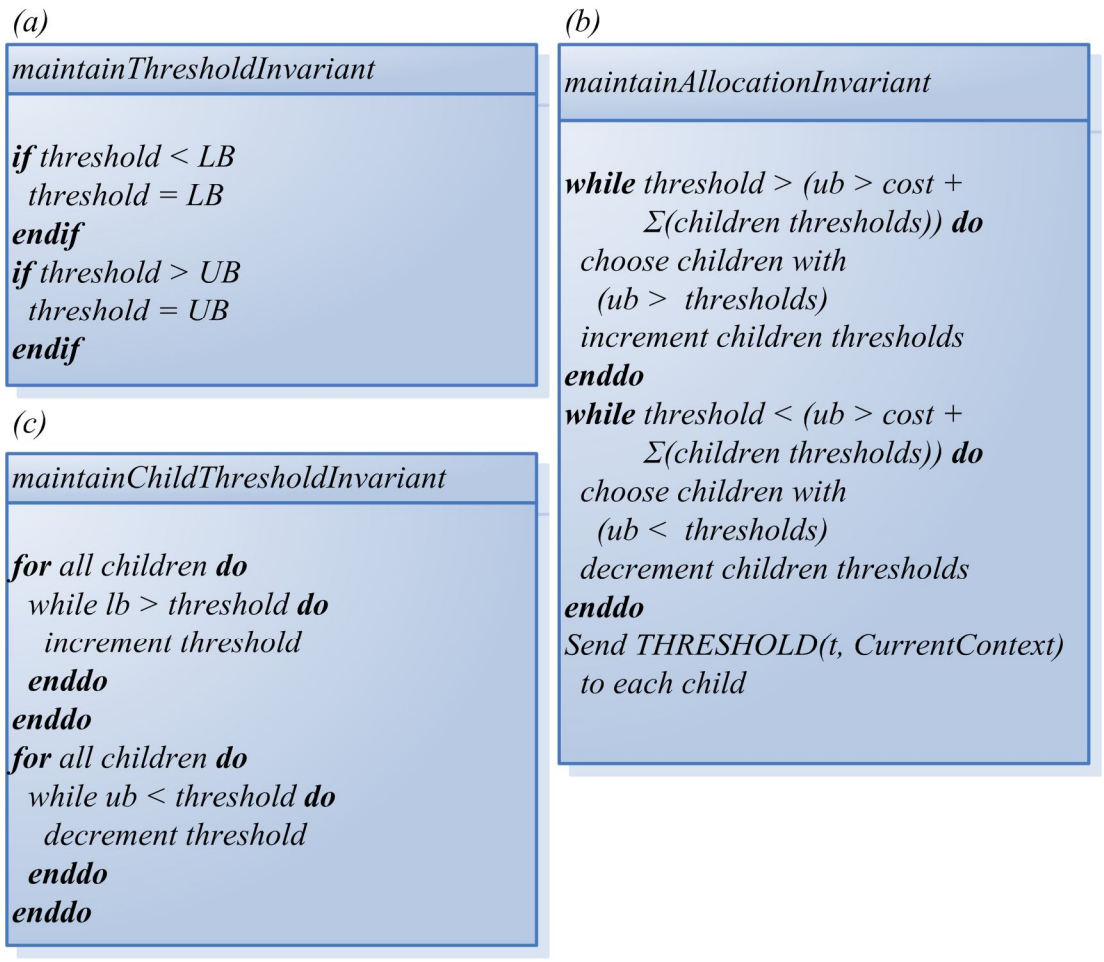


Figure 6.5: Adopt procedures for updating Backtrack Thresholds.

6.4.5 Adopt: Example of Algorithm Execution

For the agent group in Example 6.1 we have a complex task (t_i) that enders the agent network at agent A_1. Agent A_1 is a gateway agent. Therefore after checking its routing indices decides to form the DFS tree depicted in Figure 6.2. Now agents A_1 A_2 and A_4 will try to accommodate the task using Adopt.

As cost function we use the constraints between the variables. If the constraint is satisfied then cost is zero. In case of a violated constraint, the cost is the difference between the second and the first part of the constraint. This means that between X_1 and X_2 for instance, if $X_2 = 0$ and $X_1 = 0$ the cost is $max((X_1 + 1 - X_2), 0) = max((0 + 1 - 0), 0) = max (1,0) = 1$. In case X_2 changes it value to $X_2 = 3$ then cost is $max((X_1 + 1 - X_2), 0) = max((0 + 1 - 3), 0) = max (-2,0) = 0$.

Since Adopt is asynchronous, there are multiple possible event combinations until a solution can be found. A possible sequence of events in order to solve the formed DisCOP could be as follows:

- Agents initialize the variables they control choosing the first values in their domain. Thus $X_1 = X_2 = X_3 = 0$ (Figure 6.6.b).
- Agent A_2 receives this value through a VALUE (Figure 6.6.b) message and re-evaluates its own value. If, for instance, it was $X_2 = 0$ this will change to $X_2 = 2$ as this is the first variable in its domain that minimizes its *LB* (Figure 6.6.c and 6.6.d). At this point for A_2 $LB(2) = \delta(2) + lb(2, X_4) = 0 + 0 = 0$. $UB = \delta(2) + ub(2, X_4) = 0 + inf = inf$.
- Then A_2 sends a COST message back to A_1 (Figure 6.6.c). Since the constraint between A_1 and A_2 is satisfied the COST message is $COST(X_2,0,0,inf)$.
- A_2 also sends a VALUE message to A_4 (Figure 6.6.d).
- A_1 upon receiving the COST message keeps its value set to $X_1 = 0$, since it cannot further minimize its *lower bound*.
- A_1 cannot send a TERMINATION message to A_2 since for A_1 $LB(0) \neq UB(0)$ and subsequently these agents continue their DisCOP solution procedure by responding to incoming messages.
- Under the same procedure agent A_4 eventually receives the VALUE message A_2 has sent. A_4 updates its *CurrentContext* to $\{(X_2,2)\}$and backtracks (Figure 6.7.a).
- A_4 re-evaluates its own value, setting $X_3 = 4$. Now for A_4 $LB(4) = \delta(4) = 0$ and $UB(4) = \delta(4) = 0$. Therefore, A_4 (Figure 6.67a) sends to A_2 the message $COST(X_4,2,0,0)$.
- A_2 receives the COST message and since the its *context* is compatible with the *CurrentContext* A_4 has included in the COST message it registers $lb(2, X_4) = 0$ and $ub(2,X_4) = 0$.
- A_2 does not change its value since its LB is minimal but re-evaluates its upper bound which now is $UB(2) = \delta(2) + ub(2, X_4) = 0 + 0 = 0$.
- Now at the end of the backtracking procedure A_2 sends to A_1 (Figure 6.7.a) the message $COST(X_2,0,0,0)$. Note that in A_2 now is $LB = UB$ but A_2 cannot send TERMINATE to A_4 since it has not received TREMINATE itself from A_1, the root agent.
- When A_1 receives the new COST message registers $ub(0,A_2) = 0$ and changes its upper bound to $UB(0) = \delta(0) + ub(0,A_2) = 0 + 0 = 0$. Now at the root agent $LB = UB$.
- Therefore, A_1 sends TEMINATE to A_2, and stops itself (Figure 6.7.b).
- Now A_2 after having received TERMINATE from its parent sends TERMINATE to A_4, and stops itself as well (Figure 6.7.c).
- A_4 since it has no children stops itself too.
- The set of all assignments for each variable at this moment $(\{(X_1,0),(X_2,2),(X_4,4)\})$, form the solution to the initial DisCOP (Figure 6.7.d).

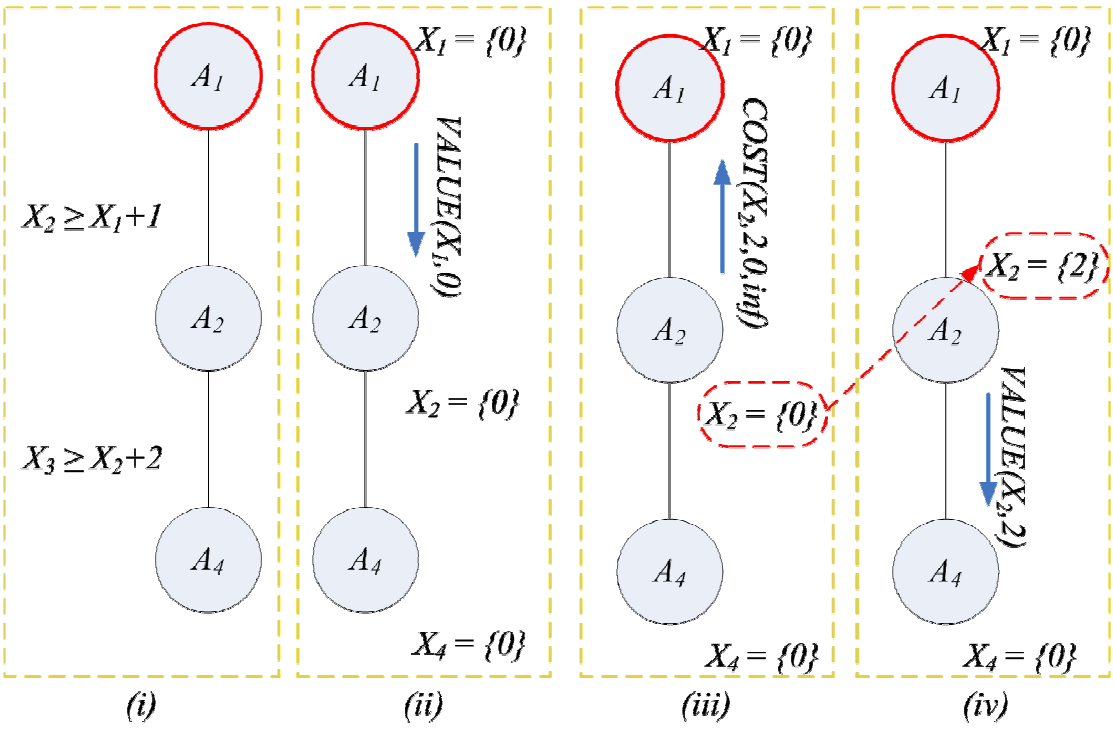


Figure 6.6: Adopt solution procedure: a possible message exchange sequence.

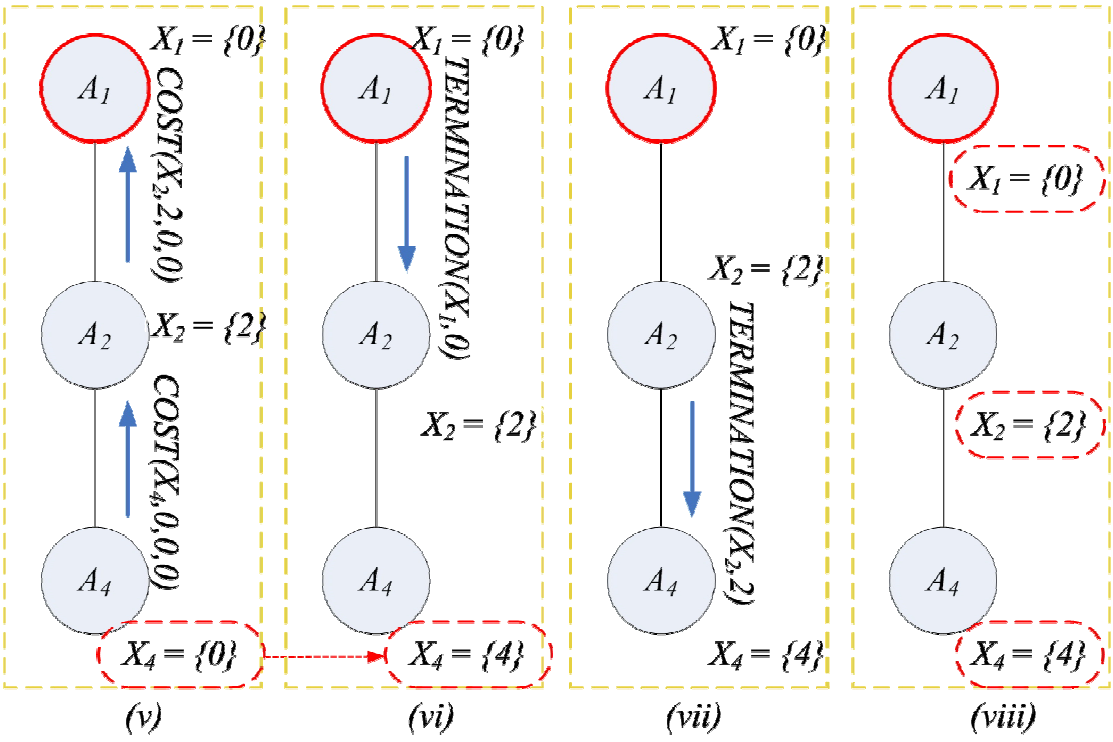


Figure 6.7: Adopt solution procedure: a possible message exchange sequence.

CHAPTER 7

7. Searching, Task Allocation and Scheduling

This chapter describes the interplay of the methods for the scheduling of tasks, with the methods for the (re-) formation of teams in large networks of agents. Special emphasis is given to the searching of agents and to the allocation of tasks to agents, forming a network of potential teammates (PTN). Generally, in cases where systems have to allocate and schedule joint activities that involve temporally interdependent subsidiary tasks, the process involves at least (a) searching: finding agents that have the required capabilities and resources, (b) task allocation: allocating tasks to appropriate agents, and (c) scheduling: constructing a commonly agreed schedule for these agents. Each of these issues has received considerable interest in the literature. Here we present a framework that integrates the aforementioned procedures.

7.1 Agent Network Organization and Arrangement

The primary task in a network of acquaintances *AN*, is to organize itself into a network where a set of agents form a connected overlay sub-network of "gateways" (*GN)*. Each time a change occurs in *AN* (due to uncontrollable events), agents may need to reorganize themselves forming a new *GN*. Self-organization happens by means of agents' local criteria using the algorithm explained in Chapter 5 for the computation of dominating nodes.

Given an arbitrary *GN*, each of the non-gateway agents connects to at least one gateway agent. To facilitate searching and maintenance of routing indices, gateway agents maintain routing indices for the resources and capabilities available to non gateway neighbours. Moreover, every gateway agent in *GN*, stores aggregated indices of its gateway neighbours with lower-priority. This forms an aggregated and approximate view of the network state, resulting in a jointly fully observable setting (Goldman and Zilberstein, 2004).

Gateways' views (i.e. routing indices) are maintained by means of capability-informing and resource-informing tokens. Requests concerning atomic or complex tasks enter the agent network in an arbitrary fashion. Any agent can be considered as an entry point for a demand over the network's resources.

This dynamic self-organizing searching infrastructure supports the formation of teams for the performance of joint activities: Given a request for a joint task *t* originated by an agent in the network, then all its sub-tasks $\{g_1,..,g_i,...,g_k\}$ must be allocated to the appropriate agents, i.e. the agents that have the resources and the capabilities to perform each subtask. The search for the appropriate agent for each of these atomic tasks proceeds as it will be described in the remainder of this chapter. The appropriate agents (i.e. have the required capabilities and resources) form a logical network of potential teammates (PTN), who jointly try to schedule their activities with respect to

the constraints associated to *t*, conjunctively with constraints that must hold for their other activities.

This is done by means of one of the DisCOP algorithms of Chapter 6 (DSA or Adopt). In case they are not successful in forming a common schedule, and depending on the violated constraints, they reform the PTN until a team is formed successfully (i.e. a team that has successfully scheduled all subsidiary tasks). There is a maximum number of PTNs that are formed for each complex task. A time-to-live (TTL) parameter is attached to every complex task and is decreased each time the PTN formed to find resources for the complex task in discussion fails to do so. If the TTL factor becomes zero then request for the joint activity to accommodate the task expires. Depending on the DisCOP algorithm used, failure in a PTN occurs either when the algorithm determines inconsistency (in the case of Adopt) or is trapped in a local minimum (in the case of DSA).

7.2 Approaches to Searching and Task Allocation

Upon the arrival of a complex task t*i*, one of the following two cases holds:

- *Aj* is a gateway agent, then depending on whether ti is a complex or an atomic task, it tries to locate the appropriate agent(s).

- *Aj* is a non-gateway agent. In that case we have considered two possible modes of operation:

  - **Inactive non-gateway mode:** the non-gateway agent immediately forwards the tasks to a gateway agent. In this case, the complex task is forwarded to the one-hop away gateway agent that has the highest priority among the gateways covering *Aj*.
  - **Active non-gateway mode:** The non-gateway agent performs a quick local placement effort, checking whether it can satisfy the request itself. According to this mode, the non-gateway *Aj* checks whether its own resources and capabilities can satisfy all requirements concerning t*i*. In the case of a single atomic task, the task start time, end time and actual demand are the sole parameters taken into consideration. Similarly, in the case of a complex task, *Aj* will check if it has the time resources to accommodate all the subtasks, while respecting the constraints between them. Since the agent has a clear view of the whole task, this is straightforward. At this time point, *Aj* being aware of the task requirements can decisively conclude whether it is capable to satisfy these requirements. Note that in this case there is no need to formulate a DisCOP for the given task. If the agent decides that it can successfully accommodate the requested task, it updates its timeline appropriately to reflect the current situation. The task is marked as satisfied and *Aj* continues receiving requests from other agents. In case *Aj* decides that it cannot accommodate the requested task using only its own resources, it will send this task to one of its gateway agents. As in the inactive non-gateway mode, it will send the task to the one-hop away gateway agent with the highest priority.

Comparing the two alternatives, we expect the active mode to speed up the system as some requests will be immediately handled by the agents used as entry points. On the other hand, we expect that the inactive mode will result in more efficient allocation as this process will be handled only by the gateway agents who, through their routing indices, have a better view of the agents' availability.

Henceforth we assume that the complex task is in the hands of a gateway agent, either because this agent has been used as the entry point to the system, or because the non-gateway agent that first received the request has sent it to a gateway agent. This may have happened either immediately, or after the non-gateway agent has decided that it cannot satisfy the task requirements on its own. In the sub-sections that follow we specify two different methods for searching and allocating tasks.

We have tried two different approaches to the process of PTN reformation that are detailed below in Sections 7.3 and 7.4. Briefly, in the first approach reformation consists of forwarding the whole complex task to another gateway agent. In the second approach, the request originator asks one (randomly selected) agent involved in a constraint violation to release its subtask and then propagates the request for this subtask. In the first approach, the new PTN formed may involve a completely different set of agents, while in the second approach the new PTN involves only one different agent than before. Reformation of a PTN may proceed as long as the corresponding joint task exceeds its TTL. In this case the task is considered as unsatisfied.

We have to point out that agents are allowed to reconsider their existing schedules when they face requests for participation in new joint activities as long as they have not committed their resources to already successfully allocated tasks. That is, if an agent is involved in a PTN that has not yet been resolved and at some point the agent joins another PTN (i.e. it now participates in two PTNs at the same time) then it can try to accommodate all subtasks that have been assigned to it by making the necessary shifts in its schedule. However, successfully allocated tasks are cannot backtracked in order to find a better overall allocation after a future search. The approach followed in this work, in its current design and implementation, is tuned to greedily try and accommodate incoming requests in the best possible way. Once agents commit to certain tasks, they allocate the appropriate parts of their timeline to these tasks and these commitments cannot be undone in order to accommodate requests arriving later.

We now turn our attention to the searching and allocation tasks, and describe two alternative methods (Method A and B, respectively) for achieving them.

7.3 Method A

According to this method, the gateway node A_j that has received the task will first examine whether the task can be served by any of its non-gateway neighbours (i.e. by any of the agents it covers). If the task is atomic the course of action is straightforward, meaning that A_j only searches for the first agent with adequate capabilities and resources.

In the case of a complex task A_j breaks it down into individual subtasks and derives the constraints between them. Then it searches for neighbouring agents, including

itself, that have the required resources and capabilities to accommodate subtasks. In both cases of an atomic task and a subtask of a complex task the first agent that presents the desired capabilities and available resources to serve it, is chosen to check if it can accommodate the task (atomic or part of a complex task). When all subtasks have been assigned, the agents form a PTN and subsequently, they form a new c-overlay network, as required by the DisCOP algorithm. As they are now aware of the constraints between the subtasks allocated to them, they start executing the DisCOP algorithm to determine if there is a solution.

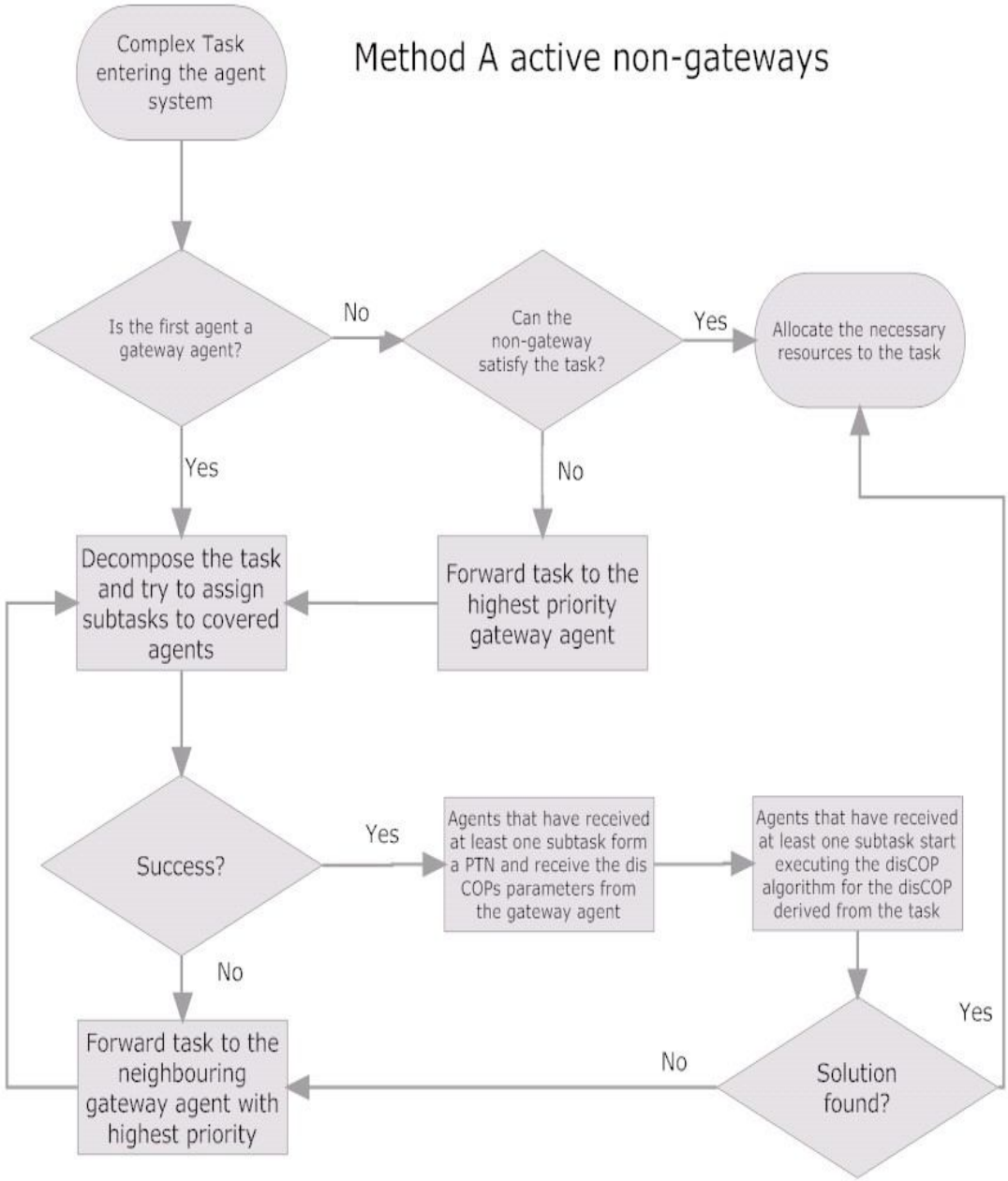


Figure 7.1: Flowchart for Method A.

If A_j cannot locate any non-gateway agent with the necessary resources and capabilities, or if the PTN formed cannot solve the DisCOP, then A_j forwards the request to its gateway neighbour with the highest priority. In case there is no gateway

neighbour with higher priority than A_j, then A_j propagates the request to all of its gateway neighbours. Since requests propagate through many different gateways, it is possible that there will be more than one agent that can serve a request. In such a case, all these agents inform the request originator about their availability, and the originator decides to whom the task shall be allocated (for example, based on their workload). The execution of method A is summarized as a flowchart in Figure 7.1.

7.4 Method B

As an alternative to the approach described above, we introduce a method that is based on a more elaborate initialization process between neighbouring gateway agents. While the main course of action is the one described in the previous paragraph when the gateway agent A_j cannot accommodate a task in its immediate neighbourhood (i.e. in the one-hop away non-gateway agents), the procedure thereafter is different. Recall that through the use of routing indices, a gateway agent is fully aware not only of its own resources but of the resources and capabilities that are available via its neighbours: Therefore a gateway agent is aware of (a) its own resources and capabilities at any given time, (b) the resources and capabilities of all non-gateway agents that exist in its neighbourhood and (c) the resources and capabilities of the agents that can be reached by the gateway agents and have lower priority while existing at a one hop distance.

Consequently, if A_j can not satisfy the requirements for resources or capabilities upon receiving a request, it tries to form a group with the "eligible" subtask recipients in order to satisfy the task. Eligible candidates are all neighbouring non-gateway agents and lower priority gateway agents. However, the gateway agent's view of its neighbours is not always accurate. This is because gateway agents have no way of knowing what task requests the other agents process at any moment, since routing indices are not updated immediately after each change in the availability of the agents. For instance, if an agent is in the process of using constraint solving techniques to decide if a previously submitted task or subtask can be accommodated by it, we may have a situation in which its own accurate view pertaining to its own resources is not in accordance with the (out of date) view that neighbouring gateway agents have.

Consequently, in method B the gateway agent A_j consults its neighbouring agents in order to decide whether a certain part of the task is going to be forwarded to one of them. Before doing this the gateway agent checks its routing indices to decide which agent seems most capable for receiving one or more subtasks. These remote agents are contacted and the final decision is made by each one of them after they have checked their own accurate view of their resources' availability. The first agent that answers positive to a request concerning a specific subtask is the one that receives it. The procedure for resolving the constraints involved cannot begin before the allocation procedure for the particular task has ended.

The major difference between Methods A and B is the gateway's 'view' during the allocation process. In Method A the gateway allocates each subtask based merely on its routing indices. On the contrary, in Method B the gateway consults its neighbours to acquire a clearer up-to-date view of their resources. The routing indices act as a first lead but the procedure continues and the gateway asks for an accurate snapshot of the potential recipients' timeline. Another point of difference is that in Method A the

gateway searches for potential candidates in its immediate neighbourhood only. In method B the request may propagate through gateway agents with lower priority. Therefore, in Method B a subtask can be allocated to an agent that resides several hops away.

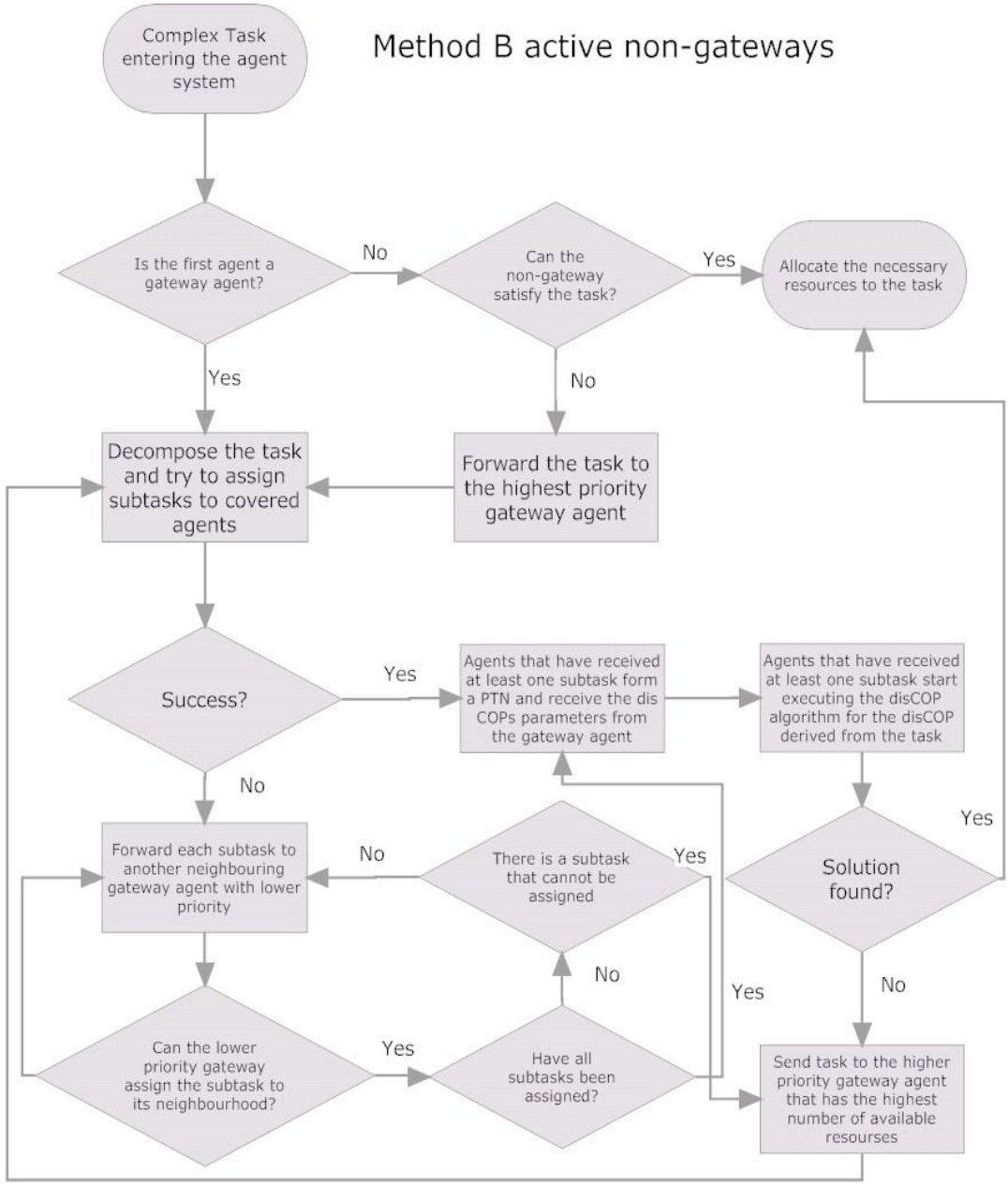


Figure 7.2: Flowchart for Method B.

At this point the status of the recipient agent (being gateway or not) can lead to alternative ways in order to accomplish the assignment process:

- If the recipient is a non-gateway agent it can respond to a request by simply checking its own view. Therefore, the answer is plainly positive or negative.
- If the remote agent is a gateway agent (note that its priority is always lower than the one of the agent that initiated the assignment procedure) and this agent cannot handle the subtask or subtasks in question, an additional step is involved before

the final answer. The receiver (lower priority gateway agent that lacks capabilities or resources) starts a similar procedure to the one already started by the requesting (sender) gateway node, trying to forward the requests in discussion to its own neighbourhood. If unsuccessful, the higher priority gateway is notified that the lower priority gateway agent cannot satisfy the request. Otherwise, the lower priority agent sends the agents' ids that can possibly satisfy each subtask. Due to gateway priorities, it is impossible for a given subtask to circle in the agent network during the allocation process. In case an agent cannot be assigned the subtask requested by the gateway, the next neighbouring agent that seems capable to accommodate the specific fraction of the initial request is consulted.

Consecutive negative answers from all neighbouring agents result in sending the task to a higher priority one-hop-away gateway agent. Since lower priority gateway agents do not have information about resource availability of their higher priority gateway agents, the decision is based on a simple request over the amount of total resources and capabilities that the higher priority gateways have in their view. This means that if a gateway agent cannot accommodate a task, and consequently must send it to a higher priority gateway agent, it only asks for the resources (time units) available via its one-hop-away gateways. The actual recipient is the one among them that possesses the highest number of available time units in its view. The execution of method B is summarized as a flowchart in Figure 7.2.

Comparing the two methods for searching and task allocation we can say that Method A uses simple (and fast) means for propagating the requests to those directions where it seems that there is a high possibility to locate the appropriate agents. In contrast, Method B uses a slightly more sophisticated (and hence slower) approach by first contacting neighbouring agents and acquiring a more accurate view of their availability. Therefore, Method B helps gateways in having more options during the task allocation process and increases the possibility of directing requests towards parts of the network where it is more likely that the requested task will be successfully allocated and scheduled.

7.5 Discussion

Analyzing the complexity of designing organizations Horling (Horling, 2006) has shown that the complexity of constructing an organization template and allocating agents to organizational roles is NEXP-Complete. This agrees with complexity results by Nair, Tambe and Marsella (Nair et al., 2003). Knowledgeable and heuristic methods (Sims and Lesser, 2008) may prune the search space for constructing suboptimal organizations. Our approach does not deal with designing organizational templates.

Given a set of complex tasks and the network of acquaintances, agents need to be assigned to specific atomic tasks with respect to the required and own resources and capabilities. Therefore, the organizational structures are quite simple with respect to the roles agents need to play (i.e. the tasks to perform) and the network has to be clustered in specific teams of agents that can jointly perform the requested tasks. In this setting, each agent may be assigned multiple atomic tasks (i.e. participate in multiple teams), which need to be scheduled consistently to the atomic and team constraints. According to this, given a specific overlay network of gateways, the

complexity of our approach lies mostly to the searching and constraint problem solving tasks.

The searching task aims at reducing the complexity of allocating agents to specific tasks by exploiting indices of agents' resources and capabilities. Searching is bounded by the number of gateways and the maximum number of immediate acquaintances of each gateway agent. The decision of which agents to participate in a team is distributed among the agents and it is subject to their joint ability to resolve the constraints: In such a case a simple method (e.g. a contract net protocol) would not be suitable for decision making given that each agent needs to satisfy jointly with its potential teammates the constraints of a task, in conjunction to all the constraints imposed by the other teams in which it aims to participate. According to the above, our approach is mostly suitable in cases where the gateway agents are proportionally less than the number of agents in the acquaintance network, and in cases where there are complex tasks that need to be jointly performed by teams of agents.

CHAPTER 8

8. System Implementation, Experiments and Results

The experiments we carried out aimed at evaluating two important aspects of our methods. More specifically, we evaluated the two methods for DisCOP solving described in Chapter 6 and the various combinations of methods for searching and task allocation put forward (Sections 7.3 and 7.4). We first discuss the way test problems were generated and then we present the experimental results.

8.1 Problem Generation

We experimented with two models for the generation of the agent network. The first one generates networks of randomly deployed agents, assuming that the geographical distance among agents determines the topology of the system. Each node A establishes connections with all nodes that exist in a specific distance from it, according to a given radius r. That is, all nodes located in a cycle with center A and radius r are neighbors of A. Networks are constructed by distributing randomly |N| agents in an n×n area, each with a "visibility" ratio equal to r. The acquaintances of an agent are those that are "visible" to the agent and those from which the agent is visible (since edges in the acquaintance network are bidirectional). Although this method of generation, which we will call geographic henceforth, distributes agents randomly in an area, the "visibility" ratio ensures that only agents that are "close by" can communicate directly. Hence, some kind of structure is introduced. The experiments in Sections 8.2 and 8.3 concern networks *AN=(N,E)*, with *|N|=500* nodes, randomly placed in a 250x250 grid. The radius parameter r was set to 25 and only connected networks were considered in the experiments. Note that since the area of placement is large and r is small, these settings tend to create relatively sparse networks where a message originating at some agent may need to travel through many edges in order to reach distant agents. In Section 8.4 we discuss an alternative random generation method that constructs the agent network in a way that allows for messages to reach any other agent by traveling through fewer edges on average.

We assume that each agent *Ai* possesses an amount of Si time units. Agents are simultaneously requested to jointly fulfill a set of tasks *T={t1,t2, ... ,tn}*, where *ti=<ai, starti, endi,Capi>* such that $\sum_{i=1}^{n} a_i = \sum_{i=1}^{|N|} S_i$: This is a worst-case scenario where the system has to simultaneously fulfill the maximum number of tasks that fit its resource capacity. In the experiments below, Si was uniformly set to 10 for all agents meaning the total available, and required; capacity was 500x10=5000 time units.

For simplicity we assume that the cardinality of each *Capi* is *1*, which means that a unique type of capability is sufficient for ti's satisfaction. We assume that each Ai ∈ N is also attributed with a unique type of capability, such that $\bigcup_{\forall t_i \in T} Cap_{t_i} = \bigcup_{\forall A_i \in N} Cap_{A_i} = Cap_{AN}$. We divided our experiments into three sets in terms of the capabilities that agents have

and that tasks require. The first set includes experiments where all agents have the same capability (i.e. *Capi =1*) and all tasks require an agent with *Capi =1*. In the second set there are two possible values for *Capi*, simply denoted by *1* and *2*. Hence, every agent has *Capi =1* or Capi =2 and accordingly, each atomic task either requires one agent with *Capi =1* or one with *Capi =2*. Finally, in the last set of experiments there are three possible values for *Capi (1, 2, and 3*).

The complex tasks in set of task requests T are generated sequentially in a way such that the total duration of all subtasks does not exceed the network's total capacity. In the experiments presented below each complex task consists of, at maximum, 3 or 7 subtasks. The TTL of all tasks was set to 10. The actual number of subtasks for each complex task is chosen randomly with a uniform distribution. The next step in the generation of a complex task involves deciding the duration of its subtasks. This is done by randomly setting the duration of each subtask so that their total duration is at most *Si*, i.e. at most equal to the uniform capacity of the agents. To be more specific, assuming a complex task with 3 subtasks, this is done as follows. We first pick a subtask *t*j and randomly set its duration *dj* to a number between *1* and *Si*. We then select another *subtask tk* and randomly set its duration *dk* to a number between *1* and *Si - dj*. Finally, the duration of the remaining subtask is randomly set to a number between *1* and *Si - dj - dk*. If at some point during this process there is no available choice for the duration of a subtask, because of previously set durations of other subtasks, then the process is restarted.

We then generate the set of constraints between the subtasks. Given *X* subtasks in a complex task, this is done by first selecting randomly *y%* of the possible constraints between the subtasks (i.e. *[(y/*100*)* x |*X*| x *(X*-1*)/2)]* constraints). For example, when y=50 in a complex task with 7 subtasks we generate randomly *[(0.5x (7 x 6)/2)] =10* constraints between the subtasks. For all the experiments presented below, y was set to 50. This value resulted in creating complex tasks that are relatively hard while at the same time rarely being overconstrained. Then for each constraint we choose a random label among the following set: {>,<,=, ≥,≤)}. For instance, if the chosen label is '>' it means that the start time of the second subtask participating in the constraint must be greater than the end time of the first. For example, if the first subtask's duration is 2 and its start time is 1, the second subtask's start time must be greater than *3*.

As a final step, we use ADOPT to check if the generated complex task is actually satisfiable. If it happens to be overconstrained then it is dropped and a new complex task is generated. In all experiments all complex tasks enter the agent system through a randomly chosen agent at the start of the system run.

In the reported experiments we report averages over 10 experiments for each individual case of parameter settings. Throughout the following sections we compute and compare three basic measures:

1. Benefit: The percentage of complex tasks scheduled over the complex tasks requested to be scheduled.
2. Messages: This is the total number of messages exchanged between any two agents throughout the task allocation and scheduling process

3. Message Gain: The ratio of the benefit over the total number of exchanged messages.

In some cases we also report additional useful information such the number of gateway agents created and the numbers of PTNs formed during the task allocation process.

8.2 Adopt vs Local Search for solving DisCOPs

Experiments here compare Local Search and Adopt for solving the DisCOPs derived from the temporal interdependencies of tasks. That is, we evaluate the performance of the system's scheduling component when either Local Search or Adopt is used to solve the DisCOPs. For a fair comparison, experiments for both algorithms ran in a system that uses the same method for searching and task allocation (Method A with active non-gateways). Therefore, the experiments presented here illustrate the contrast between a complete (Adopt) and an incomplete (Local Search) algorithm for solving DisCOPs generated from complex tasks. As we detail below, results show that Adopt, compared to Local Search, improves the efficiency without considerably increasing the cost in terms of exchanged messages.

The parameters of the generated geographic networks are as follows:

- 500 nodes
- n=250, r=25
- (1) or (1 and 2) or (1 and 2 and 3) distinct capabilities
- 3 or 7 subtasks per task at maximum

Note that with the above settings for n, r, and 500 nodes, the average density of the generated networks is around 1.95%. The average number of nodes that obtained gateway status was 227.

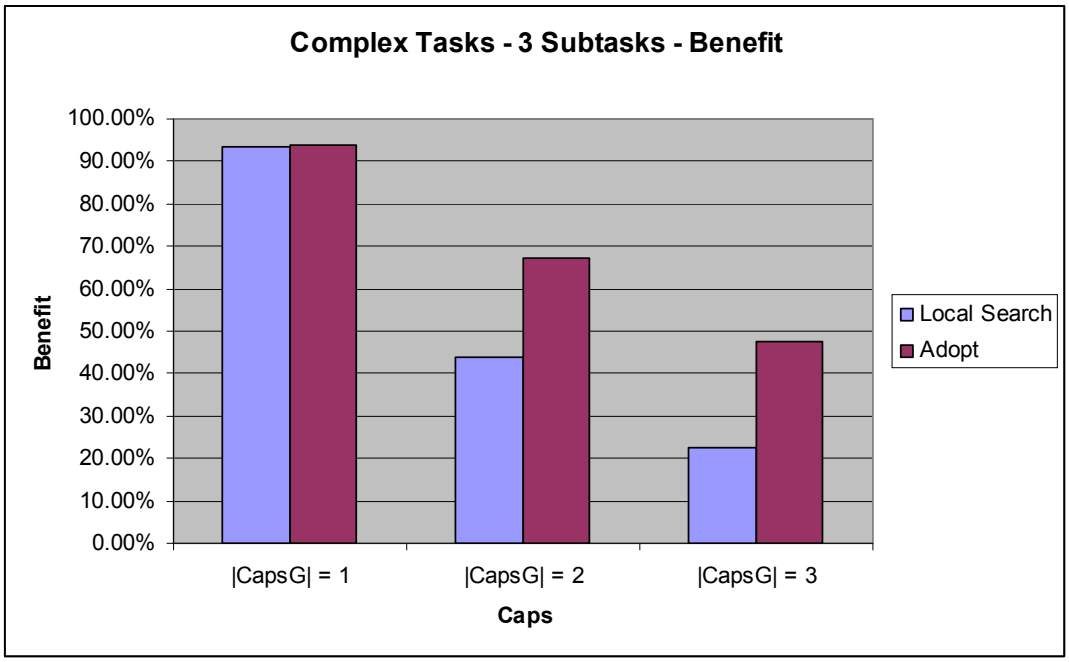


Figure 8.1: Benefit achieved by Adopt and Local Search when there are at most 3 subtasks per complex task.

Figures 8.1 and 8.2 give the benefit and message gain of the compared methods when complex tasks include at maximum 3 subtasks. Accordingly, Figures 8.3 and 8.4 give the same information when complex tasks consist of 7 subtasks at maximum. As displayed in Figures 10 and 11, when Cap_i is 1, both Adopt and Local Search achieve a very high benefit while Local Search displays higher message gain because of the increased message exchange that Adopt incurs. However, as the number of available capabilities increases and the problems become harder, Adopt achieves a much higher benefit and also outperforms Local Search in terms of message gain. This is because Adopt is able to solve more DisCOPs and hence successfully schedule more complex tasks than Local Search. This success outweighs Adopt's extra message cost and results in higher message gain.

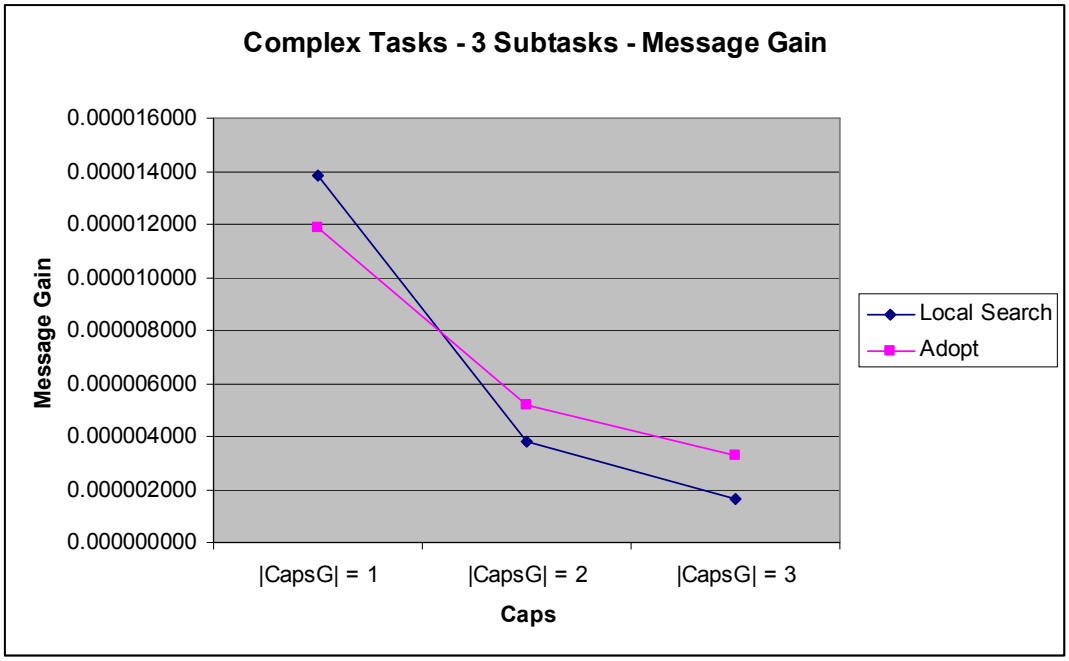


Figure 8.2: Message gain achieved by Adopt and Local Search when there are at most 3 subtasks per complex task

As displayed in Figures 8.3 and 8.4, when the complexity of the task requests increases Adopt is constantly better than Local Search both in terms of benefit and message gain for all values of Cap_i. It is notable that when there are 3 types of capabilities (which is the hardest case) a system that uses Adopt can achieve nearly three times the benefit achieved by Local Search.

To summarize, the use of Adopt displayed increased efficacy as far as the obtained benefit is concerned. Furthermore, the number or exchanged messages did not increase considerably compared to Local Search. As a result, Adopt demonstrated a better message gain than Local Search. Therefore, we can safely conclude that, on hard problems, the method that applies Adopt for scheduling is more efficient than the one that applies Local Search.

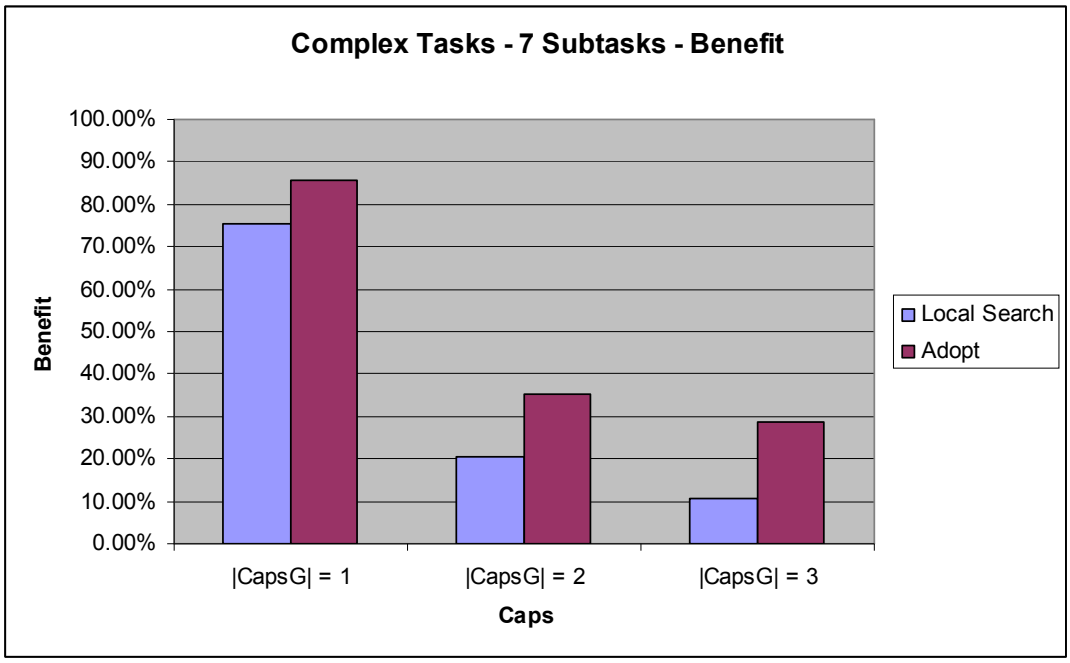


Figure 8.3: Benefit achieved by Adopt and Local Search when there are at most 7 subtasks per complex task.

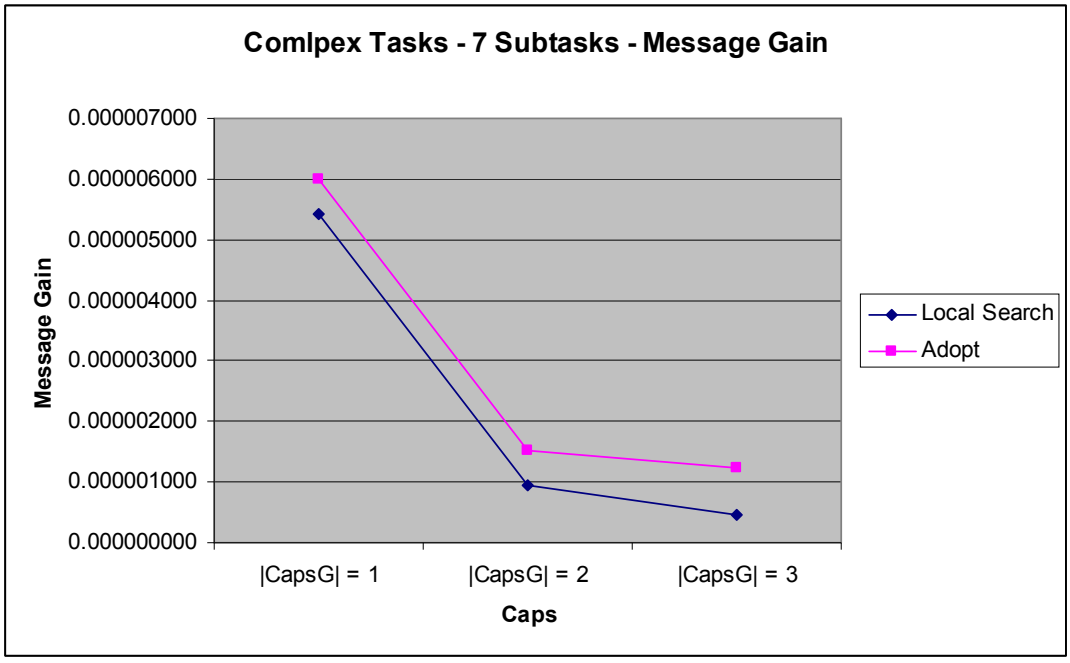


Figure 8.4: Message gain achieved by Adopt and Local Search when there are at most 7 subtasks per complex task.

8.3 Evaluating different approaches to searching and task allocation

Having established that Adopt offers considerable advantages in terms of the system's benefit, we now evaluate the different approaches to searching and task allocation using the same networks as in Section 8.2 (i.e. |N|=500, r=25, n=250). For all these

experiments we used Adopt for scheduling despite the relative increase in the number of exchanged messages that it sometimes occurs compared to Local Search. We evaluated the following methods outlined in Sections 7.3 and 7.4:

1. *Method A with active non-gateways*
2. *Method A with inactive non-gateways*
3. *Method B with active non-gateways*
4. *Method B with inactive non-gateways*

Considering the two methods (A and B) for performing tasks allocation and scheduling, recall that as discussed in Section 7.4, Method B is in theory more flexible in finding agents with free resources and consequently in forming PTNs. Indeed, experiments verified this as Method B consistently demonstrated better results than Method A in each single experiment (10 experiments for each parameter setting). Both variations of Method B performed better than any variation of Method A in terms of the total number of satisfied complex tasks. A small downside is that Method B produced more messages. However, the average difference in message gain between the two methods was 4.79% when each complex task has at most 3 subtasks and 3.41% in the case of 7 subtasks, both in favor of Method B. These are the average variations in message gain between Methods A and B with either active or inactive non-gateway agents and caps 1,2 or 3. Method A achieved a slightly better average message gain only in the (simple) case of complex tasks with at most 3 subtasks and caps = 1. Therefore, in the rest of Section 8.3 we opt to present results from the two variations of Method B (active and inactive non-gateways) only. There is also a detailed statistical analysis comparing Method A to Method B. We now first present some information regarding the generated sets of tasks, and then we give results from the two variants of Method B.

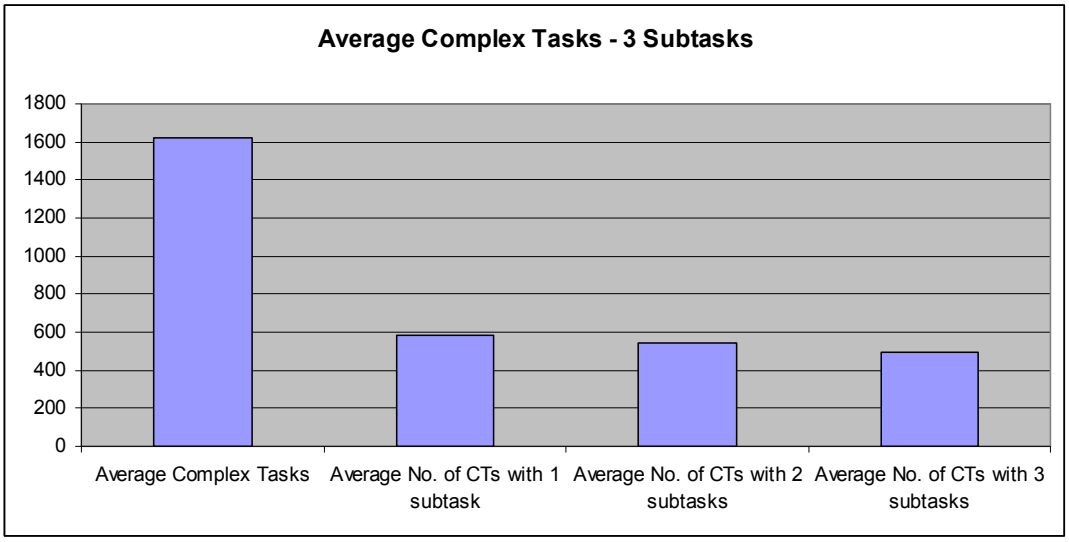


Figure 8.5: Average number of complex tasks for 3 subtasks at maximum and average number of complex tasks with exactly 1,2 and 3 subtasks respectively.

In the first column of Figures 8.5 and 8.6 we show the average number of complex tasks created in a set of task requests *T* when the maximum number of subtasks per complex task is 3 and 7 respectively. We also give the average number of complex

tasks with exactly 1,2,3 subtasks for the first case and 1,2,…,7 subtasks for the second case. We can see that although the number of subtasks per complex task was uniformly selected, complex tasks with few subtasks appeared more often than ones with many subtasks. This is due to the fact that preprocessing with Adopt detected unsatisfiability for many of the latter complex tasks, and therefore they were dropped from the generated set *T*.

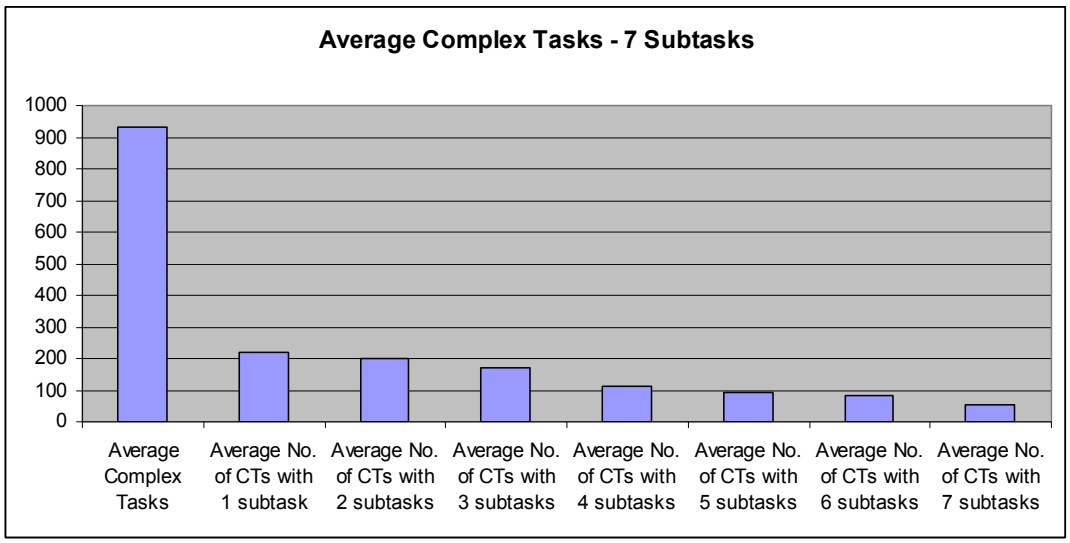


Figure 8.6: Average number of complex tasks for 7 subtasks at maximum and average number of complex tasks with exactly 1,2,…,7 subtasks respectively.

Figures 8.7 and 8.8 present the average duration of the complex tasks in a set of tasks *T* for the two cases (3 or 7 subtasks at maximum). In these figures, we also give the average duration of the complex tasks broken down to the number of subtasks. As expected, in the first case shorter tasks are generated, meaning that their allocation and scheduling is more likely to succeed as they can often be assigned to a single agent.

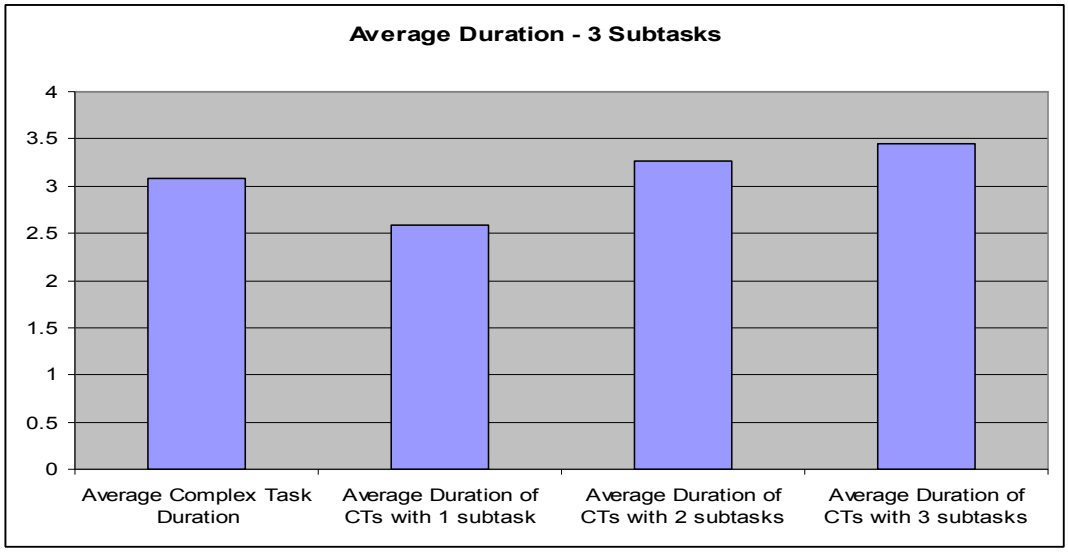


Figure 8.7: Average duration of complex tasks of 3 subtasks at maximum and average duration of complex tasks with exactly 1,2 and 3 subtasks respectively

In contrast, complex tasks with many subtasks have average duration closer to the capacity of the agents which means that most likely a PTN with several agents needs to be formed in order to serve them, especially in the case of 3 possible capabilities.

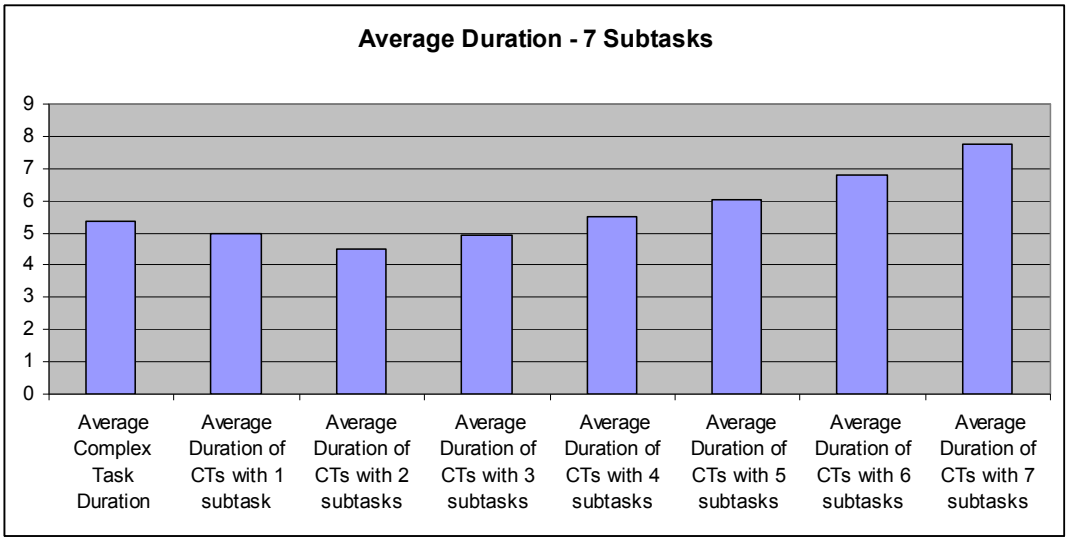


Figure 8.8: Average duration of complex tasks of 7 subtasks at maximum and average duration of complex tasks with exactly 1,2 … 6 and 7 subtasks respectively.

Figures 8.9 and 8.10 report the number of PTNs formed while trying to resolve a complex task, the number of times these attempts succeeded in finding resources for a complex task, and the number of times they failed. In Figure 18 we give these measurements for the case of 3 subtasks at maximum, while Figure 19 contains this data for 7 subtasks at maximum. Note that in both cases there are two methods (the two versions of Method B) and three possible sets of Caps in each.

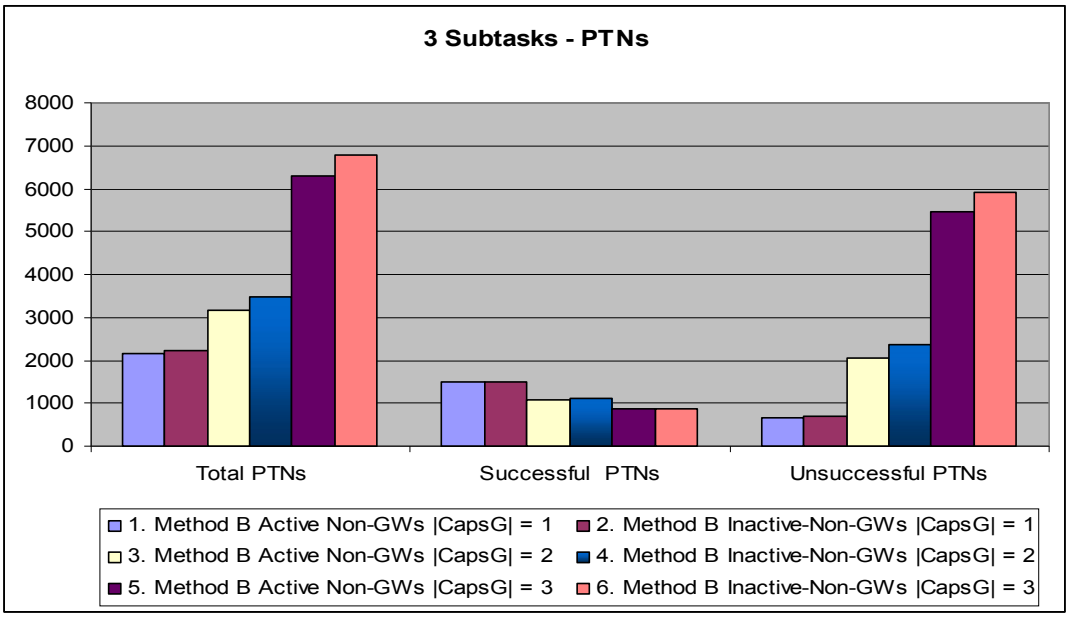


Figure 8.9: Number of total PTNs formed for each case of problems concerning complex tasks with maximum 3 subtasks, followed by the number of those that were successful and unsuccessful.

Both figures demonstrate that the inactive non-gateways method forms a larger number of PTNs on average. Although the number of successful PTN formations does not vary much between the two methods, the number of unsuccessful PTNs is visibly higher for the inactive non-gateways variant of Method B. The average number (over all values of Caps) of unsuccessful PTNs for each successfully allocated task ranges from 1.05 when there are at most 3 subtasks within any complex task up to 2.7 unsuccessful PTNs for each successful one when there are at most 7 subtasks. These numbers are approximately 1.8% higher on average for the inactive non-gateways method compared to the active non-gateways method. The active gateways method forms fewer PTNs because in some cases a task that enters the system through a non-gateway node can be directly accommodated by this node. In contrast, in the non-active gateways method the task will be immediately forwarded to a gateway agent and therefore the process of searching for a PTN to accommodate the task will begin. Despite this, and as results below demonstrate, the non-active gateways method provides greater flexibility to the system since non-gateways that receive incoming tasks do not engage their resources immediately, and in this way fill up their timeline, but allow for gateway agents that a have a more accurate view of the system's available resources to forward the task to agents that may have better availability to serve it.

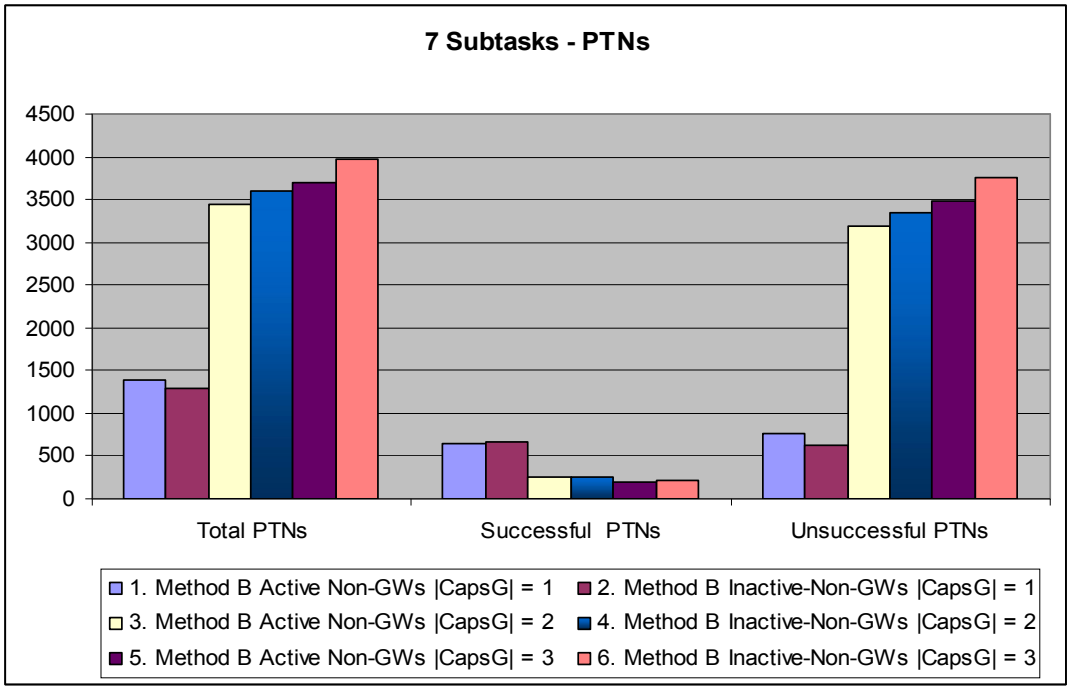


Figure 8.10: Number of total PTNs formed for each case of problems concerning complex tasks with maximum 7 subtasks, followed by the number of those that were successful and unsuccessful.

Concluding this section we give the average benefit and message gain for the two alternatives of Method B. Figures 8.11 and 8.12 depict the benefit for the cases of 3 and 7 maximum subtasks per complex task respectively, while Figures 8.13 and 8.14 depict the message gain for the two cases. In each figure we give the average numbers for problems with 1, 2, and 3 possible capabilities.

As displayed, the inactive non-gateways method achieves slightly better performance compared to the active non-gateways method, especially in the case of 7 subtasks. Despite the fact that the inactive non-gateway method forms more PTNs on average, the increase in the number of messages exchanged incurred is not important enough to affect the message gain factor considerably. Hence, the inactive non-gateways method also obtains a higher message gain. However, the difference in the message gain obtained is reduced as the number of subtasks in each complex task increases. This is to be expected as a higher number of subtasks within each complex task decreases the potential of each active non-gateway agent to serve a request, resulting in more messages being exchanged while searching for PTNs to serve the tasks.

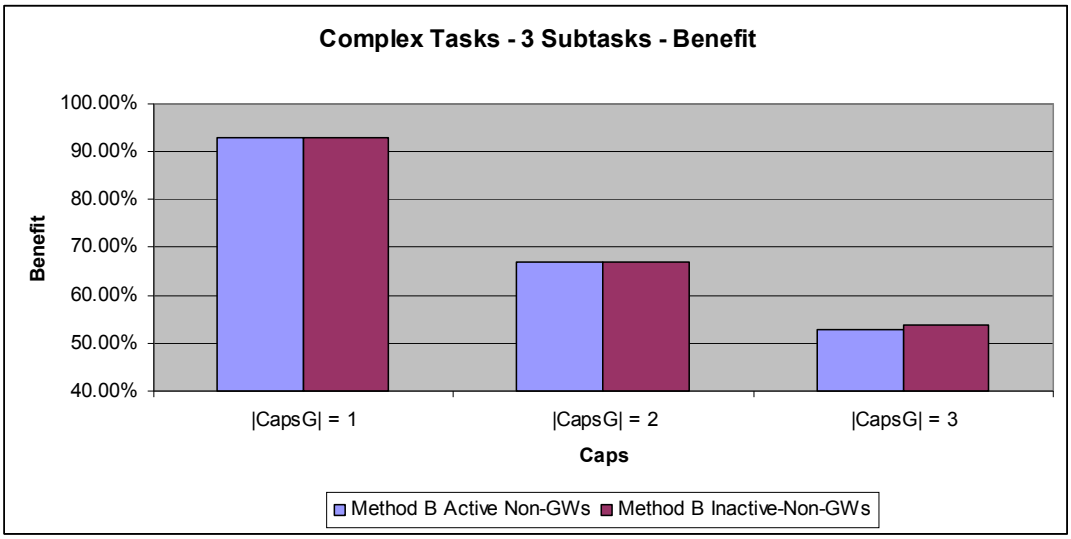


Figure 8.11: Benefit achieved when there are 3 subtasks per complex task.

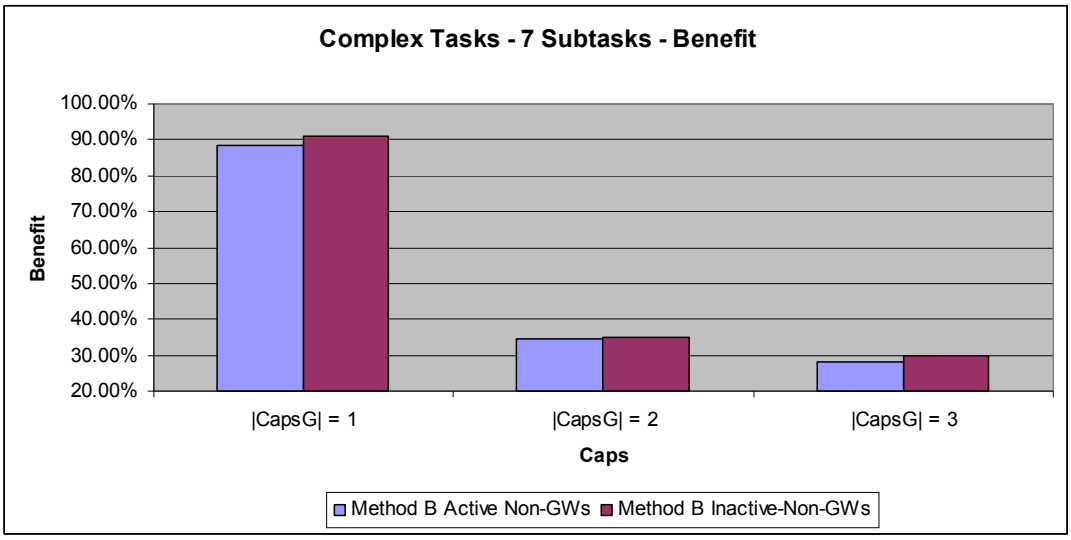


Figure 8.12: Benefit achieved when there are 7 subtasks per complex task.

Relating these results to the discussion on Figures 8.5 – 8.8, it is interesting to note that the decline in benefit achieved when moving from fewer to more subtasks per complex task is not significant (see the first two bars in Figures 8.11 and 8.12) in the

case of a single capability for all agents. However, the decline is rapid as the number of capabilities increases as it becomes increasingly difficult to locate the appropriate agents and successfully form PTNs.

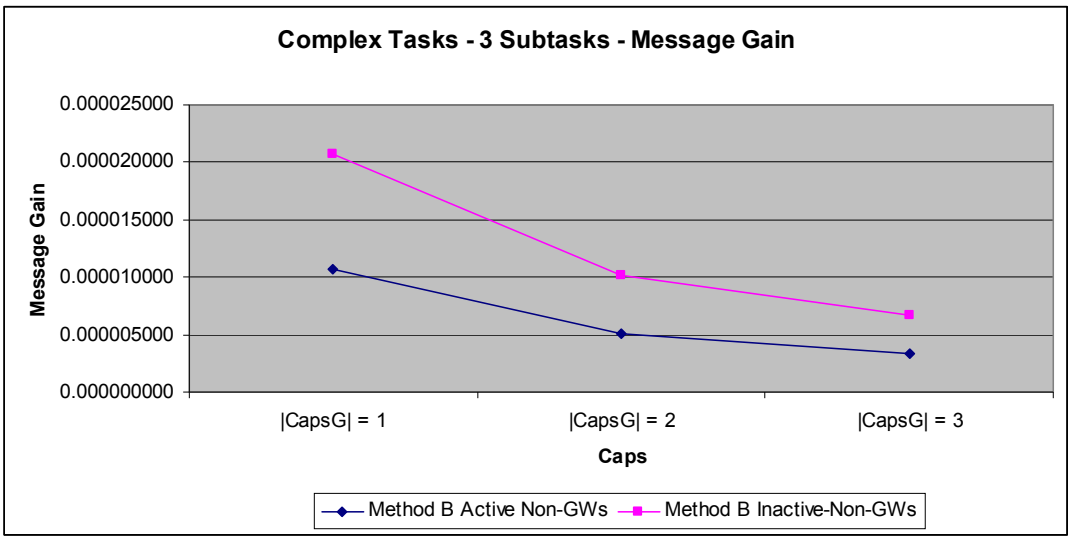


Figure 8.13: Message Gain when there are 3 subtasks per complex task.

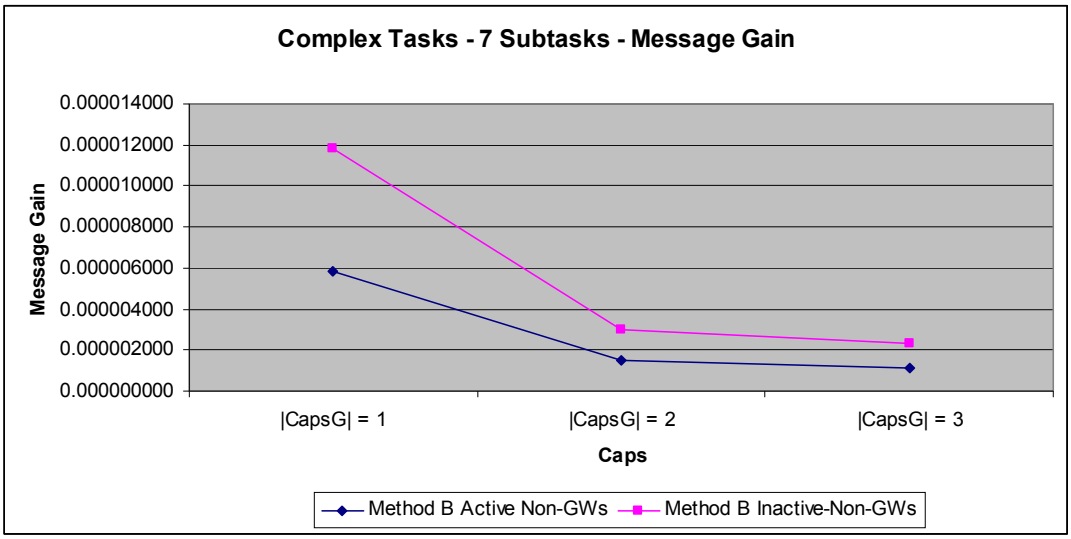


Figure 8.14: Message Gain when there are 7 subtasks per complex task.

8.4 Evaluating different approaches to searching and task allocation on networks with random connections

In this section we study the performance of the proposed methods in different agent network settings aiming to investigate whether and to what extent the topology of the network influences the performance of our proposed techniques.

The networks in this set of experiments were generated in a way such that connections can be created between any two nodes in the network, discarding the radius parameter. Hence, we call this the random_connections generation method. The

generator takes a parameter n denoting the maximum number of edges that can be attached to any node in the agent network. That is, the maximum number of agents each agent can have in its immediate neighborhood. For each agent, the actual number m of its neighbors is selected randomly between 1 and n. The generation process starts by randomly selecting an agent Ai and a number m, m > 0. Then, we randomly select m agents to connect to Ai and the corresponding edges are added to the network. This process continues until the connections of all agents have been determined. When determining the connections of an agent we take into account the connections it may have already acquired through previous steps in the process. We ensure that the resulting network is connected by randomly adding edges between any disconnected components. For the experiments presented here n varied from 3 to 15 with an increasing step of 2.

The average number of complex tasks generated for the experiments of this section was 1,637 for the case of 3 subtasks and 933 for the case of 7 subtasks. Note that the generation of complex tasks is not influenced by the underlying network's topology and therefore their characteristics are very close to the ones shown in Figs. 8.5 and 8.6. The average graph density of the graphs generated for n = 3, 5, 7, . . . , 15 is shown below (Fig. 8.15). Note that the generated graphs are sparse as the average graph density ranges from 0.48% for n=3 to 2.52% for n=15.

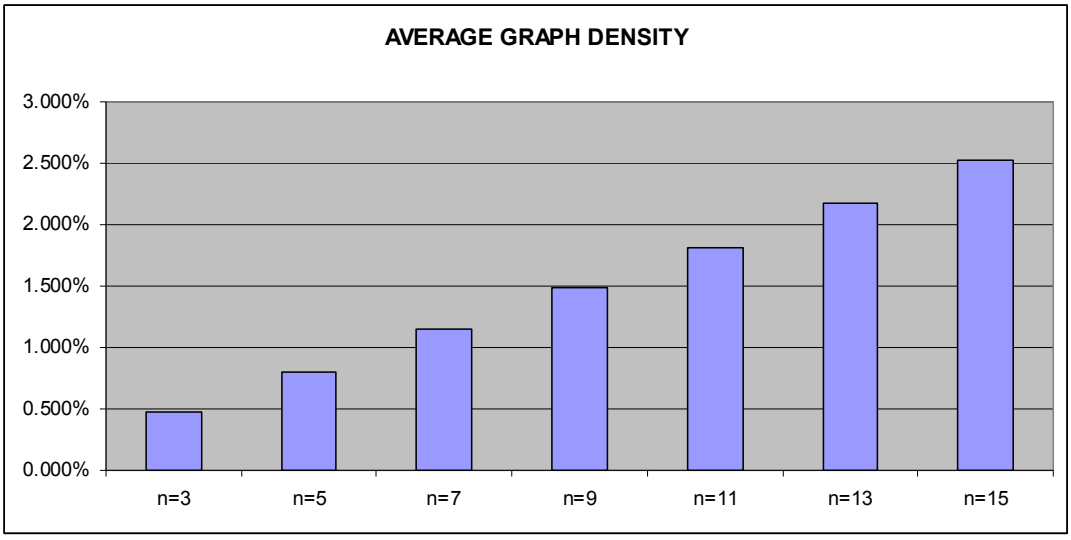


Figure 8.15: Average graph density for n= 3,5,…,15.

After each agent network is generated and each agent acquires knowledge of its neighbors we decide which ones will be assigned the gateway status. Information on this is presented in Figure 8.16 where we give the average number of agents that opted for gateway status over the maximum number of connections for each agent. As expected, as the density of the network increases, the number of gateways decreases.

We have performed a detailed evaluation of the proposed searching and allocation methods (the variants of Method A and Method B) on the problems generated with the random connections generation method whose characteristics are explained in Figs. 8.15 and 8.16. Before presenting the results of our most competitive methods

(the two variants of Method B), we first give a statistical analysis confirming the (slight) advantage of Method B over Method A.

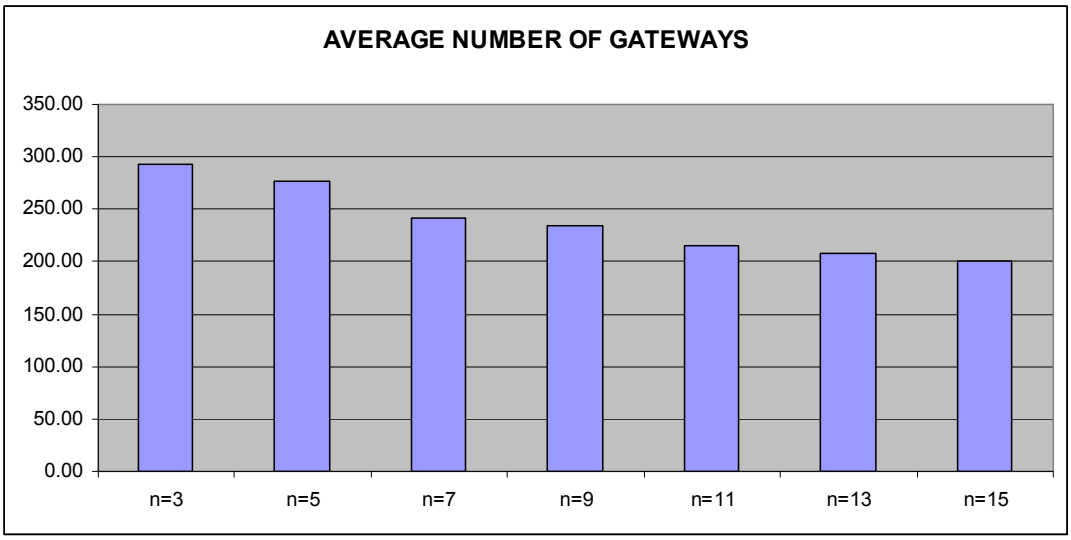


Figure 8.16: Average number of gateway agents for n maximum possible immediate neighbors.

8.4.1 Method A versus Method B

As it is also reported in Section 8.3, Method B is generally slightly better than Method A. We now give results from a statistical analysis, performed using non-parametric Wilcoxon signed-rank tests that verify this. The Wilcoxon signed-rank test is a non-parametric statistical hypothesis test for the case of two related samples or repeated measurements on a single sample. The test is based on the magnitude of the difference between the pairs of observations. The Wilcoxon signed-ranks tests assume a continuous value distribution. It can be used as an alternative to the paired Student's t-test when the population cannot be assumed to be normally distributed. Table 8.1 compares Method B with active non-gateways to Method A with active non-gateways, while Table 8.2 gives similar results for the case of inactive non-gateways. We give the mean and standard deviation of the difference in benefit and the difference in the number of messages sent by the compared methods. We also give the Z-value and the p-value for each combination of Capability types/Max. subtasks. If the computed p-value is equal or less to 0.05, then the two methods can be said to differ significantly. Otherwise, no statistically significant difference can be assumed between the two methods.

Results from Tables 8.1 and 8.2 show that in all cases, except in the case of 1 capability and 3 subtasks per task in Table 8.1, Method B achieves a higher benefit on average than Method A. Moreover, in most cases the difference is statistically significant as the p-values indicate. In contrast, in all cases Method B has a higher average cost than Method A, measured by the messages sent. Also, in most cases the difference in the number of messages is statistically significant. Having established that Method B achieves a higher benefit than Method A, in the rest of Section 8.4 we only present results from the two variants of Method B.

Caps / Max. Sub-tasks	Benefit_B_i - Benefit_A_i				Messages_B_i - Messages_A_i			
	Mean	**SD**	***Z*-value**	***p*-value**	**Mean**	**SD**	***Z*-value**	***p*-value**
1/3	-0.11	1.40	-0.296(b)	0.767	3398	4080	-2.090(a)	0.037
2/3	1.66	1.90	-2.191(a)	0.028	3751	4339	-2.090(a)	0.037
3/3	1.74	1.97	-2.701(a)	0.007	6394	4720	-2.599(a)	0.009
1/7	1.97	2.22	-2.191(b)	0.028	6663	5341	-2.803(b)	0.005
2/7	2.48	3.00	-2.191(a)	0.028	1115	5701	-0.255(a)	0.799
3/7	1.48	3.07	-1.362(a)	0.173	14584	6865	-2.803(a)	0.005

Table 8.1: Statistical results comparing Method B with active non-gateways (denoted by B_i) to Method A with active non-gateways (denoted by A_i). Column 1 gives the number of capability types and the maximum number of subtasks per complex task. Columns 2–5 give statistics regarding the difference in achieved benefit. Columns 6–9 give statistics regarding the difference in sent messages. p-Values and Z-values that indicate statistical significance are underlined. A sample of 10 instances was used for each combination of values for the capabilities and the maximum subtasks per complex task

Caps / Max. Sub-tasks	Benefit_B_i - Benefit_A_i				Messages_B_i - Messages_A_i			
	Mean	**SD**	***Z*-value**	***p*-value**	**Mean**	**SD**	***Z*-value**	***p*-value**
1/3	0.36	1.65	-0.663(a)	0.508	8494	4492	-2.803(a)	0.005
2/3	1.95	1.83	-2.395(a)	0.017	1504	3338	-1.274(a)	0.203
3/3	3.01	2.94	-2.395(a)	0.017	6574	6685	-2.701(a)	0.007
1/7	6.39	2.12	-2.803(b)	0.005	5387	6903	-2.090(b)	0.037
2/7	2.51	1.89	-2.599(a)	0.009	2127	6358	-0.968(a)	0.333
3/7	1.98	1.91	-2.293(a)	0.022	11861	8850	-2.599(a)	0.009

Table 8.2: Statistical results comparing Method B with inactive non-gateways (denoted by B_i) to Method A with inactive non-gateways (denoted by A_i). Column 1 gives the number of capabilities and the maximum number of subtasks per complex task. Columns 2–5 give statistics regarding the difference in achieved benefit. Columns 6–9 give statistics regarding the difference in sent messages. p-Values and Z-values that indicate statistical significance are underlined. A sample of 10 instances was used for each combination of values for the capabilities and the maximum subtasks per complex task

8.4.2 Active versus Inactive non-gateways

In Figs. 8.17 and 8.18 we compare the benefit achieved by the two variations of Method B in problems with complex tasks consisting of at most 3 and 7 subtasks respectively, while in Figs. 8.19 and 8.20 we show the message gain for the same classes of problems. The numbers given are averages for all values of the maximum number of agents' neighbors n (3 up to 15). We do not present separate results for the different values of n because changes in the graph's density in the range of 0.48–2.52 did not have a significant impact on the system's benefit. Of course, for much denser

networks this may not be true and we intend to investigate this in more detail in the future. While the average density for networks with n=3 was substantially lower than the one for networks with n=15 (around 5 times lower), the system's benefit throughout all sets of experiments had a variation of only 1.5% between the highest and the lowest benefit obtained for networks of any density, noting that slightly higher benefit was achieved as the density was increased.

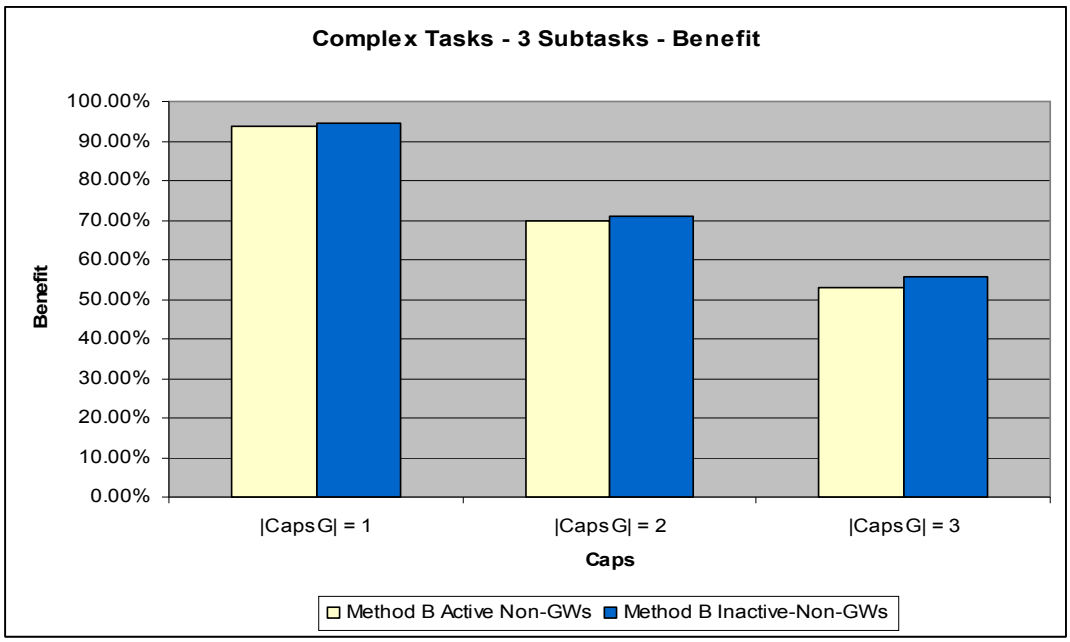


Figure 8.17: Benefit achieved when there are at most 3 subtasks per complex task.

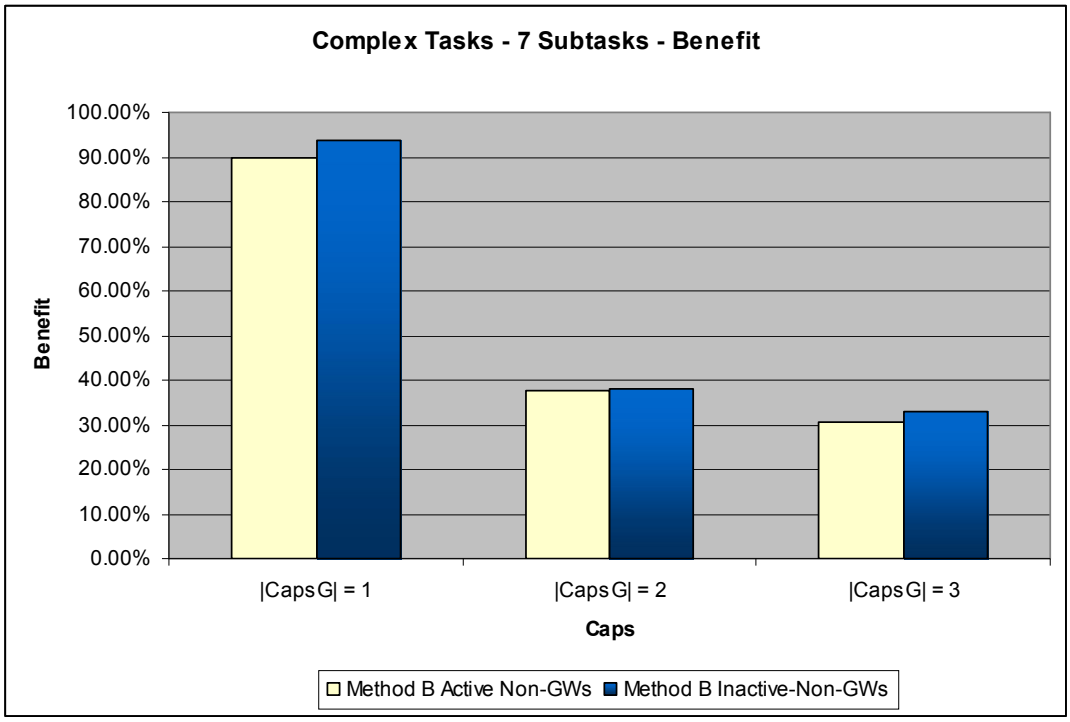


Figure 8.18: Benefit achieved when there are at most 7 subtasks per complex task.

Hence, it seems that the proposed method is quite robust regarding the network's density, at least for the random_connections generation method. Results show that the system's benefit remains high independent of the allocation method selected.

The second method (Method B with inactive non-gateways "non-GWs") performs slightly better compared to the other one in terms of benefit but it incurs more exchanged messages, as in Section 8.3. Hence, the active non-gateways method achieves slightly better message gain in the case of 3 subtasks per complex task. However, in the case of 7 subtasks this is reversed due to the higher difference in benefit in favor of the inactive non-gateways method. To obtain a better understanding of the difference in performance between the two variants of Method B, we performed a statistical analysis similar to the one presented in Section 8.4.1. Table 8.3 presents the results of this analysis that confirm that the inactive non-gateways variation always achieves a higher benefit on average compared to the active gateways variation. However there is only a small difference which is not always statistically significant. On the other hand, the inactive non-gateways method incurs a slight increase in the number of exchanges messages, but this increase is very small and rarely has a statistical significance.

Caps / Max. Sub-tasks	**Benefit_B_i - Benefit_A_i**				**Messages_B_i - Messages_A_i**			
	Mean	**SD**	***Z*-value**	***p*-value**	**Mean**	**SD**	***Z*-value**	***p*-value**
1/3	0.63	1.47	-1.073(a)	0.283	5473	5792	-2.395(a)	0.017
2/3	0.88	2.21	-0.866(a)	0.386	442	3440	-0.153(b)	0.878
3/3	2.66	2.00	-2.803(a)	0.005	1981	4899	-1.274(a)	0.203
1/7	3.98	2.20	-2.803(b)	0.005	1913	5169	-0.866(b)	0.386
2/7	0.32	2.30	-0.255(b)	0.799	2244	6860	-1.070(a)	0.285
3/7	2.22	2.23	-2.090(a)	0.037	1504	6635	-0.866(a)	0.386

Table 8.3: Statistical results comparing Method B with active non-gateways (denoted by B_a) to Method B with inactive non-gateways (denoted by B_i). Column 1 gives the number of capability types and the maximum number of subtasks per complex task. Columns 2–5 give statistics regarding the difference in achieved benefit. Columns 6–9 give statistics regarding the difference in sent messages. p-Values and Z-values that indicate statistical significance are underlined. A sample of 10 instances was used for each combination of values for the capabilities and the maximum subtasks per complex task

Comparing the results of Section 8.4 with the ones presented in Section 8.3 we can conclude that the proposed method for allocation and scheduling is quite robust with respect to the problem generation method (at least for geographical and random_connections networks). That is, the benefit achieved does not vary considerably between the two generation methods. Specifically, the variation between the results of Sections 8.3 and 8.4 with respect to the system's benefit was 1.9% on average, with maximum value of 3.85%, for the case of 3 subtasks per complex task. In the case of 7 subtasks the variation was 2.6% on average and the maximum difference was 3.14%. In all cases the benefit achieved on the networks generated with the random_connections method is higher. Comparing the results from the geographical and random_connections types of networks that have the same average

density (i.e. around 1.95%) the average benefit variation is roughly 1%, again in favor of the random_connections generation method.

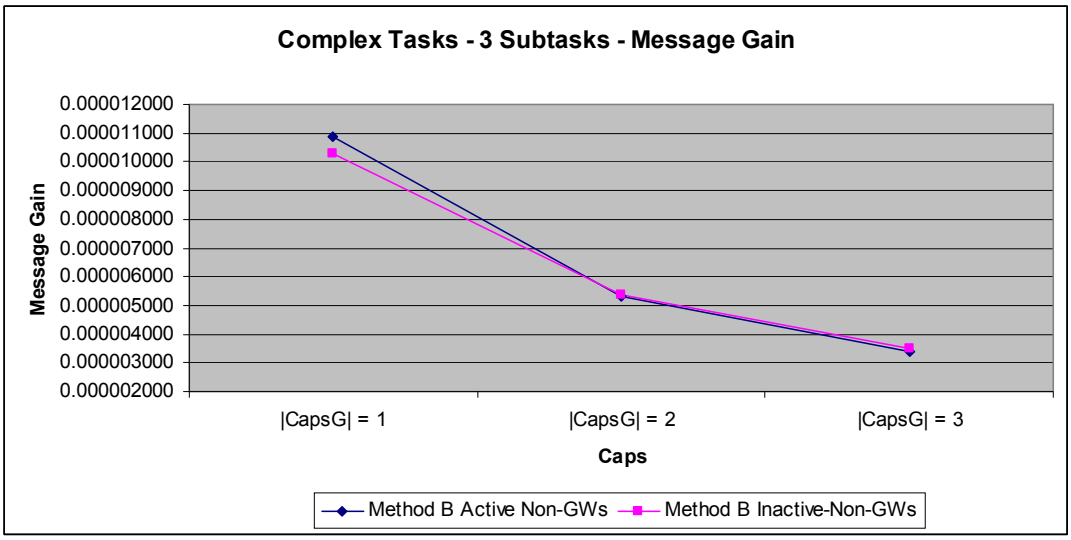


Figure 8.19: Message Gain when there are 3 subtasks per complex task.

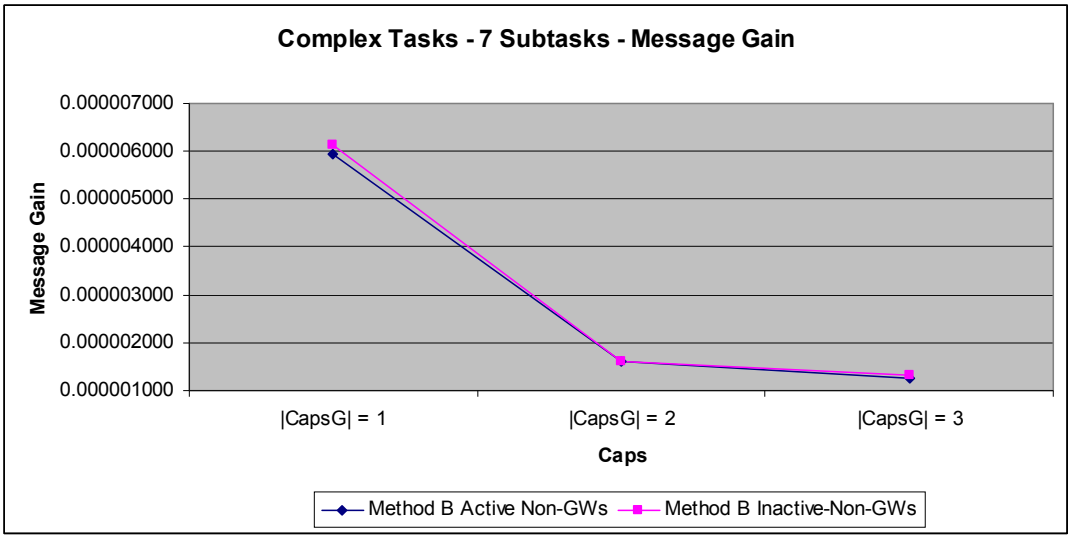


Figure 8.20 Message Gain when there are 7 subtasks per complex task.

Although this is not significant statistically, we conjecture that on the random_connections networks the higher benefit may be due to that a message can reach any node in the network in fewer hops (on average) than in geographical networks. This means that a task request that enters through a specific agentAi in a geographical network, and requires resource availability and capabilities offered by distant agents only, may not be served because its TTL may expire before the appropriate agents are located and the corresponding PTN is formed.

In contrast, in random_connections networks the appropriate agents could be reached faster (because of the random shortcuts), within the TTL, and the request might

therefore be served. Of course, this conjecture requires further and more detailed experimental evidence.

8.4.3 Changing the Distribution of Capabilities

Up to this point capabilities have been assigned to agents using a uniform distribution. In practice though, many systems tend to behave in a different way as certain agent capability types are frequent and others are scarce. Hence, we also experimented with problem sets where capabilities are distributed in a different way. To be precise, the agent networks here were randomly generated using the random_connections method and capabilities of 3 distinct types were assigned to the agents (each agent getting only one type). Capability assignment followed the Zipfean distribution. In this case, the number of agents having capability type 1 was much higher than the one of agents having capability type 2. In turn, agents with capability type 2 are more than ones with capability type 3. The performance of the system is shown in Figures 8.21 (benefit) and 8.22 (message gain). As can be seen, there is a sharp drop in the benefit achieved compared to Figures 8.17 and 8.18, especially in the case of 3 subtasks. For example, in this case, the benefit of the non-active gateways method drops from approx. 60% down to just over 30%. Similar results hold with relevance to the message gain obtained.

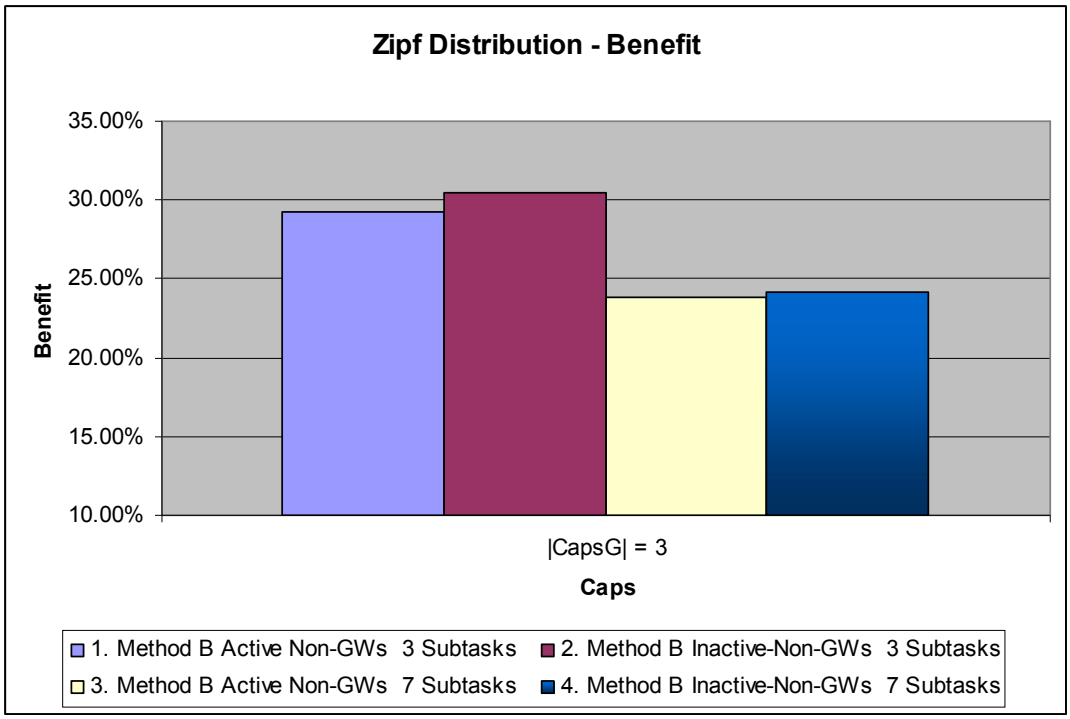


Figure 8.21: Benefit for networks with possible capability values 1,2 or 3 following the Zipfean distribution.

These results are not unexpected since the system has increasing difficulty in finding agents capable of serving atomic tasks that require capabilities 2 or 3. Hence, most of the complex tasks that include subtasks requiring these capabilities will not be served, resulting in reduced benefit. In addition, the difficulty to locate appropriate agents results in a larger volume of messages exchanged. On the other hand, if all the

subtasks in a complex task require agents with capability type 1, it will be very easy to locate them, form a PTN and subsequently successfully schedule the task. However, this is not a very common case. Note that we did not use the Zipfean distribution to assign required capabilities to subtasks, but rather the uniform one. If we had used the Zipfean distribution for the subtasks then the distribution of capabilities to agents would follow that of capabilities to subtasks, resulting in easier problems.

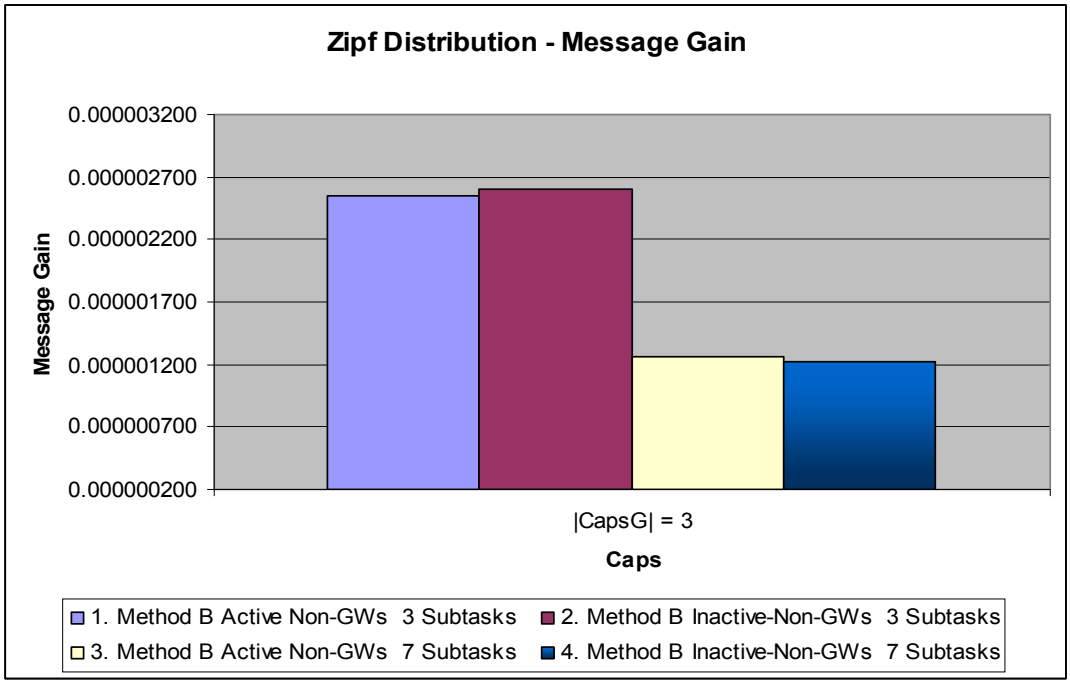


Figure 8.22: Message Gain for networks with possible capability values 1,2 or 3 following the Zipfean distribution.

To conclude the experimental evaluation, viewing the task allocation and scheduling as a two-step process we can note that different approaches to each step have different repercussions on the system's efficiency. As demonstrated in Section 8.2, different approaches to the scheduling step can lead to significant variance in the amount of successfully scheduled tasks (benefit). Hence moving from local search to a sophisticated complete DisCOP algorithm like Adopt is indeed very beneficial. On the contrary, in this and the previous subsections we have shown that trying to improve the task allocation process through more elaborate mechanisms does not have a similarly significant impact on the system's benefit since the use of the gateway/routing indices infrastructure is by itself very effective. However, it remains to be seen if even more intricate task allocation processes can have a significant impact on the system's benefit for complex problems.

8.5 Evaluating performance when agent availability is limited

In this section, we study how the system responds and performs when not all agents are available at any given moment. Of the proposed methods, we tested limited agent availability for Method B in different agent network settings. Since we have established that the different network topologies we experimented with have little influence on the system's performance, we chose to use the random_connectiocs

generation method for these experiments. The networks in this set of experiments were again generated so that connections can be created between any two nodes in the network (Section 8.4, random_connections generation method). For the experiments in this section n varied from 3 to 11 with an increasing step of 2. We tested complex tasks of 3 and 5 subtasks at most. Capabilities were kept to Cap = 1 for all cases. In this section, the capability factor was not stretched further because we aimed to test the impact availability would impose to our system. We chose Method B (inactive non-gateway mode) as this method is backed up by a complete algorithm for solving DisCOPs and has performed better than Method A (the other method that uses a complete algorithm for DisCOPs) in virtually all our tests. Similar tests with local search for solving DisCOPs can be found in [Theocharopoulou et all].

The experiments presented in this section test the impact availability factor has on method B when agent networks remain the same while the complex task set varies. More specifically, first we created an agent network using the random_connections generation method, having set the maximum number of connections to 3. Then, we created two super-sets of random complex task sets (with total duration equal to the network's capacity), setting the maximum number of subtasks to 3 and 5 respectively. Each super-set consisted of ten complex task sets. Afterwards, we tested Method B (inactive non-gateway mode) using the agent network against each of the complex task sets setting the agents to be 100% available. The exact same procedure was followed after only changing the availability factor, which now was 80%. The same overall procedure was repeated for networks where each agent could have a maximum number of 5, … , 11 edges.

The derived results were grouped into two major categories. One pertaining to results for complex task sets of 3 subtasks at maximum and one for 5 subtasks at maximum. Within these categories, we used a secondary grouping taking into account the availability factor (80% and 100%). For instance, the experiments for n = 3 in Figures 8.25 and 8.26 were done using the same agent network. In figure 8.25 the results for the two columns over n = 3 were derived after using the same complex task sets as well (first super-set with 3 subtasks at maximum). The same holds for the two columns over n = 3 in Figure 8.2 (second super-set with 5 subtasks at maximum). The percentages the columns represent are averages of the results for all complex task sets within each super-set.

In this section, our purpose was to explore how our framework would respond to unpredictable shortages on two aspects. First is the overall capacity of our system. When agents are unavailable, their free time units become unavailable as well. Therefore, there are periods when the demand for resources exceeds the offer. Secondly, an unavailable agent forces the process of team formation to look for alternative routes. Thus, unavailable agents subtly change the agent network's configuration. Considering the architecture and concepts behind Method A and Method B, the choice of Method B over Method A seems straightforward. Additionally, the experiments presented in the previous sections of this chapter dictate that if we used Method A, the individual results could be slightly different but the overall picture would be similar. A parameter, which should be explored in future works, is how the capability factor coupled with the availability factor would affect our system.

For networks of 200 agents, the average number of complex tasks generated for the experiments of this section was 914 for the case of 3 subtasks and 604 for the case of 5 subtasks. The generation of complex tasks as already stated is not influenced by the underlying network's topology thus bearing characteristics very close to the ones shown in Figs. 8.5 and 8.6. The average graph density of the graphs generated for n = 3, 5, 7, . . . , 11 is shown below (Fig. 8.23). Note that the generated graphs are sparse as the average graph density ranges from 1.2% for n=3 to 4.53% for n=11.

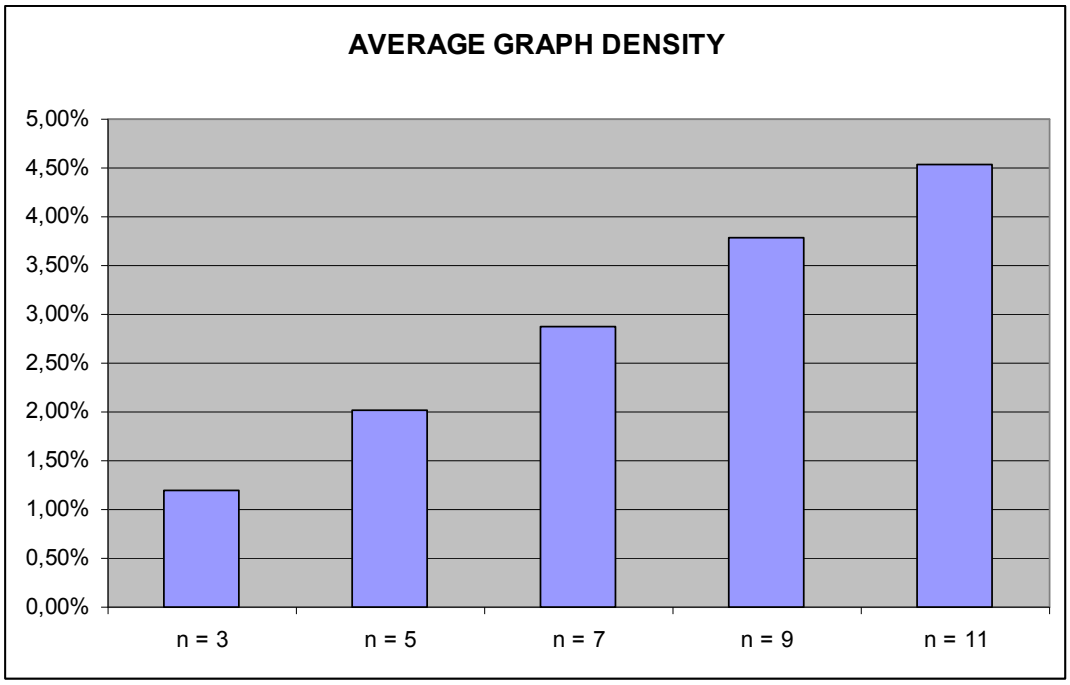


Figure 8.23: Average graph density for n = 3,.., 11

Graph density may seem somewhat offset when compared to previously depicted graph densities. Nevertheless, this is solely due to network sizes. Our network generation method produces network edges whose number grows linearly as the network size increases. On the other hand, the maximum number of edges grows polynomially ($O(n^2)$)) in relation to the network's size. Thus if the number of nodes are halved, the number of edges is almost halved as well. Nevertheless, the number of edges of a fully connected graph is 4-fold diminished. Therefore, the networks in this section are not as sparse as the ones in the previous one. However, given the percentages presented in Figure 8.23 it is obvious that our networks remain very sparse. Consequently, this factor can be eliminated as an effector when comparing the results of this section with results of other sections.

As far as the number of gateways is concerned, Figure 8.24 displays the average percentage of gateway nodes for each value of n. In order to compare Figure 8.24 with Figure 8.16 data is shown as percentages of gateway nodes. Data in Figure 8.16 for n = 3,.., 11 was converted accordingly.

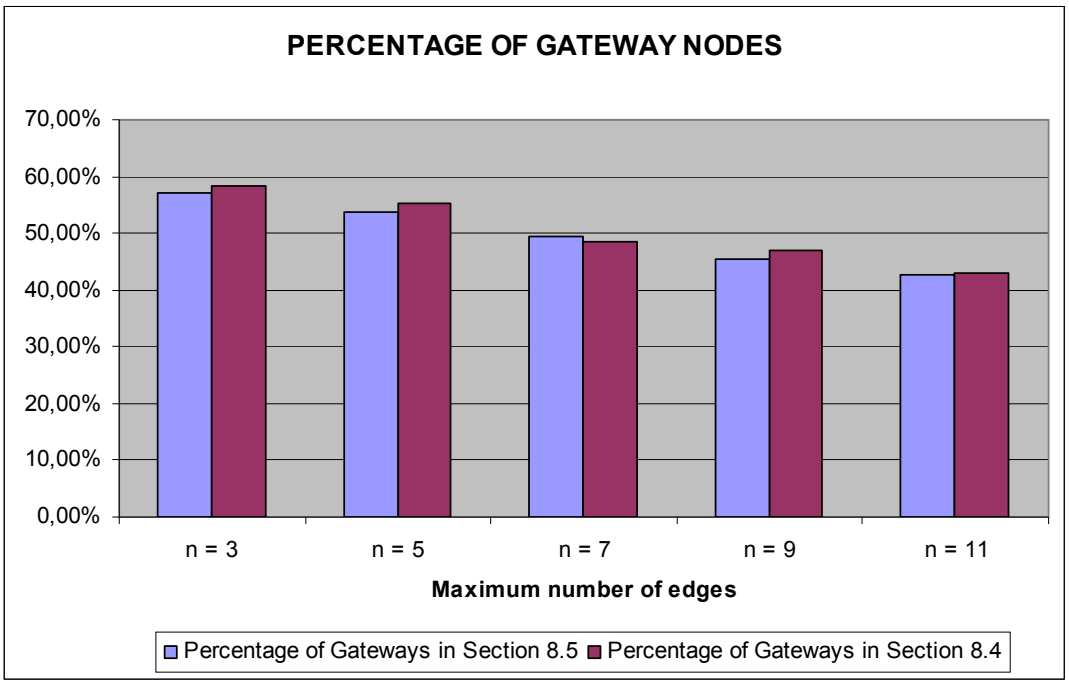


Figure 8.24: Percentage of gateway nodes for the networks presented in Sections 8.4 and 8.5. Maximum number of nodes in each case is n = 3, …, 11.

As can be seen, differences in gateway percentages are insignificant. As far as the experiments are concerned we created a number of complex task sets and tested this section's implementation against these sets when agent availability is at its maximum (100%). Note that this is identical with the tests in 8.4. Only the network size is different but this is a factor that could be scaled accurately. After that, we run the same tasks sets on networks with agent availability at 80%. This means that if an agent is asked for collaboration (PTN formation) it declares its status as available for negotiation at 80 out of 100 times. We present these results in the remainder of this section.

As already mentioned we created complex task sets for 200 agents with 3 and 5 subtasks at maximum. Again, total task duration equals the system's total capacity and agents are 100% available. Figure 8.25 compares the Benefit achieved, with the results for the same category (3 subtasks maximum) when each agent is available for 80% of the total requests made to it. The results for the experimental sets were derived using the inactive non gateway mode of Method B in both availability percentages.

Figure 8.26 is almost identical to Figure 8.25. The only difference is that here complex tasks can have up to 5 subtasks at maximum. Considering the data presented in figures 8.25 and 8.26 we can safely derive that reduced availability did not affect heavily the system's Benefit. When we have complex tasks with 3 subtasks there is little difference in terms of Benefit. The highest margin comes up when there are no more than 3 edges for any network node. Even so this difference is only 1%. What is relatively unexpected is the fact that agent networks with reduced availability can rival the performance of maximum availability networks.

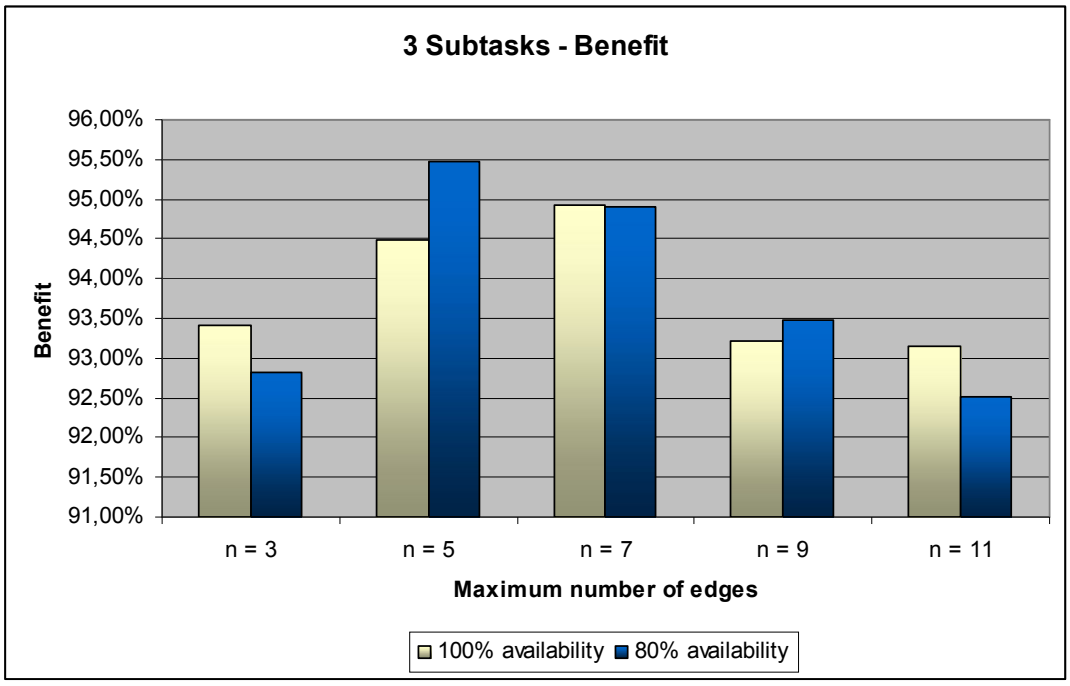


Figure 8.25: Benefit for the inactive non gateway mode of Method B when there are 3 subtasks at maximum in each complex task. The maximum number of edges is n = 3,…, 11. Agents were either 100% or 80% available.

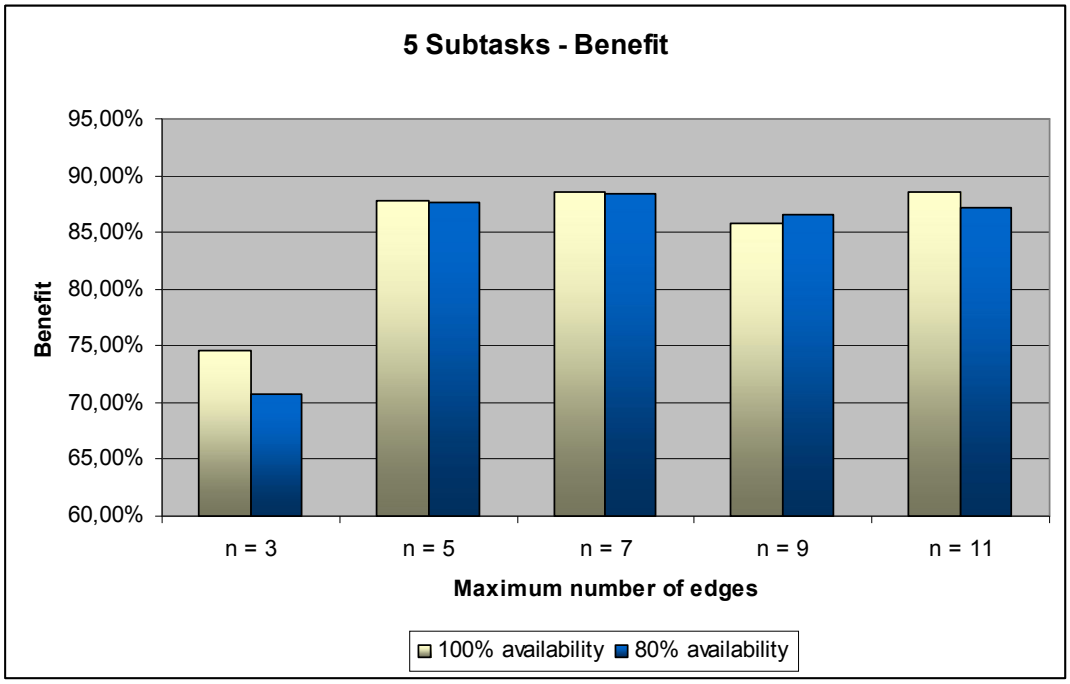


Figure 8.26: Benefit for the inactive non gateway mode of Method B when there are 5 subtasks at maximum in each complex task. The maximum nunber of edges is n = 3,…, 11. Agents were either 100% or 80% available.

The results in this section's figures are categorized by taking into account the maximum number of edges each node can have. The logic behind this decision is that

when availability is reduced, much is expected from the extra choices available to an agent when a peer has refused collaboration (is unavailable).

In order to derive reliable conclusions it would be appropriate to examine the Message Gain in each case, as well. Figure 8.27 presents the Message Gain factor for the experiments conducted for the purposes of this section.

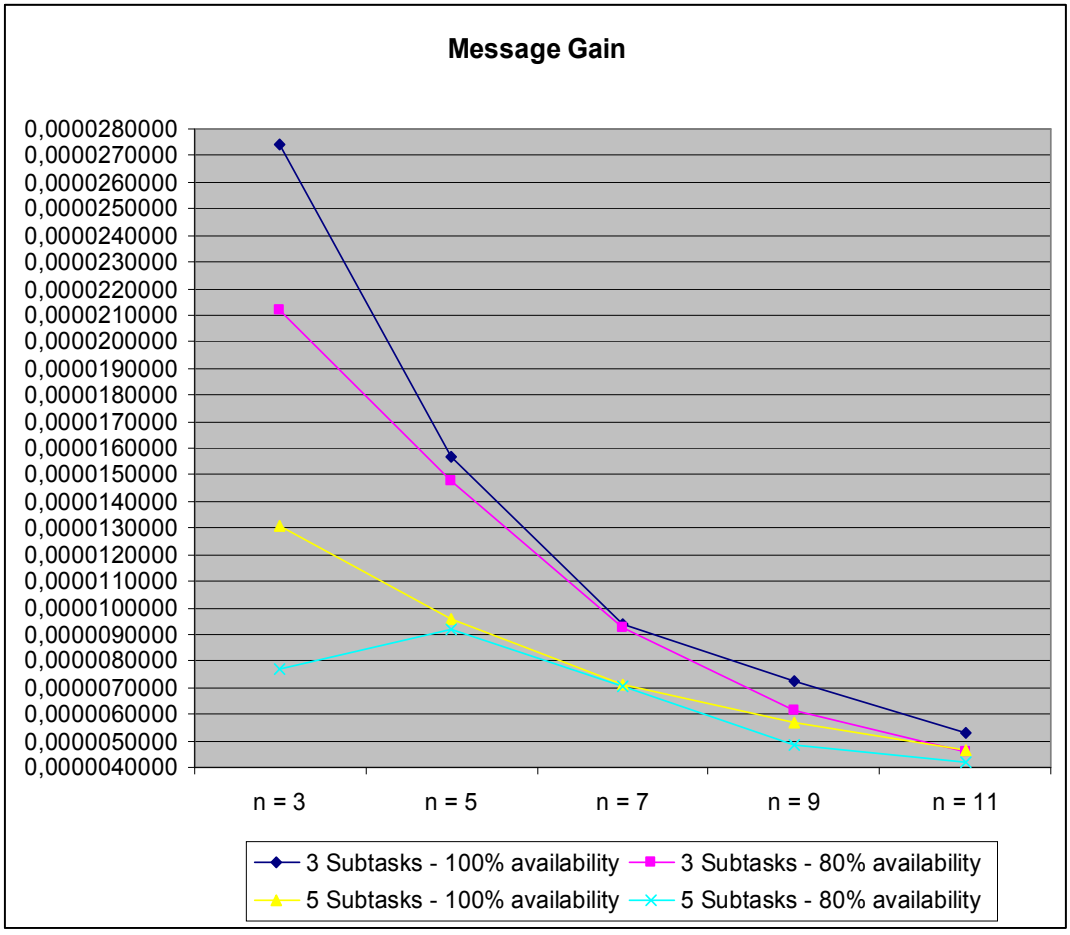


Figure 8.27: Message Gain for experiments with agent networks of 200 agents and maximum number of edges n = 3,..,11. Results stand for 3 and 5 subtasks at maximum with 100% or 80% agent availability.

After considering Figures 8.25, 8.26 and 8.27 we can conclude that only in the case when n = 3 there is a significant difference in the system's performance. When we have n = 3 and no more than 3 subtasks the Benefit is 93.41% for 100% available agents and 92.83% for 80% available agents (difference less than 1%). The same figures for n = 3 and no more than 5 subtasks are 74.60% and 70.75% respectively (difference less than 4%). These differences are not considered as noteworthy but when it comes to Message Gain the landscape is different. For 3 subtasks at maximum the difference for the Message Gain factor is 22.59% in favor of the 100% availability case. For 5 subtasks at maximum this difference increases to a much more considerable 41.25%.

In all other cases, differences in the Benefit factor are minimal and the Message Gain reduction varies between 1% and 15%. The conclusion for such a behavior is as pointed out the choices an agent has. If during a PTN formation for instance a gateway agent receives refusal from another agent its immediate action is to turn to another agent. When n = 3 this is more difficult than the cases where n is greater. Therefore, a lot more messages have to be transmitted in order to form PTNs. Thus the system's performance is much lower. Progressively, as n increases the system is virtually unaffected, because many connections (edges) give many more choices in negotiation terms. For the availability factor that was tested here (80%) it is evident that as soon as n rises to 5 the 20% reduction in availability is balanced by the increased routing alternatives that become available. Moreover as n becomes even higher the system gradually converges as far as Message Gain is concerned (Figure 8.27). A question of major interest is how availability losses can be counter measured by routing alternatives. Furthermore, would the system converge in a homogeneous behavior if n increases towards its maximum possible valuefor any availability percentage?

8.6 Measuring optimality

The experimental results also lead to the conclusion that the benefit achieved by the system implemented in this thesis sharply declines as the problems become harder. Although this is natural, we have to focus mainly on declining system performance when the number of available capabilities rises and the number of subtasks per complex task is large. A natural question that follows is whether this is due to some inherent approach oriented deficiency or to the extreme hardness of the generated instances.

One way to answer this question is to measure the distance between the optimal benefit that can be achieved in the generated instances and the benefit achieved by our system. However, this is far from easy to do in practice since the problems we deal with are highly combinatorial and hence very hard. In practice, finding the optimal benefit in large enough random instances would require collecting all the generated tasks in a single agent and solving the resulting problem in a centralized way through some elaborate branch & bound search algorithm.

To obtain an indication of our method's ability to approach the optimal benefit, we have run experiments with an alternative task generation method which guarantees that all generated tasks are satisfiable, not only on their own, but also considered collectively. Recall that the generation methods detailed above ensure that each task is satisfiable when considered individually but do not guarantee that all tasks are satisfiable when considered together. This alternative generation method proceeds as follows.

At first, the agents' timelines are processed one by one and broken into smaller fragments. Each fragment's duration is determined randomly and is between 1 and 5 time units (half of the agent's full capacity). Then, the complex tasks are constructed. Assuming a task with x subtasks is required, we randomly choose x time fragments from the previously broken down timelines. Each individual fragment of every timeline is chosen only once. For instance, when generating a complex task with 3 subtasks, we randomly select 3 timeline fragments from 3 agents (which are not

necessarily different). Each fragment becomes an individual subtask within the complex task. Constraints are then posted between the subtasks making sure that they are satisfied. For example, assuming two subtasks (timeline fragments) t1=[1,3] and t2=[5,8], a constraint S1+3<S2 may be posted. After a complex has been created, the corresponding timeline fragments are removed from the respective agents' timelines and the process is repeated until no more satisfiable complex tasks can be created. Any remaining timeline fragments are considered as atomic tasks.

This generation method guarantees that all tasks are satisfiable. This is because, ideally, any subtask of a given task can be allocated to the agent where the corresponding timeline fragment, from which it was created, belongs. Then this agent can schedule this subtask in exactly the "correct" timeline fragment. Therefore, the optimal benefit under this generation method is 100%.

We ran experiments with Method B in combination with inactive non-gateways on randomly generated problems with 200 agents where the total capacity required by the generated tasks is equal to the total capacity of the agents. The results are given in Table 8.4. The first column gives the number of capabilities and the maximum number of subtask per complex task. Columns 2 to 4 give the average performance of our method measured in the metrics used across all our experiments. Column 5 gives the average % coverage of the total capacity of the agents (i.e. the aggregation of their timelines) once the scheduling of the tasks has been completed. For instance, 96.78% coverage in the case of one capability and 3 subtasks per task at maximum means that once the allocation and scheduling of the tasks was completed, the 3.22% of the total capacity was free (i.e. not allocated to any task). Column 6 gives the uniform TTL of the generated tasks.

Capabilities – Max. Subtasks	**Benefit**	**Messages**	**Ben/Mess**	**%Timeline Covered**	**TTL**
1 - 3	97.38%	102368	0.000009513	96.78%	10
1 - 7	95.17%	157422	0.000006046	94.83%	10
3 - 3	63.27%	167347	0.000003781	62.76%	30
3 - 7	39.83%	198563	0.000002006	39.21%	30
3 - 3	67.49%	173168	0.000003897	68.15%	50
3 - 7	43.84%	203722	0.000002152	42.87%	50

Table 8.4: Average performance of Method B with inactive non-gateways on problems where all tasks are guaranteed to be satisfiable.

Table 8.4 demonstrates that as the TTL of the tasks increases the benefit achieved also increases. Recall that the TTL in all experiments presented above was set to 10, which is relatively low. This explains, to some extend, the drop in the achieved benefit as the tasks get harder. However, even with TTL=50, the benefit of the system is still under 50% in the case of 3 capabilities and 7 subtasks per complex task. This is of course very far from the optimal 100% benefit but we must note that it is considerably higher than the benefit achieved under the previously used task generation methods. We believe that these results do not demonstrate a deficiency in our approach but they rather show that the problems we tackle are very hard. As explained, the generation method we try here guarantees that all tasks can be satisfied, but in practice it will be

nearly impossible for any heuristic method to locate the appropriate agents and schedule the subtasks to the "correct" timeline fragments for all the generated tasks.

8.7 Experimental results, synopsis and conclusions

In the experiments we have presented there are two models for the generation of the agent network. The first one generates networks of randomly deployed agents, assuming that the geographical distance among agents determines the topology of the system. Networks are constructed by distributing randomly |N| agents in an n×n area, each with a "visibility" ratio equal to r. The alternative method generates agent networks in a way such that connections can be created between any two nodes in the network, discarding the grid and radius parameter.

First, we compare two algorithms, one incomplete and on complete, for solving the DisCOPs we have formed based on the temporal interdependencies of tasks. The experiments for both algorithms (Local Search and Adopt) used the same method for searching and task allocation, thus pointing out the contrast between a complete (Adopt) and an incomplete (Local Search) algorithm for solving DisCOPs generated from complex tasks. Results showed that Adopt manages to outperform Local Search in the number of successfully allocated tasks without increasing the cost produced by the relatively higher number of exchanged messages. This was a rather expected outcome as Adopt contrary to Local Search guarantees to either find the solution if it exists or decisively determine that the problem has no solution.

Then, we used again the networks that helped evaluate Local Search vs Adopt to explore the approaches to searching and task allocation in Sections 7.2 and 7.4. For all experiments, PTNs used Adopt for allocating tasks after solving the related DisCOPs. Experiments verified that Method B consistently demonstrated better results than Method A in each experiment. Both variations of Method B performed better than any variation of Method A in terms of the total number of satisfied complex tasks. A small downside is that Method B produced more messages. This was nevertheless not noteworthy as even with a slight increase at the number of messages Method B consistently produced higher the Message Gain. The results justify the additional negotiating ability inherent to Method B when compared to Method A. Better negotiation leads to forming more PTNs as well as greater numbers of exchanged messages. Moreover, when we compared the two variations of Method B (active and inactive non-gateway mode) we came under the conclusion that the inactive non-gateway mode performed slightly better in virtually all the experiments we did. This is merely due to the fact that when non-gateways are active, they tend to allocate large part of their timeline considerably faster than gateway nodes. Thus, certain nodes or parts of the agent network become quickly overloaded. Moreover, this can reduce the probabilities of successful PTN formation.

The experimental session focused on the impact the change of the network generation method would impose on our framework. Until now, we have agent networks generated when a random visibility ratio (r) was applied on each agent, while all agents occupied positions on an n×n grid (n = 250, r = 25). For this part of the experiments, we used a method that could possibly connect any two agents. The changing parameter was the maximum number of neighbours an agent could have. The reason behind this approach is twofold. First, we wanted to compare Methods A

and B under a different and arguably more random setting for agent networks. Secondly, we aimed to explore how the two variations of our general framework would be affected by being given a set of complex tasks under constant alterations of its physical setting. The results showed that in all cases, but one, Method B achieves a higher benefit on average than Method A. Nevertheless, Method B has again a higher average cost than Method A, measured by the messages sent. As far as approaches to team formation is concerned, Method B with inactive non-gateways "non-GWs" performs slightly better compared to the other one in terms of benefit but it incurs more exchanged messages. Hence, the active non-gateways method achieves slightly better message gain in the case of 3 subtasks per complex task. However, in the case of 7 subtasks this is reversed due to the higher difference in benefit in favor of the inactive non-gateways method. These results conform with the results of the previous experimental sessions showing that our framework behaves in a consistent manner irrespectively of alterations in the topology of the agent network. At the end of this experimental session, we experimented with problem sets with a different distribution of capabilities. The agent networks were randomly generated using the random_connections method but capabilities followed the Zipfean distribution. This led to a considerable decrease in the benefit achieved.

Then we studied how the system performs when not all agents are available at any given moment. We tested limited agent availability for Method B in different agent network settings. The networks in this set of experiments were again generated so that connections can be created between any two nodes in the network. An unexpected outcome was that agent networks with reduced availability achieve high performance standards. Moreover, as the maximum number of connections per agent increases the impact restricted availability has, is diminished.

The experimental results also lead to the conclusion that the benefit achieved by our system declines as the problems become harder. Therefore, we aimed to explore whether this was an inherent deficiency to our system. Thus, we used an alternative task generation method, which secured that all generated tasks for a complex task set would be satisfiable. Experimental results did not reveal any deficiencies rather than the fact that the problems we deal with are very hard to solve.

CHAPTER 9

9. Conclusions and future work

The thesis proposes a novel method for allocating atomic and complex tasks in large-scale networks of homogeneous or heterogeneous cooperative agents. In contrast to prior work, we treat searching, task allocation and scheduling as a single problem and propose a decentralized method for all these tasks where no accumulated or centralized knowledge or coordination is necessary.

Efficient searching for agent groups that can facilitate task requests is accomplished through the use of a dynamic overlay structure of gateway agents and the exploitation of routing indices. A connected subset of agents forms a dominating set of nodes. The dominating set of nodes preserves and maintains the "coverage" of all the other nodes in the network. This makes the solution procedure much easier. The members of the dominating set (called gateway agents) keep track of their own resources and capabilities as well as the available resources and capabilities of the agents they cover. Thus, gateway agents can form local knowledge clusters. Local knowledge is subsequently used in order to decide if they can form teams of agents that will be able to allocate and schedule incoming complex tasks.

The task allocation and scheduling of complex tasks is accomplished by combining dynamic reorganization of agent groups and distributed constraint optimization methods. Agents form Potential Teammates' Networks (PTNs) and try to find the necessary resources for scheduling incoming tasks. We have introduced two different approaches for forming PTNs with two different variations for each. After PTNs are formed, a DisCOP is derived based on the constraints each complex task carries. Then the agents comprising the PTN try to solve the derived DisCOP. If successful, they can schedule the complex task. Alternatively, they can re-route it to another gateway and the same process restarts.

Experimental results displayed the efficiency of the proposed approach for successfully accommodating complex task sets with total capacity requirements equaling that of the system. Especially in the cases where we used a complete algorithm for solving DicCOPs the proposed system displayed high performance standards at relatively low communication costs. Performance declined as problems became more complex. Nevertheless, this is expected, as such problems are widely considered very hard. Our approach also proved itself very consistent when tested alternative agent network topologies or reduced the availability of the agents participating in the agent network.

In the immediate future, we plan to perform a more in-depth experimental investigation of the effect that the various parameters have on the system's performance. Specifically, we are referring to the way networks are generated, their density (as dictated by n and r for geographical networks for example), the size of the complex tasks in terms of subtasks they include, the density of the constraint graph for each complex task, and the number of available capabilities.

Further work targets investigating in more detail the trade-off between DCOP solving and dynamic reorganization of PTNs during the scheduling process. First we intend to clarify through extensive experimentation the contribution that the DCOP algorithm has to the resolution of complex tasks compared to the dynamic reorganization of PTNs. That is, we will measure the percentage of complex tasks that are successfully allocated without reorganization and the extent of reorganization that occurs on average. This investigation will hopefully lead to the development of efficient heuristic methods for choosing whether to continue employing sophisticated DCOP algorithms or simply reorganize the PTN once conflicts are encountered during the scheduling process.

Other, more general, directions for future work include extending our framework to consider other types of resources and tasks with uncertain durations as, as well as to situations where the agents have preferences on the tasks they can serve. Also, it would be interesting to study the use of alternative DCOP algorithms within our method, such as DPOP and its extensions.

Finally, a direction we aim to pursue is the extension of our approach so that changes to agent commitments can be made as tasks arrive dynamically so that a better overall allocation can be achieved. This requires modifying the scheduling and allocation methods involved in our approach, and will most likely increase their run times, but may well result in higher system benefit being obtained.

References

Aarts, E. and Lenstra, J. K., editors (1997). *Local Search in Combinatorial Optimization*. John Wiley & Sons, Inc., New York, NY, USA.

Angluin, D., Aspnes, J., Chen, J., Wu, Y., and Yin, Y. (2005). Fast construction of overlay networks. In *Seventeenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 145–154.

Anussornnitisarn, P., Nof, S. Y., and Etzion, O. (2005). Decentralized control of cooperative and autonomous agents for solving the distributed resource allocation problem. *International Journal of Production Economics*, 98(2):114–128.

Apt, K. R. (2003). *Principles of Constraint Programming*.

Atlas, J. and Decker, K. (2007). A complete distributed constraint optimization method for non-traditional pseudotree arrangements. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–8, New York, NY, USA. ACM.

Bacchus, F. and van Run, P. (1995). Dynamic variable ordering in csps. In *CP*, pages 258–275.

Bain, S., Thornton, J., and Sattar, A. (2005). Evolving variable-ordering heuristics for constrained optimisation. In *CP*, pages 732–736.

Barták(i), R. (1999). Constraint programming - what is behind? In *In Proceedings of the Workshop on Constraint Programming for Decision and Control (CPDC99*, pages 7 – 15.

Barták(ii), R. (1999). Constraint programming: In pursuit of the holy grail. In *Proceedings of Week of Doctoral Students (WDS99)*, pages 555–564.

Béjar, R., Domshlak, C., Fernández, C., Gomes, C., Krishnamachari, B., Selman, B., and Valls, M. (2005). Sensor networks and distributed csp: communication, computation and complexity. *Artificial Intelligence*, 161(1-2):117–147.

Bernuy, A. and Joyanes, L. (2008). *Innovations and Advanced Techniques in Systems, Computing Sciences and Software Engineering*, chapter Collaborative e-Business and Software Agents, pages 121–126. Springer.

Bessière, C. and Cordier, M.-O. (1993). Arc-consistency and arc-consistency again. In *AAAI*, pages 108–113.

Bessière, C., Freuder, E. C., and Régin, J.-C. (1999). Using constraint metaknowledge to reduce arc consistency computation. *Artificial Intelligence*, 107(1):125–148.

Bessière, C., Maestre, A., and Meseguer, P. (2001). Distributed dynamic backtracking. In *CP '01: Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*, page 772, London, UK. Springer-Verlag.

Bordini, R. H., Dastani, M., Dix, J., and Fallah-Seghrouchni, A. E., editors (2005). *Multi-Agent Programming: Languages, Platforms and Applications*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*. Springer.

Bradshaw, J. M., editor (1997). *Software Agents*. AAAI Press, Menlo Park, CA.

Brito, I. and Meseguer, P. (2003). Distributed forward checking. In *CP*, pages 801–806.

Carle, J. and Simplot-Ryl, D. (2004). Energy-efficient area monitoring for sensor networks. *Computer*, 37(2):40–46.

Chechetka, A. and Sycara, K. (2006). No-commitment branch and bound search for distributed constraint optimization. In *Proceedings of Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 1427 – 1429.

Chmeiss, A. and Jégou, P. (1998). Efficient path-consistency propagation. *International Journal on Artificial Intelligence Tools*, 7(2):121–142.
Cooper, P. R. and Swain, M. J. (1994). Arc consistency: parallelism and domain dependence. pages 207–235.
Cost, R. S., Labrou, Y., and Finin, T. W. (2001). Coordinating agents using agent communication languages conversations. In *Coordination of Internet Agents: Models, Technologies, and Applications*, pages 183–196.
Craenen, B. G. W., Eiben, A. E., and van Hemert, J. I. (2003). Comparing evolutionary algorithms on binary constraint satisfaction problems. *IEEE Trans. Evolutionary Computation*, 7(5):424–444.
Crespo, A. and Garcia-Molina, H. (2002). Routing Indices For Peer-to-Peer Systems. In *ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, page 23, Washington, DC, USA. IEEE Computer Society.
Davin, J. and Modi, P. J. (2006). Hierarchical variable ordering for distributed constraint optimization. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 1433–1435, New York, NY, USA. ACM.
Davis, M., Logemann, G., and Loveland, D. (1962). A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397.
Davis, M. and Putnam, H. (1960). A computing procedure for quantification theory. *J. ACM*, 7(3):201–215.
Dechter, R. (1990). Enhancement schemes for constraint processing: backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41(3):273–312.
Dechter, R. and Pearl, J. (1985). The anatomy of easy problems: A constraint-satisfaction formulation. In *IJCAI*, pages 1066–1072.
Dechter, R. and Pearl, J. (1987). Network-based heuristics for constraint-satisfaction problems. *Artif. Intell.*, 34(1):1–38.
Dignum, V. (2004). *A model for organizational interaction: based on agents, founded in logic*. PhD thesis, Universiteit Utrecht.
Dignum, V., VΓ˜zquez-salceda, J., and Dignum, F. (2004). Omni: Introducing social structure, norms and ontologies into agent organizations. In *in β€□PROMAS*, pages 181–198. Springer.
dos Santos, F. and Bazzan, A. L. (2009). An ant based algorithm for task allocation in large-scale and dynamic multiagent scenarios. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09, pages 73–80, New York, NY, USA. ACM.
Eiben, A. E., Hemert, J. I. v., Marchiori, E., and Steenbeek, A. G. (1998). Solving binary constraint satisfaction problems using evolutionary algorithms with an adaptive fitness function. In *PPSN V: Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, pages 201–210, London, UK. Springer-Verlag.
Eiben, A. E. and Ruttkay, Z. (1997). *Handbook of Evolutionary Computation*, chapter 5, pages 1–8. IOP Publishing Ltd., Bristol, UK.
Endriss, U., Maudet, N., Sadri, F., and Toni, F. (2006). Negotiating socially optimal allocations of resources. *J. Artif. Int. Res.*, 25(1):315–348.
Ephrati, E. and Rosenschein, J. S. (1997). A heuristic technique for multi-agent planning. *Annals of Mathematics and Artificial Intelligence*, 20(1-4):13–67.
Esteva, M., Rodrguez-Aguilar, J. A., Sierra, C., Garcia, P., and Arcos, J. L. (2001). Ón the formal specifications of electronic institutions. In *Agent Mediated Electronic*

Commerce, The European AgentLink Perspective., pages 126–147, London, UK. Springer-Verlag.
F., D. and J., W. (2003). Distributed dominant pruning in ad hoc networks. In *In Proc. IEEE 2003 Intβ€™l Conf. Communications (ICC 2003)*, pages 353 – 357.
Findler, N. V. and Elder, G. D. (1995). Multi-agent coordination and cooperation in a distributed dynamic environment with limited resources. *Artificial Intelligence in Engineering*, 9:229–238.
Finin, T. W., Joshi, A., Labrou, Y., and Peng, Y. (2000). Multi-agent systems, internet and applications - introduction. In *HICSS*.
Fitzpatrick, S. and Meertens, L. G. L. T. (2001). An experimental assessment of a stochastic, anytime, decentralized, soft colourer for sparse graphs. In *SAGA '01: Proceedings of the International Symposium on Stochastic Algorithms*, pages 49–64, London, UK. Springer-Verlag.
Freuder, E. C. (1978). Synthesizing constraint expressions. *Commun. ACM*, 21(11):958–966.
Freuder, E. C. and Quinn, M. J. (1985). Taking advantage of stable sets of variables in constraint satisfaction problems. In *In IJCAIβ€™85*, pages 1076–1078.
Frost, D. and Dechter, R. (1995). Look-ahead value ordering for constraint satisfaction problems. In *IJCAI'95: Proceedings of the 14th international joint conference on Artificial intelligence*, pages 572–578, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
Fruhwirth, T. and Abdennadher, S. (2003). *Essentials of Constraint Programming*. Springer.
Garcia-Molina, H. and Crespo, A. (2003). Semantic overlay networks for p2p systems. Technical Report 2003-75, Stanford InfoLab.
Gaschnig, J. (1974). A constraint satisfaction method for inference making. In *Annual Conf. on Circuit System Theory*.
Gaschnig, J. (1978). Experimental case studies of backtrack vs. waltz-type vs. new algorithms for satisficing assignment problems. In *2nd Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, pages 268 – 277.
Geelen, P. A. (1992). Dual viewpoint heuristics for binary constraint satisfaction problems. In *ECAI '92: Proceedings of the 10th European conference on Artificial intelligence*, pages 31–35, New York, NY, USA. John Wiley & Sons, Inc.
Genesereth, M. R. and Ketchpel, S. P. (1994). Software agents. *Communications of the ACM*, 37(7):48–53.
Gent, I. P., MacIntyre, E., Prosser, P., and Walsh, T. (1996). The constrainedness of search. In *AAAI/IAAI, Vol. 1*, pages 246–252.
Gershman, A., Meisels, A., and Zivan, R. (2006). Asynchronous forward-bounding for distributed constraints optimization. In *Proceeding of the 2006 conference on ECAI 2006: 17th European Conference on Artificial Intelligence August 29 – September 1, 2006, Riva del Garda, Italy*, pages 103–107, Amsterdam, The Netherlands, The Netherlands. IOS Press.
Gershman, A., Zivan, R., Grinshpoun, T., Grubshtein, A., and Meisels, A. (2008). Measuring distributed constraint optimization algorithms. In *Proc. of the 10th international workshop on Distributed Constraint Reasoning at AAMAS'08*, Estoril, Portugal.
Ginsberg, M. L. (1993). Dynamic backtracking. *J. Artif. Intell. Res. (JAIR)*, 1:25–46.
Ginsberg, M. L. and McAllester, D. A. (1994). Gsat and dynamic backtracking. In *PPCP*, pages 243–265.

Glover, F. (1996). Tabu search and adaptive memory programing β€“ advances, applications and challenges. In *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer.
Glover, F. and Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA.
Goldberg, D. (1989). *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley.
Goldman, C. V. and Zilberstein, S. (2003). Optimizing information exchange in cooperative multi-agent systems. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 137–144, New York, NY, USA. ACM.
Goldman, C. V. and Zilberstein, S. (2004). Decentralized control of cooperative systems: categorization and complexity analysis. *J. Artif. Int. Res.*, 22(1):143–174.
Golomb, S. W. and Baumert, L. D. (1965). Backtrack programming. *J. ACM*, 12(4):516–524.
Grinshpoun, T. and Meisels, A. (2008). Completeness and performance of the apo algorithm. *J. Artif. Int. Res.*, 33:223–258.
Hamadi, Y. (2001). Distributed interleaved parallel and cooperative search in constraint satisfaction networks. In *In Proc. IAT-01, Singappore*.
Hamadi, Y. (2002a). Interleaved backtracking in distributed constraint networks. *International Journal on Artificial Intelligence Tools*, 11(2):167–188.
Hamadi, Y. (2002b). Optimal distributed arc-consistency. *Constraints*, 7(3/4):367–385.
Hamadi, Y., Bessière, C., and Quinqueton, J. (1998). Backtracking in distributed constraint networks. In *ECAI*, pages 219–223.
Hamadi, Y. and Ringwelski, G. (2010). Boosting distributed constraint satisfaction. *Journal of Heuristics*.
Hannoun, M., Boissier, O., Sichman, J. S. a., and Sayettat, C. (2000). Moise: An organizational model for multi-agent systems. In *Proceedings of the International Joint Conference, 7th Ibero-American Conference on AI: Advances in Artificial Intelligence*, IBERAMIA-SBIA '00, pages 156–165, London, UK. Springer-Verlag.
Hao, J.-K. and Dorne, R. (1994). A new population-based method for satisfiability problems. In *ECAI*, pages 135–139.
Haralick, R. M. and Elliott, G. L. (1979). Increasing tree search efficiency for constraint satisfaction problems. In *IJCAI'79: Proceedings of the 6th international joint conference on Artificial intelligence*, pages 356–364, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
Harchol-Balter, M., Leighton, T., and Lewin, D. (1999). Resource discovery in distributed networks. In *In Symposium on Principles of Distributed Computing*, pages 229–237.
Hindriks, K. V., Boer, F. S. d., Hoek, W. v. d., and Meyer, J.-J. C. (2001). Agent programming with declarative goals. In *ATAL '00: Proceedings of the 7th International Workshop on Intelligent Agents VII. Agent Theories Architectures and Languages*, pages 228–243, London, UK. Springer-Verlag.
Hindriks, K. V., De Boer, F. S., Van Der Hoek, W., and Ch. Meyer, J.-J. (1999). Agent programming in 3apl. *Autonomous Agents and Multi-Agent Systems*, 2(4):357–401.
Hirayama, K. and Yokoo, M. (1997). Distributed partial constraint satisfaction problem. In *Principles and Practice of Constraint Programming*, pages 222–236.

Hirayama, K. and Yokoo, M. (2005). The distributed breakout algorithms. *Artificial Intelligence*, 161(1-2):89–115.
Hoos, H. and Stutzle, T. (2004). *Stochastic Local Search: Foundations & Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
Horling, B. (2006). *Quantitative Organizational Modeling and Design for Multi-Agent Systems*. PhD thesis, University of Massachusetts at Amherst.
Hübner, J. F., Sichman, J. S. a., and Boissier, O. (2002). Moise+: towards a structural, functional, and deontic model for mas organization. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, AAMAS '02, pages 501–502, New York, NY, USA. ACM.
Hutter, F., Tompkins, D. A. D., and Hoos, H. H. (2002). Scaling and probabilistic smoothing: Efficient dynamic local search for sat. In *CP '02: Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, pages 233–248, London, UK. Springer-Verlag.
Jennings, N. and Wooldridge, M. (1998). *Agent Technology: Foundations, Applications and Markets*. Springer Computer Science.
Jennings, N. R. and Wooldridge, M. (1995). Applying agent technology. *Applied Artificial Intelligence*, 9(4):357–369.
Jennings, N. R. and Wooldridge, M. J. (1996). Software agents. *IEE Review*, pages 17–20.
Johnson, D. S., Aragon, C. R., McGeoch, L. A., and Schevon, C. (1989). Optimization by simulated annealing: an experimental evaluation. part i, graph partitioning. *Oper. Res.*, 37(6):865–892.
Jussien, N., Debruyne, R., and Boizumault, P. (2000). Maintaining arc-consistency within dynamic backtracking. In *CP '02: Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming*, pages 249–261, London, UK. Springer-Verlag.
Kasif, S. (1990). On the parallel complexity of discrete relaxation in constraint satisfaction networks. *Artif. Intell.*, 45(3):275–286.
Kask, K., Dechter, R., and Gogate, V. (2004). Counting-based look-ahead schemes for constraint satisfaction. In *CP*, pages 317–331.
Katsirelos, G. and Bacchus, F. (2005). Generalized nogoods in csps. In *AAAI'05: Proceedings of the 20th national conference on Artificial intelligence*, pages 390 – 396. AAAI Press.
Kim, H.-K. (2005). A component-based approach for integrating mobile agents into the existing web infrastructure. In *Third ACIS International Conference on Software Engineering, Research, Management and Applications (SERA 2005), 11-13, August 2005, Mt. Pleasant, MI, USA*, pages 75–85.
Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.
Koes, M., Nourbakhsh, I., and Sycara, K. (2005). Heterogeneous multirobot coordination with spatial and temporal constraints. In *AAAI'05: Proceedings of the 20th national conference on Artificial intelligence*, pages 1292–1297. AAAI Press.
Kolp, M., Giorgini, P., and Mylopoulos, J. (2006). Multi-agent architectures as organizational structures. *Autonomous Agents and Multi-Agent Systems*, 13:3–25.
Kschischang, F. R., Frey, B. J., and andrea Loeliger, H. (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47:498–519.
Land, A. H. and Doig, A. G. (1960). An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520.

Liang, J., Ko, S. Y., Gupta, I., and Nahrstedt, K. (2005). Mon: on-demand overlays for distributed system management. In *Proceedings of the 2nd conference on Real, Large Distributed Systems - Volume 2*, WORLDS'05, pages 13–18, Berkeley, CA, USA. USENIX Association.
Lieberman, H. (1997). Autonomous interface agents. In *Proceedings, CHI*, Atlanta, GA. ACM.
Liu, J.-S. and Sycara, K. P. (1998). Readings in agents. chapter Multiagent coordination in tightly coupled task scheduling, pages 164–171. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
Lynch, N. A. (1996). *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
M. Boddy, B. Horling, J. P. R. G. R. V. A. C. L. R. K. and Maheswaran, R. (2006). *C-TAEMS language spec. v. 2.02*.
Mackworth, A. K. (1977a). Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118.
Mackworth, A. K. (1977b). On reading sketch maps. In *IJCAI'77: Proceedings of the 5th international joint conference on Artificial intelligence*, pages 598–606, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
Mackworth, A. K. and Freuder, E. C. (1985). The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25(1):65–74.
Mackworth, A. K., Mulder, J. A., and Havens, W. S. (1985). Hierarchical arc consistency: Exploiting structured domains in constraint satisfaction problems. *Computational Intelligence*, 1(3):118–126.
Maheswaran, R. T., Szekely, P., Becker, M., Fitzpatrick, S., Gati, G., Jin, J., Neches, R., Noori, N., Rogers, C., Sanchez, R., Smyth, K., and Vanbuskirk, C. (2008). Predictability & criticality metrics for coordination in complex environments. In *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, pages 647–654, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.
Mailler, R. (2006). Using prior knowledge to improve distributed hill climbing. In *IAT*, pages 514–521.
Mailler, R. and Lesser, V. (2003). A Mediation Based Protocol for Distributed Constraint Satisfaction. pages 49–58, Acapulco, Mexico.
Mailler, R. and Lesser, V. (2004). Solving Distributed Constraint Optimization Problems Using Cooperative Mediation. In *Proceedings of Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, pages 438–445. IEEE Computer Society.
Mailler, R. and Lesser, V. R. (2006a). Asynchronous partial overlay: A new algorithm for solving distributed constraint satisfaction problems. *J. Artif. Intell. Res. (JAIR)*, 25:529–576.
Mailler, R. and Lesser, V. R. (2006b). A cooperative mediation-based protocol for dynamic distributed resource allocation. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 36(1):80–91.
Marriott, K. and Stuckey, P. J. (1998). *Programming with Constraints: an Introduction*. MIT Press.
Mayfield, J., Labrou, Y., and Finin, T. W. (1995). Evaluation of kqml as an agent communication language. In Wooldridge, M., Muller, J. P., and Tambe, M., editors, *ATAL*, volume 1037 of *Lecture Notes in Computer Science*, pages 347–360. Springer.

Meisels, A. and Zivan, R. (2003). R.: Asynchronous forward-checking on discsps. In *In: Proceedings of the Workshop on Distributed Constraints (DCR-03*.
Meisels, A. and Zivan, R. (2007). Asynchronous forward-checking for discsps. *Constraints*, 12(1):131–150.
Minton, S., Johnston, M. D., Philips, A. B., and Laird, P. (1990). Solving large-scale constraint-satisfaction and scheduling problems using a heuristic repair method. In *AAAI*, pages 17–24.
Minton, S., Johnston, M. D., Philips, A. B., and Laird, P. (1994). Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. pages 161–205.
Modi, P. J., Jung, H., Tambe, M., Shen, W.-M., and Kulkarni, S. (2001). A dynamic distributed constraint satisfaction approach to resource allocation. In *CP '01: Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*, pages 685–700, London, UK. Springer-Verlag.
Modi, P. J., Shen, W. M., Tambe, M., and Yokoo, M. (2005). Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1-2):149–180.
Modi, P. J. and Veloso, M. (2004). Multiagent meeting scheduling with rescheduling. In *Distributed Constraint Reasoning, (DCR) 2004*.
Mohr, R. and Henderson, T. C. (1986). Arc and path consistence revisited. *Artif. Intell.*, 28(2):225–233.
Montanari, U. (1974). Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132.
Morris, P. (1993). The breakout method for escaping from local minima. In *Proc. of AAAI-93*, pages 40–45, Washington, DC.
Moyaux, T., Chaib-draa, B., and D'Amours, S. (2003). Multi-agent coordination based on tokens: reduction of the bullwhip effect in a forest supply chain. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, AAMAS '03, pages 670–677, New York, NY, USA. ACM.
Musliner, D. J. and Goldman, R. P. (2006). Coordinated plan management using multiagent mdps. In *In AAAI Spring Symposium*, pages 73–80. AAAI Press.
Nair, R., Tambe, M., and Marsella, S. (2003). Role allocation and reallocation in multiagent teams: towards a practical analysis. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 552–559, New York, NY, USA. ACM.
Nguyen, T. and Deville, Y. (1998). A distributed arc-consistency algorithm. *Sci. Comput. Program.*, 30(1-2):227–250.
Nguyen, V., Sam-Haroud, D., and Faltings, B. (2004). Dynamic distributed backjumping. In *In Proc. 5th workshop on distributed constraints reasoning DCR-04*.
Nwana, H. S., Lee, L. C., and Jennings, N. R. (1997). Co-ordination in multi-agent systems. In Nwana, H. S. and Azarmi, N., editors, *Software Agents and Soft Computing*, volume 1198 of *Lecture Notes in Computer Science*, pages 42–58. Springer.
Ostwald, J., Lesser, V., and Abdallah, S. (2005). Combinatorial auction for resource allocation in a distributed sensor network. In *RTSS '05: Proceedings of the 26th IEEE International Real-Time Systems Symposium*, pages 266–274, Washington, DC, USA. IEEE Computer Society.
Paredis, J. (1993). Genetic state-space search for constrained optimization problems. In *IJCAI'93: Proceedings of the 13th international joint conference on Artifical*

intelligence, pages 967–972, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
Parkes, D. C. and Shneidman, J. (2004). Distributed implementations of vickrey-clarke-groves mechanisms. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 261–268, Washington, DC, USA. IEEE Computer Society.
Perlin, M. (1992). Arc consistency for factorable relations. *Artif. Intell.*, 53(2-3):329–342.
Petcu, A. and Faltings, B. (2004). A value ordering heuristic for local search in distributed resource allocation. In *In CSCLP04, Lausanne Switzerland, Febuary*.
Petcu, A. and Faltings, B. (2005a). An Efficient Constraint Optimization Method for Large Multiagent Systems. In *AAMAS 05 - LSMAS workshop (Large Scale Multi-Agent Systems)*.
Petcu, A. and Faltings, B. (2005b). A scalable method for multiagent constraint optimization. In *IJCAI'05: Proceedings of the 19th international joint conference on Artificial intelligence*, pages 266–271, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
Petcu, A. and Faltings, B. (2007). Mb-dpop: a new memory-bounded algorithm for distributed optimization. In *IJCAI'07: Proceedings of the 20th international joint conference on Artifical intelligence*, pages 1452–1457, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
Pokahr, A., Braubach, L., and Lamersdorf, W. (2005). Jadex: A bdi reasoning engine. In R. Bordini, M. Dastani, J. D. and Seghrouchni, A. E. F., editors, *Multi-Agent Programming*, pages 149–174. Springer Science+Business Media Inc., USA. Book chapter.
Prosser, P. (1993). Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9:268–299.
Pullan, W. J. and Zhao, L. (2004). Resolvent clause weighting local search. In *Canadian Conference on AI*, pages 233–247.
Pynadath, D. V. and Tambe, M. (2002). Multiagent teamwork: analyzing the optimality and complexity of key theories and models. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 873–880, New York, NY, USA. ACM.
Rao, A. S. (1996). Agentspeak(l): Bdi agents speak out in a logical computable language. In *MAAMAW '96: Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world : agents breaking away*, pages 42–55, Secaucus, NJ, USA. Springer-Verlag New York, Inc.
Ricci, A., Viroli, M., and Omicini, A. (2007). Cartago: a framework for prototyping artifact-based environments in mas. In *Proceedings of the 3rd international conference on Environments for multi-agent systems III*, E4MAS'06, pages 67–86, Berlin, Heidelberg. Springer-Verlag.
Ringwelski, G. (2005). An arc-consistency algorithm for dynamic and distributed constraint satisfaction problems. *Artif. Intell. Rev.*, 24(3-4):431–454.
Rojas, M. C. R. (1996). From quasi-solutions to solution: An evolutionary algorithm to solve csp. In *CP*, pages 367–381.
Rossi, F., Beek, P. v., and Walsh, T. (2006). *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA.
Rothkopf, M. H., Pekec, A., and Harstad, R. M. (1995). Computationally manageable combinatorial auctions. Technical report.

Russell, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition.

Sabin, D. and Freuder, E. C. (1994). Contradicting conventional wisdom in constraint satisfaction. In *PPCP '94: Proceedings of the Second International Workshop on Principles and Practice of Constraint Programming*, pages 10–20, London, UK. Springer-Verlag.

Samal, A. and Henderson, T. (1987). Parallel consistent labeling algorithms. *Int. J. Parallel Program.*, 16(5):341–364.

Sandholm, T. (2000). Issues in computational vickrey auctions. *Int. J. Electron. Commerce*, 4(3):107–129.

Scerri, P., Farinelli, A., Okamoto, S., and Tambe, M. (2005). Allocating tasks in extreme teams. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 727–734, New York, NY, USA. ACM.

Shoham, Y. (1993). Agent oriented programming. *Artificial Intelligence*, 60(1):51–92.

Silaghi, M.-C. and Faltings, B. (2005). Asynchronous aggregation and consistency in distributed constraint satisfaction. *Artif. Intell.*, 161(1-2):25–53.

Silaghi, M.-C., Sam-Haroud, D., and Faltings, B. (2000). Asynchronous search with aggregations. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 917–922. AAAI Press.

Sims, M., C. D. and Lesser, V. (2008). Knowledgeable automated organization design for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 16(2):151 – 185.

Smith, S. F., Gallagher, A., and Zimmerman, T. (2007). Distributed management of flexible times schedules. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–8, New York, NY, USA. ACM.

Subrahmanian, V. S., Bonatti, P. A., Dix, J., Eiter, T., Kraus, S., Ozcan, F., and Ross, R. B. (2000). *Heterogenous Active Agents*. MIT Press.

Sultanik, E. A., Modi, P. J., and Regli, W. C. (2007). On modeling multiagent task scheduling as a distributed constraint optimization problem. In *IJCAI'07: Proceedings of the 20th international joint conference on Artifical intelligence*, pages 1531–1536, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Theocharopoulou, C., Partsakoulakis, I., Vouros, G. A., and Stergiou, K. (2007). Overlay networks for task allocation and coordination in dynamic large-scale networks of cooperative agents. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems, AAMAS '07, Article 55*, pages 1 – 8.

Thornton, J., Pham, D. N., Bain, S., and Ferreira, V. (2004). Additive versus multiplicative clause weighting for sat. In *AAAI'04: Proceedings of the 19th national conference on Artifical intelligence*, pages 191–196. AAAI Press.

Tompkins, D. A. D. and Hoos, H. H. (2003). Scaling and probabilistic smoothing: dynamic local search for unweighted max-sat. In *AI'03: Proceedings of the 16th Canadian society for computational studies of intelligence conference on Advances in artificial intelligence*, pages 145–159, Berlin, Heidelberg. Springer-Verlag.

Tsang, E., Yung, P., and Li, J. (2004). Eddie-automation, a decision support tool for financial forecasting. *Decis. Support Syst.*, 37(4):559–565.

Tsang, E. P. K. and Warwick, T. (1990a). Applying genetic algorithms to constraint satisfaction optimization problems. In *ECAI*, pages 649–654.

Tsang, E. P. K. and Warwick, T. (1990b). Applying genetic algorithms to constraint satisfaction optimization problems. In *ECAI*, pages 649–654.
Van Hentenryck, P. (1989). *Constraint satisfaction in logic programming*. MIT Press, Cambridge, MA, USA.
Vernooy, M. and Havens, W. S. (1999). An examination of probabilistic value-ordering heuristics. In *AI '99: Proceedings of the 12th Australian Joint Conference on Artificial Intelligence*, pages 340–352, London, UK. Springer-Verlag.
Vouros, G. A. (2007). Information searching and sharing in large-scale dynamic networks. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, AAMAS '07, pages 49:1–49:8, New York, NY, USA. ACM.
Vouros, G. A. (2008). Searching and sharing information in networks of heterogeneous agents. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 3*, AAMAS '08, pages 1525–1528, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.
Wallace, R. J. and Freuder, E. C. (2005). Constraint-based reasoning and privacy/efficiency tradeoffs in multi-agent problem solving. *Artificial Intelligence*, 161(1-2):209–227.
Walsh, W. E. and Wellman, M. P. (1998). A market protocol for decentralized task allocation. In *ICMAS '98: Proceedings of the 3rd International Conference on Multi Agent Systems*, page 325, Washington, DC, USA. IEEE Computer Society.
Waltz, D. L. (1972). Generating semantic descriptions from drawings of scenes with shadows. Technical report, Cambridge, MA, USA.
Warwick, T. and Tsang, E. (1994). Using a genetic algorithm to tackle the processors configuration problem. In *SAC '94: Proceedings of the 1994 ACM symposium on Applied computing*, pages 217–221, New York, NY, USA. ACM.
Wellman, M. P. (1996). *Market-oriented programming: some early lessons*, chapter 4, pages 74–95. World Scientific Publishing Co., Inc., River Edge, NJ, USA.
Xinghuo Yu, Weixing Zheng, B. W. and Yao, X. (1998). A novel penalty function approach to constrained optimization problems with genetic algorithms. *Advanced Computational Intelligence and Intelligent Informatics*, 2(6):208–213.
Xu, Y., Scerri, P., Yu, B., Okamoto, S., Lewis, M., and Sycara, K. (2005). An integrated token-based algorithm for scalable coordination. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 407–414, New York, NY, USA. ACM.
Xuan, P., Lesser, V., and Zilberstein, S. (2001). Communication decisions in multi-agent cooperation: model and experiments. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pages 616–623, New York, NY, USA. ACM.
Yang, Y., Rana, O. F., Walker, D. W., Georgousopoulos, C., Aloisio, G., and Williams, R. (2002). Agent based data management in digital libraries. *Parallel Computing*, 28(5):773–792.
Yen, J., Yin, J., Ioerger, T. R., Miller, M. S., Xu, D., and Volz, R. A. (2001). Cast: collaborative agents for simulating teamwork. In *IJCAI'01: Proceedings of the 17th international joint conference on Artificial intelligence*, pages 1135–1142, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
Yeoh, W., Felner, A., and Koenig, S. (2008). Bnb-adopt: an asynchronous branch-and-bound dcop algorithm. In *AAMAS '08: Proceedings of the 7th international joint*

conference on Autonomous agents and multiagent systems, pages 591–598, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.
Yokoo, M. (1994). Weak-commitment search for solving constraint satisfaction problems. In *AAAI '94: Proceedings of the twelfth national conference on Artificial intelligence (vol. 1)*, pages 313–318, Menlo Park, CA, USA. American Association for Artificial Intelligence.
Yokoo, M. (1995). Asynchronous weak-commitment search for solving distributed constraint satisfaction problems. In *CP '95: Proceedings of the First International Conference on Principles and Practice of Constraint Programming*, pages 88–102, London, UK. Springer-Verlag.
Yokoo, M. (1997). Why adding more constraints makes a problem easier for hill-climbing algorithms: Analyzing landscapes of csps. In *CP*, pages 356–370.
Yokoo, M., Durfee, E. H., Ishida, T., and Kuwabara, K. (1992). Distributed constraint satisfaction for formalizing distributed problem solving. In *ICDCS*, pages 614–621.
Yokoo, M., Durfee, E. H., Ishida, T., and Kuwabara, K. (1998). The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10:673–685.
Yokoo, M. and Hirayama, K. (1996). Distributed breakout algorithm for solving distributed constraint satisfaction problems. In *Proceedings of the Second International Conference on Multi-Agent Systems*, pages 401 – 408.
Zambonelli, F., Jennings, N. R., and Wooldridge, M. (2003). Developing multiagent systems: The gaia methodology. *ACM Trans. Softw. Eng. Methodol.*, 12:317–370.
Zhang, W., Wang, G., Xing, Z., and Wittenburg, L. (2005). Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence*, 161(1-2):55–87.
Zhang, Y. and Mackworth, A. K. (1991). Parallel and distributed algorithms for finite constraint satisfaction problems. In *in Proc. of Third IEEE Symposium on Parallel and Distributed Processing*, pages 394–397.
Zhang, Y. and Mackworth, A. K. (1992). Parallel and distributed finite constraint satisfaction: Complexity, algorithms and experiments. Technical report, Vancouver, BC, Canada, Canada.
Zhang, Y., Volz, R. A., loerger, T. R., and Yen, J. (2004). A decision-theoretic approach for designing proactive communication in multi-agent teamwork. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 64–71, New York, NY, USA. ACM.
Zivan, R. and Meisels, A. (2004a). Concurrent backtrack search on discsps. In *FLAIRS Conference*.
Zivan, R. and Meisels, A. (2004b). Concurrent dynamic backtracking for distributed csps. In *CP*, pages 782–787.
Zivan, R. and Meisels, A. (2006). Concurrent search for distributed csps. *Artificial Intelligence*, 170(4):440–461.