Distributed Constraint Optimization
and Scheduling in Dynamic Environments

Sankalp Khanna
BEng(ElectEng), MInfTech

Institute for Integrated and Intelligent Systems
Science, Environment, Engineering and Technology
Griffith University

Submitted in fulfilment of the requirements of the degree of
Doctor of Philosophy

July 2010
for my parents ...
First and foremost I would like to thank my supervisors, Prof Abdul Sattar and Dr Bela Stantic for their constant guidance and support at every step of this thesis. I am especially indebted to Prof. Sattar, for being a friend, philosopher, mentor and guide, and for dragging me back on track every time I strayed. Without you I would not be here today.

I would like to thank the Australian e-Health Research Centre, for providing an extremely generous scholarship and a vibrant research environment, and for supporting me through my extended health crisis. Watching research translate to impact all around you is very inspiring for a PhD student. Equally inspiring were the Friday Challenges - thanks Marilla. I am especially grateful for the mentoring and supervision provided by Prof. David Hansen and Prof. Anthony Maeder. Your kindness and ongoing support has proved invaluable through this journey.

I would also like to thank the staff at the Princess Alexandra Hospital, Brisbane, for their support and help in understanding the hospital scheduling environment. I am particularly grateful to Dr. David Cook and Helen Werder, for their initial discussions, and to Dr. Peter Moran, who welcomed all my questions and provided precious understanding of the intricacies of the problem.

I acknowledge the assistance I have received from the following people throughout this thesis. Dr. Aditya Ghose, for his invaluable advice at the confirmation stage, Prof. Makoto Yokoo, for his insightful tips on how to proceed, Tim Cleaver for his help in creating the proof-of-concept application, Chris Stanbridge for helping set up the server for the experimental evaluation, Zim, Deanne, Mario and Rohit for ongoing assistance with Java, and Bevan, for last minute Latex troubleshooting.

I have to also express my gratitude to Marion O’Connor and Natalie Dunstan. I think without your ongoing advice and assistance on how to cut red tape and get
things done, we would all be running around like headless chickens.

Most importantly, I have to thank my family, for pushing me, believing in me, and above all, for always being there for me. This thesis is dedicated to my parents. Last, but not the least, I wish to thank my wife and soulmate, Awanika, for her unfailing support through some very trying times, and for giving me not one, but two reasons to always smile through it all - Khushi and Sachi.
Abstract

Efficient scheduling of resources in a complex dynamic environment poses a significant research challenge. The problem is compounded in the case of a distributed real world environment where several departments are working at optimizing their own resources. Multiple departmental schedules thus need to be optimized simultaneously to ensure proper utilization of resources.

Much of the current efforts at solving scheduling problems with these characteristics fail to model the inherent complexity and dynamism, leaving the task of making the actual scheduling decisions to the user. The work described in this thesis aims to provide a multi-agent framework, driven by a distributed constraint optimization strategy, to model and efficiently solve this class of problems.

The foundation case studied in this thesis is the elective surgery scheduling problem, as it is an intrinsically distributed and complex real world problem. Efficient scheduling of elective surgery is of critical importance to the optimum utilization of the public health system, a topical issue with an aging population.

A summary of the main contributions of this study is as follows:

- We present an intelligent agent based approach for solving complex distributed scheduling problems in dynamic environments. Motivated by the challenge of solving the real world problem of scheduling elective surgery in an under-resourced and encumbered health system, we propose a methodology which can be used to build intelligent software agents that generate optimal schedules on behalf of their respective departments. In our solution, each agent, trained with the appropriate constraints, preferences and priorities, optimizes schedules for their respective department and then negotiates to resolve inter-agent constraints. The architecture of each agent incorporates an interface module to handle internal and external communication, an intelligence mod-
ule to perform decision making and learning, and a Distributed Constraint Optimization Problem (DCOP) engine to drive the optimization. [Part of this work was published at the MedInfo 2007 World Congress (Khanna et al., 2007b)].

• We propose an algorithm, the Dynamic Complex Distributed Constraint Optimization Problem (DCDCOP) algorithm, for solving Dynamic Complex DCOP problems. DCDCOP does not rely on the static organization of agents into depth first search trees for problem solving, and is capable of handling complex sub-problems, found in many real world DCOP problems, without using decomposition or compilation. Further, it allows each agent the flexibility to choose its own local solver. We also introduce a novel metric called Degree of Unsatisfaction ($DU$) and use it for guiding inter-agent negotiation for problem solving in the DCDCOP algorithm. Empirical evaluation of the DCDCOP algorithm and the $DU$ metric is reported to establish their suitability for the class of problem we seek to address. [Parts of this work were published at the AAMAS 2009 conference (Khanna et al., 2009c) and the WI/IAT 2009 conference (Khanna et al., 2009c)].

• We introduce ASES, an Automated Scheduler for Elective Surgery. Developed as a proof-of-concept prototype application, ASES represents the novel marriage of rational agency and distributed constraint optimization necessitated by the problem domain. It aptly demonstrates the suitability of using our proposed methodology for providing ongoing schedule optimization and can also be used to study the effect of fluctuation in staffing or resource levels on theatre utilization. [Part of this work was published at the PRIMA 2010 Conference (Khanna et al., 2010)].
Statement of Originality

This work has not previously been submitted for a degree or diploma in any university. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made in the thesis itself.

Sankalp Khanna
July 2010
## Contents

1 Introduction ................................................................. 1
   1.1 Research Contributions ............................................. 4
   1.2 Outline .............................................................. 5

2 Literature Review ......................................................... 7
   2.1 The Scheduling Problem ............................................. 7
      2.1.1 Types of Scheduling ............................................ 8
      2.1.2 Complex Dynamic Scheduling Environments ................. 8
      2.1.3 Scheduling for Distributed Environments .................. 10
      2.1.4 Constraint Based Scheduling .................................. 11
   2.2 The Constraint Reasoning Paradigm ................................. 12
      2.2.1 Constraint Satisfaction Problems ............................ 12
      2.2.2 Constraint Optimization Problems ............................ 14
      2.2.3 Constraint Guided Search .................................... 15
   2.3 Multi-Agent Systems .................................................. 17
      2.3.1 Negotiation in Multi-Agent Systems .......................... 21
      2.3.2 Distributed Problem Solving Using Multi-Agent Systems .... 21
   2.4 Distributed Constraint Reasoning .................................... 22
      2.4.1 Distributed Constraint Satisfaction ........................... 23
2.4.2 An Example of DisCSP ........................................ 23
2.4.3 Solving DisCSPs ............................................. 26
2.4.4 Search in DisCSPs ........................................... 26
2.4.5 Distributed Constraint Optimization Problems ............... 36
2.4.6 The DCOP Formalism ....................................... 36
2.4.7 Current State of the Art ................................... 37
2.4.8 Solving Dynamic Complex Problems .......................... 44
2.5 Summary ...................................................... 46

3 Scheduling in the Health Domain ................................ 47
3.1 Properties of Health Domain .................................... 48
3.2 Elective Surgery and Waiting Lists ............................... 48
3.3 Elective Surgery at the PA Hospital ............................... 52
3.4 Sharing of Resources between Elective and Emergency Surgery ... 57
3.5 Current State-of-the-Art ....................................... 57
   3.5.1 DISA - Distributed Interactive Scheduling with Abstractions 58
   3.5.2 The GPGP Approach ....................................... 60
   3.5.3 ADAPT .................................................. 61
   3.5.4 Medical Path Agents (MedPAge) ............................. 62
   3.5.5 Policy-Agents ............................................ 63
   3.5.6 Agent Based Information Logistics in Anaesthesiology (AGIL) 65
   3.5.7 Scheduling Organ Transplants ............................... 67
   3.5.8 Discussion ............................................... 68
3.6 Intelligent Management of Elective Surgery .................... 68
3.7 Proposed Architecture of Agent .................................. 70
   3.7.1 The DCOP Engine ......................................... 71
   3.7.2 Intelligence Module ....................................... 71
4 An Efficient Algorithm for Solving Dynamic Complex DCOP Problems 75

4.1 Degree of Unsatisfaction (DU) for DCOP 76

4.1.1 Defining $D C D$ and $D U$ 76

4.1.2 An Example of Calculating $D U$ 78

4.2 The DCDCOP Algorithm 80

4.2.1 Example of DCDCOP Execution 82

4.3 Soundness of the DCDCOP Algorithm 83

4.4 CostDCOP - Using Cost instead of $D U$ 87

4.5 Experimental Evaluation 87

4.5.1 Preliminary Evaluation 90

4.5.2 Extended Evaluation 90

4.6 Summary 96

5 ASES: an Automated Scheduler for Elective Surgery 99

5.1 Domain Mapping 100

5.2 The DCOP Engine 102

5.3 Implementation 102

5.4 Evaluation 105

5.5 Summary 108

6 Conclusion 109

6.1 Summary of Contributions 109

6.2 Future Work 111

Bibliography 113

Appendix A: Related Publications 131
# List of Figures

2.1 A Map Colouring Problem ........................................ 13
2.2 A Classification of Agent Types (Nwana, 1996) .............. 18
2.3 Nwana’s Smart Agent (Nwana, 1996) .......................... 18
2.4 Using Agents to Model the Physical World (Muller et al., 2004) ... 19
2.5 Using Agents to Model the Entire Healthcare Domain (Schweiger and Krcmar, 2004) ........................................ 20
2.6 Contract Net Protocol at Work (Moreno et al., 2001a) ........ 20
2.7 A Distributed Map Colouring Problem .......................... 24
2.8 The DDCR Adapter Approach to Solving Dynamic DCOPs (Lass et al., 2008) ........................................ 45

3.1 Median Waiting Time for Elective Surgery .................. 49
3.2 Patients Waiting for Elective Surgery as at 1 Jan 2010 (Queensland Health, 2010) ........................................ 50
3.3 Patients Waiting for Elective Surgery as at 1 Jan 2010 by Surgical Speciality (Queensland Health, 2010) .................. 51
3.4 Current Model for Scheduling Elective Surgery at the PAH ...... 53
3.5 Emergency Admissions in Queensland 2005/06 to 2009/10 (Queensland Health, 2010) ........................................ 56
3.6 Temporal Multi-Agent Topology of DISA (Friha, 1998a) ........ 58
3.7 Functional Diagram of DISA (Berry et al., 1994) ............. 59
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.8</td>
<td>A Typical Patient Scheduling Episode in GPGP (Decker and Li, 1998)</td>
<td>60</td>
</tr>
<tr>
<td>3.9</td>
<td>ADAPT: Simulating Patient Scheduling for Radiation Therapy (Heine and Kirn, 2004)</td>
<td>61</td>
</tr>
<tr>
<td>3.10</td>
<td>MedPAge Coordination Protocol (Paulussen et al., 2004b)</td>
<td>62</td>
</tr>
<tr>
<td>3.11</td>
<td>Conventional Process of OT Scheduling (Czap and Becker, 2003)</td>
<td>64</td>
</tr>
<tr>
<td>3.12</td>
<td>Individual Agent Decision Making in Policy Agents (Czap and Becker, 2003)</td>
<td>64</td>
</tr>
<tr>
<td>3.13</td>
<td>Entities (Agents) and Information Flow (Knublauch et al., 2000)</td>
<td>65</td>
</tr>
<tr>
<td>3.14</td>
<td>Agents for Information Management in a Future Hospital (Knublauch et al., 2000)</td>
<td>66</td>
</tr>
<tr>
<td>3.15</td>
<td>Proposed Agent Based Theatre Management System (Sedlmayr et al., 2000)</td>
<td>66</td>
</tr>
<tr>
<td>3.16</td>
<td>MAS Architecture for Scheduling Organ Transplants (Moreno et al., 2001a)</td>
<td>67</td>
</tr>
<tr>
<td>3.17</td>
<td>Agent Interaction for Scheduling Elective Surgery at the PAH</td>
<td>69</td>
</tr>
<tr>
<td>3.18</td>
<td>Proposed Architecture of Agent</td>
<td>70</td>
</tr>
<tr>
<td>3.19</td>
<td>Intelligent Decision Support</td>
<td>72</td>
</tr>
<tr>
<td>4.1</td>
<td>Calculating Degree of Unsatisfaction - An Example</td>
<td>79</td>
</tr>
<tr>
<td>4.2</td>
<td>DCDCOP: Agent Decision Flow</td>
<td>83</td>
</tr>
<tr>
<td>4.3</td>
<td>DCDCOP Execution - Step 1</td>
<td>84</td>
</tr>
<tr>
<td>4.4</td>
<td>DCDCOP Execution - Step 2</td>
<td>85</td>
</tr>
<tr>
<td>4.5</td>
<td>DCDCOP Execution - Step 3</td>
<td>86</td>
</tr>
<tr>
<td>4.6</td>
<td>Performance of ADOPT vs DCDCOP ($LD = 2$)</td>
<td>88</td>
</tr>
<tr>
<td>4.7</td>
<td>Performance of ADOPT vs DCDCOP ($LD = 3$)</td>
<td>89</td>
</tr>
<tr>
<td>4.8</td>
<td>Average Time (sec) Vs Problem Size (Variables)</td>
<td>91</td>
</tr>
<tr>
<td>5.1</td>
<td>Using Agents for Scheduling Elective Surgery at the PAH</td>
<td>100</td>
</tr>
<tr>
<td>5.2</td>
<td>Mapping the Problem</td>
<td>101</td>
</tr>
<tr>
<td>5.3</td>
<td>Bookings Agent</td>
<td>103</td>
</tr>
</tbody>
</table>
5.4 Resources Agent ......................................................... 104
5.5 Nursing Agent .......................................................... 105
5.6 Agentspeak(L) for the Nursing Agent ......................... 106
5.7 Scheduling Conflicts Vs Theatre Utilization ................ 107
List of Tables

3.1 Classifying the Elective Surgery Scheduling Problem (ESSP) . . . . . 48
3.2 Resource Managers in Elective Surgery Scheduling . . . . . . . . . . 53

4.1 Average Time (sec) Vs Problem Size (Variables) . . . . . . . . . . . 91
4.2 Evaluating the Control Set: Time (sec) . . . . . . . . . . . . . . . . . 92
4.3 Evaluating the Control Set: Messages Sent . . . . . . . . . . . . . . . 93
4.4 Evaluating the Control Set: Cycles . . . . . . . . . . . . . . . . . . . 94
4.5 Evaluating the Control Set: Final Solution Quality . . . . . . . . . . 95
4.6 The Benchmark Dataset . . . . . . . . . . . . . . . . . . . . . . . . 97
# List of Algorithms

1. A Sketch of the Asynchronous Backtracking Algorithm (Yokoo et al., 1992a) .......................... 28
2. A Sketch of the Asynchronous Weak Commitment Search Algorithm (Yokoo, 1995) .......................... 30
3. A Sketch of the Distributed Stochastic Algorithm (Zhang et al., 2003) .......................... 31
4. A Sketch of the Distributed Breakout Algorithm (Zhang and Wittenburg, 2002) .......................... 32
5. A Sketch of the Multi-Variable Asynchronous Weak Commitment Search Algorithm (Yokoo and Hirayama, 1998) .......................... 33
6. The Dynamic Agent Ordering Algorithm (Zhou et al., 2003) .......................... 34
7. Modified Assign Local Variables Procedure for Dynamic Agent Ordering and Nogood Repairing (Zhou et al., 2004) .......................... 35
8. A Sketch of the ADOPT Algorithm (Modi et al., 2003) .......................... 39
9. A Sketch of the DPOP Algorithm (Petcu and Faltings, 2005c) .......................... 40
10. A Sketch of the OptAPO Algorithm (Mailler and Lesser, 2004b) .......................... 42
11. A Sketch of the NCBB Algorithm (Chechetka and Sycara, 2006b) .......................... 43
12. The DCDCOP Algorithm .......................... 82
13. The CostDCOP Algorithm .......................... 87
the most exciting research topics to me are those inspired by challenging real-world problems

Scheduling has been a well studied area in Computer Science research. Incorporating knowledge from areas of Operations Research, Artificial Intelligence and Multi-Agent Systems (MAS), several previous efforts have focused on finding practical ways of solving scheduling problems in real world domains.

However, the state-of-the-art methods suffer from several shortcomings when applied to real world scheduling problems. In particular:

- Real world scheduling problems are often more complex than those used for developing such systems. The problems addressed in literature are simplified versions, and, while most address real world scheduling requirements, they either model only some of the constraints, optimize only some of the functional requirements, or are built to optimally solve only specific instances of a problem.

- Real world scheduling problems are often dynamic. Unforeseen and unaccounted for events can adversely affect the schedule, rendering it obsolete. Thus, schedules that have gone through the costly computation process of optimization may no longer be valid because of small changes in the environment. In fact, a harder problem than creating a schedule is adapting the schedule to the changing environment (Sauer, 2000).

- Real world scheduling problems are often distributed. While centralized deci-
sion making is easier to design, it does not effectively capture the inherently distributed decision making nature of large organizations that contain multiple departments. While all these departments logically work in unison to meet the organization’s goals, each department’s scheduling activity primarily focuses on optimizing its own performance. Scheduling for such an organizational structure becomes the task of creating and managing several schedules and keeping all of them aligned and consistent.

Of significant importance to this study are problems that embody all of the above characteristics, i.e. are very complex, inherently distributed, and riddled with unforeseen happenings. Finding solutions for problems with these characteristics poses a significant research challenge.

The health domain presents several inherently distributed complex scheduling problems in very dynamic environments where the complexity and changeability of interacting factors demands a flexible and dynamic solution, in order to achieve a high level of utilization and cater for many different competing priorities. A particularly complex problem is the scheduling of patients, medical staff and resources for elective surgery in an overburdened public health system. Statistics show that as at 1 January 2010, 34,480 Queenslanders were waiting for elective surgery, of whom almost 20% had waited longer than a clinically desirable time (Queensland Health, 2010). Krempels and Panchenko (2006b) reveal that in the Operation Theatre Scheduling domain they study, it takes one person 3-5 full working days to create a Nurse Roster. Aldea et al. (2001) reveal that between 15% and 30% patients in Europe die waiting for transplant operations and propose a multi-agent based decision support system to improve the coordination process. Several collaborative research projects all over the world are working at improving coordination and scheduling at various levels in the health domain.

A study conducted as part of this research aimed to identify the state-of-the-art scheduling systems used in the health domain. The findings, however, revealed that most scheduling activity is still being carried out manually. In departments where computers are being used, these are still only being used to record manually optimized schedules. While several recently proposed health-related scheduling systems including DISA (Friha, 1998b), MedPage (Paulussen et al., 2006a), and Policy Agents (Krempels and Panchenko, 2006c), use multi-agent systems to model their domains, distributed schedule optimization has been largely overlooked or proposed as future aims. Further, since transient elective surgery scheduling data is not captured in any current mechanisms, there is a lack of benchmark problems in this domain. We believe that, while all of these methods help to improve the state-of-the-art, what is missing is an flexible intelligent methodology that can adapt itself
to the complexity of the problem, without modification or down-scaling. Handling changes caused by the dynamic nature of the environment in a timely manner is also a non trivial challenge.

The Multi-Agent Systems (MAS) paradigm offers expressively rich and natural fit mechanisms for modeling and negotiation for solving distributed problems. Despite being a relatively young research area (the first asynchronous Distributed Constraint Satisfaction Problem (DisCSP) algorithm was proposed in 1992 (Yokoo et al., 1992b), and the first complete Distributed Constraint Optimization Problem (DCOP) algorithm, ADOPT, was proposed in 2003 (Modi et al., 2003)), the Distributed Constraint Reasoning formalism has also developed rapidly to offer efficient and sophisticated algorithms to model and solve a variety of naturally distributed multi-agent problems. Several notable DCOP approaches employing techniques from search (e.g. ADOPT and its several variants), dynamic programming (e.g. DPOP (Petcu and Faltings, 2005c) and its several variants) and cooperative mediation (e.g. APO (Mailler and Lesser, 2004b)) have emerged and are being successfully used to model and solve problems in many fields, including sensor networks (Lesser et al., 2003), (Modi et al., 2005), (Scerri et al., 2003), meeting scheduling (Maheswaran, 2004b) and coordination of unmanned aerial vehicles (Schurr et al., 2005).

Current complete DCOP algorithms, however, largely fail to scale well enough to solve large complex problems, typically the class of problems we seek to address. Further, while DCOP algorithms can theoretically utilize decomposition or compilation to deal with complex sub-problems, this generally results in blowing the distributed problem size out of proportion. Local search algorithms, that trade off completeness for practical efficiency, have been proposed for dealing with DCOPs, but are generally synchronous and thus unsuitable for addressing the class of problems that interest our study. Also, while DCOP offers an excellent mapping for representing many real world problems, the general design of DCOP algorithms is static. Most DCOP algorithms are based on tree structures, which are static in nature and need to be continually rebuilt in dynamic environments. In addition, given the nature of the real world domains relevant to our research, partial centralization based strategies would not be a good fit here because of obvious departmental privacy and decision control concerns.

We propose a multi-agent architecture for modeling and solving dynamic complex distributed optimization problems, like the Elective Surgery Scheduling Problem. In our model, intelligent agents, armed with the constraints, preferences, and priorities of the administrators, optimize schedules for their respective departments. They then negotiate in a privacy-preserving manner (i.e. without sharing more information than is essential) to resolve inter-agent constraints. The architecture of each
CHAPTER 1. INTRODUCTION

agent incorporates an interface module to handle internal and external communication, an intelligence module to handle decision making and learning, and a DCOP engine to drive the optimization.

Another key contribution of this thesis is a novel asynchronous DCOP algorithm, the Dynamic Complex Distributed Constraint Optimization Problem Algorithm (DCDCOP). DCDCOP preserves the decentralized decision control mechanisms of the problem at hand and offers a robust, flexible, and efficient mechanism for modeling and solving dynamic complex problems. The algorithm is experimentally evaluated to show that DCDCOP scales better and outperforms the state-of-the-art asynchronous DCOP algorithms.

1.1 Research Contributions

The main contributions of this thesis are as follows:

- A comprehensive case study of elective surgery scheduling procedures in the public health domain. Detailing the current encumbrances affecting the health system, the case study describes current procedures followed at one of the leading adult surgical hospitals in Australia and identifies shortcomings of current systems in offering an efficient solution to the problem. It also provides insight into the type of inter-departmental conflicts and the negotiation strategies employed to overcome these conflicts. The case study provides the foundation for the multi-agent model and distributed optimization algorithms presented in the thesis.

- The development of an intelligent agent based approach for modeling and solving complex dynamic distributed optimization problems such as the scheduling of elective surgery. The architecture of each agent incorporates an interface module to handle internal and external communication, an intelligence module to handle decision making and learning, and a DCOP engine to drive the optimization.

- The introduction of novel Static Cost Density, Dynamic Cost Density and Degree of Unsatisfaction (DU) metrics for DCOP problems. Our DU metric represents how far an agent’s instantiation is from an optimal solution.

- Development of an efficient asynchronous DCOP algorithm for solving complex distributed optimization problems in dynamic environments, the Dynamic Complex Distributed Constraint Optimization Problem (DCDCOP) algorithm. The DCDCOP algorithm preserves the decentralized decision control
nature of distributed departmental structures found in problems like the Elective Surgery Scheduling Problem and uses our measure of $DU$ to dynamically guide agent ordering in the problem solving process. A proof of soundness for the DCDCOP algorithm is also presented.

- Empirical evaluation of the performance of the DCDCOP algorithm as compared to several state-of-the-art DCOP algorithms. Composed of three sets of experiments, the evaluation provides liberal evidence of various advantages offered by DCDCOP over existing asynchronous DCOP algorithms.

- Empirical evaluation of the performance of the $DU$ metric within the DCDCOP algorithm by comparing it to a novel variant of DCDCOP, CostDCOP, that uses total solution cost instead of $DU$ to guide agent ordering. The evaluation offers an important insight into the actual impact of $DU$ on convergence and final solution quality.

- Development of a proof-of-concept prototype implementation of our intelligent agent based approach for solving the elective surgery scheduling problem. The Automated Scheduler for Elective Surgery (ASES) system provides ongoing negotiation for schedule optimization and also demonstrates the effect of fluctuation in staffing and resource levels on theatre utilization.

### 1.2 Outline

The remainder of this thesis is organized as follows:

Chapter 2 provides an overview of background concepts related to scheduling, constraint reasoning, multi-agent systems, distributed constraint satisfaction and distributed constraint optimization. The state-of-the-art in DisCSP and DCOP algorithms are discussed and shortcomings in dealing with complex dynamic domains identified.

Chapter 3 looks at the problem of scheduling as applied to the health domain. The environment is classified, a case study of elective surgery scheduling is presented and the state-of-the-art research efforts at addressing surgery scheduling are discussed. This is followed by our Multi-Agent methodology for modeling and solving the Elective Surgery Scheduling Problem. We then discuss the agent architecture and various functional components, especially focusing on the requirements for the DCOP engine.

Chapter 4 describes our novel DCDCOP algorithm. First we discuss the novel $DU$ metric as applied to DCOP problems and explain it with an example. The
CHAPTER 1. INTRODUCTION

DCDCOP algorithm is presented next and algorithm execution is discussed with another example. This is followed by details of the implementation and experimental evaluation of the DCDCOP algorithm and the $DU$ metric.

Chapter 5 presents ASES, a proof-of-concept demonstration of our multi-agent methodology. We discuss the mapping of the Elective Surgery Scheduling Problem to the DCOP notation, and then present particulars of the implementation of the ASES system. We conclude with a description of ongoing development.

Chapter 6 summarizes the research contributions of this thesis and discusses research challenges and future work.
Chapter 2

Literature Review

“Although the distributed solution protocol currently has some limitation mainly in its efficiency, we believe that it is one of the promising directions of distributed problem solving”

Hirayama (2006)

This study is primarily focused on the constraint reasoning approach to solving complex dynamic distributed optimization problems. Motivated by the problem of scheduling elective surgery in modern hospitals, we investigate a flexible intelligent Multi-Agent approach to model the problem and a distributed constraint reasoning approach to optimally solve it. This chapter presents our understanding of the background acquired in trying to understand and address the problem.

In this Chapter, we introduce the scheduling problem. We then discuss constraint based reasoning, the paradigm underlying the application of AI techniques for solving scheduling problems. We then introduce the Multi-Agent Systems (MAS) paradigm followed by detailed discussion on Distributed Constraint Reasoning (DCR), focussing on the state-of-the-art Distributed Constraint Optimization Problem (DCOP) algorithms. This is followed by a discussion of inadequacies in current DCOP algorithms when applied to dynamic complex problems.

2.1 The Scheduling Problem

The words planning, scheduling, and resource allocation have been used somewhat loosely in defining the actual problem. While some researchers mark specific bound-
aries between planning (what to do), scheduling (when and how to do it) and resource allocation (order/assignment) tasks, others use the terms interchangeably or inclusively. For the purpose of this study, we will take a broader view and encompass the tasks of deciding what to do, when and how to do it, and allocation, into the scheduling problem. We thus adopt the broader definition of scheduling:

**Definition 1.** *Scheduling deals with the temporal assignment of activities to limited resources where a set of constraints has to be regarded* (Sauer, 2000).

### 2.1.1 Types of Scheduling

The task of scheduling itself can be further classified into three types depending on the type of problems it handles (Sauer, 2000):

- **Predictive Scheduling:** Predictive scheduling deals with the creation of the schedule in advance based on available knowledge. Predictive scheduling systems assume environmental stability and solution executability and reduce the problem to a comparatively simple optimization problem. Historically, these have dominated scheduling research.

- **Reactive Scheduling:** Reactive scheduling systems are built on the understanding that scheduling is an ongoing reactive process where evolving and changing circumstances continually force reconsideration and revision of pre-established plans (Smith, 1995). These generally follow an iterative repair approach, reacting to changes in the environment using appropriate actions to handle each event.

- **Interactive Scheduling:** Interactive scheduling is built about the actual scheduling process of an organization, combining predictive and reactive scheduling with human interaction. It, therefore, acts as a decision support tool, optimizing initial schedules, and reacting to changes while allowing users to interact with the decision making process. While this is logically sound, most “interactive scheduling systems” are not yet truly intelligent and leave the actual task of decision making to the user.

### 2.1.2 Complex Dynamic Scheduling Environments

Scheduling problems are decidedly complex, combinatorial problems that are in general NP-hard (Fromherz, 2001). The task becomes more complicated if the scheduling environment is complex and dynamic. As there are no finite measures
for dynamics and complexity of environment, and as this study is focused on solving such problems, we briefly discuss complex dynamic environments and their effect on scheduling.

Norvig and Cohn (1997) define the complexity of a computing environment as being the contribution of three dimensions - increased number of users, increased number of interactions between them, and the increased number of goals of the environment. In keeping with this definition and the currently accepted definition of "complex Distributed Constraint Satisfaction Problems" (Yokoo and Hirayama, 1998) as having more than one local variable, we define a complex scheduling environment:

**Definition 2.** A complex scheduling environment is one which has a large number of variables, a large number of constraints between them, and multiple, often conflicting, scheduling goals.

Environmental stability is rare in real world environments. A dynamic environment is one where the environment, and thus the scheduling requirements associated with it, undergoes changes with time. Disruptions in the schedule can broadly be classified into three major groups (Kocjan, 2002):

- **Activity Changes:** These are typically requests for new activities, change in activity duration, or removal from the schedule. These could be caused by various reasons including last minute orders, cancellations, improving utilization, etc.
- **Resource Changes:** These are typically requests to assign a different resource and could be requested due to machine failure, availability of better/faster machines, cost reduction, etc.
- **Temporal Changes:** These are typically requests to modify the temporal assignment of an activity. These could be short term changes (to cope with resource or activity changes), or long term changes (to regularise or optimize workflow).

Dealing with these changes, i.e. rescheduling to adapt to the changed environment, is often more difficult than the initial task of scheduling (Sauer, 2000).

Scheduling systems thus tend to suffer from one or more of the following problems (Prieditis et al., 2004):

- They do not react well to dynamic environments. When the unexpected happens, the remaining schedule is usually discarded because it is no longer relevant. On the other hand, a lot of time is spent on scheduling events far into the
future, events that in a world full of uncertainty have little chance of occurring as scheduled.

- They do not react well to time. On one hand, scheduling is not real-time, i.e. the systems don’t run within the time constraints of the task. On the other hand, they don’t improve given more time. One would expect that a system should improve its decision-making, given more time.

- Their performance is sub-optimal. Complex scheduling decisions, like ordering competing tasks, are left up to the user of the system, and the decisions returned are often not optimal.

- They are expressively limited. They lack the ability to model all types of domain constraints and are thus not good for modeling complex system requirements.

### 2.1.3 Scheduling for Distributed Environments

A key characteristic of a number of resource allocation problems is that they are distributed. The distributed nature is defined in that the control of resources and thus decision making power does not rest with a central scheduling authority. In the case of an organizational structure, each department would control its own resources and set its own goals. However, they must all also work together towards achieving the organization’s central goals.

With regard to scheduling, this leads to one of the following:

- Each department creating its own schedule and working according to it isolated from the actions of other departments or the organization as a whole.

- Each department creating its own schedule but modifying its schedule to ensure it is working in coordination with other departments.

- Several departments coordinating with each other to work on a central schedule. Each department may also be creating a departmental schedule in coordination with the central schedule.

The first of the above scenarios does not pose a grave problem and can in fact be viewed as several small centralized scheduling problems.

The second and third scenarios, however, greatly complicate the task of scheduling as they bring complex constraints, competing priorities, data homogeneity and privacy issues into the task. For example, several tasks may need resources from more
2.1. The Scheduling Problem

than one department and no department would be able to make a decision about allocating these resources without collaboration with other departments, thus leading to resource contention. This study is aimed at solving problems of this nature.

2.1.4 Constraint Based Scheduling

Approaches to solving the scheduling problem find their roots in several different research areas. While scheduling was traditionally looked at as an Operations research problem, it was introduced to the AI community by Fox et al. (1982) with ISIS, a hierarchical intelligent constraint based scheduling system to solve the job-shop scheduling problem. It soon became the preferred tool for solving scheduling problems because of its "rich set of tools for capturing domain knowledge and novel techniques for exploiting that knowledge" (Fox, 1990). Several AI based scheduling systems followed, including OPIS (Ow and Smith, 1988), SONIA (Collinot et al., 1988), YAMS (Parunak et al., 1985), DAS (Burke and Prosser, 1991), and REDS (Hadavi et al., 1992). YAMS was based on the Contract Net architecture (Davis and Smith, 1983) and is credited with being the first intelligent scheduler to exploit distributed AI techniques. DAS and REDS built upon YAMS and REDS introduced the notion of distributing the scheduling problem across a society of agents to allow agents to optimize on different criteria. Several modern systems followed, with systems like DISA (Friha, 1998b), MedPage (Paulussen et al., 2006a), and Policy Agents (Krempels and Panchenko, 2006c) specifically focussing on the healthcare domain.

Scheduling has been dubbed as the “killing application” for constraint based reasoning (Bartak et al., 2010). This is attributed to the ability of constraint based techniques to integrate OR techniques in general AI algorithms and combine the complimentary strengths of both approaches. It is thus well accepted as the dominant choice for modeling and solving scheduling problems.

Employing this approach, the scheduling problem is reduced to a constraint reasoning problem. The tasks to be scheduled are represented by variables, possibly restricted by constraints that define conditions on them. The search space is the space of possible assignments. While some scheduling problems can employ a constraint satisfying solution, we often demand an optimal schedule. In this case, an objective function is used to define the optimality criterion and this represents the goal of the scheduling process, often to minimize cost or maximize utility.
2. Literature Review

2.2 The Constraint Reasoning Paradigm

Constraint-based reasoning has been a feverishly researched area of automated reasoning in artificial intelligence. Finding application in many different domains, the popularity of the Constraint reasoning paradigm lies in its ability to effectively model and solve many different types of real world problems.

2.2.1 Constraint Satisfaction Problems

Modeling a problem as a Constraint Satisfaction Problem (CSP) translates the objective of solving the problem to that of finding a combination of instantiations that satisfy the constraints using effective and efficient generic problem solving techniques that exploit the CSP structure.

Formally, a Constraint Satisfaction Problem (Mackworth, 1975; Mackworth and Freuder, 1985) consists of:

1. A finite ordered set of variables \( V = \{V_1, V_2, V_3, \ldots, V_n | n \in \mathbb{Z}^+ \} \),
2. A domain set \( D = \{D_1, D_2, D_3, \ldots, D_n \} \), containing a finite and discrete domain \( D_i \) for each variable \( V_i \),
3. A constraint set \( C = \{C_1, C_2, \ldots, C_m \}, m \in \mathbb{Z}^+ \) where each \( C_j, \forall j \in [1, m] \) is a predicate defined on the cartesian product of the domains of variables involved in \( C_j \) and is satisfied iff the value assignment satisfies \( C_j \), and
4. An ordered solution set \( S = \{v_1, v_2, v_3, \ldots, v_n | v_i \in D_i, \forall j \in [1, n] \} \) where each \( v_i \) is an instantiation of \( V_i \) such that all constraints in the constraint set \( C \) are satisfied.

Simple examples for demonstrating the concept of Constraint Satisfaction include the N-Queens Problem and the Map/Graph Colouring Problems.

In the N-Queens Problem, the aim is to place \( n \) queens on an \( n \times n \) chessboard such that no queen can attach any other. Formalising this problem as a CSP for a 8-Queens Problem would give us 8 variables, say \( x_1, x_2, \ldots, x_8 \), each representing a position of a queen in rows 1..8. The domain of each variable would be \( \{1, 2, 3, 4, 5, 6, 7, 8\} \) and the constraints between variables would be represented as \( \{(x_i \neq x_j \land |i-j| \neq |x_i - x_j|) \forall i, j : i \neq j \} \) where \( i \) and \( j \) represent variable positions for \( x \).

The Graph/Map Colouring Problem similarly aims to assign a colour to each node in graph or adjacent country in a map such that no two neighbouring nodes/countries
2.2. The Constraint Reasoning Paradigm

(a) Map of Australia

(b) Constraint Graph

(c) Sample Solution

Figure 2.1: A Map Colouring Problem
2. Literature Review

have the same colour. Figure 2.1 shows a Map Colouring problem (with possible solution and constraint graph) for the map of Australia (adapted from (Russell and Norvig, 2003)). Formalising this problem as a CSP would give us 7 variables, say $x_1, x_2, ..., x_7$, each representing a state of Australia. The domain of each variable would be $\text{Red}, \text{Green}, \text{Blue}$ and the constraints between variables would be represented as $\{(x_i \neq x_j)\}$ where $x_i$ and $x_j$ represent variables with common borders.

### 2.2.2 Constraint Optimization Problems

While CSPs are a powerful tool for modeling and solving difficult problems, the CSP paradigm fails when the problem is not solvable, i.e. there exists no instantiation of variables that satisfies all constraints. This is especially true of several real-world problem domains. However, in several such cases, some constraints are not necessarily mandatory and can be violated (relaxed) in solutions without causing such solutions to be unacceptable. These problems are defined as Over-constrained Constraint Satisfaction Problems (OCSPs) or Constraint Optimization Problems (COPs) and the objective in this case is to find an instantiation of variables that minimizes the number of unsolved constraints or minimizes/maximizes a cost/utility function defined on the constraints. Constraints are appropriately weighted to represent their individual priority. Another common practice is to classify the constraints as hard or soft to differentiate between mandatory and optional constraints. For the rest of this study, we assume that the objective function in optimization problems seeks to minimize total cost.

Formally, we can then define a Constraint Optimization Problem as consisting of:

1. A finite ordered set of variables $V = \{V_1, V_2, V_3, ..., V_n | n \in Z^+\}$,

2. A domain set $D = \{D_1, D_2, D_3, ..., D_n\}$, containing a finite and discrete domain $D_i$ for each variable $V_i$,

3. A constraint set $C = \{C_1, C_2, ..., C_m\}, m \in Z^+$ where each $C_j, \forall j \in [1, m]$ is defined as a cost function on a pair of variables, $f_{i,i'} : D_i D_{i'} \rightarrow N, \forall V_i, V_{i'} \in V$, and

4. An ordered solution set $S = \{v_1, v_2, v_3, ..., v_n | v_i \in D_i, \forall j \in [1, n]\}$ where the aggregate cost $F = \sum_{(x_i, x_{i'} \in V)} f_{i,i'}(d_i, d_{i'})$, $x_i \leftarrow d_i, x_{i'} \leftarrow d_{i'}$ is minimized.
2.2.3 Constraint Guided Search

Constraint guided search sits as the core technology behind solving CSPs and COPs. As a research area, it has received as much, if not more, academic attention than scheduling. Understanding constraint guided search is critical to modeling any solution to the scheduling problem.

Constraint guided search algorithms can be broadly classified into three types, those that propose a systematic search methodology or Exact Search methods, those that propose a somewhat random search methodology or Local Search methods, and those that combine features of both the above or Hybrid Search methods.

Exact Search

Exact Search methods, or systematic search or complete search methods, guarantee that they can find a solution to the problem if it exists, or conclude that the problem has no solution. Broadly, these algorithms incrementally extend a partial solution by repeatedly assigning a value to one or more variables using a chosen heuristic/scheme. When no acceptable value can be found, the algorithms backtrack (retract/undo) an assignment based on a chosen heuristic/scheme, thus the name backtracking, which was coined by American mathematician D. H. Lehmer in the 1950s.

Several forms of Backtracking have been introduced in literature. Russell and Norvig trace backtracking methods back to the 19th century, though the most generic algorithm for Chronological Backtracking is credited to Golomb and Baumert (1965). Several modifications have been proposed since, some improving backtracking such as by Bitner and Reingold (1975), others modifying the method to propose Backjumping (Gaschnig, 1978), Conflict Directed Backjumping (Prosser, 1993), Graph Based Backjumping (Dechter, 1990), Backmarking (Gaschnig, 1977), Min-Conflict Backtracking (Jiang et al., 1994), and Forward Checking (Haralick and Elliott, 1980) to name a few.

Exact search methods have been extremely successful for solving problems that are small in size. However, the very nature of the algorithms makes exact search methods an impractical methodology for larger size problems. In fact, the use of a specific heuristic that might seem to improve the search can sometimes make Backtracking even more impossible to apply to larger problems (Baker, 1994).
2. Literature Review

Local Search

Local Search methods offer a quick, but incomplete, solution to solving CSPs. The starting point in the algorithm is a complete but inconsistent assignment to all variables in the problem. The algorithm then iteratively tries to improve the solution (using a specified cost metric) until a consistent (or acceptable in the case of COP) solution is obtained. However, these algorithms suffer from the problem of getting stuck in local minima i.e. the algorithm cannot find a cost improving move and thus does not know where to go. Several sophisticated strategies have been proposed to deal with local minima and Local Search algorithms based on these, such as GSAT (Selman et al., 1992), Breakout (Morris, 1993), WalkSAT (Selman et al., 1994), Tabu Search (Glover, 1989, 1990), PAWS (Hutter et al., 2002), and SAPS (Thornton et al., 2004) to name a few, form the most sophisticated and preferred tools for solving large size CSPs and COPs.

The primary problem in Local Search methods, however, is that, since they are not complete, they are unable to recognise when a CSP is unsolvable or when a COP has reached the best possible solution.

Hybrid Search

Hybrid Search algorithms combine Exact Search and Local Search methods to try and overcome limitations of each method. The most popular of these, Weak-Commitment Search (Yokoo, 1994), is based on Min-Conflict Backtracking, but like Local Search, it abandons the whole partial solution when it is unable to proceed further. Another method, BOBT (Eisenberg and Faltings, 2003), named thus as it combines the breakout algorithm (BO) with backtracking (BT), first executes BO and if it is unable to complete in a predefined number of steps, it terminates, derives a fail-first variable order from the constraint weights and the graph structure, and starts BT.

Improving Constraint Guided Search

Several strategies can be employed to improve constraint guided search. The general idea is to preprocess the CSP to help solve it better. Of these, the most accepted are Consistency Algorithms, i.e. preprocessing algorithms that prune the search space, propagating the implications of constraints to help reduce the search space. This technique greatly benefits systematic search. Well received consistency algorithms include Forward Checking (Haralick and Elliott, 1980), Arc Consistency (Mackworth, 1975) and k-consistency (Freuder, 1978, 1982). Variable Ordering and Value
2.3 Multi-Agent Systems

Multi-Agent systems (MAS) are a popular paradigm for modeling distributed systems. Simply put, a MAS is a loosely networked society of agents. The term agent, however, is one that is much more difficult to define. Several definitions have emerged for agents (see (Franklin and Graesser, 1996; Wooldridge and Jennings, 1995b) for fairly comprehensive lists) and the one we choose to adopt is from Pattie Maes:

Definition 3. Autonomous agents are computational systems that inhabit some complex, dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed (Maes, 1995).

Wooldridge and Jennings (1995a) classify the notion of agency as being weak or strong depending on the properties exhibited by the agent. The so-called Weak Notion of Agency is now well accepted as the following minimum set of properties that a system must exhibit to be an agent:

- Autonomy: the ability to operate without direct intervention and to have some kind of control over its actions and internal state.
- Social Ability: the ability to interact with other agents and humans
- Reactivity: the ability to perceive its environment and to react to changes in a timely manner.
- Pro-activeness: the ability to take initiative and exhibit goal directed behaviour.

Other properties that have been attributed to agents in being a stronger notion of agency include mobility, veracity, benevolence, rationality, unpredictability, emotion, trustworthiness, cooperativity, competitiveness and accountability (Wooldridge and Jennings, 1995a). Several agent types have also emerged (see Fig. 2.2) with each agent type exhibiting a subset of the above properties. Of these, Nwana’s Smart Agent (Nwana, 1996) (see Fig. 2.3), or Intelligent Agent as it is more popularly known, is of special interest to this study and is characterized as an agent displaying cooperation, learning and problem solving over and above the basic property set.
2. Literature Review

Figure 2.2: A Classification of Agent Types (Nwana, 1996)

Figure 2.3: Nwana’s *Smart Agent* (Nwana, 1996)
2.3. Multi-Agent Systems

In order to carry out their tasks, and increase the problem solving scope of the system, agents need to be able to interoperate and coordinate with each other in peer-to-peer interactions. To define a MAS formally thus, we adopt Wooldridge’s definition:

**Definition 4.** A **Multi-Agent system is a system that consists of a number of agents, which interact with each other, typically by exchanging messages through some computer network infrastructure** (Wooldridge, 2002).

Additionally, a MAS has been defined as having the following characteristics (Sycara, 1998):

- each agent has a limited viewpoint i.e. incomplete problem information.
- there is no global system control
- data are decentralized
- computation is asynchronous

A good framework, and effective communication, coordination, and negotiation protocols, are thus as important in an agent as is its individual problem solving ability. Coordination and negotiation strategies are especially well researched areas. We discuss negotiation briefly as a good understanding of effective negotiation strategies is important for solving the scheduling problem.
2. Literature Review

Figure 2.5: Using Agents to Model the Entire Healthcare Domain (Schweiger and Krcmar, 2004)

Figure 2.6: Contract Net Protocol at Work (Moreno et al., 2001a)
2.3. Multi-Agent Systems

2.3.1 Negotiation in Multi-Agent Systems

An advantage of using the multi-agent systems paradigm to model the problem environment is that agents can use the same mechanisms as the units they model, i.e. negotiation instead of just simple search in solving the problem at hand (Shen, 2002). Effective negotiation and coordination are key issues in the problem solving capability of a multi-agent system. In fact, according to Sauer and Appelrath (2003), “The central problem of multi-agent systems is how to achieve coordinated action among agents in a way yielding problem solving capabilities that exceed those of any individual agent.” Consequently, most current research that aims to tackle complex distributed problems like hospital scheduling focuses on negotiation and coordination strategies. Several negotiation strategies have evolved as a result of this ongoing research. Among these, the most popular are the Contract Net Protocol (Davis and Smith, 1983; Smith, 1980, 1988), blackboard like systems, and market mechanisms, especially auction mechanisms.

2.3.2 Distributed Problem Solving Using Multi-Agent Systems

While several solution protocols have been proposed for solving distributed problems, these can be broadly divided into the following three categories (Hirayama, 2006):

- Centralized Problem Solving: In this approach, each agent sends its knowledge and problem to a central agent/server that receives all sub-problems, constructs a global problem, solves it, and then returns the solution to each agent. While simple to implement, this involves translating the problem to a common form (Yokoo, 2001; Bessiere et al., 2003), adding communication and processing overhead (Hirayama, 2006), and cannot be used in domains where privacy or security of information is an issue (Yokoo et al., 2002).

- Decentralized Problem Solving: In this approach, each agent solves its own problem autonomously but it interacts with other agents through a central agent/server that acts as a coordinator. An example of this approach may be seen in (Kutanoglu and Wu, 1999). While this may seem to address the privacy/security issues, the central server is still able to deduce at least partial agent knowledge. In addition, this still contributes a significant communication and processing overhead to the problem solving process.

- Distributed Problem Solving: In this approach, the agents follow true distributed problem solving in that no central coordinator/server is used and they try to solve the problem by communicating and negotiating with each
2. Literature Review

other. Private information is not shared with agents unless it is required to do so. This approach is especially suitable because it can effectively model the decision making process of distributed and complex domains. Good examples of this approach can be found in several distributed constraint reasoning algorithms though efficiency is still somewhat lacking (Hirayama, 2006).

Given its mediator-free, truly distributed structure, the last approach forms the focus of this study.

2.4 Distributed Constraint Reasoning

Essentially, a Distributed CSP/COP is a CSP/COP in which the variables and constraints are distributed among agents. An agent is said to own its variables. As with most CSP/COP algorithms, for the purpose of simplicity but without losing generality (Yokoo, 2001), constraints are assumed to be binary.

There are two types of constraints - intra-agent constraints between variables belonging to the same agent, and inter-agent constraints between variables belonging to different agents. Each agent is only familiar with the partial problem associated with its constraints. The global solution thus consists of the sum of all partial problems.

Communication among agents plays an important part in solving the global problem. Algorithm efficiency, therefore, also needs to take into account the cost of inter-agent communication. To better model this factor, the following communication model proposed by Yokoo et al. (1992a) is universally adopted.

• Agents communicate by sending messages.

• An agent can send messages to other agents iff the agent knows the addresses/identifiers of the agents.

• The delay in delivering the messages is finite, though random.

• For the transmission between any pair of agents, messages are received in the order in which they are sent.

For the purpose of distributed problem solving, it is also implicitly assumed that agents do know the addresses/identifiers of the other agents they need to contact.
2.4. Distributed Constraint Reasoning

2.4.1 Distributed Constraint Satisfaction

Formally, a Distributed Constraint Satisfaction Problem (DisCSP) consists of:

1. A finite ordered set of Agents $A = \{A_1, A_2, A_3, ..., A_n | n \in Z^+\}$,

2. For each Agent, there exists:
   (a) A finite ordered set of variables $V = \{V_1, V_2, V_3, ..., V_n | n \in Z^+\}$,
   (b) A domain set $D = \{D_1, D_2, D_3, ..., D_n\}$, containing a finite and discrete domain $D_i$ for each variable $V_i$,
   (c) An intra-agent constraint set $C = \{C_1, C_2, ..., C_m\}, m \in Z^+$, where each $C_j, \forall j \in [1, m]$ is a predicate defined on the cartesian product of the domains of intra-agent variables involved in $C_j$ and is satisfied iff the value assignment satisfies $C_j$,
   (d) An inter-agent constraint set $IC = \{IC_1, IC_2, ..., IC_m\}, m \in Z^+$, where each $IC_j, \forall j \in [1, m]$ is a predicate defined on the cartesian product of the domains of inter-agent variables involved in $IC_j$ and is satisfied iff the value assignment satisfies $IC_j$, and
   (e) An ordered solution set $S = \{v_1, v_2, v_3, ..., v_n | v_i \in D_i, \forall i \in [1, n]\}$ where each $v_i$ is an instantiation of $V_i$ such that all constraints in the constraint sets $C$ and $IC$ are satisfied.

3. The solution set of the DisCSP $S^*$ is defined as the set of the solution sets of each agent.

2.4.2 An Example of DisCSP

To further understand DisCSPs, we look at the distributed model of the Map Colouring CSP example in Section 2.2. Figure 2.7 shows a distributed CSP map colouring problem where the map of Australia is distributed among three agents $A$, $B$, and $C$, that have been given the task of colouring their respective states with the colours $Red$, $Blue$ or $Green$ such that no adjacent states have the same colour.

We therefore define the DisCSP as:

1. A set of Agents $X = \{A, B, C\}$,

2. For Agent $A$, there exists:
   (a) A set of variables $V_A = \{WA\}$,
2. Literature Review

(a) A Distributed Map of Australia

(b) Constraint Graph

(c) Sample Solution

Figure 2.7: A Distributed Map Colouring Problem
2.4. Distributed Constraint Reasoning

(b) A domain set \( D_A = \{\text{Red, Green, Blue}\} \),

(c) A blank intra-agent variable constraint set \( C_A = \text{null} \),

(d) An inter-agent variable constraint set \( IC_A = \{C_1, C_2\} \), where,
   - \( C_1 = WA \neq NT \)
   - \( C_2 = WA \neq SA \)

3. For Agent B, there exists:

(a) A set of variables \( V_B = \{NT, SA\} \),

(b) A domain set \( D_B = \{\text{Red, Green, Blue}\} \),

(c) A intra-agent variable constraint set \( C_B = \{C_3\} \), where,
   - \( C_1 = NT \neq SA \)

(d) An inter-agent variable constraint set \( IC_B = \{C_1, C_2, C_4, C_5, C_6, C_7\} \), where,
   - \( C_1 = WA \neq NT \)
   - \( C_2 = WA \neq SA \)
   - \( C_4 = NT \neq Qld \)
   - \( C_5 = SA \neq Qld \)
   - \( C_6 = SA \neq NSW \)
   - \( C_7 = SA \neq Vic \)

4. For Agent C, there exists:

(a) A set of variables \( V_C = \{Qld, NSW, Vic, Tas\} \),

(b) A domain set \( D_A = \{\text{Red, Green, Blue}\} \),

(c) A intra-agent variable constraint set \( C_C = \{C_8, C_9\} \), where,
   - \( C_8 = Qld \neq NSW \)
   - \( C_9 = NSW \neq Vic \)

(d) An inter-agent variable constraint set \( IC_C = \{C_4, C_5, C_6, C_7\} \), where,
   - \( C_4 = NT \neq Qld \)
   - \( C_5 = SA \neq Qld \)
   - \( C_6 = SA \neq NSW \)
   - \( C_7 = SA \neq Vic \)

5. Upon solving the DisCSP, we get the following solution sets

- For the Agent A, \( S_A = \{WA = \text{Red}\} \)
- For the Agent B, \( S_B = \{NT = \text{Green}, SA = \text{Blue}\} \)
2. Literature Review

- For the Agent C, \( S_C = \{Qld = \text{Red}, NSW = \text{Green}, Vic = \text{Red}, Tas = \text{Red}\} \)
- For the DisCSP, \( S^* = \{WA = \text{Red}, NT = \text{Green}, SA = \text{Blue}, Qld = \text{Red}, NSW = \text{Green}, Vic = \text{Red}, Tas = \text{Red}\} \)

2.4.3 Solving DisCSPs

Like any other technology, distributed CSPs bring along with themselves a host of other important issues to complicate the problem (Faltings and Yokoo, 2005).

- While they eliminate the need for a central overhead, the currently known algorithms for solving DisCSPs perform very poorly compared to their centralized peers. Much needed are different algorithms better suited to a distributed environment.
- While most distributed algorithms can handle some amount of agent failure, research has not adequately identified the kind of failures allowed for each algorithm.
- More research is needed on algorithms that minimize the number of constraint evaluations when needed, measures of privacy loss and on algorithms that balance the trade off between privacy loss and efficiency.

However, the most important advantage obtained from using DisCSPs is that the underlying agent based architecture offers an excellent paradigm for modeling a distributed domain and the flow of organizational decision making while allowing for the preservation of privacy if required.

2.4.4 Search in DisCSPs

As with CSPs, search in DisCSPs can be broadly classified into three types - Exact Search, Local Search and Hybrid Search. Given the complex nature of DisCSP problems, however, several other forms of classifications are possible - for example methods may be classified into those that handle single variable or multiple variables, or into those that support centralized, decentralized, or truly distributed problem solving. In addition, given that not too many efficient methods have been proposed so far, we present the methods and their variants individually and classify them into the above possibilities in pursuit of an efficient DisCSP search method that is efficient, complete, can handle multiple variables and supports distributed problem solving.
Asynchronous Backtracking

The simplest way to extend backtracking to DisCSPs would obviously be to establish a total order among agents and have each agent solve its local solution and pass it on to the next agent. This form of *Synchronous Backtracking* would, however, defeat the very purpose of using the multi-agent paradigm.

The first asynchronous complete algorithm proposed for DisCSPs, Asynchronous Backtracking (ABT) (Yokoo et al., 1992a) allows agents to run concurrently and asynchronously to solve a DisCSP (see Algorithm 1). It assumes that each agent has only one variable, all constraints between agents are binary and directed (i.e. enforced by an *outgoing* agent/variable onto an *incoming* one) and that a total order exists among agents (necessary to avoid infinite processing loops). Each agent instantiates its variables and sends the values out on outgoing links, asking agents on the other of the links to check these values. On receiving this *ok?* message, each agent adds the new values received to its *view* and checks for any inconsistencies. If the agent’s *view* is not consistent with the new values received, it tries to reassign its variables to make its view consistent, and then sends an *ok?* message on its outgoing links. If, however, it finds no instantiation that can make its view consistent, it records its current *view* as a *nogood*, i.e. a bad solution, and adds a new link (constraint) to any unconnected variable in the nogood. It then broadcasts the nogood to the lowest priority agent in the nogood, removing the value of this agent from its view and rechecks its view. In doing so, it eventually reaches a consistent state or arrives at an empty nogood set, in which case it infers that a solution is not possible.

Several improvements have been proposed to the basic ABT algorithm. Most differ in the way no-goods are stored and used and in how the variables/agents are ordered. All of these methods, however, suffer from the common backtracking drawback in needing an exhaustive search when a bad decision is made by a higher priority agent. Further, all of these methods are limited to one variable per agent and are thus unable to effectively model any complex real world DisCSP, the problem domain we are interested in solving.
2. Literature Review

**Algorithm 1**: A Sketch of the Asynchronous Backtracking Algorithm (Yokoo et al., 1992a)

<table>
<thead>
<tr>
<th>Instantiate variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send ok? message on outgoing links</td>
</tr>
<tr>
<td>Receive messages on incoming links</td>
</tr>
<tr>
<td><strong>when ok? message received do</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>end do</strong></td>
</tr>
<tr>
<td><strong>when nogood received do</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>end do</strong></td>
</tr>
<tr>
<td><strong>Procedure: check_agent_view</strong></td>
</tr>
<tr>
<td><strong>while agent view and new values are not consistent do</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>end</strong></td>
</tr>
</tbody>
</table>

Stop when complete solution found or no solution possible.
Asynchronous Weak Commitment Search

Asynchronous Weak Commitment Search (AWC) was introduced by Yokoo (1995) as an improvement over ABT based on the principle of Weak Commitment Search (Yokoo, 1994) (see Algorithm 2). It starts like ABT with each agent instantiating its variables and sending out \textit{ok?} messages but, unlike ABT, each agent is assigned a priority (initially set to 0), which is transmitted as part of the message. The instantiations are based on the min-conflict heuristic (in that the agent has to align itself with higher priority agents but prefers a value that minimizes the number of constraint violations with lower priority agents), the links are not directed and messages are sent out to all neighbours. Also, when the agent cannot find an instantiation that is consistent with its \textit{view}, it sends a nogood message to other agents, records the sent nogood, abandons the whole partial solution, increases its priority (to the highest priority among neighbours) and restarts its search. Similarly, when an agent receives a nogood message, it only acts on it if it is a new nogood.

The ability of an agent to change its priority, if it cannot find a solution that satisfies higher priority agents, helps AWC overcome any bad decisions that may have been taken by a previously higher priority agent. In addition, while the Distributed Breakout Algorithm (Yokoo and Hirayama, 1996) has been shown to outperform AWC, a modification of AWC that uses resolvent-based nogood learning (Hirayama and Yokoo, 2000) has been shown to outperform DBA in case of large communication delays. In both these forms, however, AWC is still unable to deal with more than one variable per agent.

Distributed Stochastic Algorithm

The Distributed Stochastic Algorithm (DSA) represents a group of uniform stochastic algorithms (Fabiunke, 1999; Fitzpatrick and Meertens, 2001; Macready et al., 1996; Zhang et al., 2003). The general idea behind DSA is simple. Each agent adopts a random instantiation and then exchanges its values with its neighbours. Agents then assume their neighbours will maintain their values and randomly decide on whether to keep or change their value. Agents only change values if this change can help reduce the number of unsatisfied constraints. The probability factor $p$ in DSA represents the degree of parallel executions as it controls how frequently neighbouring agents change their values.
2. Literature Review

Algorithm 2: A Sketch of the Asynchronous Weak Commitment Search Algorithm (Yokoo, 1995)

```
Instantiate variables
set priority ← 0
Send ok? message (including priority) to all neighbours
Receive messages on incoming links
when ok? message received do
    add values to agent view
    check_agent_view
end do
when new nogood received do
    record nogood as new constraint
    record nogood in nogood_list
    create new local link & add value to agent view if required
    add nogood to agent view
    check_agent_view
end do
Procedure: check_agent_view
while agent view and new values are not consistent do
    Select new value for variable consistent with agent view and minimizing
    constraint violations with lower priority agents
    if new value found then
        send ok? message to all neighbours
    else
        record nogood
        if new nogood then
            record nogood in sent_nogoods list
            if nogood is an empty set then
                broadcast “no solution possible” to everyone
            else
                send nogood message to all agents in nogood
                change priority to become highest priority
            end
        end
    end
end
Stop when complete solution found or no solution possible.
```
Algorithm 3: A Sketch of the Distributed Stochastic Algorithm (Zhang et al., 2003)

Randomly choose a value

while no termination condition is met do
    if (a new value is assigned) then
        send the new value to neighbours
    end
    collect neighbour’s values, if any
    compute the best possible conflict resolution $\triangle$
    if ($\triangle > 0$) or ($\triangle = 0$ but there is a conflict) then
        change to a value giving $\triangle$ with probability $p$
    end
end

Distributed Breakout Algorithm

The Distributed Breakout Algorithm (DBA) was originally proposed by Yokoo and Hirayama (1996). Based on local search and the breakout heuristic (Morris, 1993), the algorithm (see Algorithm 4) works on the following two main principles:

- Agents exchange values of possible improvements and only the agent that can make the maximum improvement is allowed to make the changes. Obviously, if two agents are not neighbours, they can make changes concurrently.

- Instead of attempting to detect local minima, the agents detect quasi-local-minima, a weaker state that can be detected by local communication. An agent is said to be in quasi-local-minima if it is violating some constraint and the possible improvement of the agent, and of all its neighbours is 0.

In its original proposed form, DBA is incomplete and can deal with only one variable per agent. It, however, offers two advantages over the ABT and AWC. As it is a local search method, it is much faster than AWC, especially for critical problems. Also, unlike ABT and AWC, DBA has termination detection built into the algorithm and does not need a separate procedure.

Zhang and Wittenburg (2002), however, show that DBA is complete for acyclic graphs. They also propose stochastic variations for DBA and show that a weak probabilistic variant (wp-DBA) can help overcome the incompleteness of DBA on graphs with cycles.

Hirayama and Yokoo (2005) present further work on DBA wherein they rename the original DBA as Single-DB and introduce a multi-variable form of DBA (Multi-DB) and two stochastic variations of the latter (Multi-DB$^+$ and Multi-DB$^{++}$), though
Algorithm 4: A Sketch of the Distributed Breakout Algorithm (Zhang and Wittenburg, 2002)

```
set the local weights of constraints to 1
value ← a random value from domain

while no termination condition met do
    exchange value with neighbours
    WR ← BestPossibleWeightReduction()
    send WR to neighbours and collect their WRs
    if WR > 0 then
        if it has the biggest improvement among neighbours then
            value ← the value that gives WR
        end
    else
        if no neighbour can improve then
            increase violated constraints’ weights by one
        end
    end
end
```

these are designed for solving the Distributed Satisfiability (DisSAT) problem.

Multi-Variable Asynchronous Weak Commitment Search

Yokoo and Hirayama (1998) propose a multi-variable form of the AWC (mv-AWC) drawing from Armstrong and Durfee’s work (Armstrong and Durfee, 1997) on dynamic agent ordering. While Armstrong and Durfee propose an algorithm that uses a central coordinating agent and a central nogoods processor, mv-AWC modifies the AWC algorithm to introduce the following changes (see Algorithm 5):

- each agent instantiates its variables sequentially (decreasing priority).
- if an agent cannot find a value for a variable, it increase the variable’s priority to highest among all local variables and re-instantiates its local variables.
- when all local assignments are complete, it sends an ok? message to related agents.
- to maintain completeness, all nogoods are transmitted as soon as they are known.
Algorithm 5: A Sketch of the Multi-Variable Asynchronous Weak Commitment Search Algorithm (Yokoo and Hirayama, 1998)

Instantiate variables

set \textit{priority} \leftarrow 0

Send \textit{ok}? message (including priority) to all neighbours

Receive messages on incoming links

\textbf{when} \textit{ok}? \textit{message received} \textbf{do}

add values to agent view

\textbf{while} agent view and new values are not consistent \textbf{do}

\textbf{|} \textit{check\_agent\_view}

\textbf{end}

\textbf{end do}

\textbf{Procedure: check\_agent\_view}

\textbf{if} agent view and current assignments are consistent \textbf{then}

\textbf{|} communicate changes to related agents

\textbf{else}

select highest priority local variable, $x_k$ violating constraint with higher priority variable

set value of $x_k$ consistent with agent view and minimizing constraint violations with lower priority variables

\textbf{if} new value found \textbf{then}

\textbf{|} \textit{check\_agent\_view}

\textbf{else}

record and communicate a nogood

\textbf{when new nogood do}

set $x_k$’s priority to highest among related variables

select value for $x_k$ which minimizes constraint violations with lower priority variables

\textbf{|} \textit{check\_agent\_view}

\textbf{end do}

\textbf{end}

\textbf{end}

Stop when complete solution found or no solution possible.
Dynamic Agent Ordering with Nogood Repairing

Zhou, Thornton and Sattar present two algorithms for handling multi-variable DisCSPs. They propose a novel dynamic parameter, degreeUnsat, which is used for dynamic agent ordering. In the first algorithm (see Algorithm 6), Dynamic Agent Ordering (DAO) (Zhou et al., 2003), each agent concurrently instantiates its variables and sends its local solution to all neighbouring agents. All agents then work to solve their local solution. If the agents are unable to do so, and an inter-agent variable needs to be changed, they compare their values of degreeUnsat and the agent with a higher value of degreeUnsat is allowed to reassign its value. If no suitable values for local variables are found, the agent solves as many constraints as possible and the state is recorded as a nogood and the nogood is transmitted to related agents. After assigning its own variables, the agent sends its instantiation and value of degreeUnsat to neighbouring agents that again try to solve their local solution. The search stops when each agent detects that the value of degreeUnsat for itself and all its neighbours is zero.

Algorithm 6: The Dynamic Agent Ordering Algorithm (Zhou et al., 2003)

```plaintext
while received(Sender_id, variable_values, degreeUnsat) do
    calculate local_degreeUnsat
    if local_degreeUnsat and all other agents’ degreeUnsats = 0 then
        the search is terminated;
    else
        add(Sender_id, variable_values, degreeUnsat) to agent_view
        if local_degreeUnsat > degreeUnsat then
            assign_local_variables
            calculate local_degreeUnsat
            send(Sender_id, variable_values, degreeUnsat) to neighbouring agents
        end
    end
end

Procedure: assign_local_variables
Backtracking to construct optimal_local_partial_solution
if optimal_local_partial_solution is not local_solution then
    add the culprit_variables with their values, agent_ids to the nogood_set
    if the nogood is new then
        record the new nogood
        send nogood_set message to the related agents
    end
end
```

DAO is shown to perform faster than AWC and a static ordering algorithm, and stores lesser nogoods, but has higher communication costs.
2.4. Distributed Constraint Reasoning

In the second algorithm (see Algorithm 7), Dynamic Agent Ordering with Nogood Repairing (DAONR) (Zhou et al., 2004), each agent concurrently instantiates its variables and sends its local solution to all neighbouring agents. All agents then work to solve their local solution, considering only those inter-agent constraints that are with an agent having a lower value of degreeUnsat. If no suitable values for local variables are found, the state is recorded as a nogood and the agent autonomously chooses the best among three options available to repair the nogood. After assigning its own variables, the agent sends its instantiation and value of degreeUnsat to neighbouring agents that again try to solve their local solution. The search stops when each agent detects that that the value of degreeUnsat for itself and all its neighbours is zero.

Algorithm 7: Modified Assign_Local_Variables Procedure for Dynamic Agent Ordering and Nogood Repairing (Zhou et al., 2004)

Procedure: Assign_Local_Variables
if local instantiation is consistent with agent view from neighbouring agents, and degreeUnsat < local degreeUnsat then
  send(Sender_id, variable_values, degreeUnsat) to neighbouring agents
else
  select an inconsistent variable v with the highest priority and assign a value from its domain
  if no value for the variable then
    if nogood is new then
      Nogood_Repairing(v)
    end
  else
    assign a value with minimal violations to the variables with lower priorities
  end
  Assign_Local_Variables
end

Procedure: Nogood_Repairing(v)
if nogood is intra-nogood or mix-nogood then
  v’s priority = the highest priority of local variables in nogood +1
  if no values for v then
    priority of the local agent = minimum of degreeUnsat of related neighbouring agents -\( \triangle \)
  else
    assign a value with minimal violations to the variables with lower priorities
  end
else
  priority of the local agent = minimum of degreeUnsat of related neighbouring agents -\( \triangle \)
end
DAONR is shown to outperform mv-AWC but DAO presents a better runtime performance than DAONR despite generating a much larger set of instantiations. The authors credit this result to the fact that the analysis is performed on a single machine, thus making the problem of message sending less significant. It is expected than in a true MAS implementation, DAONR would perform better than DAO because it produces significantly lesser instantiations and thus requires fewer inter agent messages.

2.4.5 Distributed Constraint Optimization Problems

As with COPs, the DisCSP paradigm fails when the problem is unsolvable, i.e. over-constrained. For example, given an overconstrained DisCSP, ABT and AWC would broadcast “no solution possible” and terminate while DBA would never terminate. Further, several complex problems require constraints of varying importance and thus require the ability to qualify constraints with costs or utility measures.

The solution lies in generalizing DisCSPs to Distributed Constraint Optimization Problems (DCOP) and modeling constraints as cost/utility functions. The objective of problem solving is now to find an instantiation that maximizes/minimizes the global objective function. As in the centralized case, we assume cost minimization as the problem objective.

DCOPs typically consist of multiple agents that have mutually-related sub-problems and a common goal that can be described as a global objective function.

2.4.6 The DCOP Formalism

In solving a DCOP, the goal for each agent is to assign values to its variables such that a given global objective function is minimized. The cost functions in DCOP are analogous to constraints in DisCSP, and DCOP is thus regarded as a generalization of the DisCSP formalism. For simplicity, we use the term constraints and cost functions interchangeably. Formally, we can define a DCOP as consisting of:

1. A finite ordered set of Agents $A = \{A_1, A_2, ..., A_k | k \in \mathbb{N}^*\}$, where, for each Agent $A$, there exists:

   (a) A finite ordered set of variables $V = \{V_1, V_2, ..., V_n | n \in \mathbb{N}^*\}$,

   (b) A domain set $D = \{D_1, D_2, ..., D_n\}$, containing a finite and discrete domain $D_i$ for each $V_i$,
2.4. Distributed Constraint Reasoning

(c) A constraint set \( C = \{C_1, C_2, ..., C_m \mid m \in \mathbb{N^*}\} \), where each \( C_j, \forall j \in [1, m] \), is defined as a cost function \( f \) on a pair of variables \((i, i')\). i.e. \( C_j = f_{ii'} : D_i \times D_{i'} \rightarrow \mathbb{N}, \forall V_i, V_{i'} \in V \), and

(d) An ordered solution set \( S = \{v_1, v_2, ..., v_n \mid v_i \in D_i, \forall i \in [1, n]\} \) where each \( v_i \) is an instantiation of the variable \( V_i \) and the aggregate cost of the assignment \( F(S) = \sum_{(x_i, x_{i'} \in V)} f_{ii'}(d_i, d_{i'}), x_i \leftarrow d_i, x_{i'} \leftarrow d_{i'} \in S \).

2. The solution set of the DCOP \( S^* \) is defined as the set of the solution sets of each agent.

In keeping with the norm (Modi et al., 2003), it is assumed that all constraints are binary, and optimization functions are associative, commutative, and monotonic. Modi et al. (2003) also point out, however, that generalization to \( n \)-ary constraints is achieved quite simply for most DisCSP algorithms. In dealing with complex DCOPs, however, we do not make the general assumption of one variable per agent.

The computational complexity of DCOP has been shown to be NP-Hard (Modi, 2003).

2.4.7 Current State of the Art

Since Hirayama and Yokoo (1997) introduced Synchronous Branch and Bound (SBB) and Iterative Distributed Breakout (IDB) as the first DCOP algorithms, several notable DCOP approaches employing techniques from search (e.g. ADOPT (Modi et al., 2003) and its variants), dynamic programming (e.g. DPOP (Petcu and Faltings, 2005c) and its variants) and cooperative mediation (e.g. APO (Mailler and Lesser, 2004b)) have emerged and are being successfully used to model and solve problems in sensor networks, meeting scheduling, etc. Due to space constraints, we focus on the following key algorithms, each representing a significantly different approach. In addition, we do not critique each variant as they all still have the same shortcomings when applied to dynamic complex problems. More recent and promising work in this field is also discussed and later related with our ongoing work.

Asynchronous Distributed OPTimization (ADOPT)

Asynchronous Distributed OPTimization, or ADOPT (Modi et al., 2003), is a complete and asynchronous DCOP algorithm. In ADOPT, agents are first prioritized into a Depth First Search (DFS) tree, whereby each agent maintains lower and upper bounds for the subtree rooted at their node. The agents then use opportunistic best-first search to assign their variables such that the lower bound is minimized. Cost
messages propagate up the tree and *threshold* and *value* messages are sent down the tree, iteratively tightening the lower and upper bounds until the lower bound of the minimum cost solution is equal to its upper bound. If an agent detects this condition and its parent has terminated, then an optimal solution is found and it may also terminate. The other key idea in ADOPT is to store lower bounds as a threshold and discard partial solutions before they are proven to be definitely suboptimal, thus maintaining linear space complexity at each agent. In the worst case, ADOPT may require an exponential number of messages to arrive at a solution. Definitely the most popular and most extended DCOP algorithm (Scerri et al., 2003; Ali et al., 2005; Bowring et al., 2006; Davin and Modi, 2006; Pecora et al., 2006; Silaghi and Yokoo, 2006; Matsui et al., 2008; Yeoh et al., 2008; Silaghi and Yokoo, 2009; Yeoh et al., 2009), ADOPT is well regarded as the gold standard among search based algorithms for DCOP.

**Distributed Pseudotree Optimization Procedure (DPOP)**

Distributed Pseudotree Optimization Procedure, or DPOP (Petcu and Faltings, 2005c), is a complete dynamic programming algorithm that involves a three phase process (see Algorithm 9). Similar to ADOPT, the first phase involves the formation of the DFS tree. Phase two involves calculating and propagating the utility (cost) from the bottom up, i.e. from the leaves upwards to the root. Phase three involves a downward value propagation, initiated by the root node. Each agent then calculates its optimal value based on the utility message received from its subtree and the value message received from its parent. DPOP thus generates only a linear number of messages, but the message size grows with every traversal up the tree and the algorithm thus requires a large amount of memory, up to space exponential in the induced width of the problem. Several variants have been proposed to improve the algorithm’s performance when applied to dynamic environments (Petcu and Faltings, 2005a, 2007b), self interested agents (Petcu et al., 2008) and limited memory domains (Petcu and Faltings, 2007a). Other variants that address privacy (Silaghi et al., 2006; Faltings et al., 2008) and anytime performance (Petcu and Faltings, 2005b), and allow local search (Petcu and Faltings, 2007c), partial centralization (Petcu et al., 2007), and cross-edges (Atlas and Decker, 2007), have also been proposed.

**Optimal Asynchronous Partial Overlay (OptAPO)**

Optimal Asynchronous Partial Overlay (OptAPO) (Mailler and Lesser, 2004b) is an alternative approach to DCOP that utilizes partial centralization to solve diffi-
Algorithm 8: A Sketch of the ADOPT Algorithm (Modi et al., 2003)

initialize
threshold ← 0; CurrentContext ← {}
forall d ∈ Di, xi ∈ Children do
    lb(d, xi) ← 0; t(d, xi) ← 0; lb(d, xi) ← ∞; context(d, xi) ← {}
end do
when received (THRESHOLD, t, context) do
    if context compatible with CurrentContext then
        threshold ← t
        maintainThresholdInvariant; backtrack
    end
end do
when received (VALUE, (xj, dj)) do
    if TERMINATE not received from parent then
        add (xj, dj) to CurrentContext
        forall d ∈ Di, xi ∈ Children do
            if context(d, xi) incompatible with CurrentContext then
                lb(d, xi) ← 0; t(d, xi) ← 0; lb(d, xi) ← ∞; context(d, xi) ← {}
            end
        end do
        maintainThresholdInvariant; backtrack
    end
end do
when received (COST, xk, context, lb, ub) do
    d ← value of xi in context
    remove (xi, d) from context
    if TERMINATE not received from parent then
        forall (xj, dj) ∈ context and xj is not my neighbour do
            add (xj, dj) to CurrentContext
        end do
        forall d′ ∈ Di, xi ∈ Children do
            if context(d′, xi) incompatible with CurrentContext then
                lb(d′, xi) ← 0; t(d′, xi) ← 0; lb(d′, xi) ← ∞; context(d′, xi) ← {}
            end
        end do
    end
end do
Algorithm 9: A Sketch of the DPOP Algorithm (Petcu and Faltings, 2005c)

**Phase 1: Pseudotree creation**

elect leader node

if leader node then

  initiate pseudotree creation

end

**Phase 2: UTIL message propagation**

if agent is a leaf node then

  Compute utility

  Send UTIL message to parent

end

activate UTIL_Message_handler

**Phase 3: VALUE message propagation**

activate VALUE_Message_handler

END ALGORITHM

**PROCEDURE: UTIL_Message_handler**

store received utility

if UTIL messages received from all children then

  if agent is a root agent then

    Choose Optimal Value

    Send VALUE message to all children

  else

    Compute utility

    Send UTIL message to parent

  end

end

**PROCEDURE: VALUE_Message_handler**

add value to agent view

Choose Optimal Value

Send VALUE message to all children
2.4. Distributed Constraint Reasoning

cult portions of a DCOP problem. OptAPO works by constructing a good list and maintaining an agent view (see Algorithm 10). The agent view stores information about linked agents and the good list holds names of directly or indirectly related agents. During initialization, agents check their agent view to identify conflicts with neighbours. If a conflict is found, it expresses a desire to act as a mediator. The agent with the highest priority assumes the role of the mediator. During mediation, an internal Branch and Bound is used to solve sub-problems and when solutions of overlapping sub-problems have conflicts, the solving agents increase the centralization to resolve them. The algorithm has been shown to be complete and optimal and outperforms ADOPT on graph colouring problems (Mailler and Lesser, 2004b). More recently though, it has been proven (Grinshpoun and Meisels, 2008) that the original APO (Mailler and Lesser, 2004a, 2006) and OptAPO are incomplete algorithms and complete variants have been proposed.

No-Commitment Branch and Bound (NCBB)

No-Commitment Branch and Bound (NCBB) (Chechetka and Sycara, 2006b) is a distributed branch and bound search strategy for distributed optimization. It allows different agents to search non intersecting parts of the search space concurrently. It also allows incremental computation and communication of lower bounds on solution cost. Similar to ADOPT and DPOP, agents are prioritized in a DFS tree. During the initialization stage, agents compute global upper and lower bounds on the solution cost. Agents choose their values greedily so as to minimize their individual costs. These costs propagate up the tree and the agents set their bound variables to play the role of upper bounds for their individual subtrees. After initialization, agents execute the main search loop (Algorithm 11). Agents start actively searching only after they receive SEARCH messages from their parents. The search process exploits the inherent parallelism of the problem to compute tighter upper bounds and eagerly propagates changes in cost effected by lower bound changes resulting in more precise lower bounds and thus better pruning of the search space. A caching scheme has also been proposed (Chechetka and Sycara, 2006a) and this has been shown to further improve the performance of the algorithm.

Local Search Methods for DCOP

While both DBA and DSA have been extended to apply to DCOP (Zhang et al., 2003), DBA itself cannot be applied to a truly distributed system for solving DCOP, as it requires global knowledge of solution quality. (Maheswaran et al., 2004a). Maheswaran et al. (2004a) propose a simpler variant of the DBA algorithm, the MGM
Algorithm 10: A Sketch of the OptAPO Algorithm (Mailler and Lesser, 2004b)

**Procedure initialize**
- initialize $d_i, F^*\_i, p_i, m_i$
- $\text{mediate} \leftarrow \text{none}$
- add $x_i$ to the $\text{good\_list}$
- send $\text{init}$ message to neighbours
  $\text{initList} \leftarrow$ neighbours

**end initialize**

**when init message received do**
- Add message context to $\text{agent\_view}$
  - if $x_j$ is a neighbour of some $x_k \in \text{good\_list}$ then
    - add $x_j$ to the $\text{good\_list}$
    - add all $x_l \in \text{agent\_view}$ and $x_l \notin \text{good\_list}$
      that can now be connected to the $\text{good\_list}$
    - $p_i \leftarrow \text{sizeof}(\text{good\_list})$
  - end
  - if $x_j \notin \text{initList}$ then send $\text{init}$ message to $x_j$ else
    - remove $x_j$ from $\text{initList}$
    - $\text{check\_agent\_view}$
  - end

**end do**

**when value message received do**
- Add message context to $\text{agent\_view}$
- $\text{check\_agent\_view}$

**end do**

**Procedure check\_agent\_view**
- if $\text{initList} \neq 0$ or $\text{mediate} \neq \text{false}$ then return
- Compute new mediate intention $m'_i$
  - if $m'_i = \text{active}$ and (no higher priority mediator) then
    - if ($\exists d'_i : F_i = F^*_i$ and changes with lower priority neighbours) then
      - change value of $d_i$
      - send $\text{value}$ messages to all $x_j \in \text{agent\_view}$
    - else mediate($m'_i$)
  - else if $m'_i = \text{passive}$ then mediate($m'_i$)
  - else if $\text{mediate flag or conflict set changed}$ then
    - send $\text{value}$ messages to all $x_j \in \text{agent\_view}$
  - else if $m'_i = \text{none}$ then
    - update $\text{good\_list}$ if needed
- end
2.4. Distributed Constraint Reasoning

Algorithm 11: A Sketch of the NCBB Algorithm (Chechetka and Sycara, 2006b)

function mainLoop()
  if not(Agent is root) then
    updateContext()
  end
  while true do
    search()
    if (not(Agent is root) or updateContext()) then
      break
    end
  end
  set costs[resultValue] to 0
  initiate SubtreeSearch for all children
  send “STOP” to all children
end mainLoop()

FUNCTION: updateContext()
while true do
  receive messages from ancestors
  if “search” received then
    set bound to message.BOUND
    return false
  else if “y=d” received then
    set context[y]=d
    send new lower bound to y
  else if “STOP” received then
    return true
  end
end
(Maximum Gain Message) algorithm. MGM is a modification of the DBA algorithm in that it focuses only on gain message passing. Further, being synchronous algorithms, they fail to offer a strategy that can be applied in a truly distributed environment. However, we introduce a DSA-like asynchronous algorithm in Section 4.4 and evaluate its performance against the state-of-the-art algorithms discussed above and against our DCDCOP algorithm.

More recently Kiekintveld et al. (2010) have introduced an asynchronous local search algorithm for DCOP. Based on a new concept of t-distance optimality, the algorithm utilizes partial centralization where the group leader uses a DPOP-like centralized algorithm to explore an optimal solution for the group. A standard lock/commit pattern and partial synchronization is then utilized to negotiate with other group leaders.

Asymmetric Distributed Constraint Optimization Problems (ADCOP)

Recently, Grubshtein et al. (2009, 2010) have proposed the Asymmetric Distributed Constraint Optimization (ADCOP) formalism that offers an elegant model for DCOP problems where each agent participating in a constraint can assign a different cost to it based on its own valuation. They also propose asymmetric versions of SyncBB and AFB and four new synchronous local search algorithms: Proposal Based Search (PBS), Asymmetric Coordinated Local Search (ACLS), Minimal Constraint Sharing MGM (MCS-MGM) and Guaranteed Convergence Asymmetric MGM (GCA-MGM).

2.4.8 Solving Dynamic Complex Problems

Although DCOP offers an excellent mapping for representing many real world problems, the general design of DCOP algorithms is static. Since most DCOP algorithms utilize static DFS tree structures, changes to the constraints would often result in the need for the tree to be rebuilt. Further, since ADOPT discards no-goods to maintain linear space complexity, changes to the constraints would also result in bounds being discarded and the search restarted.

Both ADOPT and DPOP also offer variants to deal with dynamic environments. Modi (2003) offers a formalism for mapping and solving dynamic resource allocation problems but this is applied in the DisCSP domain. This is extended to map over-constrained problems into DCOP but can handle only static problems as the author concedes to the lack of an effective DCOP algorithm for dynamic problems. Petcu and Faltings (2005a, 2007d) propose S-DPOP and RS-DPOP, which utilize
self stabilizing DFS trees to guarantees optimal solution stability in distributed continuous-time combinatorial optimization problems. Matsui and Matsuo (2005) deal with dynamic DCOP problems by applying ADOPT to solve a set of trees that are similar to the DFS-tree of, and built bottom up from, the constraint network. Trees are discarded as current search breaks down and nodes quickly construct new trees to start the next search. ADOPT is modified in this case to obtain quasi-optimal solutions. Lass et al. (2008) deal with the complicating factor of dynamism by wrapping ADOPT in an Adapter (see Fig. 2.8) that receives and handles dynamic event requests. Once again, search is restarted when the current tree structure is no longer valid. Zivan et al. (2009) also present a model, DCOP_MST, for representing mobile sensor teams by enabling the representation of variant dynamic elements, such as location, set of neighbours, and set of domains. Though specific to the domain, this model recognizes the unsuitability of complete search methods and local search algorithms are used to explore the search space. All of the above variants, however, still suffer from working off a static tree structure that needs rebuilding from time to time.

In dealing with the issue of complex sub-problems, algorithms can theoretically utilize decomposition or compilation. Several ADOPT variants use techniques such as decomposition (Modi, 2003), compilation (Davin and Modi, 2006), interleaving...
2. Literature Review

(Burke, 2008), and relaxation (Burke, 2008), to deal with complex sub-problems. In practice, however, decomposition results in failure to exploit the inherent benefit of domain centralization, and also blows the distributed problem size out of proportion. Burke and Brown (2006) show that the compilation outperforms decomposition in case of large local sub problems but only small domain size, whereas decomposition is more appropriate when the number of inter-agent constraints and domain size is large but only for small problems. Further, applying decomposition to techniques like DPOP would result in a significant increase in the message sizes, while compilation would need a novel mechanism of calculating the agent utility for different combinations of local variable assignments.

We thus conclude that while the above DCOP algorithms are optimal in a static environment, there is need for a more flexible robust algorithm, which can model the complexity and adapt better to a dynamic environment.

2.5 Summary

In this Chapter, we introduce concepts that form the foundation of this study. The scheduling problem is presented and scheduling is discussed in the context of complex, dynamic, and distributed scheduling environments to identify shortcomings of current systems in addressing these problem characteristics. Constraint based scheduling is identified as the dominant choice for modeling and solving scheduling problems. The constraint reasoning paradigm and various forms of constraint guided search are also discussed. The Multi-Agent Systems paradigm is presented as a natural way to model the structure and the underlying decision making in complex distributed environments. Negotiation and distributed problem solving are also briefly discussed.

We then introduce Distributed Constraint Satisfaction and Distributed Constraint Optimization and review the state-of-the-art DisCSP and DCOP techniques. This reveals that while distributed constraint reasoning in general offers an excellent framework for modeling and solving distributed real world problems, current DCOP techniques fall short of offering an efficient solution for solving complex optimization problems in dynamic environments. While several sophisticated complete DCOP algorithms have been proposed, they generally fail to scale as well as their centralized counterparts and thus offer sub-optimal performance for complex dynamic problems. While efficient local search counterparts have been proposed, these rely on a global bound or synchronous operation and are also thus not well suited to the class of problems we seek to address.
Scheduling in the Health Domain

“Hospital scheduling is an inherently distributed problem, not amenable to a centralized solution because of the human organizational authorities involved.”

Decker and Li (2000)

While scheduling research has made marked progress in the last two decades, significantly less progress has been made in applying this research to complex real world problems. The problem lies in the inability of current systems to effectively model the complexity of real world domains (Prieditis et al., 2004). This work is focused on developing a state-of-the-art flexible intelligent methodology for solving complex dynamic distributed scheduling problems. Though very common in all real world domains, such problems are exemplified in the health domain. Scheduling emergency and elective surgery, patient workflow management, and the sharing of limited and expensive lifesaving equipment among hospitals are among many examples of instances in the health domain where it is hard to schedule, and even harder to maintain the schedule, given the dynamic nature of the environment. Further, optimizing scheduling for the health domain would also provide much needed relief to the encumbered public health sector. For this reason, the health domain is chosen as the real world domain where we will apply and evaluate the methodology developed in this study. In keeping with this, a study of scheduling problems in the health domain was carried out and the elective surgery scheduling problem was identified as an ideal candidate for the study. A comprehensive case study of processes involved in elective surgery scheduling at the Princess Alexandra Hospital, Brisbane, has also been carried out. We have also reviewed the state-of-the-art in
3. Scheduling in the Health Domain

Russell and Norvig (2003) define six properties of task environments. They further define the hardest case as an environment that is partially observable, stochastic, sequential, dynamic, continuous, and multi-agent. We use these properties to classify the elective surgery scheduling problem (see Table 3.1) for the purpose of understanding the complexity and providing a platform to compare it to other scheduling problems. It is observed that the problem meets Russell and Norvig’s description of the hardest case to solve.

### Table 3.1: Classifying the Elective Surgery Scheduling Problem (ESSP)

<table>
<thead>
<tr>
<th>Property Characteristic</th>
<th>ESSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully Observable vs. Partially Observable</td>
<td>Partially Observable</td>
</tr>
<tr>
<td>Deterministic vs. Stochastic</td>
<td>Stochastic</td>
</tr>
<tr>
<td>Episodic vs. Sequential</td>
<td>Sequential</td>
</tr>
<tr>
<td>Static vs. Dynamic</td>
<td>Dynamic</td>
</tr>
<tr>
<td>Discrete vs. Continuous</td>
<td>Continuous</td>
</tr>
<tr>
<td>Single Agent vs. Multi-Agent</td>
<td>Multi-Agent</td>
</tr>
</tbody>
</table>

The rest of this chapter is arranged as follows. We start by characterizing the health domain, specifically the elective surgery scheduling problem, based on Russell and Norvig’s classification (Russell and Norvig, 2003) of task environments. This is done to better understand the underlying complexity and to provide future research with a means of comparing this to other domains. We then present the problem of scheduling elective surgery at the Princess Alexandra Hospital, Brisbane, Australia. This is followed by a review of the state-of-the-art of other research seeking to address scheduling in the health domain. We then present our intelligent agent methodology for solving the Elective Surgery Scheduling Problem. We also discuss the agent architecture and discuss various components, focusing especially on the DCOP engine.

3.1 Properties of Health Domain

Russell and Norvig (2003) define six properties of task environments. They further define the hardest case as an environment that is partially observable, stochastic, sequential, dynamic, continuous, and multi-agent. We use these properties to classify the elective surgery scheduling problem (see Table 3.1) for the purpose of understanding the complexity and providing a platform to compare it to other scheduling problems. It is observed that the problem meets Russell and Norvig’s description of the hardest case to solve.

3.2 Elective Surgery and Waiting Lists

“Elective surgery is surgery that, in the opinion of the treating clinician, is necessary but can be delayed for at least 24 hours. (Queensland Health, 2010)”. In Australia, patients requiring elective surgery are placed on a register, or waiting list, so their
3.2. Elective Surgery and Waiting Lists

surgery can be planned. The scheduling of patients on a waiting list is determined according to the patients clinical need and the likelihood of their condition deteriorating or becoming an emergency. These patients are assigned one of three nationally standard clinical urgency categories:

- **Category 1** - admission within 30 days desirable for a condition that has the potential to deteriorate quickly to the point that it may become an emergency.

- **Category 2** - admission within 90 days desirable for a condition causing some pain, dysfunction or disability but which is not likely to deteriorate quickly or become an emergency.

- **Category 3** - Admission at some time in the future acceptable for a condition causing minimal or no pain dysfunction or disability, which is unlikely to deteriorate quickly and which does not have the potential to become an emergency.

If a patient does not receive surgery within the clinically desirable time defined by their category, they are classified as a *long-wait* patient. For category 3 patients, this time is defined as 365 days.
3. Scheduling in the Health Domain

Figure 3.2: Patients Waiting for Elective Surgery as at 1 Jan 2010 (Queensland Health, 2010)
3.2. Elective Surgery and Waiting Lists

Figure 3.3: Patients Waiting for Elective Surgery as at 1 Jan 2010 by Surgical Speciality (Queensland Health, 2010)

The escalating demand for elective surgery, compounded by a shortage of fully trained surgeons, anaesthetists, and nurses, means that efficient scheduling of elective surgery is critical to ensure optimum utilization of the public health system. Efficient scheduling of elective surgery is, however, an extremely difficult and time consuming given the complex, distributed, and dynamic nature of the health domain coupled by individual constraints and preferences of the doctors, patients, and other stakeholders that contribute to the scheduling process. Further, in most Australian public hospitals, resources including beds and surgical staff are shared between both elective and emergency services. For this reason, when emergency demand increases, the ability of hospitals to provide elective surgery services is compromised.

Recent statistics (Australian Medical Association, 2009) show that, despite repeated government intervention, elective surgery wait times continue to grow in Australia (Fig. 3.1). Though slightly better, Queensland statistics follow similar trends. Figure 3.2 represents the change in number of patients waiting for elective surgery over the past four years in each category. These represent patients from various surgical specialities (Fig. 3.3). As of 1 January 2010, 34480 patients were waiting for elective surgery in Queensland of whom almost 20% had waited longer than a clinically desirable time (Queensland Health, 2010). Despite government initiatives such as a 38% increase in clinical staff (since June 2005) and 11.7 million dollars spent on treating public elective surgery patients in private hospitals (2009/10), this
number represents a 59.1% increase in long-waits for Category 1 patients and a 22.5% increase in long-waits for Category 2 patients since October 2009.

Any improvement in the scheduling processes would not only result in improved staff and resource utilization, but also lead to reduced patient waiting and patient-in-care times, increased patient and staff satisfaction, and increased hospital revenue.

3.3 Elective Surgery at the PA Hospital

We discuss current scheduling processes at a leading public hospital in Queensland to help establish a better understanding of the intricacies involved.

The PA hospital offers 21 operation theatres (OT) that can be utilized by the various departments for emergency and elective surgery. Of these, one is run 24 hours and caters for emergencies while at least one theatre is available at any given time during working hours to tackle trauma cases.

For the sake of scheduling the theatre schedule is divided into AM and PM slots of 3.5 hours each. These slots are allocated to various doctors, departments, trauma or emergency.

Responsibility of ensuring smooth and efficient running of the OT and contributing staff and resources for scheduled procedures rests with several departments represented by their managers (see Table 3.2). Each of these managers carries a dual responsibility. On one hand they have to ensure that no constraints are violated at their end and that their individual schedules are optimal. On the other hand, they have to ensure that their departmental schedules are aligned and the resulting OT schedule is conflict free.

Slots on the theatre schedule are currently preallocated to various surgical teams to allow easier coordination of departmental schedules. This allows surgical teams to plan their weekly procedures more efficiently. Nursing can use this preallocation to loosely classify staff speciality requirements on various days. Resource conflicts can also be minimized using this strategy. However, the specific needs of individual procedures is only known as they are booked in, and the various departments constantly modify their schedules to adapt to this ongoing change. If slots are not filled, they may be reallocated to other surgical teams while extremely complicated surgery may need neighbouring slots to be freed up and reallocated.

For elective surgery, the bookings department receives booking requests from the Doctors/departments for their respective slots. This information is entered into the
3.3. Elective Surgery at the PA Hospital

<table>
<thead>
<tr>
<th>Department/Resource</th>
<th>Responsible Person</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation Theatres</td>
<td>Theatre Manager</td>
</tr>
<tr>
<td>Anaesthetists</td>
<td>Director of Anaesthesia</td>
</tr>
<tr>
<td>General Nurses</td>
<td>Charge Nurse</td>
</tr>
<tr>
<td>ORMIS</td>
<td>ORMIS Supervisor</td>
</tr>
<tr>
<td>Bookings</td>
<td>Bookings Supervisor</td>
</tr>
<tr>
<td>Doctors</td>
<td>Director of Surgery</td>
</tr>
<tr>
<td>Anaesthesia Nurses</td>
<td>Anaesthetic Charge Nurse</td>
</tr>
</tbody>
</table>

Table 3.2: Resource Managers in Elective Surgery Scheduling

Figure 3.4: Current Model for Scheduling Elective Surgery at the PAH
3. Scheduling in the Health Domain

ORMIS	extsuperscript{1} system. ORMIS (Operating Room Management Information System) is an Operating Suite application which manages the patient episode from the first contact, through the waiting list, pre-operative, peri-operative, and recovery stages and offers an rich source for information for analysis. It allows the bookings department to schedule procedures based on requests received. The complex task of optimizing this schedule is, however, left up to the operators of the system.

While the bookings department handles requests for surgery, each department connected with the surgery, i.e. allocating staff or other resources to the surgery, carries out their individual scheduling activity. For example, the Charge Nurse allocates nurses to theatre slots. As procedures are booked into ORMIS, they become visible to each of these departments and any rescheduling required because of the nature or demand of the booking is carried out. For example, the Charge nurse may have to allocate a nurse with a specialized skill for a specific procedure, or the Director of Anaesthesia may find that he does not need to provide an anaesthetist for a private patient booked into a slot. If they encounter any clashes, such as if the Theatre Manager finds that three procedures booked at the same time require special equipment but the hospital has only two, they note this information down as a conflict. If the problem is major, it is sometimes relayed back immediately to bookings or to the doctor concerned for redressal.

Every Thursday, the managers of the different departments involved in the above process meet and review bookings for the next day to the Friday in the week ahead. Each session is discussed and if any of the managers has a problem, they bring it to everyone’s notice and a solution is worked out by negotiation. Negotiation can generally be classified as belonging to one of the following categories:

• Requests:
  – Change of Time Request - do we leave as is or do we change ?
  – Change of Day Request - do we leave as is or do we change ?
  – Change of Theatre Requested - do we leave as is or do we swap two sessions or do we change ? - eg. doctor likes (or may need) to use a particular table.
  – Close a Slot Request - do we leave as is or do we close ?
  – Open a Previously Closed Slot Request - Do we leave as is or do we open ?

• Conflicts:

\footnote{http://www.isoftplc.com/corporate/products/2593.asp, last visited 13/04/2010}
3.3. Elective Surgery at the PA Hospital

- Staff Not Available - do we continue to hold / close / reallocate?
- Scheduled Doctor Away - do we close / reallocate?
- Slot is Empty - do we continue to hold / close / reallocate?
- Too Many Bookings - leave as is or request to reschedule.
- Too Few Bookings - leave as is or ask if they will fill up or offer to someone else.
- Conflicting Booking - need to correct / reschedule - eg. two parallel bookings need same single resource.
- Staff/Resource Needed - need to provide / reschedule.
- Staff/Resource Not Needed - need to reschedule.
- List Mismatch - whose is correct - generally ORMIS e.g. case information fax exists but not entered on list - need to check and add.
- Inefficient Booking Identified - need to correct / reschedule - eg. Doctor has a long session booked after a session that is bound to run over - move to other OT.
- Incorrect Booking Identified - need to correct / reschedule - eg. Kidney Transplant booked for 8am - not possible to have donor kidney ready by that time - move to PM.
- Typo/ Logical Error Identified - need to correct.

- Predictive:
  - Understaffed Situation (Workshops etc.) Coming Up - need to close slots before bookings come in.
  - Typical Overload Situation (Holidays etc.) Identified - need to schedule for Emergency/Trauma slots before bookings come in.
  - Typically Underutilized Slot Identified - consider allocating it to someone else.

All changes made to the system after the meeting are dealt with individually by the departments. All conflicts arising are negotiated and resolved on a case-by-case basis using conventional communication such as telephone and emails, or even by face-to-face meetings. This further leads to huge delays in decision making and can result in the creation of a bad or compromised schedule.

This process is faced with several problems. Picking up conflicts in the schedule is left up to the keen eye of the scheduling specialist. If the concerned scheduling specialist is busy and unable to attend the meeting, their replacements may not do
3. Scheduling in the Health Domain

Figure 3.5: Emergency Admissions in Queensland 2005/06 to 2009/10 (Queensland Health, 2010)

as good a job as them. In addition, the schedule is never complete on a Thursday morning and some slots may not be filled until just a day or two before. Further, unexpected emergencies, change in patient health state, sudden changes in staffing, and surgeon availability etc. lead to several changes being required from time to time. In meeting with the dynamics of the domain, the schedule needs to be updated quickly and efficiently. This is often not possible, because of the delays in communicating with the different departments to make the change, and often leads to an easy but inefficient solution. For example, if a procedure is cancelled at the last minute because new medical reports say it is no longer required, bookings may be able to offer the slot to another doctor but may be unable to confirm availability of specialist staff or equipment because the charge nurse or the theatre manager were temporarily unavailable. This may lead to the slot being unused.

Scheduling elective surgery thus presents a very complex scheduling problem where the dynamics of interacting factors affecting the schedule demands a coordinated, intelligent, flexible, and dynamic solution, in order to achieve a high level of utilization and cater for many different competing criteria.
3.4 Sharing of Resources between Elective and Emergency Surgery

Emergency admissions are regarded as the first priority of the hospital system (Queensland Health, 2010). In most Australian public hospitals, resources including surgical staff and operating theatres are shared between elective and emergency services. At the PAH, this is dealt with by blocking off one theatre for emergency services and holding another to handle trauma cases. However, recent statistics show that there is a lot of fluctuation in demand for emergency services (see Fig. 3.5). When emergency demand increases, elective surgery procedures are generally put on hold and rescheduled to a later time. Quick rescheduling measures are thus required to allow departments to confirm availability of staff and resources at the earliest to ensure this process does not inconvenience patients too much and to minimize the impact of this rescheduling on the rest of the elective surgery schedule. Further, reduced demand for emergency services provides an opportunity for opening up these available beds for elective surgery if efficient scheduling mechanisms are available. Recently introduced tools like the Patient Admissions Prediction Tool (Boyle et al., 2008a,b) allow accurate prediction of patient load for the next hour, the rest of the day, into next week, or even on holidays. Data from these tools could also be used effectively to better manage the sharing of resources between elective and emergency surgery if better scheduling mechanisms were available.

3.5 Current State-of-the-Art

A study conducted as part of this research looked at identifying the state-of-the-art scheduling software used in the health domain. Several discussions and interviews revealed that the most trusted form of scheduling being used was still the white board and in areas where computers were being used, these were still only being used to record decisions being made by the people doing scheduling.

In relation to work proposed by previous research, a review and analysis of previously proposed health-related scheduling systems was carried out and it was observed that most systems proposed lacked realism as they were based on simplistic case studies and did not map the complexities of the domain they were modeling.

Historically, a number of solutions to the scheduling problem in the area of intelligent (or semi-intelligent) scheduling can be found in literature. The first “intelligent scheduling system” to be reported, ISIS (Fox et al., 1982) also introduced scheduling (or specifically job shop scheduling) to the AI community. Over the last two decades,
several research efforts have been directed at solving the scheduling problem, though
most have been directed at the classical “job shop scheduling problem” (Zweben and
Fox, 1994; Prosser and Buchanan, 1994; Jones and Rabelo, 1998). Further, research
in the Operations Research domain has also looked at the problem of scheduling for
Operating Theatres and proposed efficient solvers (Jebali et al., 2006; Lamiri et al.,
2008; Pham and Klinkert, 2008; Lamiri et al., 2009) to handle the task, but most
such solvers approach the problem as a centralized one. We focus our research on
distributed Multi-Agent representations of the problem.

Multi-Agent systems are actually a very popular paradigm for modeling and solving
distributed problems. Hospitals are inherently distributed in nature and control and
decision making rests with each individual department. Thus, multi-agent technol-
yogy offers expressively rich modeling of the hospital scheduling problem.

3.5.1 DISA - Distributed Interactive Scheduling with Abstractions

DISA (Distributed Interactive Scheduling with Abstractions) (Berry et al., 1994;
Friha, 1998a) addresses resource management in uncertain domains, specifically the
task of the allocation of operating theatres and technicians to surgical operations in
a large hospital, using the Multi-Agent Systems paradigm. It extends DAS (Burke
and Prosser, 1991) in that it follows a hierarchical structure and uses heterogeneous
agents, and like ReDS (Hadavi et al., 1992), it uses a temporal constraint tree, i.e.
hierarchies are divided according to the time parameter.

The architecture of DISA is thus hierarchical (see Fig. 3.6), with each agent’s posi-
tion reflecting its temporal influence. The agents are heterogeneous in function, each working depending on its position in the hierarchy. The P-Agent sits at the highest level and is a long term agent responsible for strategic planning, say for operating theatre requirements and nursing levels for a whole year. The next level contains the medium term S-Agents, which are scheduling agents coordinating activities corresponding to say a week or month of operations or nursing shifts. These carry out the task of sequencing the tasks onto the principal resource. Finally, at the lowest level there are the short-term agents, the A-Agents, which are responsible for the allocation of specific resources to sequenced tasks and which correspond to the day to day operational schedule.

DISA utilizes temporal abstractions in the form of summarizations and generalizations for problem reduction. To prevent over simplification, it employs a clustering algorithm which applies a Value Assignment Delay heuristic followed by temporal summarizations based on work by Choueiry and Faltings (1993). A conflict resolution procedure guided by user input, a generalization process that provides feedback for conflict resolution and upper levels of hierarchy and a dynamic structure which forms as the problem is solved are other characteristics of the system. A high-level functional diagram of DISA is shown in Figure 3.7.
3. Scheduling in the Health Domain

The DISA system is based on a Geneva Hospital Case Study but no actual results of the system performance are published. The last published work (Friha, 1998a) reports that real data has been collected but that discussions are still on with Hospital of Geneva. No actual testing is documented. It further identifies the need to integrate learning capability, to perform empirical analysis of the scheduling algorithm, to explore conflict resolution, and to improve scheduling performance, as areas that need addressing. On further communication with one of the authors (Berry, 2006), it was confirmed that the project folded before real world testing being carried out.

3.5.2 The GPGP Approach

Decker and Li (1998, 2000) present an approach for scheduling resources in hospitals. Specifically, the research focuses on extending Generalized Partial Global Planning (GPGP) (Decker and Lesser, 1995; Decker, 1995) to include mutually exclusive resources, i.e. the patient, and then to help map patient pathways as captured by nursing units (see Fig. 3.8). It recognizes that no single coordination mechanism is sufficient to model all kinds of coordination and allows the agents to choose from one of multiple coordination mechanisms available.

While the authors seek to preserve human organization and authority structures and cater for increasing complexity in hospital scheduling, the work is based on a very simplistic and out-of-date case study of a hospital provided in (Ow et al., 1989). In addition, it does not consider the dynamics of the patient scheduling problem and the medical priorities of the patients. The authors also recognize the need to increase the model’s realism by using more detailed data from real hospitals.
3.5. Current State-of-the-Art

3.5.3 ADAPT

The ADAPT project (Heine et al., 2003; Heine and Kirn, 2004, 2005) focused on developing an information system that can substantially increase the efficiency of hospital process management. One of the sub-goals of the project is the improvement of distributed appointment scheduling. It recognizes the fact that scheduling in hospitals is inherently distributed between various organizational units and numerous interdependencies exist between the processes, making the task a complex one. It aims at finding new more sophisticated scheduling and negotiation strategies through simulation and analysis of real world scenarios. In pursuing this goal, an agent based simulation system, SeSAm (Shell for Simulated Agent Systems) is proposed to provide an integrated environment for modeling and simulating complex multi-agent systems.

A prototype system of the Simulator has been developed along with a simplistic scheduling scenario for patient scheduling in a clinic for radiation therapy (see Fig. 3.9), but demonstrating the practical use of the system and evaluating and deploying the experimental agent systems to existing information system infrastructure are identified as future goals.
3. Scheduling in the Health Domain

3.5.4 Medical Path Agents (MedPAge)

The Medical Path Agents (MedPAge) project (Bartelt et al., 2002; Paulussen et al., 2003, 2004a,b) presents a multi-agent based distributed patient-centred approach for inter-unit patient scheduling in hospitals. It identifies the shortcomings of traditional (operations research) based approaches especially in failing to model the dynamics of the patient scheduling problem and the medical priorities of the patients. MedPAge uses two types of agents - Patient Agents (that endeavor to minimize their stay time) and Hospital Agents (that endeavor to minimize resource idle times). To reduce complexity, all resources needed for a specific medical action are represented by the hospital unit (resource agent) responsible for them. For inter-agent coordination, a market mechanism is used as it facilitates efficient solutions with low communication needs and allows agents to take actions according to their current (dynamically changing) situation based upon private information and preferences. The agents trade resources until no agent can improve its schedule without harming another agent. The mechanism works on a worth-oriented environment, where the degree of goal achievement is evaluated by continuous worth functions based on cardinal measurement of the health state progress over time that enables the agents to compromise in order to achieve a better solution.

However, while MedPAge identifies the shortcomings of other approaches, the project is specifically based on developing coordination mechanisms and an auction protocol without looking into sophisticated DisCSP processing, though reducing backtracking with the inclusion of more domain knowledge is highlighted as a future goal. In addition, while the project reports field study in 5 hospitals, no testing on data is
reported and validation of current coordination mechanisms is identified as a future aim.

More recent work (Paulussen et al., 2006b; Zoller et al., 2006) reintroduces the utility functions and coordination mechanism and a simulation based on real hospital data to show that considering the health state of the patient for scheduling reduces the waiting time for patients with more severe diseases. Once again, the authors identify the use of scheduling heuristics as possible future extensions to the model.

3.5.5 Policy-Agents

The Policy-Agents project (Becker et al., 2003; Czap and Becker, 2003; Czap et al., 2005) aims at providing a multi-agent system for hospital scheduling, specifically for scheduling of central operating theatres. It is based on a case study carried out at a German partner hospital to identify the conventional process of scheduling (see Fig. 3.11). It identifies that scheduling is a two step process, one where long term allocations are made, and the other step dealing with scheduling and rescheduling for the present and immediate future. It also identifies the latter step as being a process with an outcome that is highly dependent on situational variables that cannot be predicted accurately.

The major focus of the project, however, is on improving the coordination mechanism. The authors argue that Expected-Utility-Theory (EU-Theory) is not very well suited for the problem at hand as physicians, nurses and so on, who are the decision makers in our hospital setting, are not specially trained in this process. They propose a conjoint-analysis based model to create a utility function based on systematic profiling of user preferences, coupled with a learning mechanism to cope with dynamically changing preferences.

Scheduling of actions and resources in the system is carried out in two separate stages. Initially, the agent system creates a preliminary plan taking only medical and organizational demands and constraints into account. In this process, the system interacts with the domain expert, offering him sub plans for modification. In the second stage, the agents, acting as personal assistants, try to negotiate the best working schedule considering the individual preferences without sacrificing medical or efficiency goals.

More recent work (Krempels and Panchenko, 2006a,b) presents a semi automated scheduling system where the scheduling problem is broken up into 4 sub-problems, each solved with a different preference based heuristic, and presented to a human expert for accepting all, part, or none of the schedule. The process continues until
3. Scheduling in the Health Domain

Figure 3.11: Conventional Process of OT Scheduling (Czap and Becker, 2003)

Figure 3.12: Individual Agent Decision Making in Policy Agents (Czap and Becker, 2003)
the scheduling process is complete. It also identifies evaluation in a real-world setting as a future task to be carried out in cooperation with the university hospital.

### 3.5.6 Agent Based Information Logistics in Anaesthesiology (AGIL)

Knublauch et al. (2000) present an approach for management of information related to preoperative planning and theatre management carried out by the anaesthesiology departments. Based on the tasks carried out by the clinical personnel and information flow between them (see Fig. 3.13), they distinguish between three types of agents - Interface Agents (that communicate with personnel), Task Agents (that carry out tasks on behalf of clients), and Information Agents (that collect, manage, and provide data).

While this model is used to describe a future hospital (see Fig. 3.14), their study does not focus on any actual process scheduling or negotiation between agents.

They also propose a mobile agent-based theatre management system (Sedlmayr et al., 2000) (see Fig. 3.15) which is motivated from the understanding that the domain is dynamic and it is important to get information of changes to the domain/scheduling expert as soon as possible but once again the emphasis is on timely information management and aims to decrease information and workload and shorten communication paths. The actual scheduling decision making is seem-
Figure 3.14: Agents for Information Management in a Future Hospital (Knublauch et al., 2000)

Figure 3.15: Proposed Agent Based Theatre Management System (Sedlmayr et al., 2000)
3.5. Current State-of-the-Art

Figure 3.16: MAS Architecture for Scheduling Organ Transplants (Moreno et al., 2001a)

ingly left up to the user. Further, this system is proposed as a “vision” and is not actually implemented.

3.5.7 Scheduling Organ Transplants

Moreno et al. (2001a,b) present a multi-agent approach to solving the task of scheduling an organ transplant operation. The system assumes that the task of selecting the donor and transporting the organ is already completed and is thus very similar to the procedure of scheduling elective surgery that we are modeling. The proposed system models the hospital structure quite accurately and uses the contract net protocol for negotiation between agents. It also presents a heuristic based on scoring to optimize allocation of Operation Theatre slots. A prototype of the system has also been presented. However, the proposed system is fairly basic and assumes that all theatres are identical, all doctors involved are specialists capable of performing all organ transplants, and that nurses and doctors have no specialisations. The authors also acknowledge that so far the system has been tested on only “toy examples” and term the theatre scheduling heuristic as “rudimentary”.

67
3.5.8 Discussion

While DISA offers a formal architecture that seeks to address the entire scheduling problem, a theoretically sound problem solving methodology, and is based on a real hospital case study, no actual testing is documented and most system components, including integrating learning capability, empirical analysis of the scheduling algorithm, exploring conflict resolution, and improving scheduling performance, are reported as future aims.

The GPGP approach caters for flexibility between coordination mechanisms but fails to model the dynamics of the patient scheduling problem and the medical priorities of the patients. Further, the study is based on an out of date case study and the authors acknowledge the lack of the model’s realism.

While ADAPT and AGIL seek to address scheduling, both projects again have no reported progress in this field. In addition, the approach by Moreno, Valls and Bocio is admittedly simplistic and “rudimentary” and fails to model the complexity of real world scheduling.

MedPAge offers a unique approach where patient agents negotiate with hospital agents using state of health. It also provides excellent economic models for the auction mechanism but this approach is nowhere near the real scheduling approach in hospitals. Once again, exploring sophisticated DisCSP processing is presented as a future aim.

The Policy Agents project, like DISA, proposes the involvement of the domain expert into the scheduling architecture, but once again, like DISA, no testing or development is reported. It, however, seeks to divide the scheduling process into a two step process where negotiation in order to include preferences follows scheduling based on medical priorities and we see this as a sub-optimal process. More recent work by some of the authors reports a four-step semi-automated scheduling approach and once again real-world testing is reported as future aims.

3.6 Intelligent Management of Elective Surgery

As discussed in Chapter 2, the Multi-Agent Systems paradigm offers a natural fit solution for modeling the Elective Surgery Scheduling Problem given the distributed nature of the domain. It also captures the autonomy of the hospital departments in constructing and managing their individual schedules. In order to ensure the optimality and compatibility of the departmental schedules, the Distributed Constraint Optimization formalism is chosen to guide coordination and resolution of
inter-departmental schedule conflicts. Given the complexity of the numerous interdependencies that affect the scheduling process, and the lack of structured scheduling rules to control the process of automated scheduling, it is also essential that the proposed methodology allows for domain expert interaction to guide and complement the automated scheduling process. An intelligent system should also be able to learn from this interaction, and integrate its knowledge to aid future scheduling.

Figure 3.17: Agent Interaction for Scheduling Elective Surgery at the PAH

We propose an agent-oriented methodology where each department involved in the scheduling of its resources, be they patients, staff or equipment, is represented by an intelligent agent. Similar to the concept of a personal assistant, these agents are customized to the constraints, preferences, and priorities of the department they represent and carry out scheduling for their respective departments (Fig. 3.17). It is also the responsibility of the agents to send, receive, and react to messages from other agents. As necessary, the agents then negotiate in a privacy-preserving manner to resolve inter-agent constraints and optimize their local schedules. Where necessary, the agents initiate domain expert interaction and incorporate learning from this user feedback. These features are discussed in further detail as we discuss the functional architecture of the agents.
3.7 Proposed Architecture of Agent

A high level functional model of the architecture of individual agents is presented in Figure 3.18. The proposed model focuses on the elements that contribute directly to scheduling process, while other architectural components and core services that do not directly impact this model are abstracted away. The model consists of a number of modules. An interface module handles communication with other agents and users. Decision support and learning is handled by the intelligence module. Negotiation and optimization is driven by the DCOP engine.

The agents, thus, have a number of capabilities. The environment is monitored for changes necessitating updates to the schedule. Advanced DCOP algorithms are used to optimize local schedules while ensuring efficient alignment of the global schedule. They use logical reasoning to identify the need for and to guide negotiation. They can also learn user preferences and domain knowledge through domain expert interaction.
3.7.1 The DCOP Engine

The DCOP algorithm we utilize needs to be robust in a number of ways. It must be scalable to the variety and complexity of the involved agents’ sub-problems. Negotiation resolution must be timely with respect to the environment under which the negotiation is taking place. The ability to separate the communication protocol from the details of the local solver is also essential as this facilitates the customization of the local solver to each agent’s unique problem while maintaining communication compatibility.

In Chapter 4, we present the Dynamic Complex Distributed Constraint Optimization Problem (DCDCOP) algorithm that has been designed to meet the above criteria. In DCDCOP, agents solve their local sub-problem using a local solver of their choice and then employ a novel metric called Degree of Unsatisfaction to guide inter-agent negotiation and solve inter-agent constraints. DCDCOP outperforms the state-of-the-art DCOP algorithms, by more an order of magnitude. In chapter 5, we choose DCDCOP to drive the DCOP engine in ASES, our proof-of-concept Automated Scheduler for Elective Surgery application.

3.7.2 Intelligence Module

The Intelligence Module is designed to provide decision support for negotiation requests and conflict resolution. The decision flow diagram of this module is presented in Figure 3.19. When a new negotiation request is received, the module checks against its knowledge bank for any domain rules or previous knowledge that may pertain to the request. The previous knowledge is formed by the agent’s ongoing learning from user interaction. A confidence score is calculated based on any matches found and if this confidence score is above the nominated threshold, then the negotiation request is accepted and the schedule is modified to accommodate it. If this is not the case, the system refers the request to the domain expert for a decision to be made and actions the user input appropriately. The module also incorporates learning from user feedback. Any interaction with the user leads to the system learning the request and its outcome to aid future decision support. The system thus gradually learns to model the domain expert’s decision making process.

3.8 Summary

In a bid to demystify the complexity of the health domain, the chosen elective surgery environment is classified, only to arrive at the conclusion that it represents the
The chosen elective surgery scheduling problem at Princess Alexandra Hospital, Brisbane, is discussed in detail. It is concluded from the case study that the current procedures followed for elective surgery scheduling are far from optimum and, in addition to using up a lot of valuable staff time, often lead to an inefficient or compromised schedule.

Further, the study of current approaches for solving the scheduling problem reveals that while several good research efforts are being made, none seems to offer a complete, or even acceptable, solution from the real-world perspective.

We believe that while all of the methods make a positive contribution to improv-
The state-of-the-art of scheduling, what is missing is a flexible and intelligent methodology that provides easy and effective modeling of the inherent distribution of resources and decision making, while allowing for easy integration of sophisticated problem solving and negotiation strategies. Also needed are good real-world benchmarks that can be used to evaluate proposed scheduling strategies.

To overcome these limitations, we model the Elective Surgery Scheduling Problem domain using the MAS paradigm and present a DCOP based problem solving methodology where intelligent agents, trained with the constraints, preferences, priorities etc. of the administrators, optimize schedules for their respective departments. They then negotiate in a privacy-preserving manner (i.e. without sharing more information than is essential) to resolve inter-agent constraints. The architecture of each agent incorporates an interface module to handle internal and external communication, an intelligence module to handle decision making and learning, and a DCOP engine to drive the optimization. Essential attributes for the DCOP engine are also discussed.

Using this methodology, the system can translate from the current practice of resolving conflicts during weekly meetings to one where ongoing negotiation ensures that the departmental schedules are largely conflict free at all times, thus making the weekly meetings redundant. The use of agents also significantly reduces delays in inter-departmental information flow and negotiation. Though delays resulting from waiting for user interaction are unavoidable, the need for such interaction will also decrease as the system learns and builds its knowledge bank for automated decision support.
3. Scheduling in the Health Domain
Chapter 4

An Efficient Algorithm for Solving Dynamic Complex DCOP Problems

“DCOP is NP-hard, so an important line of work focuses on developing fast incomplete solution algorithms for large-scale applications”

Kiekintveld et al. (2010)

Our research efforts are focused on tackling dynamic complex Distributed Constraint Optimization Problems (DCOP). In constraint-speak, these can be understood as distributed problems, with multiple variables per agent, where the constraint graph undergoes changes from time to time. In chapter 3, we studied the Elective Surgery Scheduling Problem and found that it presented an excellent real world example of this class of problems. We also presented a Multi-Agent methodology for modeling and efficiently solving this class of problems, and outlined desirable attributes for our DCOP solver. Essentially, our algorithm of choice needs to model the problem at hand instead of trying to cast it to suit the algorithm. Further, maintaining the inherent decentralization of control and departmental privacy, while ensuring efficiency, needs to be recognized as an integral part of the problem solving process.

In chapter 2, we reviewed the state-of-the-art in DCOP algorithms and concluded that while complete algorithms like ADOPT (Modi et al., 2003), DPOP (Petcu and Faltings, 2005c), and NCBB (Chechetka and Sycara, 2006a) offer excellent solution to several real world optimization problems, they fail to offer efficient solutions to complex dynamic DCOP problems. Further, synchronous algorithms like DBA (Zhang et al., 2003), DSA (Zhang et al., 2003), and MGM (Maheswaran et al., 2004a), and algorithms that utilize partial centralization, like OptAPO (Mailler and
4. An Efficient Algorithm for Solving Dynamic Complex DCOP Problems

Lesser, 2004b), are not well-suited to the class of problems we seek to address.

In this Chapter, we propose novel measures of Static Cost Density (SCD) and Dynamic Cost Density (DCD) and use these measures to calculate the metric of Degree of Unsatisfaction (DU) as applied to DCOP problems. This extends previous work (Zhou et al., 2004) in the DisCSP domain. We then present the Dynamic Complex Distributed Constraint Optimization Problem (DCDCOP) algorithm, where agents solve their sub-problem using a local solver of their choice and then use the DU metric to guide inter-agent negotiation and solve inter-agent constraints. The algorithm’s execution is discussed with an example. A proof of soundness is also presented. We also propose a DSA-like variant of this algorithm, called CostDCOP, that uses Cost instead of DU to guide inter-agent negotiation, to help validate the efficiency of the DU metric. We then present empirical evaluation of the DCD-COP algorithm as compared to CostDCOP and other state-of-the-art asynchronous algorithms and discuss our findings.

4.1 Degree of Unsatisfaction (DU) for DCOP

In section 2.4.4 we discussed how the DAO and DAONR algorithms use the measures of Static Constraint Density, Dynamic Constraint Density, and Degree of Unsatisfaction (DU) to handle complex and dynamic constraint reasoning problems efficiently. These, however, are suited only to satisfaction and constraint relaxation approaches as the measures do not take varying constraint costs into account. In this section, we redefine these measures as applied to the DCOP domain.

4.1.1 Defining DCD and DU

We generalize the definitions of Zhou et al. (2004) to define the following new static measures of Intra-Agent Cost Density (IACD) and Inter-Agent Cost Density (IAcad):

\[
IACD_i = \begin{cases} 
0, & \text{if } |\text{intra}V_i| = 0 \\
\sum_{j=1}^{\text{intra}C_i} (\delta_m (\text{intra}C_i^j))_{\text{intra}V_i}, & \text{otherwise} 
\end{cases} \quad (4.1)
\]

\[
\text{IAcad}_i = \begin{cases} 
0, & \text{if } |\text{inter}V_i| = 0 \\
\sum_{j=1}^{\text{inter}C_i} (\delta_m (\text{inter}C_i^j)) + \sum_{l=1}^{\text{intra}C_i} (\delta_m (\text{intra}C_i^l)) \frac{1}{|\text{inter}V_i|}, & \text{otherwise} 
\end{cases} \quad (4.2)
\]
4.1. Degree of Unsatisfaction (DU) for DCOP

where,

- \( intraC_i \) is the set of intra-agent constraints for agent \( i \),
- \( \delta_m \) represents the maximum cost of the constraint,
- \( intraC^j_i \) is the \( j^{th} \) intra-agent constraint of agent \( i \),
- \( intraV_i \) is the set of variables constrained by \( intraC_i \),
- \( interC_i \) is the set of inter-agent constraints for agent \( i \),
- \( interC^j_i \) is the \( j^{th} \) inter-agent constraint of agent \( i \),
- \( \kappa C^j_i \) is the set of intra-agent constraints belonging to \( i \) and connected to \( interC^j_i \), and
- \( interV_i \) is the set of variables constrained by \( interC_i \) and controlled by agent \( i \).

The measure of \( IACD \) takes into account the interconnectedness of the variables that are attached to an inter-agent constraint. The higher the cost of intra-agent constraints attached to this variable, the greater the impact of the variable on the cost density. This is justified, as changing the value of this variable would attract much higher effort towards optimizing the problem. In addition, the measures of \( IACD \) and \( IACD \) both equate to zero if there are no intra-agent or inter-agent variables connected to constraints respectively. Thus, no variables, and a consequent cost density of 0, would imply that this component of the problem does not need to be solved further.

Using these measures, we can now define Static Cost Density (SCD).

**Definition 5.** Static Cost Density of an agent is defined as the sum of the maximum possible Intra-Agent and Inter-Agent Cost Densities.

\[
SCD_i = IACD_i + IACD_i
\] (4.3)

In equations (4.1) and (4.2) we can replace \( \delta_m \) by \( \delta_c \), which gives us the current cost of the constraint, to get our dynamic measures of IntraUnsat (\( IU \)), and InterUnsat (\( I_U \)). These represent the dynamic intra-agent and inter-agent cost densities respectively. We utilize these to define the measure of Dynamic Cost Density (DCD).

\[
IU_i = \begin{cases} 
0, & \text{if } |IACD_i| = 0 \\
\sum_{j=1}^{|intraC^j_i|} \left( \delta_c \left( intraC^j_i \right) \right) / |intraV_i|, & \text{otherwise}
\end{cases}
\] (4.4)
4. An Efficient Algorithm for Solving Dynamic Complex DCOP Problems

\[ I_U_i = \begin{cases} 
0, & \text{if } |I_{ACD_i}| = 0 \\
\sum_{j=1}^{\text{inter}C_i} \left( \delta_c(\text{inter}C_j) + \sum_{l=1}^{\text{intra}C_i} \delta_c(\text{intra}C_l) \right) / |\text{inter}V_i|, & \text{otherwise}
\end{cases} \]  

(4.5)

**Definition 6.** Dynamic Cost Density of an agent is defined as the sum of the current Intra-Agent and Inter-Agent Cost Densities.

\[ DCD_i = I_U_i + I_{U_i} \]  

(4.6)

We can now also redefine the measure of Degree of Unsatisfaction for agent \( i \) \( (DU_i) \).

**Definition 7.** Degree of Unsatisfaction of an agent is defined as the ratio of the Dynamic(current) to Static Cost Densities.

\[ DU_i = \frac{DCD_i}{SCD_i} \]  

(4.7)

\( DU \) provides a measure of how far away an agent’s current instantiation is from reaching an optimal state. It does not provide a direct measure to compare the level of complexity to two agents’ problems or the time it may take to solve them. However, unlike a simple summation of max cost or current cost, it attaches a higher cost to the changing of more interconnected constraints, thus providing a more realistic measure of the complexity of the solution.

Adding together the \( DU \) of each agent would then give us the Degree of Unsatisfaction of the MAS.

\[ DU_{MAS} = \sum_{j=1}^{\text{intra}C_i} (DU_j) / |i| \]  

(4.8)

In the absence of a global optimization function, an instantiation that minimizes \( DU_{MAS} \) would represent an optimal or near-optimal solution.

### 4.1.2 An Example of Calculating \( DU \)

To better understand the above measures, we utilize a simple example based on the DCOP problem shown in Figure 4.1. Here, we calculate the values of \( SCD, DCD \) and \( DU \) for agent \( D \), which has four variables, three intra-agent constraints, and three inter-agent constraints. The max cost of each constraint (from the cost table shown in the figure) is 1. In calculating the static measures for the problem, we have:
4.1. Degree of Unsatisfaction (DU) for DCOP

\[ I_{ACD_D} = \frac{(1 + 1 + 1)}{4} = 0.75 \]
\[ I_{UACD_D} = \frac{((1 + (1)) + (1 + (1)) + (1 + (1)))}{2} = 3 \]
\[ SCD_D = 0.75 + 3 = 3.75 \]

Now, assuming a snapshot view of the scenario, where each agent knows each other’s instantiation, we can calculate the dynamic measures:

\[ IU_D = \frac{(0 + 0 + 0)}{4} = 0 \]
\[ I_{UD} = \frac{((1 + (0)) + (0 + (0)) + (0 + (0)))}{2} = 0.5 \]
\[ DCD_D = 0 + 0.5 = 0.5 \]
\[ DU_D = \frac{0.5}{3.75} = 0.13 \]

Note that the constraint between variables 1 and 2 of agent D is counted twice in the calculation of \( I_{ACD_D} \) and \( IU_D \). Similarly, for Agent A:
4. An Efficient Algorithm for Solving Dynamic Complex DCOP Problems

\[
I_{ACD} = \frac{(1+1)}{3} = 0.67
\]
\[
I_{-ACD} = \frac{(1+(1))+(1+(1+1))}{2} = 2.5
\]
\[
SCD = 0.67 + 2.5 = 3.17
\]

\[
IU = \frac{(0+0)}{3} = 0
\]
\[
I_U = \frac{(1+(0))+(1+(0+0))}{2} = 1
\]
\[
DCD = 0 + 1 = 1
\]
\[
DU = \frac{1}{3.17} = 0.32
\]

These calculated values of \( DU \) are exchanged with neighbours and used to determine agent ordering in the DCDCOP algorithm.

### 4.2 The DCDCOP Algorithm

For the purpose of better explaining the algorithm flow, we assume each agent’s \( CurrentContext \) consists of two parts - an \( InternalContext \) that holds the values of its own variables, and an \( ExternalContext \) that holds values of external variables with whom the agent shares a constraint. When an agent \( A \) sends a message to agent \( B \), the message consists of its current value of \( DU \) and an appropriate \( CurrentContext \), i.e. \( A \)'s \( InternalContext \) plus an \( ExternalContext \) that consists of \( A \)'s stored values for \( B \)'s variables. If this \( ExternalContext \) matches with the values \( B \) has for these variables in its \( InternalContext \), \( A \)'s message is said to be compatible with \( B \)'s \( CurrentContext \). Further, while \( DU \) is used to dynamically guide agent ordering, a default ordering is predefined on the agents to resolve cases where two agents have the same value of \( DU \). For the purpose of all experiments conducted as part of this study, a lexicographic ordering is assumed, though a natural default exists or can be easily defined for most real world problem instances.

The DCDCOP algorithm is implemented as follows:

- All agents start by calculating the values of \( IACD \), \( I_{-ACD} \) and \( SCD \). Then they instantiate their local variables using the Branch and Bound algorithm (Freuder, 1989), thus ensuring that the current cost is the minimum allowed
4.2. The DCDCOP Algorithm

by its current context. Each agent then calculates its dynamic measures of \( DCD \) and \( DU \) and sends \( DU \) and the related context to its neighbours (i.e. those agents with whom it shares a constraint).

- All agents start to receive on incoming links. When a message is received, the message context is checked to ensure that it is compatible with the receiving agent’s \( CurrentContext \). If not, the message is discarded and the agent continues to listen on incoming links. If compatible, the \( messageContext \) is added to \( CurrentContext \) and \( messageDU \) is compared with the receiving agent’s \( DU \). If \( DU \) is higher, the agent will reassign its variables, recalculate its dynamic measures, and resend messages on outgoing links. If \( DU \) is lower than \( messageDU \), it will not reassign its variables, but if relevant, it will recalculate its dynamic measures, and resend messages on outgoing links. In the event that \( messageDU = DU \), the agent with a higher predefined ordering will reassign its variables, recalculate its dynamic measures and send messages on outgoing links.

- The search stops when each agent has achieved a stable state and no more messages are transacted. In the case of a solvable problem, this happens when the agent and all its neighbours arrive at \( DU = 0 \). In the case of an optimization problem, this happens when any agent with a higher \( DU \) no longer changes its local solution as doing so would raise the cost of its solution.

The pseudo code of the algorithm is shown in Algorithm 12.

Figure 4.2 shows the decision flow diagram for an agent. Whenever a compatible message is received, each negotiation is handled in one of the following ways: if the values of \( DU \) are not identical, the agent with a higher value of \( DU \) wins the right to reinstantiate its variables; if the values of \( DU \) are identical, the agents follow a fixed predefined ordering between them to decide who wins the right to reinstantiate its variables. The agent that wins the right to reinstantiate its variables will attempt to do so, and will send messages on outgoing links to indicate its new instantiation and \( DU \). The other agent will recalculate its \( DU \) to reflect the new values received in the message. If either agent receives no new values for its neighbour’s variables, it will not perform any functions and will continue to wait for further messages. When an agent with a higher \( DU \) finds no better instantiation for its local variables, it will return the same, thus reaching a steady state until it receives a message from another agent that forces it to reinstantiate its variables.

Note that though this version of DCDCOP uses fixed cost functions, an agent with a lower \( DU \) can force the agent with a higher \( DU \) to negotiate by raising the cost attached to the inter-agent constraint.
Algorithm 12: The DCDCOP Algorithm

Calculate static measures

Solve_local_problem

Calculate dynamic measures

Send message \((DU, \text{CurrContext})\) to all neighbours

Receive messages

\[
\text{when received (messageDU, msgContext) do}
\]

\[
\begin{align*}
\text{if } \text{msgContext and CurrContext are consistent then} \\
\text{add msgContext to CurrContext} \\
\text{if } DU > \text{msgDU} \text{ then} \\
\text{| Solve_local_problem} \\
\text{end} \\
\text{else if } DU = \text{msgDU} \text{ and higher order then} \\
\text{| Solve_local_problem} \\
\text{end} \\
\text{Calculate dynamic measures} \\
\text{Send message } (DU, \text{CurrContext}) \text{ to all neighbours} \\
\end{align*}
\]

end do

Procedure: Solve_local_problem

Branch and Bound to solve local problem

4.2.1 Example of DCDCOP Execution

In continuing the example from section 4.1.2, we start at a point where all agents have calculated their respective \(DU\) values and sent messages out on outgoing links (Fig. 4.3).

Now, assuming agents \(A\), \(B\), and \(C\) are first to receive their messages, \(A\) and \(C\) will both reassign their variables and send out new messages, as they have received \(DU\) messages less than theirs. This results in the state shown in Figure 4.4.

Now assume \(D\) and \(E\) receive both messages together. Acting on the newer messages, both \(D\) and \(E\) will attempt to reassign their variables because their last calculated \(DU\) is higher than the \(messageDU\) from \(A\) and \(C\) respectively, but with the updated values from \(A\) and \(C\), both will arrive at the same local solution as being the lowest cost. Similarly, \(B\) will arrive at the same local solution as being the lowest cost and they will send out their \(DU\) messages (Fig. 4.5). Since all agents now have a \(DU\) of 0, no more messages will flow between them and the MAS will reach a stable state.
4.3 Soundness of the DCDCOP Algorithm

In this section we show that the DCDCOP algorithm is sound. The following assertions are stated to support the proof.

**Assertion 1.** All constraint links are bidirectional. i.e. if variable $v_1$ of Agent $A_1$ shares a constraint with variable $v_2$ of Agent $A_2$, and $A_1$ has $A_2$ in its list of neighbours, then $A_2$ will also have $A_1$ in its list of neighbours.

**Assertion 2.** An agent’s context is divided into two parts - an InternalContext that represents values of its own variables and an ExternalContext that represents values of its neighbours.

**Assertion 3.** A solution set that minimizes $D_{U_{MAS}}$ for the MAS represents an optimal, or near optimal, solution.

**Assertion 4.** An agent is considered to be in a stable state if:

- it is waiting for messages but no message ever reaches it.
- it receives a message but does not send out any messages as it changes neither its InternalContext nor its ExternalContext.
4. An Efficient Algorithm for Solving Dynamic Complex DCOP Problems

Assertion 5. The MAS is considered to be in a stable state when all the agents are in a stable state.

Theorem 1. The DCDCOP algorithm is sound. i.e., the MAS reaches a stable state only if the algorithm finds a solution that minimizes $DU_{MAS}$.

Proof:
From Assertion 5, we know that the MAS would reach a stable state if and only if all agents reached a stable state. To verify if all messaging steps take agents towards a stable state, we evaluate all possible combinations of messages that an agent may receive and evaluate their outcome. From Figure 4.2, we see that a message can result in six possible scenarios. We consider these as individual cases and discuss the resulting action in each case. Note that incompatible messages are discarded without any action, and are thus accounted for in Case 6, where the agent is waiting but does not receive a compatible message.

- Case 1: An agent receives a compatible message that has a lower $DU$ than its own, and contains no new InternalContext from the sender. This represents the scenario where the sender is in a stable state and has only updated its ExternalContext based on a previous message from this agent. In this case
4.3. Soundness of the DCDCOP Algorithm

Figure 4.4: DCDCOP Execution - Step 2

the receiver agent will not send out any messages and will continue to be in a stable state.

- **Case 2**: An agent receives a compatible message that has a higher DU than its own, and contains no new *InternalContext* from the sender: This represents the scenario where the sender is in a stable state and has only updated its *ExternalContext* based on a previous message from this agent. In this case the receiver agent will not send out any messages and will continue to be in a stable state.

- **Case 3**: An agent receives a compatible message that has a lower DU than its own, and contains a new *InternalContext* from the sender: In this case the agent will update its own *ExternalContext* based on values received. It will then attempt to reinstantiate its variables to achieve a lower value of DU, and then send out messages to its neighbours. This will lead to further Case 3 or Case 4 negotiations. If the agent can not achieve a lower DU, it will only recalculate its own DU and send out messages to neighbours leading to further Case 1 or Case 2 negotiations.

- **Case 4**: An agent receives a compatible message that has a higher DU than its own, and contains a new *InternalContext* from the sender: In this case the agent will only update its own *ExternalContext* and send out messages
4. An Efficient Algorithm for Solving Dynamic Complex DCOP Problems

Figure 4.5: DCDCOP Execution - Step 3

to neighbours, leading to further Case 1 or Case 2 negotiations.

- **Case 5**: An agent receives a compatible message that has a $DU$ equal to its own. This would fall into one of the above four cases, depending on the predefined order between agents and whether the message contains a new InternalContext from the sender or not.

- **Case 6**: An agent has sent messages on outgoing links and has not received any compatible return messages from its neighbours: Assuming reliable communication, this scenario indicates that all its neighbours are in a stable state. Resultantly, the agent will continue to remain in a stable state.

Thus, we see that Case 3 negotiations will only continue if $DU$ can be minimized and Case 1, Case 2, Case 4, Case 5, and Case 6 negotiations will eventually lead to a stable state, in which case the agent would have minimized $DU$. When all agents will reach a stable state, it will represent a solution that minimizes $DU_{MAS}$ for the now-stable MAS. Consequently, the DCDCOP algorithm is sound. 

-
4.4 CostDCOP - Using Cost instead of \( DU \)

To help evaluate the true worth of the \( DU \) metric in the context of the algorithm, we developed a DSA-like variant of the DCDCOP algorithm where, instead of \( DU \), cost was used to guide agent ordering. The pseudo code of the CostDCOP algorithm is presented in Algorithm 13. In this case, the agents do not need to calculate static measures and the agent with the higher \( \text{Cost of Current Assignment} \) is given the right to change its instantiation. We do not discuss the algorithm in detail as all other aspects of the CostDCOP algorithm are identical to the DCDCOP algorithm.

![Algorithm 13: The CostDCOP Algorithm](image)

4.5 Experimental Evaluation

Given the need for agents to maintain departmental control and privacy, optimal approaches like OptAPO are not suitable in this context. And while DPOP and NCBB are good DCOP algorithms, we choose ADOPT as the primary evaluation benchmark because, like DCDCOP, it employs constraint-guided search and is well regarded as the gold-standard in search based DCOP algorithms. We also note that the metrics used to compare algorithms are questioned by most researchers. Silaghi and Yokoo (2006) have shown that it is possible to construct problems that can be exploited by algorithms such as ADOPT and DPOP to exhibit their superiority. To ensure a fair comparison, DCDCOP and CostDCOP were both implemented within
4. An Efficient Algorithm for Solving Dynamic Complex DCOP Problems

Figure 4.6: Performance of ADOPT vs DCDCOP ($LD = 2$)
4.5. Experimental Evaluation

Figure 4.7: Performance of ADOPT vs DCDCOP ($LD = 3$)
4. An Efficient Algorithm for Solving Dynamic Complex DCOP Problems

the original ADOPT source code package (Portway, 2008) using ADOPT’s original messaging and performance evaluation procedures.

4.5.1 Preliminary Evaluation

In a preliminary study reported at the Intelligent Agent Technology Conference (Khanna et al., 2009b), we evaluated DCDCOP on the same graph coloring problems that were used to report ADOPT’s performance in (Modi, 2003), and bundled with the source code. The original graph coloring problem data (8-40 variables) was evenly distributed between 3-5 agents. In addition, as in the original evaluation, we analyzed the performance of both algorithms on problems with link density ($LD$) of 2 and 3. The performance was compared using three measures: number of messages, number of concurrent cycles, and time (sec). To prevent a large disparity between the results, the algorithms were run with a maximum time, $timeMax$ of 30 mins. Given the large performance gain exhibited by DCDCOP, we also employed a logarithmic scale for a meaningful display of results (Figs. 4.6 and 4.7).

We observed that DCDCOP outperformed ADOPT significantly on all three scales of measurement. The speedup can be attributed partly to the algorithm exploiting domain centralization and performing each local reassignment within one cycle, and partly to the novel dynamic measures used to guide the inter-agent negotiation part of the algorithm.

The results allow some extremely interesting observations. We observe that the difference in DCDCOP’s performance for $LD = 2$ and $LD = 3$ is not as significant as in the case of ADOPT. This can be attributed to computational superiority over I/O speeds, the same factor also responsible for DPOP’s performance gains over ADOPT (Petcu and Faltings, 2005c). We also note that the performance of DCDCOP with 40 variables (in 5 agents), is reasonably similar to that of ADOPT with 8 agents (and 1 variable per agent). Further, the performance of DCDCOP does not deteriorate much as we increase the problem size from 8 variables (in 3 agents) to 40 variables (in 5 agents). This further asserts the computational superiority of the centralized optimization algorithms such as Branch and Bound, and reinforces common belief that communication is the bottleneck in distributed problem solving.

4.5.2 Extended Evaluation

The problem set used in the preliminary investigation was too small to offer any meaningful indication of the performance of algorithms under evaluation in complex real world domains. Given that such problems often involve hundreds of variables
4.5. Experimental Evaluation

Figure 4.8: Average Time (sec) Vs Problem Size (Variables)

<table>
<thead>
<tr>
<th>Num Vars</th>
<th>ADOPT</th>
<th>DCDCOP</th>
<th>CostDCOP</th>
<th>NCBB</th>
<th>DPOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>934.54</td>
<td>3.14</td>
<td>5.41</td>
<td>932.84</td>
<td>509.63</td>
</tr>
<tr>
<td>100</td>
<td>3551.98</td>
<td>156.9</td>
<td>440.28</td>
<td>7200</td>
<td>6367.1</td>
</tr>
<tr>
<td>200</td>
<td>7200</td>
<td>517.49</td>
<td>1234.52</td>
<td>7200</td>
<td>7200</td>
</tr>
<tr>
<td>400</td>
<td>7200</td>
<td>1751.99</td>
<td>2648.48</td>
<td>7200</td>
<td>7200</td>
</tr>
<tr>
<td>800</td>
<td>7200</td>
<td>5294.32</td>
<td>6827.09</td>
<td>7200</td>
<td>7200</td>
</tr>
</tbody>
</table>

Table 4.1: Average Time (sec) Vs Problem Size (Variables)

Table constrained by several agents, there was clearly a need to test the algorithms using larger problems. For this purpose, we generated a benchmark data set of graph colouring problems of varying sizes (up to 800 variables distributed among up to 40 agents) and complexity (link density 2 and link density 3), having between 10% and 30% inter-agent constraints (typical of the class of problems we seek to address). Table 4.6 presents more information about the attributes of the benchmark data set. Minor improvements were also made to the DCDCOP implementation including...
4. An Efficient Algorithm for Solving Dynamic Complex DCOP Problems

<table>
<thead>
<tr>
<th>Problem ID</th>
<th>ADOPT</th>
<th>DCDCOP</th>
<th>CostDCOP</th>
<th>NCBB</th>
<th>DPOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.10.3.2.10.0</td>
<td>3052.18</td>
<td>6.63</td>
<td>18.31</td>
<td>1584.35</td>
<td>12.72</td>
</tr>
<tr>
<td>5.10.3.2.10.1</td>
<td>124.34</td>
<td>2.14</td>
<td>3.24</td>
<td>4980.72</td>
<td>18.7</td>
</tr>
<tr>
<td>5.10.3.2.10.2</td>
<td>131.69</td>
<td>2.46</td>
<td>3.39</td>
<td>235.04</td>
<td>11.39</td>
</tr>
<tr>
<td>5.10.3.2.10.3</td>
<td>19.94</td>
<td>4.74</td>
<td>6.16</td>
<td>514.89</td>
<td>53.96</td>
</tr>
<tr>
<td>5.10.3.2.10.4</td>
<td>4.36</td>
<td>3.28</td>
<td>4.08</td>
<td>3.17</td>
<td>14.56</td>
</tr>
<tr>
<td>5.10.3.2.10.5</td>
<td>139.47</td>
<td>2.93</td>
<td>5.9</td>
<td>54.91</td>
<td>12.28</td>
</tr>
<tr>
<td>5.10.3.2.10.6</td>
<td>111.17</td>
<td>2.03</td>
<td>3.22</td>
<td>462.64</td>
<td>10.41</td>
</tr>
<tr>
<td>5.10.3.2.20.0</td>
<td>8.56</td>
<td>3.71</td>
<td>6.16</td>
<td>187.67</td>
<td>27.38</td>
</tr>
<tr>
<td>5.10.3.2.20.1</td>
<td>1794.01</td>
<td>2</td>
<td>5.32</td>
<td>179.68</td>
<td>28</td>
</tr>
<tr>
<td>5.10.3.2.20.2</td>
<td>8.44</td>
<td>2.1</td>
<td>7.14</td>
<td>1292.18</td>
<td>22.17</td>
</tr>
<tr>
<td>5.10.3.2.20.3</td>
<td>5.03</td>
<td>2.93</td>
<td>4.19</td>
<td>11.66</td>
<td>25.78</td>
</tr>
<tr>
<td>5.10.3.2.20.4</td>
<td>8.8</td>
<td>1.74</td>
<td>2.9</td>
<td>3.37</td>
<td>98.31</td>
</tr>
<tr>
<td>5.10.3.2.20.5</td>
<td>5.2</td>
<td>1.98</td>
<td>4.81</td>
<td>4.75</td>
<td>19.73</td>
</tr>
<tr>
<td>5.10.3.2.20.6</td>
<td>8.56</td>
<td>1.95</td>
<td>2.34</td>
<td>2.43</td>
<td>3545.1</td>
</tr>
<tr>
<td>5.10.3.2.20.7</td>
<td>7.11</td>
<td>2.22</td>
<td>5.49</td>
<td>16.02</td>
<td>30.92</td>
</tr>
<tr>
<td>5.10.3.2.20.8</td>
<td>10.84</td>
<td>2.17</td>
<td>2.83</td>
<td>15.91</td>
<td>21.6</td>
</tr>
<tr>
<td>5.10.3.2.30.0</td>
<td>17.41</td>
<td>2.59</td>
<td>3.24</td>
<td>68.33</td>
<td>454.74</td>
</tr>
<tr>
<td>5.10.3.2.30.1</td>
<td>19.5</td>
<td>3.67</td>
<td>6.83</td>
<td>28.82</td>
<td>39.2</td>
</tr>
<tr>
<td>5.10.3.2.30.2</td>
<td>5.3</td>
<td>2.69</td>
<td>6.8</td>
<td>2.65</td>
<td>1420.78</td>
</tr>
<tr>
<td>5.10.3.2.30.3</td>
<td>32.59</td>
<td>3.43</td>
<td>3.78</td>
<td>33.37</td>
<td>693.45</td>
</tr>
<tr>
<td>5.10.3.2.30.4</td>
<td>8.31</td>
<td>3.81</td>
<td>6.56</td>
<td>16.95</td>
<td>514.42</td>
</tr>
<tr>
<td>5.10.3.2.30.5</td>
<td>97.13</td>
<td>3.05</td>
<td>5.47</td>
<td>19.67</td>
<td>67.03</td>
</tr>
<tr>
<td>5.10.3.2.30.6</td>
<td>33.49</td>
<td>2.63</td>
<td>3.55</td>
<td>31.89</td>
<td>497.33</td>
</tr>
<tr>
<td>5.10.3.2.30.7</td>
<td>9.88</td>
<td>1.81</td>
<td>5.75</td>
<td>9.66</td>
<td>171.58</td>
</tr>
<tr>
<td>5.10.3.2.30.8</td>
<td>35.24</td>
<td>2.59</td>
<td>6.63</td>
<td>68.98</td>
<td>65.59</td>
</tr>
</tbody>
</table>

Table 4.2: Evaluating the Control Set: Time (sec)

adding the ability to report the solution quality.

ADOPT, DCDCOP, CostDCOP, DPOP, and NCBB were employed to solve this benchmark data set. The experiments were run on a 4 CPU 1593Mhz SUNW Ultra-SPARC - III machine with 16GB of RAM. All algorithms were allowed a maximum runtime of 2 hours for solving each problem. Unsolved instances were terminated after this time. For the purpose of comparison, therefore, unsolved instances were assigned a runtime of $t = 7200\text{secs}$.

As discussed in the previous section, ADOPT and DCDCOP were implemented within the original ADOPT package. CostDCOP was also implemented in a similar manner. ADOPT failed to arrive at an optimal solution in 421 of the 480 problems available. DCDCOP solved all of the 480 problems available, while CostDCOP timed out in 70 problem instances. For evaluating the performance of DPOP on the benchmark dataset, we used the DPOP implementation within the FRODO2 API (Léauté et al., 2009). Despite having more than 14GB RAM available, DPOP
4.5. Experimental Evaluation

<table>
<thead>
<tr>
<th>Problem ID</th>
<th>ADOPT</th>
<th>DCDCOP</th>
<th>CostDCOP</th>
<th>NCBB</th>
<th>DPOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>5_10.3.2.10.0</td>
<td>521339</td>
<td>62</td>
<td>118</td>
<td>6963830</td>
<td>2905</td>
</tr>
<tr>
<td>5_10.3.2.10.1</td>
<td>24264</td>
<td>79</td>
<td>128</td>
<td>23720804</td>
<td>2903</td>
</tr>
<tr>
<td>5_10.3.2.10.2</td>
<td>28405</td>
<td>57</td>
<td>86</td>
<td>1084706</td>
<td>3050</td>
</tr>
<tr>
<td>5_10.3.2.10.3</td>
<td>3953</td>
<td>59</td>
<td>96</td>
<td>2194708</td>
<td>2907</td>
</tr>
<tr>
<td>5_10.3.2.10.4</td>
<td>883</td>
<td>61</td>
<td>106</td>
<td>13076</td>
<td>2754</td>
</tr>
<tr>
<td>5_10.3.2.10.5</td>
<td>27501</td>
<td>70</td>
<td>92</td>
<td>274762</td>
<td>2748</td>
</tr>
<tr>
<td>5_10.3.2.10.6</td>
<td>23815</td>
<td>54</td>
<td>90</td>
<td>1848366</td>
<td>2899</td>
</tr>
<tr>
<td>5_10.3.2.20.0</td>
<td>2727</td>
<td>118</td>
<td>104</td>
<td>963464</td>
<td>5494</td>
</tr>
<tr>
<td>5_10.3.2.20.1</td>
<td>517747</td>
<td>102</td>
<td>156</td>
<td>759370</td>
<td>5498</td>
</tr>
<tr>
<td>5_10.3.2.20.2</td>
<td>3112</td>
<td>94</td>
<td>140</td>
<td>5242660</td>
<td>5192</td>
</tr>
<tr>
<td>5_10.3.2.20.3</td>
<td>1555</td>
<td>84</td>
<td>92</td>
<td>69328</td>
<td>5343</td>
</tr>
<tr>
<td>5_10.3.2.20.4</td>
<td>2659</td>
<td>101</td>
<td>116</td>
<td>12678</td>
<td>5347</td>
</tr>
<tr>
<td>5_10.3.2.20.5</td>
<td>1474</td>
<td>109</td>
<td>136</td>
<td>23864</td>
<td>5496</td>
</tr>
<tr>
<td>5_10.3.2.20.6</td>
<td>2532</td>
<td>144</td>
<td>176</td>
<td>8982</td>
<td>5504</td>
</tr>
<tr>
<td>5_10.3.2.20.7</td>
<td>2110</td>
<td>136</td>
<td>200</td>
<td>62962</td>
<td>5798</td>
</tr>
<tr>
<td>5_10.3.2.20.8</td>
<td>3337</td>
<td>67</td>
<td>84</td>
<td>73552</td>
<td>5645</td>
</tr>
<tr>
<td>5_10.3.2.30.0</td>
<td>7846</td>
<td>94</td>
<td>152</td>
<td>247116</td>
<td>8546</td>
</tr>
<tr>
<td>5_10.3.2.30.1</td>
<td>8547</td>
<td>136</td>
<td>152</td>
<td>118634</td>
<td>7789</td>
</tr>
<tr>
<td>5_10.3.2.30.2</td>
<td>2254</td>
<td>85</td>
<td>150</td>
<td>10638</td>
<td>7936</td>
</tr>
<tr>
<td>5_10.3.2.30.3</td>
<td>13263</td>
<td>140</td>
<td>128</td>
<td>144850</td>
<td>8393</td>
</tr>
<tr>
<td>5_10.3.2.30.4</td>
<td>3395</td>
<td>152</td>
<td>200</td>
<td>69990</td>
<td>8246</td>
</tr>
<tr>
<td>5_10.3.2.30.5</td>
<td>41718</td>
<td>101</td>
<td>136</td>
<td>73336</td>
<td>7936</td>
</tr>
<tr>
<td>5_10.3.2.30.6</td>
<td>13949</td>
<td>86</td>
<td>140</td>
<td>168108</td>
<td>7338</td>
</tr>
<tr>
<td>5_10.3.2.30.7</td>
<td>4775</td>
<td>108</td>
<td>200</td>
<td>38662</td>
<td>8242</td>
</tr>
<tr>
<td>5_10.3.2.30.8</td>
<td>18250</td>
<td>114</td>
<td>166</td>
<td>272228</td>
<td>8244</td>
</tr>
</tbody>
</table>

Table 4.3: Evaluating the Control Set: Messages Sent

managed to solve only 46 of the 480 problems available. For NCBB, we used the original source code used in (Chechetka and Sycara, 2006a). NCBB managed to solve only 28 of the 480 problems available. Figure 4.8 presents the average time taken by each algorithm as a function of increasing problem size and Table 4.1 presents the same information grouped by link density of the problem instances.

It was thus observed that for the smallest problems in this dataset with $LD = 2$, DPOP performed better than ADOPT and NCBB, but this performance quickly deteriorated and NCBB and DPOP failed to address over 90% of the problems within the set timeframe. ADOPT performed better but, once again, managed to address only the smaller quarter of the $LD = 2$ problems in the benchmark dataset. The performance of ADOPT, NCBB, and DPOP also significantly deteriorated for problems with $LD = 3$. Both DCDCOP and CostDCOP performed well, with CostDCOP’s performance being closer to that of DCDCOP for the smallest and largest problems in the dataset. Thus, while the experiments proved the unsuit-
ability of complete algorithms in solving larger problems, they created the need for
the algorithms, especially DCDCOP and CostDCOP, to be compared using other
popular metrics, including number of messages transacted, number of cycles, and
final solution quality.

Following the above performance results, a smaller control set of 25 problems which
was solvable by all of the above algorithms was used to compare these other per-
fomance metrics. Though representing the smaller problems out of our complete
dataset, all problems in the control set represented 50 variables as opposed to much
smaller problems used to evaluate DCOP algorithms in recent comparative studies
such as those by Gershman et al. (2008) (10 Variables) and Lass et al. (2007) (12-14
Variables). Tables 4.3 and 4.4 represent the total number of messages transacted
and the total number of cycles respectively. The final solution quality was also
measured to check if all algorithms, especially DCDCOP and CostDCOP, reached
optimal solutions (Table 4.5).

<table>
<thead>
<tr>
<th>Problem ID</th>
<th>ADOPT</th>
<th>DCDCOP</th>
<th>CostDCOP</th>
<th>NCBB</th>
<th>DPOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>5_10_3_2_10.i0</td>
<td>226581</td>
<td>30</td>
<td>58</td>
<td>2324309</td>
<td>NA</td>
</tr>
<tr>
<td>5_10_3_2_10.i1</td>
<td>8611</td>
<td>45</td>
<td>70</td>
<td>6911713</td>
<td>NA</td>
</tr>
<tr>
<td>5_10_3_2_10.i2</td>
<td>26261</td>
<td>33</td>
<td>46</td>
<td>315164</td>
<td>NA</td>
</tr>
<tr>
<td>5_10_3_2_10.i3</td>
<td>1511</td>
<td>36</td>
<td>58</td>
<td>708484</td>
<td>NA</td>
</tr>
<tr>
<td>5_10_3_2_10.i4</td>
<td>325</td>
<td>32</td>
<td>64</td>
<td>3316</td>
<td>NA</td>
</tr>
<tr>
<td>5_10_3_2_10.i7</td>
<td>10470</td>
<td>48</td>
<td>50</td>
<td>71025</td>
<td>NA</td>
</tr>
<tr>
<td>5_10_3_2_10.i9</td>
<td>8139</td>
<td>27</td>
<td>52</td>
<td>617424</td>
<td>NA</td>
</tr>
<tr>
<td>5_10_3_2_20.i0</td>
<td>591</td>
<td>59</td>
<td>46</td>
<td>255033</td>
<td>NA</td>
</tr>
<tr>
<td>5_10_3_2_20.i1</td>
<td>97240</td>
<td>44</td>
<td>74</td>
<td>242575</td>
<td>NA</td>
</tr>
<tr>
<td>5_10_3_2_20.i2</td>
<td>691</td>
<td>54</td>
<td>74</td>
<td>1783407</td>
<td>NA</td>
</tr>
<tr>
<td>5_10_3_2_20.i3</td>
<td>9347</td>
<td>51</td>
<td>54</td>
<td>13615</td>
<td>NA</td>
</tr>
<tr>
<td>5_10_3_2_20.i4</td>
<td>579</td>
<td>43</td>
<td>60</td>
<td>3791</td>
<td>NA</td>
</tr>
<tr>
<td>5_10_3_2_20.i5</td>
<td>13548</td>
<td>53</td>
<td>64</td>
<td>5318</td>
<td>NA</td>
</tr>
<tr>
<td>5_10_3_2_20.i6</td>
<td>522</td>
<td>70</td>
<td>84</td>
<td>2293</td>
<td>NA</td>
</tr>
<tr>
<td>5_10_3_2_20.i7</td>
<td>469</td>
<td>57</td>
<td>106</td>
<td>20732</td>
<td>NA</td>
</tr>
<tr>
<td>5_10_3_2_20.i8</td>
<td>746</td>
<td>36</td>
<td>48</td>
<td>21443</td>
<td>NA</td>
</tr>
<tr>
<td>5_10_3_2_30.i0</td>
<td>31674</td>
<td>40</td>
<td>74</td>
<td>93192</td>
<td>NA</td>
</tr>
<tr>
<td>5_10_3_2_30.i1</td>
<td>34386</td>
<td>64</td>
<td>70</td>
<td>37987</td>
<td>NA</td>
</tr>
<tr>
<td>5_10_3_2_30.i2</td>
<td>14471</td>
<td>40</td>
<td>82</td>
<td>2830</td>
<td>NA</td>
</tr>
<tr>
<td>5_10_3_2_30.i3</td>
<td>1623</td>
<td>71</td>
<td>56</td>
<td>44509</td>
<td>NA</td>
</tr>
<tr>
<td>5_10_3_2_30.i4</td>
<td>19224</td>
<td>69</td>
<td>98</td>
<td>24030</td>
<td>NA</td>
</tr>
<tr>
<td>5_10_3_2_30.i5</td>
<td>5359</td>
<td>48</td>
<td>66</td>
<td>25502</td>
<td>NA</td>
</tr>
<tr>
<td>5_10_3_2_30.i6</td>
<td>2092</td>
<td>45</td>
<td>76</td>
<td>40860</td>
<td>NA</td>
</tr>
<tr>
<td>5_10_3_2_30.i7</td>
<td>15363</td>
<td>45</td>
<td>92</td>
<td>12270</td>
<td>NA</td>
</tr>
<tr>
<td>5_10_3_2_30.i8</td>
<td>16124</td>
<td>56</td>
<td>88</td>
<td>93782</td>
<td>NA</td>
</tr>
</tbody>
</table>

Table 4.4: Evaluating the Control Set: Cycles
4.5. Experimental Evaluation

<table>
<thead>
<tr>
<th>Problem ID</th>
<th>ADOPT</th>
<th>DCDCOP</th>
<th>CostDCOP</th>
<th>NCBB</th>
<th>DPOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>5_10_3_2_10_i0</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5_10_3_2_10_i1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5_10_3_2_10_i2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5_10_3_2_10_i3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5_10_3_2_10_i4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5_10_3_2_10_i7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5_10_3_2_10_i9</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5_10_3_2_20_i0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5_10_3_2_20_i1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5_10_3_2_20_i2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5_10_3_2_20_i3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5_10_3_2_20_i4</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5_10_3_2_20_i5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5_10_3_2_20_i6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5_10_3_2_20_i7</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5_10_3_2_20_i8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5_10_3_2_30_i0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5_10_3_2_30_i1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5_10_3_2_30_i2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5_10_3_2_30_i3</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5_10_3_2_30_i4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5_10_3_2_30_i5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5_10_3_2_30_i6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5_10_3_2_30_i7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5_10_3_2_30_i8</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4.5: Evaluating the Control Set: Final Solution Quality

It was observed that DCDCOP again significantly outperformed ADOPT, DPOP, and NCBB. And while the performance of DCDCOP and CostDCOP was again somewhat comparable, looking at the solution quality revealed that CostDCOP failed to reach optimal solution in 8 of the 25 test instances. However, while it also trades off completeness for practical efficiency, DCDCOP reported optimal solution quality for all problems in our control set. The evaluation proves the effectiveness of the $DU$ metric in converging to an optimal solution. It is hypothesized that appropriately weighted inter-agent constraints would help all agents converge to a globally optimum solution. This is quite consistent with the rules represented by such constraints in the real world as inter-agent constraints are generally less flexible and would thus carry a higher cost.
4. An Efficient Algorithm for Solving Dynamic Complex DCOP Problems

4.6 Summary

Several real world optimization problems translate to agents with complex sub-problems in a dynamic environment, that need to negotiate in a manner where departmental privacy and decision making authority is preserved. The state-of-the-art in DCOP deals with such problems by offering extensions to algorithms best suited for optimization of single-variable agents in static environments and this often lead to sub-optimality. We present a flexible and robust algorithm, DCDCOP, that is capable of exploiting the inherent domain centralization found in such problems, and uses a novel measure, $DU$, to dynamically guide agent ordering during optimization. Experimental evaluation shows that DCDCOP significantly outperforms the state-of-the-art asynchronous DCOP algorithms. Comparison with a DSA-like variant, CostDCOP, also proves the effectiveness of the $DU$ metric in guiding the algorithm towards an optimal solution.

We thus observe how problems that cannot be efficiently solved by DCOP algorithms can benefit greatly by applying a flexible and robust algorithm like DCDCOP, that can help model the departmental structure of large real world optimization problems, offering not only a natural mapping from real world problem solving structure to DCOP problem solving structure, but also exploiting this to provide an order of magnitude improvement in performance.
<table>
<thead>
<tr>
<th>Num Agents</th>
<th>Num Variables</th>
<th>Link Density</th>
<th>% Inter</th>
<th>Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>10</td>
<td>2</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>2</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>2</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>3</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>3</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>3</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>2</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>2</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>2</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>3</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>3</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>3</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>2</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>2</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>2</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>3</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>3</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>3</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>2</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>2</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>2</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>3</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>3</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>3</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>2</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>2</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>2</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>3</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>3</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>3</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>2</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>2</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>2</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>3</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>3</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>3</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>40</td>
<td>10</td>
<td>2</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>40</td>
<td>10</td>
<td>2</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>40</td>
<td>10</td>
<td>2</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>40</td>
<td>10</td>
<td>3</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>40</td>
<td>10</td>
<td>3</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>40</td>
<td>10</td>
<td>3</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>40</td>
<td>20</td>
<td>2</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>40</td>
<td>20</td>
<td>2</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>40</td>
<td>20</td>
<td>2</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>40</td>
<td>20</td>
<td>3</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>40</td>
<td>20</td>
<td>3</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>40</td>
<td>20</td>
<td>3</td>
<td>30</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 4.6: The Benchmark Dataset
4. An Efficient Algorithm for Solving Dynamic Complex DCOP Problems
Chapter 5

ASES: an Automated Scheduler for Elective Surgery

“The scheduling of surgery has been described as a complex activity, a perpetually difficult problem due to an ever-changing environment, and even as a political battle”

Fitzgerald et al. (2006)

In this Chapter, we present ASES, an Automated Scheduler for Elective Surgery. Developed as a proof-of-concept demonstration of ideas discussed in chapters 3 and 4, ASES is a Multi-Agent System designed to reflect and complement the existing manual methods of elective surgery scheduling, while offering efficient mechanisms for negotiation and schedule optimization. As discussed previously, the use of the multi-agent paradigm is a natural fit given the distributed nature of the problem. It also captures the autonomy of the hospital departments in constructing and managing their individual schedules. In order to ensure the optimality and compatibility of the departmental schedules, we employ distributed constraint optimization to guide coordination and resolution of schedule conflicts. This marriage of rational agency and distributed constraint optimization is novel and necessitated by the problem domain.

The rest of this Chapter is organized as follows. We start with discussing the mapping of the Elective Surgery Scheduling Problem to the DCOP notation, and the choice of DCDCOP to drive the DCOP engine. We then present particulars of the implementation of the ASES system and discuss the feasibility and benefits of our approach. We conclude with a description of ongoing development.
5. ASES: an Automated Scheduler for Elective Surgery

5.1 Domain Mapping

The Elective Surgery Scheduling Problem (ESSP) presented in Section 3.3 can be viewed as a set of departmental scheduling problems. Each department allocating staff or other resources to the surgery prepares their own departmental schedule and then negotiates with other departments to ensure that their departmental schedules are aligned and the resulting Operating Theatre schedule is conflict free. To offer an efficient solution to the problem, we propose an agent-oriented methodology (Section 3.6) where each department is represented by an intelligent agent that is assigned the task of managing its department’s schedule (Fig. 5.1).

To map the ESSP to a multi-agent DCOP notation (Fig. 5.2), we assign each departmental scheduling problem to a single agent. The schedule slots are mapped to variables, and the staff and resources to be scheduled form the domain of values for the variables. Constraints between variables of the same agent represent conditions such as not being able to schedule a staff to two slots that run in parallel, while constraints between variables belonging to different agents represent conditions such as doctor-nurse team preference allocations. Domain rules and preferences are used to define cost functions for individual constraints.

An optimal solution to the resultant DCOP problem will now lead to an optimal elective surgery schedule.
5.1. Domain Mapping

Figure 5.2: Mapping the Problem
5.2 The DCOP Engine

The DCOP algorithm that we utilize needs to be robust in a number of ways. It must be scalable to the variety and complexity of the involved agents’ sub-problems. Negotiation resolution must be timely with respect to the environment under which the negotiation is taking place. The ability to separate the communication protocol from the details of the local solver is also essential as this facilitates the customization of the local solver to each agent’s unique problem while maintaining communication compatibility.

In Chapter 4, we presented the DCDCOP algorithm where agents solve their local sub-problem using a local solver of their choice and then employ a novel metric called Degree of Unsatisfaction to guide inter-agent negotiation and solve inter-agent constraints. DCDCOP was shown to outperform the state-of-the-art asynchronous DCOP algorithms by an order of magnitude. We thus choose DCDCOP to drive the DCOP engine in ASES. It is proposed to implement other key algorithms like ADOPT within ASES at a later stage to empirically validate our choice.

5.3 Implementation

ASES has been implemented using Jason (Bordini et al., 2007). Jason is a Java implementation of Agentspeak(L) (Rao, 1996). In addition to providing extended Agentspeak(L) syntax and semantics for the development of individual agents, Jason provides facilities for the specification of multi-agent systems. Crucial in so doing is the provision for speech-act-based communication. This speech-act-based communication underlies our DCOP communication implementation. Figure 5.6 illustrates the use of Agentspeak(L) within the Nursing agent.

ASES models the scheduling activity of 4 agents: Bookings, Nursing, Anaesthesiology, and Theatre Resources. Each agent is briefly discussed to present a better understanding of their activities.

The Bookings agent (Fig. 5.3) receives randomly generated requests to add or modify bookings. Each request includes the patient and procedure information. When a slot is allocated, the Bookings agent sends this information out to all agents concerned. If an agent is unable to provide resources, a message is returned to the Bookings agent, resulting in the allocation being cancelled and another message being sent out to all agents concerned.

The Resource agent (Fig. 5.4) calculates the equipment required for the procedure to schedule. If the required resources are unavailable, the Resource agent requests
that the Bookings agent reschedule the procedure. Thus, equipment is allocated on a first-come first-served basis. This models the hospital’s current resource allocation strategy. However, work is underway to enhance this process to utilize procedure/patient priorities required.

The Nursing agent (Fig. 5.5), receiving notification of a new procedure allocation, must then schedule the nursing staff to accommodate the new allocation. Unlike the Resource agent, the resources available to the Nursing agent are not fixed. The Nurse Unit Manager is able to hire casual/temporary nurses when necessary. However, their use is to be minimized. This is modeled by assigning a higher cost to casual/temporary nursing staff.

In managing the nursing schedule, the Nursing agent is required to ensure that for each assignment of nurse to procedure, the nurse contributes a skill necessary to the completion of the procedure. No more nurses than necessary should be assigned to a procedure. Each procedure must have its nursing skills requirements met. Should the nursing agent be unable to allocate nurses to satisfy a procedure’s requirements, a request is sent to the bookings agent to reschedule the procedure. Additional constraints representing preference, breaks, shifts, and working regulation also apply to the nursing schedule.
The Nursing agent also needs to match the allocation of nurses to procedures with other staffing agents such as Anaesthesiology. Such negotiations are often necessary to maximize the compatibility and efficiency of the operating team, and also help maintain staff morale. This is modeled using inter-agent constraints carrying appropriately high cost. An optimal solution would thus ensure that these constraints were satisfied, even if at was reached at the cost of hiring additional casual staff.

The responsibilities of the Anaesthesiology agent largely mimic those of the Nursing agent. The differences lie in the requirements of procedures, preferences, and number of staff to be assigned, use of temporary staff, and award and training requirements of the department.

Finally, all agents are able to incrementally adjust and optimize the schedules based on changing circumstances. Should a procedure be rescheduled, all schedules must reflect this in a timely manner. As scheduled procedures draw near to execution, additional constraints can be imposed to increase the stability of the approaching procedures. This would reflect the difficulty of successfully accommodating last minute changes.

However, at no point prior to the scheduled time of a procedure can a procedure booking be deemed confirmed. Emergency cases must be accommodated. Should
5.4. Evaluation

![Nursing Agent](image)

Figure 5.5: Nursing Agent

theatres, staff, or resources be required by such emergencies, the system must be capable of adjusting to these last minute needs.

In many scenarios, the system needs user-input to make a decision about a negotiation request received. For example, if a slot opening permits a procedure to be brought forward, the Bookings department may request such a change. However, the Nurse Unit Manager may accommodate the change at short notice only at her own discretion, or after explicit discussion with the staff involved. In such situations, there is no alternative to deferring the decision to the user. We are currently working on implementing an Intelligence Module within ASES that provides decision support. This module is based on the system discussed in Section 3.7.2 and is designed to mimic the behaviour of the domain expert in these scenarios and to build a knowledge bank by learning from decisions taken by the domain expert.

5.4 Evaluation

Since current hospital processes do not capture transient scheduling information, real-world data could not be used to drive the ASES system. We generated statistically significant random test data to evaluate the feasibility of our approach. The number of theatre slots, average procedure time, and number of staff per depart-
Figure 5.6: AgentSpeak(L) for the Nursing Agent

ment were selected based on data collected from interviews with domain experts and the tools currently in the hospitals employ. However, we did make some simplifying assumptions. We did not model all of the constraints we identified as crucial. This was due to the immaturity of the system and not to any technical difficulty. Further, given the absence of suitable comparison benchmarks, the efficiency of the DCDCOP algorithm was not specifically evaluated within the system.

As procedures were booked, the information flowed in real time to other agents, who updated their schedule accordingly. Conflicts were identified, and negotiation was initiated to resolve them. Similarly, cancellations resulted in resources being freed up and made available instantly. The system thus dramatically cut down any inefficiencies caused by delays in current communication and negotiation procedures. With all resources and staff available, ASES reported resolving an average of 226 conflicts at 70% theatre utilization and an average of 325 conflicts at 100% theatre utilization. When available resources were reduced by 10% (to simulate situations where equipment was unavailable), the number of conflicts increased to 384 at 70% utilization and ASES managed to achieve only a maximum of 93% theatre utilization (Fig. 5.7).

In automating the scheduling process thus, ASES significantly reduces delays in inter-departmental information flow and negotiation. The ability to automatically
generate optimal departmental schedules also offers a saving of several hours of man-
ual work that currently goes into preparing the schedules. For example, the Nurse
Unit Manager currently spends an average of 50 hours a month creating the follow-
ing month’s schedule and an average of 2 hours a day handling the rescheduling.
Though delays resulting from waiting for user interaction are unavoidable, the need
for such interaction will also decrease as the system learns and builds its knowledge
bank for automated decision support. Further, as the departmental schedules are
always maintained conflict free, ASES altogether does away with the need for weekly
meetings.

Another key enhancement offered by ASES revolves around the efficient manage-
ment of operating theatre resources. In the current manual system, procedures are
scheduled without foreknowledge of the availability of the required resources, often
resulting in a compromised scheduling. This is corroborated in the current evalu-
ation as we observe that unavailability of resources can quickly lead to poor the-
atre utilization. The knowledge generated by integrating resource management and
scheduling within the ASES system can allow sufficient time to overcome resource
shortages and improve theatre utilization.
5. ASES: an Automated Scheduler for Elective Surgery

5.5 Summary

We have presented ASES, an Automated Scheduler for Elective Surgery. ASES models the challenging Elective Surgery Scheduling Problem using the multi-agent system paradigm, and is powered by a DCOP engine capable of handling the complex and dynamic nature of the problem. Through this novel integration of multi-agent modeling and state-of-the-art artificial intelligence techniques, ASES represents a significant advance towards solving this particularly challenging class of complex distributed dynamic problems. Our preliminary evaluation of the system shows that automated scheduling using ASES offers real-world efficiency improvements.

We are currently implementing intelligent decision support and learning within ASES. This module would gradually learn to mimic the domain expert’s decision making process and help overcome delays caused by the unavailability of the domain experts. We are also implementing other DCOP algorithms within ASES to aid empirical evaluation of DCDCOP’s performance within the system.

Several challenges need to be addressed before ASES can be deployed in hospital. Firstly, much of the knowledge utilized to generate current departmental schedules is informal and undocumented. Creating domain rules that could be used to define and quantify constraint cost functions is a non trivial task. Achieving this milestone, however, would also serve the purpose of streamlining current scheduling processes. Secondly, quantifying confidence scores and managing dynamically changing priorities also poses a challenge for intelligent decision support. Manual curation of the schedules, and the system’s ability to learn from this process, however, provides a mechanism for assisting with the latter. Lastly, gaining acceptance from the end-users of the system is critical, and we are working closely with these practitioners to ensure that the system optimally serves their scheduling needs.
Chapter 6

Conclusion

“There will come a time when you believe everything is finished. Yet that will be the beginning”

Louis L’Amour

In this Chapter, we summarize the main research contributions presented in this thesis. We then identify potential future research questions and discuss ongoing work.

6.1 Summary of Contributions

Motivated by the challenge of better managing the task of scheduling elective surgery in public hospitals, this thesis has aimed to advance the state-of-the-art in modeling and solving complex distributed optimization problems in dynamic real world environments. After detailed investigation into the complexities imposed by the problem domain and the inadequacies of current systems in dealing with these complexities, we have developed a methodology where intelligent agents, trained with the constraints, preferences, and priorities of the administrators, optimize schedules for their respective departments and then negotiate in a privacy-preserving manner to resolve inter-agent constraints. The underlying DCOP engine in our approach is driven by a novel asynchronous DCOP algorithm, the Dynamic Complex Distributed Constraint Optimization Problem Algorithm (DCDCOP), that preserves the decentralized decision control mechanisms of the problem at hand and offers a robust, flexible, and efficient mechanism for modeling and solving dynamic complex problems. We have experimentally evaluated our DCDCOP algorithm against the state-of-the-art asynchronous DCOP algorithms. We have also developed a proto-
type proof-of-concept application that demonstrates the efficacy of our approach.

In Chapter 3 we present a detailed case study of elective surgery scheduling at a leading Australian hospital. A result of several months of attending the scheduling meeting and interviews with domain experts involved in the process, the case study offers invaluable insight into process intricacies and helps identify inter-departmental negotiation and schedule optimization as the key areas needing improvement. A review of the state-of-the-art in research seeking to address scheduling in the health domain also returns similar finding and establishes the need for a flexible and intelligent methodology that provides easy and effective modeling of the inherent distribution of resources and decision making, while allowing easy integration of sophisticated problem solving and negotiation strategies. Based on these finding, we propose an agent oriented methodology driven by an efficient DCOP solver to solve the problem at hand.

In Chapter 4 we extend previous work by Zhou et al. (2003) to define a new measure of Degree of Unsatisfaction ($DU$) as applied to DCOP problems and use this to guide inter-agent negotiation in our novel asynchronous DCDCOP algorithm. The $DU$ metric and the DCDCOP algorithm are explained with examples. A proof of soundness for the DCDCOP algorithm is also presented. A DSA-like variant of the DCDCOP algorithm, called CostDCOP, is introduced to assist the evaluation of the effect of the $DU$ metric on convergence and final solution quality. The experimental evaluation is discussed in detail. Our findings here indicate that DCDCOP outperforms other asynchronous state-of-the-art DCOP algorithms by more than an order of magnitude. The efficacy of the $DU$ metric is also empirically validated.

Finally, in Chapter 5 we present a proof-of-concept implementation to demonstrate our proposed methodology. Implemented in Jason (Bordini et al., 2007), the Automated Scheduler for Elective Surgery (ASES) system provides ongoing negotiation for schedule optimization as the bookings department adds/edits procedures to the elective surgery schedule. It also demonstrates the effect of fluctuation in staffing or resource levels on theatre utilization. We also discuss ongoing development of the ASES system.

While the elective surgery scheduling problem was used as the real world example for developing our methodology, the proposed methodology can be applied to any distributed problem domain. It is especially well suited to domains where distributed entities may need to work, cooperatively or competitively, to achieve their own goals while also working towards larger organizational or cross-organizational goals.
6.2 Future Work

While this thesis concludes here, knowledge gained as part of this study presents several research challenges that offer rich opportunities for future research. These can broadly be categorized as follows:

- **Knowledge Representation**: Given the inherent complexities of the health domain, much of the knowledge utilized to generate current departmental schedules is informal and undocumented. Realizing inter-departmental relationships and domain rules and using these to define and quantify constraints and their related cost functions poses a significant challenge. Achieving this milestone, however, would serve the purpose of streamlining current scheduling processes and also provide the rule base for automated decision support. Incorporating historical performance and utilization data in the scheduling process, and investigating the effect of this on the robustness of the final schedule is also an important research challenge.

- **Distributed Optimization**: Our research into efficient algorithms for solving complex dynamic DCOPs has opened several avenues for future research. While DCDCOP is an efficient algorithm, there is scope for further improvement. Exploring heuristics for accelerating search, using more efficient local solvers than branch and bound, and exploring alternate metrics for guiding agent ordering are some of the ways in which the performance of this algorithm could be improved. Investigating variants that may offer quality guarantees under specific conditions and applying Multicriteria Optimization to tradeoff additional, possibly conflicting, constraints also pose significant research challenges. In addition, it would be interesting to explore integration of Bayesian probabilistic frameworks (Pearl, 1988) into the scheduling process to help model the uncertainty and develop more robust planning and scheduling tools.

- **Decision Support**: Quantifying confidence scores and managing dynamically changing priorities pose a challenge for intelligent decision support. Manual curation of the schedules, and the system’s ability to learn from this process, however, provides a mechanism for assisting with the latter.

We are currently working on further development of the ASES prototype, specifically on incorporating intelligent decision support within the agents. We are also working on developing the DCOP engine to incorporate complex inter-agent constraints and input from the intelligence module into the DCDCOP algorithm. Using input from
the Patient Admissions Prediction Tool (Boyle et al., 2008b,a), we also plan to model
dynamic demand-governed allocation of Operating Theatres for emergency surgery
within the system. Implementing DCDCOP and other DCOP algorithms within
ASSES to evaluate them on benchmark scheduling problems is also proposed. Mah-
eswaran et al. (2004b) show that the performance of ADOPT in solving real world
problems is significantly worse than in solving similar-sized map coloring problems.
Benchmarking the algorithms within ASES would thus provide better insight on
their relative performances. Incorporating domain expert interaction and override
functionality are also future aims for the ASES system.

We are also working towards employing various local solver algorithms within DCD-
COP to produce a family of algorithms that can adapt themselves to the nature of
the problem at hand. For example, agents with larger sub-problems could employ
local search to perform faster, though incomplete, search within the node, while
agents with smaller sub problems could benefit with using faster complete search
within the node. Investigating the interaction between agents using different local
solvers and the efficacy of the $DU$ metric in this scenario is also to be investigated.

Translating domain rules to constraints, quantifying cost functions to represent these
constraints, incorporating multi-criteria optimization, and quantifying confidence
scores for intelligent decision support are also key research challenges that must be
addressed before our research can actually be fully deployed in a real hospital envi-
ronment. Lastly, gaining acceptance from the end-users of the system is a challenge
for all new systems but we expect the time already spent working closely with these
practitioners, and studying their requirements, will help bridge this gap.


Pauline Berry. Personal communication, February 2006.


D. A. Burke and K. N. Brown. A comparison of approaches to handling complex local problems in DCOP. In Sixth International Workshop on Distributed Constraint Reasoning, page 27–33, Italy, 2006.


Sankalp Khanna, Abdul Sattar, Anthony Maeder, Peter Moran, and Bela Stantic. Intelligent management of elective surgery schedules. Sixth Annual Health and Medical Research Conference of Queensland, Brisbane, Australia, November 2006a.

Sankalp Khanna, Abdul Sattar, Bela Stantic, and Anthony Maeder. Intelligent scheduling in a distributed domain. CSIRO e-Health Research Centre Colloquium, Brisbane, Australia, March 2006b.


Sankalp Khanna, Abdul Sattar, David Hansen, and Bela Stantic. An Efficient Algorithm for Solving Dynamic Complex DCOP Problems. In *WI-IAT ’09: Pro-


BIBLIOGRAPHY


BIBLIOGRAPHY


APPENDIX A
Related Publications

• Sankalp Khanna, Abdul Sattar, Bela Stantic, and Anthony Maeder. Intelligent scheduling in a distributed domain. CSIRO e-Health Research Centre Colloquium, Brisbane, Australia, March 2006b

• Sankalp Khanna, Abdul Sattar, Anthony Maeder, Peter Moran, and Bela Stantic. Intelligent management of elective surgery schedules. Sixth Annual Health and Medical Research Conference of Queensland, Brisbane, Australia, November 2006a


• Sankalp Khanna, Abdul Sattar, Anthony Maeder, Peter Moran, and Bela Stantic. Intelligent Management of Elective Surgery Schedules. CSIRO e-Health Research Centre Colloquium, Brisbane, Australia, March 2007a

• Sankalp Khanna, Abdul Sattar, Anthony Maeder, and Bela Stantic. Intelligent Scheduling of Elective Surgery. Fourth Annual CSIRO ICT Centre Science and Engineering Conference, Sydney, Australia, November 2007c (Winner - Best Student Paper/Poster Award)

• Sankalp Khanna, Abdul Sattar, David Hansen, and Bela Stantic. A New Dynamic Measure for Solving Real World DCOP Problems. Fifth Annual CSIRO ICT Centre Science and Engineering Conference, Sydney, Australia, November 2008
• Sankalp Khanna, Abdul Sattar, David Hansen, and Bela Stantic. DCDCOP: An Efficient Algorithm for Solving Dynamic Complex DCOP Problems. Sixth Annual CSIRO ICT Centre Science and Engineering Conference, Sydney, Australia, November 2009a


• Sankalp Khanna, Tim Cleaver, Abdul Sattar, David Hansen, and Bela Stantic. Multiagent Based Scheduling of Elective Surgery. The 13th International Conference on Principles and Practice of Multi-Agent Systems (PRIMA 2010), Kolkata, India, November 2010