BDI protocol and strategies : a unified agent negotiation framework for distributed constraint satisfaction

Le Dinh Bao Chau

2007

Le, D. B. C. (2007). BDI protocol and strategies : a unified agent negotiation framework for distributed constraint satisfaction. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/35729>

<https://doi.org/10.32657/10356/35729>

BDI Protocol and Strategies: A Unified Agent Negotiation Framework for Distributed Constraint Satisfaction

Le Dinh Bao Chau

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in fulfilment of the requirement for the degree of
Master of Engineering

2007

Acknowledgements

I would like to express my great thank to the following people without whom the document would have not been completed.

First and foremost, I thank my supervisor - Professor Seow Kiam Tian. With his endless guidance and advices, I have gained precious knowledge and experiences which are not only limited in the research area. He is also the co-author of my first published research paper. Words cannot express my gratitude to his effort and kindness.

I have been lucky to have great friends who have helped me a lot in my research. Le Minh Nghia, Duong Hoang Linh and Pham Huy Phuong have been giving valuable comments in both contents and presentations of the work.

Last but not least, I deeply thank my parents for always encouraging me and giving me support throughout the work.

Table of Contents

List of Figures 5

List of Tables 7

Chapter 1 Introduction 10

 1.1 Motivation 11

 1.2 Approach 11

 1.3 Contributions 12

 1.4 The Organization of the Thesis 13

Chapter 2 Literature Survey 15

 2.1 Distributed Constraint Satisfaction Problem (DCSP) 15

 2.1.1 Some basic definitions 15

 2.1.2 Problem formalization 16

 2.1.3 The need for a distributed model 17

 2.1.4 Communication model 18

 2.2 Classification of Existing DCSP Algorithms 18

 2.2.1 The Asynchronous Backtracking algorithm (ABT) 19

 2.2.2 The Asynchronous Weak Commitment Search algorithm (AWC) 21

 2.2.3 The Distributed Breakout algorithm (DBO) 22

 2.2.4 The Asynchronous Partial Overlay algorithm (APO) 23

2.3	Other DCSP Issues	24
2.3.1	Extension to multi-variable agents	24
2.3.2	Coordination, security and privacy	25
2.4	Automated Negotiation	26
2.5	The Concept of Belief-Desire-Intention (BDI)	28
2.5.1	Introduction	28
2.5.2	Existing BDI architectures	30
Chapter 3 The BDI Negotiation Framework		33
3.1	DCSP Agent Model	33
3.2	The BDI Negotiation Model	34
3.2.1	The generic protocol	35
3.2.2	Strategy formulation	40
3.3	DCSP Algorithms as Different Strategies on a Common BDI Protocol	41
3.3.1	The Asynchronous Backtracking strategy (ABT)	42
3.3.2	The Asynchronous Weak Commitment Search strategy (AWC)	45
3.3.3	The Distributed Breakout strategy (DBO)	48
3.3.4	The Asynchronous Partial Overlay strategy (APO) . .	50
Chapter 4 The Unsolicited Mutual Advice Strategy (UMA)		55
4.1	The Basic Idea	55
4.2	The Strategy Details	57
4.3	The Completeness Property	65
4.4	An Example	66
4.5	Quick comparisons among DCSP algorithms	70

Chapter 5 Performance Evaluation and Discussions 71

5.1 DCSP Simulations 71

5.2 Performance Evaluation 75

5.2.1 Evaluation with n -queens problem 76

5.2.2 Evaluation with graph coloring problem 77

5.3 Discussions 78

5.3.1 Comparison with ABT and AWC 78

5.3.2 Comparison with DBO 80

Chapter 6 Conclusion & Future Work 84

6.1 Conclusion 84

6.2 Future Work 85

Author's Publications 87

List of Figures

- 2.1 The basic BDI architecture 31
- 3.1 The BDI interaction protocol 36
- 3.2 BDI protocol with *Asynchronous Backtracking* strategy 42
- 3.3 BDI protocol with *Asynchronous Weak-Commitment* strategy 46
- 3.4 BDI protocol with *Distributed Breakout* strategy 49
- 3.5 BDI protocol with *Asynchronous Partial Overlay* strategy . . 51
- 4.1 Stronger voluntary decision due to the unsolicited advices . . . 58
- 4.2 Reluctant decision due to the unsolicited advices 59
- 4.3 Mediating two voluntary decisions 60
- 4.4 BDI protocol with *Unsolicited Mutual Advice* strategy 61
- 4.5 Example problem 66
- 4.6 The graph after round 1 68
- 4.7 The solution obtained 69
- 5.1 The implementation model of an agent used in the simulator . 72
- 5.2 The implementation prototype of Reader in the simulator . . . 74
- 5.3 Relationship between execution time and problem size 76
- 5.4 Comparison between AWC, DBO and UMA (sparse graph coloring) 78

5.5 Comparison between AWC, DBO and UMA (critical graph coloring) 79

5.6 Comparison between AWC, DBO and UMA (dense graph coloring) 80

List of Tables

4.1 Comparison between DCSP algorithms 70

5.1 Comparison in performance of three algorithms in Distributed
 n -queens 76

5.2 Comparison in performance in Sparse Graph Coloring Problem 77

5.3 Comparison in performance in Critical Graph Coloring Problem 78

5.4 Comparison in performance in Dense Graph Coloring Problem 79

Abstract

This thesis addresses the distributed constraint satisfaction problem (DCSP). It essentially involves finding a consistent combination of actions that satisfies the constraints among multiple agents in a shared environment, and is a fundamental problem that can formalize various applications in Distributed AI, such as distributed resource allocation, distributed task allocation and distributed scheduling.

Prominent among existing DCSP algorithms are Asynchronous Backtracking (ABT), Asynchronous Weak Commitment Search (AWC), Distributed Breakout (DBO) and Asynchronous Partial Overlay (APO). The thesis addresses DCSP in a new intellectual framework proposed based on automated negotiation among Belief-Desire-Intention (BDI) agents. A BDI agent reasons with its mental attitudes of belief (B), desire (D) and intention (I), representing, respectively, the information, motivational and deliberative states of the agent. Within the proposed framework, negotiation is viewed as a process of several agents searching for a solution called an agreement; and the search can be realized via a negotiation mechanism (or algorithm), by which the agents follow a high level protocol prescribing the rules of interactions, using a set of strategies devised to select their own preferences during negotiation.

More specifically, the thesis makes the following contributions:

- First, the DCSP search of all the well-known DCSP mechanisms men-

tioned above are shown to be formalized as multiple agents following the same BDI interaction protocol, but using different action selection strategies to reach their agreements.

- Second, a new BDI protocol-driven strategy called Unsolicited Mutual Advice (UMA) is proposed and formulated. A performance evaluation shows that UMA can outperform ABT and AWC in terms of the average number of computational cycles for both the sparse and critical coloring problems. A proof of the completeness property for the new strategy is also established. Importantly, this strategy development demonstrates that the proposed framework provides not only a clearer understanding of existing DCSP algorithms from a unified BDI agent perspective, but also opens up the opportunities to extend and develop new strategies for DCSP.

Finally, possible directions for future research and development are suggested.

Chapter 1

Introduction

Rapid advancements in information and communication technologies are providing a new infrastructural and communications basis, and this has opened up new challenges to developing complex systems more effectively. These systems could offer a richer variety of new or improved services, such as in telecommunication and electronic commerce. The field of multiagent systems defines a research framework to tackle these challenges, by viewing a system as an environment of distributed interacting agents - entities capable of flexible autonomous actions [33, Ch. 1].

In developing effective multiagent systems, one of the key challenges is how multiple agents could coordinate their activities, i.e., manage, among themselves, the constraints among their possible actions in a shared environment, in order to achieve a common goal. One multiagent goal is distributed constraint satisfaction [38], which has become a flourishing research area in multiagent systems. Essentially, the goal involves finding a consistent combination of actions that satisfies given inter-agent constraints, and is a fundamental problem that can formalize various application problems in Distributed AI, such as distributed resource allocation [22], distributed scheduling [29], distributed interpretation [16] and simulation [27].

This thesis addresses the distributed constraint satisfaction problem (DCSP) in a new intellectual framework, proposed based on automated negotiation [26] among agents using some Belief-Desire-Intention model [25, 5] of rea-

1.1 Motivation

soning. In this chapter, the motivation and approach of this research are presented. The contributions of this thesis are then summarized, followed by the organization for the rest of the thesis.

1.1 Motivation

Many important DCSP algorithms, such as Distributed Breakout (DBO) [39], Asynchronous Backtracking (ABT) [38], Asynchronous Partial Overlay (APO) [17] and Asynchronous Weak Commitment (AWC) [35], have been developed to address DCSP and provide the agent solution basis for its applications. While there has been no lack of efforts in this promising research field, especially in dealing with outstanding issues such as resource restrictions (e.g., limits on time and communication) [18] and privacy requirements [41], there was hitherto no well-established conceptual framework to provide model-theoretic insights into the workings of the various DCSP algorithms. It is useful to have a conceptual framework that can model DCSP algorithms and provide a clearer understanding of their reasoning processes from an agent model-theoretic perspective. In our opinion, this could motivate further research and development of new agent algorithms for DCSP.

1.2 Approach

In order to resolve the conflicts caused by the violated constraints, the DCSP agents need to interact with each other. In such interactions, information can be exchanged and taken into consideration by the agents. Subsequently, decisions are made locally by each individual agent, and a new set of violated constraints results from the changes due to these decisions. The new conflicts

1.3 Contributions

are then observed by the agents and a new round of interactions begins to resolve them. The process continues until there are no violated constraints. We can view this as agents *automatically negotiating* among themselves until they arrive at an *agreement* representing a solution to the problem. In automated negotiation [26], agents negotiate by following a certain set of common rules. This set of rules constitutes an *interaction protocol*. Given a protocol, agents can adopt different *strategies* which define the different decisions in the interaction. An interaction protocol and a strategy form a *negotiation mechanism* by which an agent decides to reach an agreement. To this end, it is apparent that a DCSP algorithm is in fact a negotiation mechanism.

To elaborate, our studies on existing DCSP algorithms show that these agent algorithms are quite similar from a model-theoretic perspective, differing only in the details of the discrete steps and the degree of synchronization between these steps. More specifically, it can be shown that existing DCSP algorithms are **negotiation mechanisms which share the same protocol but use different strategies**. And this common negotiation protocol can be prescribed using the agent-theoretic concept of Belief-Desire-Intention (BDI) [5]. In essence, the resulting *BDI negotiation framework* proposed can be said to unify the family of DCSP algorithms.

1.3 Contributions

The thesis makes several contributions to Distributed Constraint Satisfaction [38] in the field of multiagent systems. The contributions are listed below.

- A BDI negotiation framework which is able to unify existing DCSP algorithms is proposed. The framework formulates these algorithms as negotiation mechanisms each of which is formed by two components:

1.4 The Organization of the Thesis

a common protocol and a strategy. The shared protocol is established using the concept of Belief-Desire-Intention which views an agent as having certain *mental attitudes* of belief, desire and intention representing, respectively, the information, motivation and decisive stance of the agent at an arbitrary moment.

- The proposed framework not only unifies the family of DCSP algorithms to provide a clearer understanding, but also opens up the opportunities to extend existing strategies and develop new ones for DCSP. As an attempt to demonstrate this, the thesis presents a new strategy called *Unsolicited Mutual Advice* (UMA). The new strategy was evaluated to be more efficient than the ones used in ABT and AWC in terms of the average number of computational cycles for both the sparse and critical coloring problems [31]. Importantly, the new strategy also possesses the completeness property which will be shown in this thesis.

1.4 The Organization of the Thesis

Besides this introduction, the thesis contains five other chapters:

Chapter 2 provides a formal review of DCSP and the concept of BDI.

Chapter 3 presents a BDI negotiation model by which a DCSP agent reasons. In this chapter, the existing algorithms ABT, AWC, DBO and APO are shown to be as different negotiation strategies formalized on a common BDI interaction protocol.

Chapter 4 introduces a new strategy called *Unsolicited Mutual Advice* (UMA). A proof of the completeness property for UMA and one illustrative example are included in this chapter.

1.4 The Organization of the Thesis

Chapter 5 evaluates the new UMA strategy together with existing strategies ABT, AWC and DBO. Discussions which attempt to highlight the merits of the new strategy over the existing ones are provided.

Chapter 6 concludes the paper and points to some future work.

Chapter 2

Literature Survey

Automated negotiation, BDI modelling and DCSP have been active research areas in the Multiagent Systems field. This chapter presents a review of the literature most relevant to our research.

The review includes the background and foundation of DCSP, on which the unified framework for DCSP algorithms using a BDI protocol is developed.

In this chapter, the definition and formalization of DCSP are first presented, followed by a technical review of existing DCSP algorithms. An overview of automated negotiation is then presented. Finally, the generic BDI reasoning process and related BDI architectures are described.

2.1 Distributed Constraint Satisfaction Problem (DCSP)

2.1.1 Some basic definitions

There are three basic components in a DCSP [38]:

- Variable: a variable can be a property of an agent or a sequential order of actions that agent can follow. Constraints are applied on agents through their variables and the agents have to adjust the values of their variables to satisfy the constraints.

2.1 Distributed Constraint Satisfaction Problem (DCSP)

- Domain: each variable has a specific domain which contains all the possible values that can be assigned to the variable.
- Constraint: a relationship between variables. A constraint is a logic expression with multiple variables inside it. This expression will return true if the current values of the variables satisfy the relationship.

A constraint may consist of different variables belonging to different agents. An agent cannot change or modify the assignment values of other agents' variables. Therefore, in cooperatively searching for a DCSP solution, the agents would need to communicate with one another, and adjust and re-adjust their own variable assignments in the process. In DCSP, an agent may have more than one variable. So we can classify DCSP into two categories: problem with single-variable agents and problem with multiple-variable agents. Most algorithms were originally designed to work with single-variable agents.

2.1.2 Problem formalization

In this section, the formalization of DCSP is presented. The DCSP [38] considers the following environment.

- There are n agents with k variables $x_0, x_1, \dots, x_{k-1}$, $n \leq k$, which have values in domains $D_0, D_1, \dots, D_{k-1}$, respectively. We define a partial function B over the product-range $\{0, 1, \dots, (n-1)\} \times \{0, 1, \dots, (k-1)\}$ such that, that variable x_j belongs to agent i is denoted by $B(i, j)!$. The exclamation mark '!' means 'is defined'.
- There are m constraints $c_0, c_1, \dots, c_{m-1}$ to be conjunctively satisfied. In a similar fashion as defined for $B(i, j)$, we use $E(l, j)!$, ($0 \leq l < m, 0 \leq$

2.1 Distributed Constraint Satisfaction Problem (DCSP)

$j < k$), to denote that x_j is relevant to the constraint c_l .

The DCSP may be formally stated as follows.

Problem Statement: $\forall i, j$ ($0 \leq i < n$)($0 \leq j < k$) where $B(i, j)!$, find the assignment $x_j = d_j \in D_j$ such that $\forall l$ ($0 \leq l < m$) where $E(l, j)!$, c_l is satisfied.

2.1.3 The need for a distributed model

A question might be raised at this stage: why do we need to distribute the data and constraints among the agents? We can simply let one agent gather all the information and use some well-developed centralized algorithms. There are several reasons why a decentralized algorithm could be preferred to a centralized one [37]:

- Distributed model is more suitable for a multiagent system where it is difficult or infeasible to perform information aggregation due to communication cost and format variety.
- Distributed model promotes parallelism. We note that the research motivation of parallel processing and DCSP are basically different. While the former concerns with efficiency of program execution, the latter deals with finding a solution with knowledge distributed among automated agents. However, since the agents work independently and concurrently, parallel processing in solving the problem can be achieved to a certain level as a byproduct.
- Gathering all information in one agent is undesirable because of security and privacy reasons. An agent may not desire to share all information it possesses but prefers to partially reveal them as necessary until a solution is obtained.

2.2 Classification of Existing DCSP Algorithms

2.1.4 Communication model

In a distributed model, agents need to communicate with others to obtain the values of other variables in the inter-agent constraints or inform other agents about changes in the values of their variables. The simple communication model assumed in solving DCSP is described below [38]:

- Agents communicate by sending messages. An agent can send messages to another agent if only it knows the address of that agent. By default, an agent involved in a constraint will know the addresses of the other agents also involved in that constraint.
- The delay in delivering a message is finite, though random. Messages are received in the order in which they were sent.

2.2 Classification of Existing DCSP Algorithms

There are several well-developed algorithms to solve DCSP with single-variable agents. Some of them are presented in the book by M. Yokoo [37]. In general the algorithms can be classified into three different types:

- Backtracking: These algorithms build up a partial solution and extend it until the solution becomes complete. When one variable has no value to satisfy all the constraints which only consist of variables in the partial solution, the value of one other variable in the partial solution will be changed. Some heuristics can be used in building the partial solution to improve the performance. Some algorithms in this class are Asynchronous Backtracking and Asynchronous Backtracking with min-conflict heuristic.

2.2 Classification of Existing DCSP Algorithms

- Iterative improvement. This type of algorithm usually relies on some cost or evaluation function. Based on this function, the system will be directed to go through different states until a solution is obtained. No partial solution is built during the search. One algorithm in this class is Distributed Breakout.
- Hybrid-type: The algorithms involve the combination of the two types above. Asynchronous Weak Commitment Search is one of the most efficient algorithms in this class.

For simplicity in describing the algorithms, the following assumptions are made.

- Each agent has exactly one variable.
- All constraints are binary, i.e., there are at most two variables involved in one constraint.
- Each agent knows all the constraint predicates which are relevant to its variable.

Since each agent is assumed to have only one variable, the terms ‘variable’ and ‘agent’ can replace each other in this context. Section 2.3.1 shows how to extend the algorithms to the case where an agent can have multiple local variables.

2.2.1 The Asynchronous Backtracking algorithm (ABT)

Asynchronous Backtracking is one of the fundamental algorithms. Its principles are straightforward: each variable is assigned a unique priority and a

2.2 Classification of Existing DCSP Algorithms

partial solution is built up by adding in the variables according to the order of their priorities. More specifically, variables with higher priorities are added into the partial solution first and there must not be any constraint violation among variables in the partial solution. When a variable is added into the partial solution, it will try to find a value to satisfy all the constraints which only consist of variables in the partial solution. If it cannot find such a value, the variable with the lowest priority in the partial solution, i.e. the most recently added variable, will have to change its value. If this variable can choose a new suitable value, the variable that is going to be added in the partial solution will again try to search for a value which can satisfy the constraints. Otherwise if the variable in the partial solution cannot change to a different value, the variable with the lowest higher priority will be asked to change. In other words, the variables with high priorities only need to change values when all the lower priority variables cannot find a value to satisfy the constraints in the partial solution. When the variable with the lowest priority is successfully added to the partial solution, the solution is said to be complete. When the variable with the highest priority has used up all of its domain values, there is no solution for the problem.

The correctness and completeness properties of this algorithm are proved in M. Yokoo [37]. The time complexity is exponential in the problem size, i.e., the number of variables and size of their domains. The space complexity is linear in the size of the domains.

2.2 Classification of Existing DCSP Algorithms

2.2.2 The Asynchronous Weak Commitment Search algorithm (AWC)

One limitation of Asynchronous Backtracking is the fixed priority scheme. The priorities are assigned to the variables during initialization and kept fixed during execution. If a high priority variable chooses a bad value, all variables with lower priorities will perform an exhaustive search before this variable needs to change its value. In Asynchronous Weak Commitment Search [36], when a variable is added to the partial solution and it cannot find a suitable value, the priority of the variable will be increased so that it will have the highest priority. This variable then has the right to choose a value so that it will minimize the number of violated constraints, and the partial solution is cleared and rebuilt starting with this variable. Hence the agent with priority who has chosen a bad value will not commit to the bad decision since the other agents will have higher priorities and make this agent change the bad value. When a variable cannot be added to the partial solution because it cannot find a suitable value, the partial solution is called a *nogood*. The agent who has this variable will check if this *nogood* is new. If so, the agent will store this *nogood* and send it to the other agents before increasing the priority of its variable. Otherwise the agent will not do anything. The reason for storing the *nogoods* is to guarantee the completeness of the algorithm. More details on this can be found in M. Yokoo [37]. Since a *nogood* is similar to a state of the system, in the worst case the list of *nogoods* is equivalent to the list of all possible states which is exponential in the size of the problem. So the worst case space and time complexities are exponential in the problem size. However in most of the cases, this algorithm outperforms the Asynchronous Backtracking algorithm.

2.2 Classification of Existing DCSP Algorithms

2.2.3 The Distributed Breakout algorithm (DBO)

While ABT and AWC follow backtracking approach, Distributed Breakout (DBO) adopts iterative improvement. In DBO, the agents repeatedly improve the current situation, i.e., try to reduce the number of violated constraints until there is no violation. The details of DBO are described as follows [39].

- A *weight* is defined for each pair of variable values that does not satisfy constraints. Initially all the weights are set to 1.
- Each agent tries to find its maximum improvement which is the maximum difference of the total weight of constraint violating pairs solved and created due to changes in its variable's value.
- Among a group of agents, i.e., one agent and all of its neighboring agents, only the agent with the highest positive improvement can make the change to its variable's value.
- If no agent in a group has a positive improvement, the weights of all constraint violating pairs will be increased by 1.

DBO is considered as an efficient method for solving DCSP. However it does not possess the completeness property [37], that of returning a solution if one exists and terminating with no solution otherwise. The algorithm is guaranteed to complete for acyclic constraint graphs [42]. A constraint graph is a graph in which each vertex represents a variable and each edge represents a binary constraint between two agents. However for cyclic constraint graphs, the algorithm may get stuck in a local loop and never terminate.

2.2 Classification of Existing DCSP Algorithms

2.2.4 The Asynchronous Partial Overlay algorithm (APO)

Asynchronous Partial Overlay applies the idea of *Cooperative Mediation* [18], in which agents cooperate with a mediator which computes based on the information from the agents involved to propose a solution. In ABT and AWC, an agent only knows that its current value assignment does not match with other agents' value assignments to satisfy all the constraint. It does not know why the value is not accepted and hence, it has to try proposing another value in its variable's domain and wait for the replies. In cooperative mediation, since the mediator collects information about the variables and their domains from the agents involved, it can locally compute and determine which combination of values will satisfy all the constraints. A complete algorithm is presented in [17]. The steps in an execution round can be briefly described as follows:

- Each agent checks if any of their relevant constraints is violated. If there is no violation, the agent does not need to do anything.
- Otherwise if an agent is not told by any higher-priority agents that they want to mediate, it will take the role of the mediator.
- As the mediator, an agent will try to change its own value assignment first. If a possible value is found, it will be selected and a notification is sent to the neighboring agents of the mediator.
- If the mediator cannot find a local value to resolve the conflict, it will send request to its neighboring agents informing that a mediation is about to begin.

2.3 Other DCSP Issues

- When an agent receives a request, it can accept to join the mediation by sending back a reply together with its information needed for the mediator to find a solution. Otherwise an agent may refuse to join the mediation.
- Based on the information received, the mediator performs a search to find the solution among agents involved in the mediation.
- Once the solution is found, the mediator sends the result values to the respective agents.

Although the number of execution cycles is reduced comparing to AWC [17], the amount of computation required in each cycle increases significantly due to the overhead in the mediation process. This shows the impact of problem centralization, that of gathering distributed data and locally computing a solution, on the execution time of an algorithm [6].

2.3 Other DCSP Issues

2.3.1 Extension to multi-variable agents

The model of a multi-variable agent is usually used for a problem consisting of complex local problems, in which variables are clustered into groups where there are many constraints between variables in the same group and fewer constraints between variables in different groups. One agent will handle all variables in one group. This model will help to reduce the communication between agents, compared to the model in which each agent is assigned a variable. Currently there are two approaches to solve DCSP with multi-variable agents, as described in [40]:

2.3 Other DCSP Issues

- Approach 1: Find all the possible solutions for the local problems in each agent, and then use the values in solutions as the domains of the variables. The agents then communicate with others to choose values in these domains so that the inter-agent constraints are satisfied. When the local problem is large and complex, finding all solutions is infeasible.
- Approach 2: Each agent will consist of several virtual agents each of which will handle one variable and apply the algorithms for the case of single-variable agents. The shortcomings of this method are: (i) the communication between virtual agents is unnecessary and (ii) the computation is more expensive than pure local computation. Therefore the design and implementation of multi-variable agents must be done properly to minimize redundant communication and computation.

2.3.2 Coordination, security and privacy

Although the details of the algorithms are different, they basically follow two approaches in coordination: (i) backtracking-based approaches such as ABT and (ii) mediation-based approaches such as APO. As a result, the computational and communication models of algorithms following these two approaches are different and hence one execution cycle in each approach may bear a different cost. In an attempt to give a comparison between backtracking-based and mediation-based approaches, some important results have been obtained in [3]:

- On a relatively easy problem, two approaches use about the same amount of non-concurrent constraints checks.
- On high-density solvable instances, the benefit of finding solutions that minimize conflicts outside of a mediation session far outweighs the extra

2.4 Automated Negotiation

computation necessary to do so.

- The decrease in backtracking during problem solving leads to less communication as well.

Another concern in DCSP research besides computational issues is privacy, i.e., whether an agent can obtain all private information, e.g. the number of variables and their corresponding domains after a certain number of communication round. In APO, the risk of revealing private information is higher as each agent involved in the mediation needs to send its variable and domain information to the mediator and the role of mediator will be taken by different agents in different execution rounds. Although an agent does not need to send all information to the mediator, but in return the completeness and performance of the algorithm will be affected. An asynchronous algorithm called Secure DisCSP is proposed in [41]. ElGamal encryption/decryption is used to encode/decode the constraints. Although a certain level of security is achieved, the overall execution time is increased due to the overhead of encrypting/decrypting operations. This approach is more about practical implementation rather than changing the protocol.

2.4 Automated Negotiation

When we are discussing about agents which can cooperate and negotiate to achieve their goals, we are actually referring to *intelligent* agents. An intelligent agent is one that is capable of *flexible* autonomous actions in order to meet its design objectives [34]. In this context, flexibility means three things:

- **Reactivity:** intelligent agents are able to perceive their environment

2.4 Automated Negotiation

and respond in a timely fashion to meet their objectives.

- Pro-activeness: intelligent agents are able to clearly show that they are pursuing certain objectives and taking initiative to achieve these objectives.
- Social ability: intelligent agents are able to interact with other agents to meet their objectives.

Social ability includes being able to cooperate and negotiate. Although automated negotiation among agents is relatively different from that among humans, they share some similarities. When people negotiate, they may promise, threaten and compromise to eventually reach an agreement or consensus. In reaching an agreement, all parties involved must have in their minds what outcomes are acceptable. Also, there are social conventions and laws that humans follow when negotiating. Similarly, automated negotiation has three main components: (i) *space of possible deals*, (ii) *negotiation process* and (iii) *negotiation strategy*. A space of possible deals defines the kinds of agreements an agent can accept. A negotiation process, or *protocol*, specifies the interaction rules the communicating agents can use to converge to an agreement. Given a set of possible deals and interaction rules, a negotiation strategy (or set of strategies), when properly devised, should guide each agent to achieve its objectives. A protocol and a strategy (or a set of strategies) constitute a *negotiation mechanism* by which agents can come to a consensus in a negotiation.

Similar to humans, agents can be self-interested, i.e., they can behave in a manner which can save them some efforts and resources without affecting the completion of their objectives, although such behaviors could lead to unfair results for some of the agents involved in the interaction. The protocol

2.5 The Concept of Belief-Desire-Intention (BDI)

designers need to take this into consideration, to ensure that a protocol designed is fair for all parties involved. The strategy designers need to clearly understand the protocol in order to formulate a strategy so that an agent can achieve its objectives without violating the rules [26]. There are many protocols designed for agent interactions, such as in game theory [4] and robotics [7].

Our research is motivated by the fact that automated negotiation can provide a strong intellectual basis for the development of a conceptual framework for DCSP. As Chapter 3 will detail, by viewing DCSP algorithms as negotiation mechanisms, it can be shown that under different algorithms, DCSP agents would (strategically) interact differently to resolve constraint violations, although their interactions are governed by a common protocol.

2.5 The Concept of Belief-Desire-Intention (BDI)

2.5.1 Introduction

The architectural view of an agent is an interesting topic in multiagent systems. A number of different approaches have emerged to formulate the reasoning of an agent [19, 25]. Among the concrete architectures including the reactive [9] and layered [13] architectures, Belief-Desire-Intention is perhaps one of the most commonly used. The BDI model was first proposed by M. Bratman in 1987 [5] and formalized in 1991 [25]. Since then, the BDI architecture has developed into a strong foundation for building intelligent agents. The root of the BDI architecture is practical reasoning: the processing of deciding which action to perform at different moments to pursue

2.5 The Concept of Belief-Desire-Intention (BDI)

some goals. Practical reasoning involves two processes: deciding what goals to be achieved and how to achieve them, and are governed by the concepts of beliefs, desires and intentions according to [15].

In the following, we present a simple example ¹ of practical reasoning: Consider a student who is going to graduate soon; he will have to decide what he is going to do with his life. The first option is finding a job to work in the industry. If the student expects he would get a good degree, another option is available to him which is to become an academic. Given certain factors of the environment such as financial ability, family concerns and his own interest, the student believes he should work in the industry. Once he believes so, he will have a list of options of the job type. For example, the student may desire to be an engineer, analyst or consultant. Assume that the student wants to be an engineer; it now becomes his intention. The student will take a series of actions to achieve that intention: he may visit websites from different companies and send his resume to them, then go for interview if shortlisted. From the above example, we can see that the beliefs, desires and intentions must be consistent. The student cannot have the intention to be an engineer while he does not believe so. In addition, since the environmental factors may change from time to time, some beliefs can change and some intentions can be dropped. If the student does well in his final year project which is research-oriented, he may find that he is suitable for research work and hence believes that he would be a good academic. In this case, his intention to be an engineer will be dropped.

A lot of work has gone into the implementation of BDI-based agents [12, 14]. Recent cooperative agents research that adapts BDI reasoning includes collaborative linear assignment [28]. A smart application for Personal Digital

¹This example is a different version of the one presented in [34]

2.5 The Concept of Belief-Desire-Intention (BDI)

Assistant (PDA) that employs a BDI architecture is reported in [32].

2.5.2 Existing BDI architectures

Figure 2.1 shows the basic architecture of the BDI model.

In general, the basic architecture consists of seven main components as described in [34]:

- A set of current beliefs which represents information obtained from the environment.
- A belief revision function (BRF) which will generate the set of beliefs based on perceptual input from the environment and the current beliefs.
- An option generation function which will generate the set of options available to the agent given the current beliefs and intentions.
- The set of current options, representing possible courses of actions available to the agent.
- A filter function which will determine the intentions on the basis of current beliefs, desires and intentions.
- The set of current intentions, representing the agent's current focus.
- The action selection function which will decide which action to perform to achieve the intentions.

The behavior of the BDI-reasoning functions may vary in different applications but the seven components described above always exist.

Although the BDI reasoning model has been used with considerable success in practice, the BDI mechanism is simplistic in nature and hence may

2.5 The Concept of Belief-Desire-Intention (BDI)

Figure 2.1: The basic BDI architecture

not be able to address many generic aspects of human behavior and reasoning. There have been many attempts to extend the original model to cater for more complicated scenarios. An enhanced BDI architecture which allows extensibility is proposed in [1]. The basic idea of the architecture is to break up the original abstract BDI cycle (as in Figure 2.1) into a small set of self-contained meta-actions, which are invoked as needed, rather than being executed in a fixed sequence.

In the basic architecture, the BDI-reasoning functions of Perception-Belief-Desire-Intention-Execution are sequentially executed, which means an agent cannot retrieve new information before the current iteration step is finished. In a complicated and dynamic environment, each step of the process may take more time to execute. As a result, the agent may not be

2.5 The Concept of Belief-Desire-Intention (BDI)

able to react to situational changes in time and by the moment a decision is made, it may be already obsolete. Addressing this problem, a parallel BDI agent architecture is proposed [11]. In this model, there are three active entities: (i) *belief manager*, (ii) *intention generator* and (iii) *intention executor*. These entities are responsible for retrieving and updating four data structures implementing the beliefs, desires, intentions and the plan library. Importantly, these three entities run concurrently. The belief manager receives information from the environment and updates the set of beliefs. The intention generator generates the intentions based on the belief set. If there are new beliefs, the intention generator will be notified. In this case, it will generate new intentions and also remove the obsolete ones. The intention executor, which explains the intentions and executes the actions defined in the intentions, will also be notified about the changes in the intention set.

As mentioned in Section 2.4, viewing DCSP algorithms as negotiation mechanisms has unearthed a common protocol that governs existing DCSP algorithms. Interestingly, this interaction protocol can be formulated in terms of the model-theoretic concepts of B, D and I, establishing an automated BDI negotiation framework for DCSP agents. Chapter 3 will present the details of this new intellectual framework.

Chapter 3

The BDI Negotiation Framework

In this chapter, the generic model of a DCSP agent is described, followed by the description and explanation of the BDI negotiation framework. We show how the original BDI architecture can formulate the behavior of an DCSP agent and therefore establish a protocol for negotiation among such agents. Finally, four existing algorithms, ABT, AWC, DBO and APO, are reformulated as different strategies running on a common BDI negotiation protocol.

3.1 DCSP Agent Model

In general, all DCSP agents must cooperatively interact, and essentially perform the assignment and reassignment of domain values to variables to resolve all constraint violations. If the agents succeed in their resolution, a solution is found.

In order to engage in cooperative behavior, a DCSP agent needs five fundamental parameters, namely, (i) a variable [38] or a variable set [40], (ii) domains, (iii) priority, (iv) a neighbor list and (v) a constraint list.

Each *variable* assumes a range of values called a *domain*. A domain value, which usually abstracts an action, is a possible *option* that an agent may take. Each agent has an assigned *priority*. These priority values help decide the order in which they revise or modify their variable assignments. An

3.2 The BDI Negotiation Model

agent's priority may be fixed (static) or changing (dynamic) when searching for a solution. If an agent has more than one variable, each variable can be assigned a different priority, to help determine which variable assignment the agent should modify first. For simplicity in describing the BDI negotiation framework, we continue to assume that each agent has only one variable.

An agent which shares the same constraint with another agent is called the latter's *neighbor*. Each agent needs to refer to its list of neighbors during the search process. This list may also be kept unchanged or updated accordingly in runtime. Similarly, each agent maintains a constraint list. The agent needs to ensure that there is no violation of the constraints in this list. Constraints can be added or removed from an agent's constraint list in runtime.

As with an agent, a constraint can also be associated with a priority value. Constraints with a high priority are said to be more important than constraints with a lower priority. To distinguish it from the priority of an agent, the priority of a constraint is called its *weight*.

3.2 The BDI Negotiation Model

Within the scope of the DCSP framework, it can be said that the common goal pursued by all agents is finding a combination of domain values to satisfy a set of predefined constraints. In automated negotiation [26], such a solution is called an *agreement* among the agents; with negotiation being the process of several agents searching for the agreement. This search process is realized via a negotiation mechanism (or algorithm) implementing a negotiation model comprising of a high level *protocol* and a set of *strategies*. In general, given a protocol specifying the rules of interactions, different strategies can be designed for individual agents to select their own preferences among avail-

3.2 The BDI Negotiation Model

able choices at each negotiation step. Whereas a protocol is *public* in that it is agreed and followed by all the agents in a negotiation process, the strategies used by individual agents can be more *private* in that their full details can be hidden from other agents. Within this scope, we found that we were able to unearth the generic BDI behavior of a DCSP agent and formulate it in a negotiation protocol. Thus, our proposed negotiation model can be said to combine the BDI concepts with automated negotiation in a multiagent framework, allowing us to conceptually separate DCSP mechanisms into a common BDI interaction protocol and the adopted strategies.

3.2.1 The generic protocol

Figure 3.1 shows the basic reasoning steps in an arbitrary round of negotiation that constitute the new protocol. A solid line indicates a common component or transition which always exists regardless of the strategy used. A dotted line indicates a component or transition which may or may not appear depending on the adopted strategy.

Two types of messages are exchanged through this protocol, namely, the *info* message and the negotiation message.

An *info* message perceived is a message sent by another agent. The message will contain the current selected value and priority of the variable of that sending agent. The main purpose of this message is to update the agent about the current environment. *Info* message is sent out at the end of one negotiation round (also called a negotiation cycle), and received at the beginning of next round.

A *negotiation message* is a message which may be sent within a round. This message is for mediation purposes. The agent may put different contents into this type of message as long as it is agreed among the group. The format

3.2 The BDI Negotiation Model

Figure 3.1: The BDI interaction protocol

3.2 The BDI Negotiation Model

of the negotiation message and when it is to be sent out are subject to the strategy. A negotiation message can be sent out at the end of one reasoning step and received at the beginning of the next step.

Mediation is a step of the protocol that depends on whether the agent's interaction with others is synchronous or asynchronous. In a synchronous mechanism, mediation is required in every negotiation round. In an asynchronous one, mediation is needed only in a negotiation round when the agent receives a negotiation message. A more in-depth view of this mediation step is provided later in this section.

The BDI protocol prescribes the skeletal structure for DCSP negotiation. We will show in Section 3.3 that several well-known DCSP mechanisms all inherit this generic model.

The details of the six main reasoning steps for the protocol (see Figure 3.1) are described as follows for a DCSP agent. For a conceptually clearer description, we assume that there is only one variable per agent.

- **Percept.** In the scope of the proposed protocol, percept means perceiving any changes in the environment. More specifically, these are the changes in value assignments of the variables. In this step, the agent receives *info* messages from its neighbors in the environment, and using its Percept function, returns an image P . This image contains the current values assigned to the variables of all agents in its neighbor list. The image P will drive the agent's actions in subsequent steps. The agent also updates its constraint list C using some criteria of the adopted strategy.
- **Belief.** Using the image P and constraint list C , the agent will check if there is any violated constraint. If there is no violation, the agent will

3.2 The BDI Negotiation Model

believe it is choosing a correct option and therefore will take no action. The agent will do nothing if it is in a local stable state - a *snapshot of the variables assignments of the agent and all its neighbors* by which they satisfy their shared constraints. *When all agents are in their local stable states, the whole environment is said to be in a global stable state and an agreement is found.* In case the agent finds its value in conflict with some of its neighbors', i.e., the combination of values assigned to the variables leads to a constraint violation, the agent will first try to reassign its own variable using a specific strategy. If it finds a suitable option which meets some criteria of the adopted strategy, the agent will believe it should change to the new option. However it does not always happen that an agent can successfully find such an option. If no option can be found, the agent will believe it has no option, and therefore will request its neighbors to reconsider their variable assignments.

To summarize, there are three types of belief that a DCSP agent can form: (i) it can change its variable assignment to improve the current situation, (ii) it cannot change its variable assignment and some constraints violations cannot be resolved and (iii) it does not need to change its variable assignment as all the constraints are satisfied.

Once the beliefs are formed, the agent will determine its desires, which are the *options* that attempt to resolve the current constraint violations.

- **Desire.** If the agent takes Belief (i), it will generate a list of its own suitable domain values as its desire set. If the agent takes Belief (ii), it cannot ascertain its desire set, but will generate a sublist of agents

3.2 The BDI Negotiation Model

from its neighbor list, whom it will ask to reconsider their variable assignments. How this sublist is created depends on the strategy devised for the agent. In this situation, the agent will use a *virtual* desire set that it determines based on its adopted strategy. If the agent takes Belief (iii), it will have no desire to revise its domain value, and hence no intention.

- **Intention.** The agent will select a value from its desire set as its intention. An intention is the best desired option that the agent assigns to its variable. The criteria for selecting a desire as the agent's intention depend on the strategy used. Once the intention is formed, the agent may either proceed to the execution step, or undergo mediation. Again, the decision to do so is determined by some criteria of the adopted strategy.
- **Mediation.** This is an important function of the agent. Since, if the agent executes its intention without performing intention mediation with its neighbors, the constraint violation between the agents may not be resolved. Take for example, suppose two agents have variables, x_1 and x_2 , associated with the same domain $\{1, 2\}$, and their shared constraint is $(x_1 + x_2 = 3)$. Then if both the variables are initialized with value 1, they will both concurrently switch between the values 2 and 1 in the absence of mediation between them.

There are two types of mediation: local mediation and group mediation. In the former, the agents exchange their intentions. When an agent receives another's intention which conflicts with its own, the agent must mediate between the intentions, by either changing its own intention or informing the other agent to change its intention. In the latter,

3.2 The BDI Negotiation Model

there is an agent which acts as a group mediator. This mediator will collect the intentions from the group - a union of the agent and its neighbors - and determine which intention is to be executed. The result of this mediation is passed back to the agents in the group. Following mediation, the agent may proceed to the next reasoning step to execute its intention or begin a new negotiation round.

- **Execution.** This is the last step of a negotiation round. The agent will execute by updating its variable assignment if the intention obtained at this step is its own. Following execution, the agent will inform its neighbors about its new variable assignment and updated priority. To do so, the agent will send out an *info* message.

3.2.2 Strategy formulation

A strategy plays an important role in the negotiation process. Within the protocol, it will often determine the efficiency of the search process in terms of computational cycles and message communication costs.

The design space when devising or formulating a strategy is influenced by the following dimensions: (i) asynchronous or synchronous, (ii) dynamic or static priority, (iii) dynamic or static constraint weight, (iv) number of negotiation messages to be communicated, (v) the negotiation message format and (vi) the completeness property. In other words, these dimensions provide technical considerations for a strategy design.

3.3 DCSP Algorithms as Different Strategies on a Common BDI Protocol

3.3 DCSP Algorithms as Different Strategies on a Common BDI Protocol

In this section, we apply the proposed BDI negotiation model presented in Section 3.2 to expose the BDI protocol and the different strategies used for four well-known algorithms, ABT, AWC, DBO and APO. All these algorithms assume that there is only one variable per agent. Under our framework, we call the strategies applied the ABT, AWC, DBO and APO strategies, respectively.

To describe each strategy formally, the following mathematical notations are used:

- n is the number of agents, m is the number of constraints;
- x_i denotes the variable held by agent i , ($0 \leq i < n$);
- D_i denotes the domain of variable x_i ; F_i denotes the neighbor list of agent i ; C_i denotes its constraint list;
- p_i denotes the priority of agent i ; and $P_i = \{(x_j = v_j, p_j = k) \mid \text{agent } j \in F_i, v_j \in D_j \text{ is the current value assigned to } x_j \text{ and the priority value } k \text{ is a positive integer}\}$ is the perception of agent i ;
- w_l denotes the weight of constraint l , ($0 \leq l < m$);
- $S_i(v)$ is the total weight of the violated constraints in C_i when its variable has the value $v \in D_i$.

3.3 DCSP Algorithms as Different Strategies on a Common BDI Protocol

Figure 3.2: BDI protocol with *Asynchronous Backtracking* strategy

3.3.1 The Asynchronous Backtracking strategy (ABT)

Figure 3.2 presents the BDI negotiation model incorporating the Asynchronous Backtracking (ABT) strategy. As mentioned in Section 3.2, for an asynchronous mechanism that ABT is, the mediation step is needed only in a negotiation round when an agent receives a negotiation message.

For agent i , beginning initially with $(w_l = 1, (0 \leq l < m); p_i = i, (0 \leq i < n))$ and F_i contains all the agents who share the constraints with agent

3.3 DCSP Algorithms as Different Strategies on a Common BDI Protocol

i , its BDI-driven ABT strategy is described as follows.

Step 1 - Percept: Update P_i upon receiving the *info* messages from the neighbors (in F_i). Update C_i to be the list of constraints which only consists of agents in F_i that have equal or higher priority than this agent.

Step 2 - Belief: The belief function GB (P_i, C_i) will return a value $b_i \in \{0, 1, 2\}$, decided as follows:

- $b_i = 0$ when agent i can find an optimal option, i.e., if $(S_{i(v_i)} \neq 0$ or v_i is in *bad values list*) and $(\exists a \in D_i)(S_{i(a)} = 0)$ and a is not in a list of domain values called *bad values list*. Initially this list is empty and it will be cleared when a neighbor of higher priority changes its variable assignment.
- $b_i = 1$ when it cannot find an optimal option, i.e., if $(\forall a \in D_i)(S_{i(a)} \neq 0)$ or a is in *bad values list*.
- $b_i = 2$ when its current variable assignment is an optimal option, i.e., if $S_{i(v_i)} = 0$ and v_i is not in *bad value list*.

Step 3 - Desire: The desire function GD (b_i) will return a desire set denoted by DS, decided as follows:

- If $b_i = 0$, then $DS = \{a \mid (a \neq v_i), (S_{i(a)} = 0) \text{ and } a \text{ is not in the } \textit{bad value list} \}$.
- If $b_i = 1$, then $DS = \emptyset$, the agent also finds agent k which is determined by $\{k \mid p_k = \min(p_j) \text{ with agent } j \in F_i \text{ and } p_k > p_i \}$.
- If $b_i = 2$, then $DS = \emptyset$.

Step 4 - Intention: The intention function GI (DS) will return an intention, decided as follows:

3.3 DCSP Algorithms as Different Strategies on a Common BDI Protocol

- If $DS \neq \emptyset$, then select an arbitrary value (say, v'_i) from DS as the intention.
- If $DS = \emptyset$, then assign *nil* as the intention (to denote its lack thereof).

Step 5 - Execution:

- If agent i has a domain value as its intention, the agent will update its variable assignment with this value.
- If $b_i = 1$, agent i will send a negotiation message to agent k , then remove k from F_i and begin its next negotiation round. The negotiation message will contain the list of variable assignments of those agents in its neighbor list F_i that have a higher priority than agent i in the current image P_i .

Mediation: When agent i receives a negotiation message, several sub-steps are carried out, as follows:

- If the list of agents associated with the negotiation message contains agents which are not in F_i , it will add these agents to F_i , and request these agents to add itself to their neighbor lists. The request is considered as a type of negotiation message.
- Agent i will first check if the sender agent is updated with its current value v_i . The agent will add v_i to its *bad values list* if it is so, or otherwise send its current value to the sender agent.

Following this step, agent i proceeds to the next negotiation round.

3.3 DCSP Algorithms as Different Strategies on a Common BDI Protocol

3.3.2 The Asynchronous Weak Commitment Search strategy (AWC)

Figure 3.3 presents the BDI negotiation model incorporating the *Asynchronous Weak Commitment* (AWC) strategy. The model is similar to that of incorporating the ABT strategy (see Figure 3.2). This is not surprising; AWC and ABT are found to be strategically similar, differing only in the details of some reasoning steps. The distinguishing point of AWC is that when the agent cannot find a suitable variable assignment, it will change its priority to the highest among its group members $(\{i\} \cup F_i)$.

For agent i , beginning initially with $(w_l = 1, (0 \leq l < m); p_i = i, (0 \leq i < n))$ and F_i contains all the agents who share the constraints with agent i , its BDI-driven AWC strategy is described as follows.

Step 1 - Percept: This step is identical to the Percept step of ABT.

Step 2 - Belief: The belief function GB (P_i, C_i) will return a value $b_i \in \{0, 1, 2\}$, decided as follows:

- $b_i = 0$ when the agent can find an optimal option, i.e., if $(S_i(v_i) \neq 0$ or the assignment $x_i = v_i$ and the current variables assignments of the neighbors in F_i who have higher priority form a *nogood* [38]) stored in a list called *nogood* list and $\exists a \in D_i, S_i(a) = 0$ (initially the list is empty).
- $b_i = 1$ when the agent cannot find any optimal option, i.e., if $\forall a \in D_i, S_i(a) \neq 0$.
- $b_i = 2$ when the current assignment is an optimal option i.e., if $S_i(v_i) = 0$ and the current state is not a *nogood* in *nogood* list.

3.3 DCSP Algorithms as Different Strategies on a Common BDI Protocol

Figure 3.3: BDI protocol with *Asynchronous Weak-Commitment* strategy

3.3 DCSP Algorithms as Different Strategies on a Common BDI Protocol

Step 3 - Desire: The desire function $GD(b_i)$ will return a desire set DS , decided as follows:

- If $b_i = 0$, then $DS = \{a \mid (a \neq v_i), (S_{i(a)} = 0) \text{ and the number of constraint violations with lower priority agents is minimized} \}$.
- If $b_i = 1$, then $DS = \{a \mid a \in D_i \text{ and the number of violations of all relevant constraints is minimized} \}$.
- If $b_i = 2$, then $DS = \emptyset$.

Following, if $b_i = 1$, agent i will find a list K_i of higher priority neighbors, defined by $K_i = \{k \mid \text{agent } k \in F_i \text{ and } p_k > p_i\}$.

Step 4 - Intention: This step is similar to the Intention step of ABT. However, for this strategy, the negotiation message will contain the variable assignments (of the current image P_i) for all the agents in K_i . This list of assignment is considered as a *nogood*. If the same negotiation message had been sent out before, agent i will have *nil* intention. Otherwise, the agent will send the message and save the *nogood* in the *nogood* list.

Step 5 - Execution:

- If agent i has a domain value as its intention, the agent will update its variable assignment with this value.
- If $b_i = 1$, it will send the negotiation message to its neighbors in K_i , and set $p_i = \max\{p_j\} + 1$, with agent $j \in F_i$.

Mediation: This step is identical to the Mediation step of ABT, except that agent i will now add the *nogood* contained in the negotiation message received to its own *nogood* list.

3.3 DCSP Algorithms as Different Strategies on a Common BDI Protocol

3.3.3 The Distributed Breakout strategy (DBO)

Figure 3.4 presents the BDI negotiation model incorporating the *Distributed Breakout* (DBO) strategy. Essentially, by this synchronous strategy, each agent will search iteratively for improvement by reducing the total weight of the violated constraints. The iteration will continue until no agent can improve further, at which time if some constraints remain violated, the weights of these constraints will be increased by 1 to help ‘breakout’ from a local minimum.

For agent i , beginning initially with $(w_l = 1, (0 \leq l < m), p_i = i, (0 \leq i < n))$ and F_i contains all the agents who share the constraints with agent i , its BDI-driven DBO strategy is described as follows.

Step 1 - Percept: Update P_i upon receiving the *info* messages from the neighbors (in F_i). Update C_i to be the list of its relevant constraints.

Step 2 - Belief: The belief function GB (P_i, C_i) will return a value $b_i \in \{0, 1, 2\}$, decided as follows:

- $b_i = 0$ when agent i can find an option to reduce the number violations of the constraints in C_i , i.e., if $\exists a \in D_i, S_{i(a)} < S_{i(v_i)}$.
- $b_i = 1$ when it cannot find any option to improve situation, i.e., if $\forall a \in D_i, a \neq v_i, S_{i(a)} \geq S_{i(v_i)}$.
- $b_i = 2$ when its current assignment is an optimal option, i.e., if $S_{i(v_i)} = 0$.

Step 3 - Desire: The desire function GD (b_i) will return a desire set DS, decided as follows:

- If $b_i = 0$, then $DS = \{a \mid a \neq v_i, S_{i(a)} < S_{i(v_i)} \text{ and } (S_{i(v_i)} - S_{i(a)}) \text{ is maximized}\}$. $(\max\{(S_{i(v_i)} - S_{i(a)})\})$ will be referenced by h_i^{max} in

3.3 DCSP Algorithms as Different Strategies on a Common BDI Protocol

Figure 3.4: BDI protocol with *Distributed Breakout* strategy

3.3 DCSP Algorithms as Different Strategies on a Common BDI Protocol

subsequent steps, and it defines the maximal reduction in constraint violations).

- Otherwise, $DS = \emptyset$.

Step 4 - Intention: The intention function $GI(DS)$ will return an intention, decided as follows:

- If $DS \neq \emptyset$, then select an arbitrary value (say, v'_i) from DS as the intention.
- If $DS = \emptyset$, then assign *nil* as the intention.

Following, agent i will send its intention to all its neighbors. In return, it will receive intentions from these agents before proceeding to Mediation step.

Mediation: Agent i receives all the intentions from its neighbors. If it finds that the intention received from a neighbor agent j is associated with $h_j^{max} > h_i^{max}$, the agent will automatically cancel its current intention.

Step 5 - Execution:

- If agent i did not cancel its intention, it will update its variable assignment with the intended value.
- If all intentions received and its own one are *nil* intention, the agent will increase the weight of each currently violated constraint by 1.

3.3.4 The Asynchronous Partial Overlay strategy (APO)

As mentioned in Section 2, the *Asynchronous Partial Overlay* (APO) strategy was introduced in [17]. In this strategy, agents in one negotiation round

3.3 DCSP Algorithms as Different Strategies on a Common BDI Protocol

Figure 3.5: BDI protocol with *Asynchronous Partial Overlay* strategy

do not have the same role. More specifically, agents are divided into small groups called *mediation sessions*. In each session, one agent will be the *local mediator*, who receives the information from other agents in the session and proposes an agreement to satisfy all the constraints between agents in the session. The mediator will then send the proposed values to all the agents in the session, who will in turn assign these values to their variables.

Figure 3.5 presents the BDI negotiation model incorporating the APO strategy. Since APO is an asynchronous strategy, the mediation step is not always needed in a negotiation round.

3.3 DCSP Algorithms as Different Strategies on a Common BDI Protocol

For agent i , beginning initially with $(u_l = 1, (0 \leq l < m); p_i = i, (0 \leq i < n))$ and F_i contains all the agents who share the constraints with agent i , its BDI-driven ABT strategy is described as follows.

Step 1 - Percept: Update P_i upon receiving the *info* messages from the neighbors (in F_i). Update C_i to be the list of constraints which only consists of agents in F_i . In addition, each *info* message received from an agent j will have a boolean flag m_j :

- m_j is true if agent j has some conflicts with its neighbors, i.e., some of its constraints are violated.
- m_j is false if agent j does not have any conflicts with its neighbors, i.e., all of its constraints are satisfied.

Step 2 - Belief: The belief function $GB(P_i, C_i)$ will return a value $b_i \in \{0, 1, 2\}$, decided as follows:

- $b_i = 0$ when agent i can find an optimal option, i.e., if $S_{i(v_i)} \neq 0$ and $(\exists a \in D_i)(S_{i(a)} = 0)$.
- $b_i = 1$ when it cannot find an optimal option, i.e., if $(\forall a \in D_i)(S_{i(a)} \neq 0)$.
- $b_i = 2$ when its current variable assignment is an optimal option, i.e., if $S_{i(v_i)} = 0$.

Step 3 - Desire: The desire function $GD(b_i)$ will return a desire set denoted by DS , decided as follows:

- If $b_i = 0$, then $DS = \{a \mid (a \neq v_i), (S_{i(a)} = 0)\}$.
- If $b_i = 1$, then $DS = \emptyset$.

3.3 DCSP Algorithms as Different Strategies on a Common BDI Protocol

- If $b_i = 2$, then $DS = \emptyset$.

Step 4 - Intention: The intention function $GI(DS)$ will return an intention, decided as follows:

- If $DS \neq \emptyset$, then select an arbitrary value (say, v'_i) from DS as the intention.
- If $DS = \emptyset$ and $b_i = 1$, if there are no agent j in which $p_j > p_i$ and m_j is true, the intention is to take the role of a mediator. Otherwise, assign *nil* as the intention.
- If $DS = \emptyset$ and $b_i = 2$, then assign *nil* as the intention (to denote its lack thereof).

If agent i has the intention to become the mediator, it sends out negotiation messages (this message is defined as the “evaluate?” message in [17]) to agents in F_i and waits for their replies. Otherwise, agent i proceeds to the Execution step.

Step 5 - Execution:

- If agent i has a *nil* intention, it will do nothing.
- If agent i has a domain value as its intention, the agent will update its variable assignment with this value. Update p_i to be the size of F_i . Finally, agent i sends out an *info* message to update all agents in F_i .

Mediation: When agent i receives an “evaluate?” message:

- If agent i is currently involved in another session, it will reply with a “wait!” message.

3.3 DCSP Algorithms as Different Strategies on a Common BDI Protocol

- If agent i is available, it will reply with an “evaluate!” message. This message contains all possible values in D_i and the set $S = S_i(x) \forall x \in D_i$.

When agent i receives a “wait!” message from agent j , it will drop agent j from the session.

If agent i is currently the mediator, and the number of “evaluate!” and “wait!” messages received is equal to the number of “evaluate?” message sent out in Intention step, it will start finding the solution, i.e., find a combination of value assignments for all the agents still in the session:

- if a solution is found, agent i will send the proposed values to all agents in the session using a message type called “accept!”.
- if a solution cannot be found, agent i will broadcast to all agents to inform that there is no solution. The algorithm terminates here.

If agent i sent out the “evaluate!” message before and now receives an “accept!” message. It will take the value inside this message to be its intention. Agent i goes to Execution step.

Following this step, agent i proceeds to the next negotiation round.

Chapter 4

The Unsolicited Mutual Advice Strategy (UMA)

This chapter presents the development of a new strategy called Unsolicited Mutual Advice (UMA). Although the purpose of creating this strategy is to demonstrate the usefulness of having a common framework for DCSP algorithms, UMA is found to have several advantages over existing strategies. An example is given at the end of this chapter to provide a clear illustration of how the UMA strategy works.

4.1 The Basic Idea

In the following, we present an example of constraint satisfaction problem in human world. Consider a group of students in an engineering school. Each student needs to use a computer to do their tutorials and the number of computers available is much smaller than the number of students. In addition, each student is only free at some specific time slots since they may have to attend different lectures or seminars. Unfortunately, there is no computer program designed to allocate the computers and time slots to the students based on their timetables. As a result, these students need to discuss among themselves to agree on a schedule which allows every student to access a computer at one of his free time slots. First of all, each student plans the computer and time slot he wants. Subsequently, they exchange the information about their plans. When a student realizes that the computer

4.1 The Basic Idea

he intends to use at a particular time slot will also be utilized by another student at the same time, he needs to plan again. If he can find another available computer or time slot to do his tutorials, he will just simply change to use this new computer or time slot and inform the other student so that this student (who also realizes that someone else has the same plan as he does) does not have to revise his plan. If the student cannot find any other computer which is available at any of his free time slots, he *needs to tell the other student that he could not find an alternative option and advise this student to reconsider his plan*. This *unsolicited advice* has an impact on the decision process of the student who receives it. If this student has already found a new computer or time slot, this advice will strongly encourage him to adopt the new plan. In this case, the plan change is *voluntarily* done. In the other case when the student has no alternative option and receives an advice, he may *reluctantly* check other computers or time slots. Although these computers and time slots are currently occupied by other students, there is a chance that one of these students can voluntarily change his plan so that this student can use the new computer or time slot.

The above example shows the basic idea of Unsolicited Mutual Advice (UMA). Applying the idea to DCSP agent, the strategy has several key points as listed below.

- When an agent cannot find a value in its variable's domain to reduce the number of conflicts, it will send the advices to the agents which are sharing the violated constraints with this agent, asking them to change the value of their variables. Since an advice may come from both agents sharing the same violated constraints, it is called *Unsolicited Mutual Advice*

4.2 The Strategy Details

- When an agent receives an advice and it can find a value to reduce the number of conflicts, it will voluntarily decide to change its variable to the new value.
- When an agent receives an advice and it cannot find any value to reduce the number of conflicts, it will reluctantly choose another value in its variable's domain and decide to adopt the new value.
- All the decisions from the agents need to be mediated before the corresponding actions are executed. More specifically, only some of the decisions will be applied because if two agents change the values of their variables concurrently, the constraints between them may remain violated. There are two main criteria to select the decisions to be applied: (i) voluntary decision is preferred to reluctant decision and (ii) decision which reduces the highest number of constraint violations will be selected.

Figures 4.1, 4.2 and 4.3 illustrate the decision process of DCSP agents adopting UMA.

4.2 The Strategy Details

Figure 4.4 presents the BDI negotiation model incorporating the *Unsolicited Mutual Advice*(UMA) strategy.

Unlike when using the existing strategies formulated in our proposed framework in Chapter 3, a DCSP agent using UMA will not only send out a negotiation message when concluding its Intention step, but also when concluding its Desire step. The negotiation message that it sends out to conclude the Desire step constitutes an *unsolicited advice* for all its neighbors. In turn,

4.2 The Strategy Details

Figure 4.1: Stronger voluntary decision due to the unsolicited advices

4.2 The Strategy Details

Figure 4.2: Reluctant decision due to the unsolicited advices

4.2 The Strategy Details

Figure 4.3: Mediating two voluntary decisions

4.2 The Strategy Details

Figure 4.4: BDI protocol with *Unsolicited Mutual Advice* strategy

the agent will wait to receive unsolicited advices from all its neighbors, before proceeding on to determine its intention.

For agent i , beginning initially with $(w_l = 1, (0 \leq l < m), p_i = i, (0 \leq i < n))$ and F_i contains all the agents who share the constraints with agent i , its BDI-driven UMA strategy is described as follows.

Step 1 - Percept: Update P_i upon receiving the *info* messages from the neighbors (in F_i). Update C_i to be the list of constraints relevant to agent i .

Step 2 - Belief: The belief function $GB(P_i, C_i)$ will return a value

4.2 The Strategy Details

$b_i \in \{0, 1, 2\}$, decided as follows:

- $b_i = 0$ when agent i can find an option to reduce the number of violated constraints in C_i , i.e., if $\exists a \in D_i, S_i(a) < S_i(v_i)$ and the assignment $x_i = a$ and the current variable assignments of its neighbors do not form a local state stored in a list called *bad states list* (initially this list is empty).
- $b_i = 1$ when it cannot find a value a such that $a \in D_i, S_i(a) < S_i(v_i)$, and the assignment $x_i = a$ and the current variable assignments of its neighbors do not form a local state stored in the *bad states list*.
- $b_i = 2$ when its current assignment is an optimal option, i.e., if $S_i(v_i) = 0$.

Step 3 - Desire: The desire function GD (b_i) will return a desire set DS, decided as follows:

- If $b_i = 0$, then $DS = \{a \mid a \neq v_i, S_i(a) < S_i(v_i) \text{ and } (S_i(v_i) - S_i(a)) \text{ is maximized} \}$ and the assignment $x_i = a$ and the current variable assignments of agent i 's neighbors do not form a state in the *bad states list*. In this case, DS is called a *set of voluntary desires*. $\max\{(S_i(v_i) - S_i(a))\}$ will be referenced by h_i^{max} in subsequent steps, and it defines the maximal reduction in constraint violations. It is also referred to as an *improvement*).
- If $b_i = 1$, then $DS = \{a \mid a \neq v_i, S_i(a) \text{ is minimized} \}$ and the assignment $x_i = a$ and the current variable assignments of agent i 's neighbors do not form a state in the *bad states list*. In this case, DS is called a *set of reluctant desires*.

4.2 The Strategy Details

- If $b_i = 2$, then $DS = \emptyset$.

Following, if $b_i = 0$, agent i will send a negotiation message containing h_i^{max} to all its neighbors. This message is called a *voluntary advice*. If $b_i = 1$, agent i will send a negotiation message called *change advice* to the neighbors in F_i who share the violated constraints with agent i .

Agent i receives advices from all its neighbors and stores them in a list called A , before proceeding to the next step.

Step 4 - Intention: The intention function $GI(DS, A)$ will return an intention, decided as follows:

- If there is a *voluntary advice* from an agent j which is associated with $h_j^{max} > h_i^{max}$, assign *nil* as the intention.
- If $DS \neq \emptyset$, DS is a set of voluntary desires and h_i^{max} is the biggest improvement among those associated with the *voluntary advices* received, select an arbitrary value (say, v'_i) from DS as the intention. This intention is called a *voluntary intention*.
- If $DS \neq \emptyset$, DS is a set of reluctant desires and agent i receives some *change advices*, select an arbitrary value (say, v'_i) from DS as the intention. This intention is called *reluctant intention*.
- If $DS = \emptyset$, then assign *nil* as the intention.

Following, if the improvement h_i^{max} is the biggest improvement and equal to some improvement associated with one of the *voluntary advices* received, agent i will send its computed intention to all its neighbors. If agent i has a reluctant intention, it will also send this intention to all its neighbors. In both cases, agent i will attach the number of the *change advices* received

4.2 The Strategy Details

in the current negotiation round with its intention. In return, agent i will receive the intentions from its neighbors before proceeding to Mediation step.

Mediation: If agent i does not send out its intention before this step, i.e., the agent has either a *nil* intention or a voluntary intention with biggest improvement, it will proceed to next step. Otherwise, agent i will select the best intention among all the intentions received, including its own (if any). The criteria to select the best intention are listed, applied in descending order of importance as follows.

- A voluntary intention is preferred over a reluctant intention.
- A voluntary intention (if any) with biggest improvement is selected.
- If there is no voluntary intention, the reluctant intention with the lowest number of constraint violations is selected.
- The intention from an agent who has received a higher number of *change advices* in the current negotiation round is selected.
- Intention from an agent with highest priority is selected.

If the selected intention is not agent i 's intention, it will cancel its intention.

Step 5 - Execution: If agent i does not cancel its intention, it will update its variable assignment with the intended value.

Termination Condition: Since each agent does not have full information about the global state, it may not know when it has reached a solution, i.e., when all the agents are in a global stable state. Hence an *observer* is needed that will keep track of the negotiation messages communicated in the environment. Following a certain period of time when there is no more

4.3 The Completeness Property

message communication (and this happens when all the agents have no more intention to update their variable assignments), the observer will inform the agents in the environment that a solution has been found.

4.3 The Completeness Property

The proof of Completeness property is as follows:

- Each agent will maintain a list of bad states and will not select a value that will make the agent itself and its neighbors go to a known bad state. In a case that an agent cannot find any value in its domain to form its Desire set, it will have a *nil* intention. Such intention will be ignored in the mediation step which means other neighboring agents will have chances to execute their intentions. As a result, a local group of agents will keep moving to different local states without going back to the states which were known to have some constraint violations.
- If there is no solution, there exists a group of agents in which all the agents have tried all of the values in their domains and still cannot find a combination that satisfy all the relevant constraints. Let's call this group as A. We have established that as the mechanism executes, a group will not go back to a known bad state. Hence group A will eventually come to a state from which no agent in the group can find a value for its Desire set because any choice of value will bring the group to a known bad state. As a result, *if an agent cannot find a value for its Desire set and neither do its neighboring agents, there is no solution.*
- If there is a solution, in every group of agents there must be a stable state in which all the relevant constraints are satisfied. And this state

4.4 An Example

Figure 4.5: Example problem

will not be in the list of bad states. As all the bad states are stored, the number of possible states will reduce significantly to the list of stable states from which a global stable state can be formed and a solution is found.

4.4 An Example

To illustrate how UMA works, consider a 2-color graph problem [31] as shown in Figure 4.5. In this example, each agent has a color variable representing a node. There are 10 color variables sharing the same domain {Black, White}.

The following records the outcome of each step in every negotiation round executed.

Round 1:

Step 1 - Percept: Each agent obtains the current color assignments of those nodes (agents) adjacent to it, i.e., its neighbors’.

Step 2 - Belief: Agents which have positive improvements are agent 1 (this agent believes it should change its color to White), agent 2 (this agent believes it should change its color to White), agent 7 (this agent believes

4.4 An Example

it should change its color to Black) and agent 10 (this agent believes it should change its value to Black). In this negotiation round, the improvements achieved by these agents are 1. Agents which do not have any improvements are agents 4, 5 and 8. Agents 3, 6 and 9 need not change as all their relevant constraints are satisfied.

Step 3 - Desire: Agents 1, 2, 7 and 10 have the voluntary desire (White color for agents 1, 2 and Black color for agents 7, 10). These agents will send the *voluntary advices* to all their neighbors. Meanwhile, agents 4, 5 and 8 have the reluctant desires (White color for agent 4 and Black color for agents 5, 8). Agent 4 will send a *change advice* to agent 2 as agent 2 is sharing the violated constraint with it. Similarly, agents 5 and 8 will send *change advices* to agents 7 and 10 respectively. Agents 3, 6 and 9 do not have any desire to update their color assignments.

Step 4 - Intention: Agents 2, 7 and 10 receive the *change advices* from agents 4, 5 and 8, respectively. They form their voluntary intentions. Agents 4, 5 and 8 receive the *voluntary advices* from agents 2, 7 and 10, hence they will not have any intention. Agents 3, 6 and 9 do not have any intention. Following, the intention from the agents will be sent to all their neighbors.

Mediation: Agent 1 finds that the intention from agent 2 is better than its intention. This is because, although both agents have voluntary intentions with improvement of 1, agent 2 has received one *change advice* from agent 4 while agent 1 has not received any. Hence agent 1 cancels its intention. Agent 2 will keep its intention.

Agents 7 and 10 keep their intentions since none of their neighbors has an intention.

4.4 An Example

The rest of the agents do nothing in this step as they do not have any intention.

Step 5 - Execution: Agent 2 changes its color to White. Agents 7 and 10 change their colors to Black.

The new state after round 1 is shown in Figure 4.6.

Figure 4.6: The graph after round 1

Round 2:

Step 1 - Percept: The agents obtain the current color assignments of their neighbors.

Step 2 - Belief: Agent 3 is the only agent who has a positive improvement which is 1. It believes it should change its color to Black. Agent 2 does not have any positive improvement. The rest of the agents need not make any change as all their relevant constraints are satisfied. They will have no desire, and hence no intention.

Step 3 - Desire: Agent 3 desires to change its color to Black voluntarily, hence it sends out a *voluntary advice* to its neighbor, i.e., agent 2. Agent 2 does not have any value for its reluctant desire set as the only

4.4 An Example

option, Black color, will bring agent 2 and its neighbors to the previous state which is known to be a bad state. Since agent 2 is sharing the constraint violation with agent 3, it sends a *change advice* to agent 3.

Step 4 - Intention: Agent 3 will have a voluntary intention while agent 2 will not have any intention as it receives the *voluntary advice* from agent 3.

Mediation: Agent 3 will keep its intention as its only neighbor, agent 2, does not have any intention.

Step 5 - Execution: Agent 3 changes its color to Black.

The new state after round 2 is shown in Figure 4.7.

Round 3: In this round, every agent finds that it has no desire and hence no intention to revise its variable assignment. Following, with no more negotiation message communication in the environment, the observer will inform all the agents that a solution has been found.

Figure 4.7: The solution obtained

4.5 Quick comparisons among DCSP algorithms

Algorithm	ABT	AWC	DBO	APO	UMA
Properties					
Communication	Asynchronous	Asynchronous	Synchronous	Synchronous	Synchronous
Priority	Fixed	Dynamic	Fixed	Fixed	Fixed
Constraint Weight	Fixed	Fixed	Dynamic	Fixed	Fixed
Mediation Step	Separated	Separated	Integrated	Integrated	Integrated
Completeness	Yes	Yes	No	Yes	Yes

Table 4.1: Comparison between DCSP algorithms

4.5 Quick comparisons among DCSP algorithms

As mentioned in Chapter 1, one advantage of unifying DCSP algorithms in one framework is so that they can be compared with respect to aspects other than computational performance which in our opinion, can be more clearly understood from a common BDI agent perspective. Table 4.1 shows the comparison between five algorithms: ABT, AWC, DBO, APO and UMA. The comparison results were obtained by analyzing the details of the strategies adopted by these algorithms after they were reformulated using the new BDI framework. E.g., the priority of agent i is $p_i = i$ does not change in the execution of any strategies except AWC. That is why the priority is dynamic for AWC. Note that although APO stands for ‘Asynchronous Partial Overlay’, its communication type is classified as Synchronous because the mediator needs to wait for the reply from its neighboring agents before it can start the mediation session.

Chapter 5

Performance Evaluation and Discussions

In this chapter, the empirical results are presented. Several experiments and evaluations on the performance of different strategies were carried out using a simulator which was implemented as part of the work. Some observations and conclusions based on the results are also presented. In general, the results show that the Unsolicited Mutual Advice strategy produces a better performance than the Asynchronous Weak Commitment Search strategy, especially for a critically difficult problem where the ratio of the number of constraints to the number of variables is high. In general, Asynchronous Backtracking is the least efficient strategy on average.

5.1 DCSP Simulations

To facilitate performance evaluation, a simulator was implemented in Java [30]. Figure 5.1 shows the agent implementation model in the simulated environment. The design follows an Object-oriented approach [8]. Each agent has an object called Agent Body which possesses several *properties* to represent the agent's attributes such as priority and provides a certain number of *methods* to provide access to its properties. Importantly, for each agent there are two active entities: Reader and Writer. Reader and Writer will handle two buffers of the agent, Inbox and Outbox, which are used to store received and to-be-sent messages. Messages are sent and received

5.1 DCSP Simulations

Figure 5.1: The implementation model of an agent used in the simulator

following the common BDI interaction protocol. Reader reads the incoming messages in Inbox and uses an input strategy to process the information contained in the messages. Then it will generate messages to answer and inform other agents about its change in value. To do this, Reader puts the messages in Outbox by calling Send() method in Agent Body. Writer will then be notified, it will take the messages from Outbox and put them in Inboxes of the recipients by calling Receive() method of those agents. The details of the strategies were implemented inside the source code of Reader.

In terms of Java implementation, Agent Body is a passive entity which contains information of an agent and several methods to access them. As the information is shared, some of them cannot be accessed concurrently, e.g. at a moment, only one worker can access Inbox to take out or put in messages.

5.1 DCSP Simulations

Therefore Agent Body acts a *Java monitor* with *synchronized* methods [23]. Reader and Writer are active entities implemented as *Java threads* that access information in Agent Body in a mutually exclusive fashion and carry out some actions as needed.

There are several advantages of this design:

- The tasks of sending and receiving messages tasks in one agent are independent and concurrent.
- The incoming messages need to be processed in the sequence in which they are received. This requirement is met since the data structure of Inbox is a FIFO queue [2].
- The outgoing messages need to be sent in the sequence in which Reader created them. The data structure of Outbox is also a FIFO queue, thus satisfying this condition.
- Agent Body, Inbox and Outbox structures are independent of the strategy to be evaluated. Since the function of Writer is only to send out the communication messages, it can be used with different strategies without any changes required. Only Reader needs to be modified when running a new strategy.
- Extending to real distributed environment is easy by adding network communication functions to method Send() and Receive().

This thesis does not go deeply into the implementation details of each component of the simulator. Figure 5.2 shows the Java prototype of Reader from which we can see that the implementation model is similar to the theoretical model proposed in Chapter 3.

5.1 DCSP Simulations

```

public class BDIREader extends Thread {

    public BDIREader(BDI Agent boss)
    {
        // Initialize necessary variables where boss
        // is the agent whom the reader belongs to
    }

    private void init()
    {
        // Send out messages to inform agents about the initial value
        // Receive messages and store initial values of agents
    }

    private void initRound()
    {
        // This method is used to init a negotiation round
        // Some variables need to be reset before a new round
    }

    private void generateBelief()
    {
        // This method generate the belief and store in a variable
    }

    private void generateDesire()
    {
        // Generate desire based on the belief calculated
    }

    private void generateIntention()
    {
        // Generate Intention
    }

    public void run()
    {
        init();

        while (true)
        {
            initRound();

            // Read messages from inbox...

            generateBelief();
            generateDesire();
            generateIntention();

            // Additional processing

            if (solutionFound || noSolution)
            {
                return;
            }
            else
            {
                // send out messages to inform about the new value
            }
        }
    }
}

```

Figure 5.2: The implementation prototype of Reader in the simulator

5.2 Performance Evaluation

5.2 Performance Evaluation

To facilitate credible comparisons among existing strategies, we measured the execution time in terms of computational cycles as defined in [38], and built a simulator that could reproduce the published results [38] for ABT and AWC. The definition of a computational cycle is as follows:

- In one cycle, each agent receives all the incoming messages, performs local computation and sends out a reply.
- A message which is sent at time t will be received at time $t + 1$. The network delay is ignored.
- Each agent has its own clock. The initial clock value is 0. Agents attach their clock values as time-stamps in the outgoing messages and use the time-stamps in the incoming messages to update their own clock values.

Four benchmark problems [31] were considered, namely, n -queens and node coloring for sparse, dense and critical graphs. For each problem, a finite number of test cases were generated for various problem sizes n . The maximum execution time limit was set to 10000 cycles for node coloring for critical graphs and 1000 cycles for other problems. The simulator program was terminated after this period and the algorithm was considered to fail a test case if it did not find a solution by then. In such a case, the execution time for the test was counted as the maximum time limit set. The ratio of the number of test cases completed successfully (within the maximum time limit) to the total number of test cases is recorded and tabulated along with the number of cycles for each algorithm.

5.2 Performance Evaluation

N-Problem Size	ABT		AWC		UMA	
	Ratio	Cycles	Ratio	Cycles	Ratio	Cycles
10	100%	144.6	100%	46.8	100%	68.6
50	50%	434	100%	54.8	100%	98.4
100	10%	943.7	100%	61.2	100%	81.1

Table 5.1: Comparison in performance of three algorithms in Distributed n -queens

Figure 5.3: Relationship between execution time and problem size

5.2.1 Evaluation with n -queens problem

The n -queens problem is a traditional problem of constraint satisfaction. In this problem, n queens must be placed on an $n \times n$ board so that no queen is in the position endangered by other queens. 10 test cases were generated for each problem size $n \in \{10, 50, 100\}$. Table 5.1 displays the average number of computational cycles for the three algorithm: ABT, AWC and UMA.

Figure 5.3 shows the execution time in term of computational cycles for different problem sizes when ABT, AWC and UMA were run.

5.2 Performance Evaluation

Algorithm		n			
		60	90	120	150
AWC	Ratio	100%	100%	100%	100%
	Cycles	57.0	80.3	113.5	180.2
DBO	Ratio	100%	100%	100%	100%
	Cycles	82.4	120.3	200.7	298.5
UMA	Ratio	100%	100%	100%	100%
	Cycles	60.0	66.0	95.2	152.6

Table 5.2: Comparison in performance in Sparse Graph Coloring Problem

5.2.2 Evaluation with graph coloring problem

The graph coloring problem is known to be NP-complete [24] It can be characterized by three parameters: (i) the number of colors k , (ii) the number of nodes/agents n and (iii) the number of links m . Based on the ratio m/n , the problem can be classified into three types [37]: (i) sparse (with $m/n = 2$), (ii) critical (with $m/n = 2.7$ or 4.7) and (iii) dense (with $m/n = (n - 1)/4$). For this problem, we did not include ABT in our empirical results as its failure rate was found to be very high. This poor performance of ABT was expected since the graph coloring problem is more difficult than the n -queens problem, on which ABT already did not perform well (see Figure 5.3). Instead of ABT, DBO was used in our evaluation with this type of problem.

The sparse and dense (coloring) problem types are relatively easy while the critical type is difficult to solve. In the experiments, we fixed $k = 3$. 10 test cases were created using the method described in [20] for each value of $n \in \{60, 90, 120\}$, for each problem type.

The simulation results for each type of problem are illustrated in Figures 5.4 - 5.6. The numerical results are shown in Tables 5.2, 5.3 and 5.4 respectively.

5.3 Discussions

Figure 5.4: Comparison between AWC, DBO and UMA (sparse graph coloring)

Algorithm		n		
		60	90	120
AWC	Ratio	100%	96%	65%
	Cycles	513.8	2002.6	5637.4
DBO	Ratio	100%	100%	100%
	Cycles	187.5	603.4	792.1
UMA	Ratio	100%	100%	100%
	Cycles	170.0	513.8	647.2

Table 5.3: Comparison in performance in Critical Graph Coloring Problem

5.3 Discussions

5.3.1 Comparison with ABT and AWC

Figure 5.4 shows that the average performance of UMA is slightly better than AWC for the sparse problem. UMA outperforms AWC in solving the critical problem as shown in Figure 5.5. It was observed that the latter strategy failed in some test cases. However, as seen in Figure 5.6, both the strategies are very efficient when solving the dense problem, with AWC showing slightly

5.3 Discussions

Figure 5.5: Comparison between AWC, DBO and UMA (critical graph coloring)

Algorithm		n		
		60	90	120
AWC	Ratio	100%	100%	100%
	Cycles	8.5	7.8	12.0
DBO	Ratio	100%	100%	100%
	Cycles	24.0	30.6	42.7
UMA	Ratio	100%	100%	100%
	Cycles	23.2	28.8	41.2

Table 5.4: Comparison in performance in Dense Graph Coloring Problem

better performance.

The performance of UMA, in the worst (time complexity) case, is similar to that of all evaluated strategies. The worst case occurs when all the possible global states of the search are reached. Since only a few agents have the right to change their variable assignments in a negotiation round, the number of redundant computational cycles and *info* messages is reduced. As we observe from the backtracking in ABT and AWC, the difference in the ordering of incoming messages can result in a different number of computational cycles to be executed by the agents.

5.3 Discussions

Figure 5.6: Comparison between AWC, DBO and UMA (dense graph coloring)

5.3.2 Comparison with DBO

The computational performance of UMA is arguably better than DBO for the following reasons:

- UMA can guarantee that there will be a variable reassignment following every negotiation round whereas DBO cannot.
- UMA introduces one more communication round trip (that of sending a message and awaiting a reply) than DBO, which occurs due to the need to communicate unsolicited advices. Although this increases the communication cost per negotiation round, we observe from our simulations that the overall communication cost incurred by UMA was lower due to the significantly lower number of negotiation rounds. This can be explained as follows.

Consider an agent in a particular problem instance with N neighboring agents. For the DBO strategy, in one negotiation round the agent needs

5.3 Discussions

to send messages to all of its neighbors twice. So the total number of messages sent out in one negotiation round by one agent is $2 \times N$.

For the UMA strategy, in one negotiation round the agent may need to send out messages up to 3 times, depending on the belief value it has in that round:

- If the belief value is 0 which means the agent has some constraint violations and it can find a value to resolve them, N *voluntary advices* will be sent out to all the neighboring agents. Since each of the N neighboring agents can also have belief values 0, 1 or 2, we assume that $N/3$ agents will have belief 0 and will send some *voluntary advices* out as well. As a result, this agent will need to mediate its intention with these $N/3$ neighbors and finally send out N *info* messages to all its neighbors. In this case, the total number of messages sent out by this agent is $N + N/3 + N = 7 \times N/3$.
- If the belief value is 1 which means the agent has some constraint violations and it cannot find a value to resolve them, we assume that the chance for one of the N neighbors to share a constraint violation with this agent is equal to the chance that this neighbor does not share any constraint violation at all with the agent. In this case, the average number of neighboring agents that share at least one constraint violation with this agent is $N/2$. As a result, $N/2$ *change advices* will need to be sent out.

Since this agent cannot find a value to satisfy all of its related constraints, it will not have any intention if it receives a *voluntary advice*. Otherwise, it may need to mediate its intention with the $N/2$ neighbors. We assume that both cases have an equal chance

5.3 Discussions

to happen. According to probability theory, the average number of messages that need to be sent out for intention mediation is $1/2 \times 0 + 1/2 \times N/2 = N/4$. Finally, the agent will need to send out N *info* messages, each to one of its N neighboring agents. Hence the average total number of messages sent by this agent is $N/2 + N/4 + N = 7 \times N/4$.

- If the belief value is 2 which means the agent does not have any constraint violation, it will only need to send out N *info* messages to all of its neighbors. In this case, the number of messages sent out by this agent is N .

Taking the average of the three total values above, the number of messages sent out by one agent in a negotiation round is $61 \times N/36 \approx 5 \times N/3$.

Our benchmark results show that the number of computational cycles of UMA and DBO are very close to each other (except for sparse graph coloring problem where UMA has a better performance). Let us assume that they are the same and denote this value as M . From the definition of a computational cycle, we have:

- For DBO, the estimated number of negotiation rounds is $M/2$.
- For UMA, the estimated number of negotiation rounds is $M/3$.

Hence the estimated average number of messages sent out by one agent using DBO in an arbitrary problem instance is

$$M/2 \times 2N = M \times N.$$

The estimated average number of messages sent out by one agent using UMA in an arbitrary problem instance is

5.3 Discussions

$$M/3 \times 5 \times N/3 = 5/9 \times M \times N.$$

Since the above explanation is done based on estimation, the actual numbers can be different depending on each individual problem instance.

- Using UMA, in the worst case, an agent will only take 2 or 3 communication round trips per negotiation round, following which the agent or its neighbor will do a variable assignment update. Using DBO, this number of round trips is uncertain as each agent might have to increase the weights of the violated constraints until an agent has a positive improvement; this could result in an infinite loop [37].

In addition, one important advantage of UMA over DBO is the possession of the completeness property. Due to the possible infinite loop described above, DBO does not have the completeness property [42].

Chapter 6

Conclusion & Future Work

6.1 Conclusion

As a fundamental framework for formalizing many existing and emerging problems in a distributed environment, distributed constraint satisfaction is an increasingly important research problem in multiagent systems [37]. In attempting to unify existing and new distributed constraint algorithms, this thesis makes the following contributions to the field of multiagent systems.

- In Chapter 3, a unified framework is established for the distributed constraint satisfaction problem (DCSP). Following the negotiation model [26], the framework consists of two layers: a common interaction protocol and a set of strategies. The protocol defines the steps or rules of interactions that the agents involved in the negotiation process must follow. In other words, the protocol specifies the general behavior of an agent. Each strategy determines the decisive actions to be taken in each step of the common protocol. By formulating the protocol in terms of the model-theoretic concept of BDI [5], it has been shown that although the approaches are different, the behavior of each agent in existing DCSP algorithms can be prescribed by the fundamental components (see Figure 3.1) of the BDI negotiation framework. The differences among these algorithms lie in the different strategies or decisions taken in each negotiation step. We believe that the BDI char-

6.2 Future Work

acterization of agent negotiation for existing DCSP algorithms helps to better understand these algorithms, by clearly showing the *structural commonality*, namely, the BDI protocol.

- The BDI negotiation model provides the opportunity to extend the family of DCSP algorithms. Guided by the model, a new strategy called Unsolicited Mutual Advice (UMA) has also been formulated and evaluated (Chapter 4). Empirical results show that the new strategy outperforms the existing ones (Chapter 5) and enjoys certain advantages such as security. Importantly, it is also shown to possess the completeness property, that of returning a solution provided one exists.

6.2 Future Work

The research work in this thesis ushers in a conceptually clearer approach to formalizing DCSP algorithms as BDI protocol-driven strategies. Under the proposed approach, we suggest some possible directions for future research and development.

- The idea of DCSP agents using different BDI protocol-driven strategies in the same environment (i.e., resolving the same set of inter-agent constraints) will also be investigated. This will generalize the current stage-of-the-art, where DCSP agents (using the same mechanism and therefore) adopt the same strategy in a given environment.
- The UMA strategy can be extended to the case for multi-variable agents. A performance evaluation along with the existing strategies for multi-variable agents [40] will need to be done.

6.2 Future Work

- The proposed BDI negotiation framework can be extended to formalize related problems such as partial DCSP [10] and the distributed constraint optimization problem (DCOP) [21].
- In the proposed BDI negotiation framework, new strategies in addition to UMA may be discovered. Besides, insights based on the BDI concepts could also help enhance the performance of existing DCSP algorithms.
- The strategies were evaluated solely based on the number of computational cycles. A more complete evaluation where network delay and the number of messages exchanged among agents are considered will better reflect the performance of these strategies in practice.

Author's Publications

Parts of this thesis are documented in the following publications:

1. Bao Chau Le Dinh and KiamTian Seow, "Unifying Distributed Constraint algorithms in a BDI Negotiation Framework," In *Proceedings of Sixth International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2007)*, 2007. Honolulu, Hawaii, USA. (Acceptance rate: 22% or 119/531)
2. Bao Chau Le Dinh and KiamTian Seow, "Towards A Unified BDI Negotiation Framework for Distributed Constraint Agents," Journal Paper submitted, Dec 2007.

References

- [1] Lars Braubach Alexander Pokahr and Winfried Lamersdorf. A flexible bdi architecture supporting extensibility. In *Proceedings of the 2005 IEEE/ACM International Conference on Intelligent Agent Technology (IAT'05)*, 2005.
- [2] Jeffrey D. Ullman Alfred V. Aho and John E. Hopcroft. *Data Structures and Algorithms*. Addison-Wesley, 2000. Addison-Wesley Series in Computer Science and Information Pr.
- [3] Michael Benisch and Norman Sadeh. Examining dcsp coordination tradeoffs. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'06)*, pages 1405 – 1412, 2006.
- [4] Ken Binmore. *Fun and Games, A Text on Game Theory*. Basil Blackwell, Cambridge, Massachusetts, 1990.
- [5] Michael E. Bratman. *Intentions, Plans and Practical Reason*. Harvard University Press, Cambridge, M.A, 1987.
- [6] John Davin and Pragnesh Jay Modi. Impact of problem centralization in distributed constraint optimization algorithms. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'02)*, Ultrecht, Netherlands, July 2005. ACM Press.
- [7] Eithan Ephrati and Jeffrey S. Rosenschein. The clarke tax as a consensus mechanism among automated agents. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI'91)*, pages 173–178, July 1991.
- [8] Ralph Johnson Erich Gamma. Richard Helm and John Vlissides. *Design Patterns*. Addison Wesley, 1994.
- [9] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI'87)*, pages 677–682. Seattle, WA, 1987.

REFERENCES

- [10] Katsutoshi Hirayama and Makoto Yokoo. Distributed partial constraint satisfaction problem. In *Proceedings of the Third International Conference on Principles and Practice of Constraint Programming (CP'97)*, pages 222–236, Linz, Austria, 1997.
- [11] Zhang Huiliang and Huang Shell Ying. A parallel bdi agent architecture. In *Proceedings of the 2005 IEEE/ACM International Conference on Intelligent Agent Technology (IAT'05)*, pages 157–160, 2005.
- [12] M. Pischel J. P. Muller and M. Thiel. *Agent-oriented programming*, volume 60. 1993.
- [13] M. Pischel J. P. Muller and M. Thiel. *Modelling reactive behaviour in vertically layered agent architectures*, volume 890. Springer-Verlag: Berlin, Germany, 1995.
- [14] N.R. Jennings. Specification and implementation of belief, desire, joint-intention architecture for collaborative problem solving. *Journal of Intelligent and Cooperative Information System (JICIS)*, 2(3):289–318.
- [15] N.R. Jennings. Specification and implementation of a belief-desire-intention architecture for collaborative problem solving. In *International Journal of Intelligent and Cooperative Information Systems (JICIS)*, volume 2, pages 289–318, 1993.
- [16] Victor R. Lesser and Daniel D. Corkill. The distributed vehicle monitoring testbed. *Artificial Intelligence Magazine* 4, pages 63–109, Fall 1983.
- [17] Roger Mailler and Victor Lesser. Using cooperative mediation to solve distributed constraint satisfaction problems. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04)*, pages 446–453, New York, USA, 2004.
- [18] Roger Mailler, Regis Vincent, Victor Lesser, Tim Middlekoop, and Jiaying Shen. Soft Real-Time, Cooperative Negotiation for Distributed Resource Allocation. *AAAI Fall Symposium on Negotiation Methods for Autonomous Cooperative Systems*, pages 63–69, November 2001.
- [19] D. Israel Michael E. Bratman and M. E. Pollack. *Plans and resource-bounded practical reasoning*, volume 4. 349-355.
- [20] S. Minton, M. D. Johnson, A. B. Philips, and P. Laird. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, e58(1-3):161–205, 1992.
- [21] P.J. Modi. *Distributed Constraint Optimization for Multi-agent Systems*. PhD thesis, Department of Computer Science, University of Southern California, 2003.

REFERENCES

- [22] Pragnesh Jay Modi, Hyuckchul Jung, Milind Tambe, Wei-Min Shen, and Shriniwas Kulkarni. Dynamic distributed resource allocation: A distributed constraint satisfaction approach. In *Pre-proceedings of the Eighth International Workshop on Agent Theories Architectures and Languages (ATAL'2001)*, pages 181–193, Seattle, WA, USA, 2001.
- [23] Scott Oaks and Henry Wong. *Java Threads*. O'reilly, 2004.
- [24] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
- [25] A. S. Rao and M. P. Georgeff. Modeling rational agents within a bdi-architecture. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 473–484. Morgan Kaufmann Publisher, San Mateo, CA, 1991.
- [26] Jeffrey S. Rosenschein and Gilad Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers*. The MIT Press, Cambridge, M.A, USA, 1994.
- [27] Hans Schlenker and Ulrich Geske. Simulating large railway networks using distributed constraint satisfaction. In *2nd IEEE International Conference on Industrial Informatics (INDIN'04)*, pages 441– 446, Berlin, Germany, 2004.
- [28] Kiam Tian Seow and Khee Yin How. Collaborative assignment : A multiagent negotiation approach using BDI concepts. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'02)*, pages 256–263, Palazzo Re Enzo, Bologna, Italy, July 2002. ACM Press.
- [29] K. Sycara, S. Roth, N. Sadeh, and M. Fox. Distributed constrained heuristic search. In *IEEE Transactions on Systems, Man and Cybernetics (SMC'1991)*, pages 1446–1461, Novemeber/December 1991.
- [30] Sun Micro Systems. *Java Development Kit (JDK) version 1.5*. 2004.
- [31] Edward Tsang. *Foundation of Constraint Satisfaction*. Academic Press, 1993.
- [32] Anh Nguyen Wayne Wobcke, Van Ho and Alfred Krzywicki. A bdi agent architecture for dialogue modelling and coordination in a smart personal assistant. In *Proceedings of the 2005 IEEE/ACM International Conference on Intelligent Agent Technology (IAT'05)*, 2005.
- [33] Gerhard Weiss, editor. *Multiagent System : A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, London, U.K, 1999.

REFERENCES

- [34] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents : Theory and practice. *The Knowledge Engineering Review (KER)*, 10(2):115–152, 1995.
- [35] Makoto Yokoo. Asynchronous weak commitment search for solving distributed constraint satisfaction problems. In *Proceedings of the First International Conference on Principles and Practice of Constraint Programming (CP'95)*, pages 88–102, Cassis, France, 1995. Springer-Verlag. Lecture Notes in Computer Science 976.
- [36] Makoto Yokoo. Weak-commitment search for solving constraint satisfaction problem. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94)*, pages 313–318, 1998.
- [37] Makoto Yokoo. *Distributed Constraint Satisfaction : Foundations of Cooperation in Multi-Agent Systems*. Springer Verlag, 2000. Springer Series on Agent Technology.
- [38] Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara. The distributed constraint satisfaction problem : Formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering (IEEE-TKDE)*, 10(5):673–685, September/October 1998.
- [39] Makoto Yokoo and Katsutoshi Hirayama. Distributed breakout algorithm for solving distributed constraint satisfaction problems. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS'96)*, pages 401–408, Tokyo, Japan, 1996.
- [40] Makoto Yokoo and Katsutoshi Hirayama. Distributed constraint satisfaction algorithm for complex local problems. In *Proceedings of the Third International Conference on Multiagent Systems (ICMAS'98)*, pages 372–379, Grenoble, France, 1998.
- [41] Makoto Yokoo, Koutarou Suzuki, and Katsutoshi Hirayama. Secure distributed constraint satisfaction: Reaching agreement without revealing private information. *Artificial Intelligence*, 161(1-2):229–246, 2005.
- [42] Weixiong Zhang and Lars Wittenburg. Distributed breakout revisited. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI'02)*, pages 352–357, Edmonton, Alberta, Canada, 2002.