Focused Decomposition: multi-agent distributed constraint optimisation with linear communication

Terence Heung-Wing Law

December 2007

Supervised by Dr. Adrian R. Pearce

A thesis submitted to the Melbourne School of Engineering in total fulfilment of the requirements of the degree of Master of Computer Science by research

Department of Computer Science & Software Engineering
Melbourne School of Engineering
The University of Melbourne
Victoria 3010, Australia

Printed on archive paper

Preface

This thesis is based, in part, on components of a publication accepted into the internationally refereed 2006 International conference on Intelligent Agent Technology in Hong Kong.

- Terence H.-W. Law and Adrian R. Pearce, A multi-stage graph decomposition algorithm for distributed constraint optimisation. In *Proceedings of IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT)*, IEEE Computer Society, Hong Kong, pp. 506–511, December 2006.

Declaration

This is to certify that

1. the thesis comprises only my original work except where indicated in the preface,
2. due acknowledgement has been made in the text to all other material used, and
3. this thesis is less than 30,000 words in length, exclusive of figures, tables, and bibliography.

Signed,

Terence Heung-Wing Law,

December 2007.

Acknowledgements

Many thanks to my research supervisor Adrian Pearce and research committee members Leon Sterling and James Bailey. Thanks also to Susannah Soon for her assistance and advice.

Contents

List of Figures

List of Tables

Abstract

In this thesis, a novel multi-agent cooperation strategy is proposed for solving distributed constraint optimisation, which guarantees optimality and improves computation efficiency, as well as having *linear* communication complexity. The new strategy of cooperative problem solving by distributed agents is best suited for large scale multi-agent applications requiring optimal solutions efficiently.

The challenge in solving constraint problems which have exponential solution space is to guarantee optimal solutions while maintaining efficiency. When constraint problems are extended with distributed constraints, the challenge is complicated with the introduction of communication between distributed agents. The agents need to communicate because each agent only has *incomplete local knowledge* that is insufficient for global optimality. Existing distributed constraint optimisation algorithms often run into communication explosion when trying to improve efficiency through distribution.

The key to achieving performance advantages in all three desirable aspects is the mechanism we called "Focused Decomposition", which is based on the insights in information sharing, graph decomposition and hierarchical bound propagation for efficiency.

Focused Decomposition incorporates the idea of optimal sub-solution and has evolved into a mechanism for distributed agents cooperatively exploring solution optimality in distributed sub-sections of an overall problem. Using sub-optimal solution qualities as optimal bounds facilitates solution comparisons, and is effective in reducing an optimal search's enormous search space, that leads to a more efficient optimal search. In addition, it avoids the communication of solutions during search operations. The application of the repeatedly-half principle during hierarchical decomposition is another factor in the improvement of efficiency.

Communication efficiency is achieved via a four stage process of problem identification, problem distribution, solution quality report, and solution propagation. Consequently, the inefficiency of fully distributed approaches is addressed through employing centralisation as part of the cooperation strategy. It achieves linear complexity in communication over the entire cooperative problem solving session.

The new approach results in a cooperative problem solving strategy for distributed constraint optimisation that is *optimal and complete*, and *improves efficiency in both computation and communication*. It can be applied to real world multi-agent applications such as distributed travel planning.

Experimental result shows that our new algorithm is faster than a recent competitive distributed constraint optimisation algorithm for solving MaxSAT graph colouring, is extremely communication efficient, and performs better than a centralised approach based on the same search strategy.

Chapter 1

Introduction

Problem solving has been one of the central topics in artificial intelligence research. From the General Problem Solver [Newell *et al.*, 1959] in the early 60's to General Game Playing [Genesereth *et al.*, 2005] recently, researchers are equipping computers with more clever problem representations, faster solution searching algorithms, and more sophisticated computer architectures that have led to increasing overall problem solving abilities.

A lot of computational problems can be effectively modelled by *constraints*, where a constraint represents a restriction or more generally a relation of some objects. Meanwhile, the rapidly growing Internet, the popularity of personal computing devices and the continuing advancement of computer communication networks provide the necessary conditions for adopting problem solving strategies to distributed settings.

The scenario of multiple agents situated in different environments cooperating to solve global problems can be modelled as a distributed constraint problem. Depending on the exact scenario we want to model, we may want to find a solution by solving the distributed constraint satisfaction problem (DCSP), or we may want to find the best solution by solving the distributed constraint optimisation problem (DCOP). DCSP is a decision problem that seeks to find an overall assignment of all variables that satisfies all of the constraints. DCOP is the optimisation variant of the DCSP. The goal of solving DCOP is to find an overall assignment of all variables that violates the least number of constraints, sometimes more generally as maximising a global utility function.

For example in Figure 1.1, Mary needs to attend an overseas conference. She asks her personal digital assistant for a travel plan that fits her schedule. Her personal digital assistant cooperates with the university's academic calendar agent and the airliners' flight

information agents to find a flight that fits all constraints (a satisfaction problem) and also the best price among different airliners (an optimisation problem). The events that need to be scheduled are distributed among different agents. Mary is happy because her digital agent has her travel plan set in half a minute!

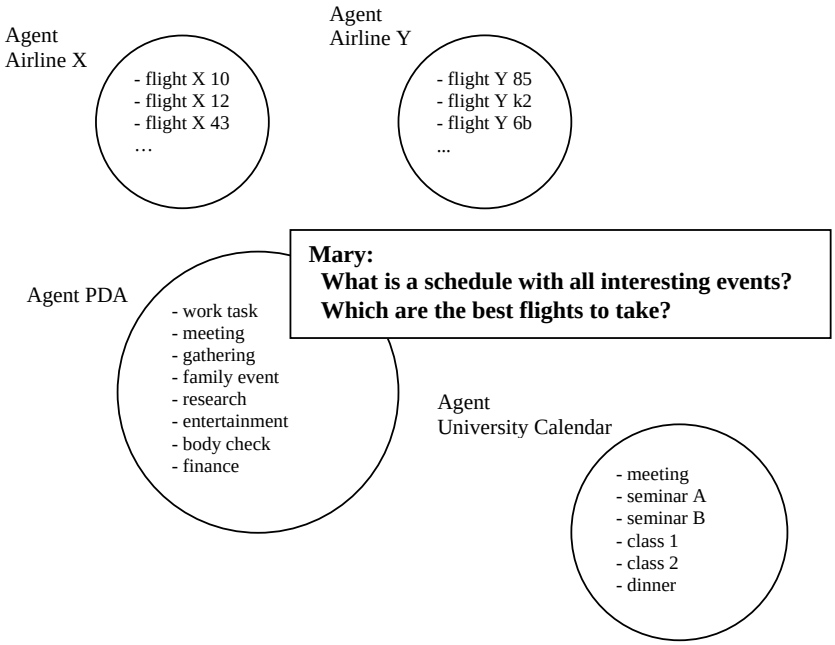

Figure 1.1: **Travel planning example:** an instance of distributed constraints. Mary wants to find a flight that fits all constraints (a DCSP problem) while also finding the best price among different airlines (a DCOP problem).

Distributed constraint optimisation problems, as simple as the above scenario, can be solved by a number of algorithms including Asynchronous Distributed Optimisation (Adopt) [Modi *et al.*, 2005] and Asynchronous Partial Overlay (OptAPO) [Mailler and Lesser, 2006a]. However, existing algorithms often are limited to solving problems with 30 or fewer variables that terminate in reasonable time. It is not only because of the inherent complexity of combinatorial optimisation problems (for many problems DCOP is NP-hard), but also a unique *communication explosion problem* faced by distributed agents, illustrated in Figure 1.2.

Unlike problem solving by a single agent, each distributed agent has a local view of the problem and has no global view of the overall problem. Agents therefore have to communicate extensively in order to find a solution that everybody agrees on. Our research

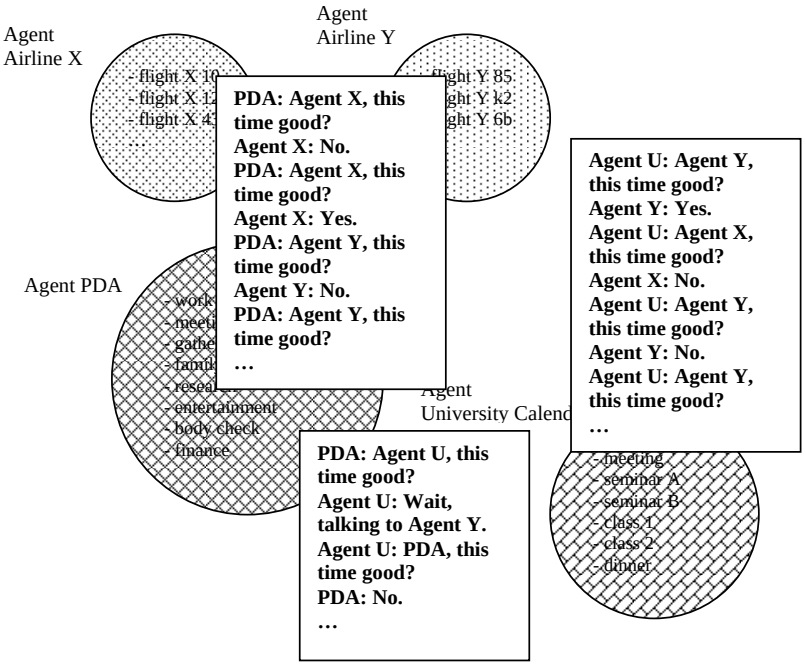

Figure 1.2: **Communication explosion:** communications involved in solving distributed constraints. In solving the DCSP and DCOP problems to serve Mary's queries, existing distributed algorithms frequently result in an explosion of communication. This problem arises due to distribution of variables among the agents.

directly addresses this challenge. We want to find a way of *solving distributed constraint optimisation problems efficiently, with optimal solution quality and minimal communication.*

In this thesis, we present the result of our investigation, a complete distributed search strategy which uses a "Focused Decomposition" mechanism that produces optimal solutions with minimal communication. The new strategy can be employed by cooperative agents for distributed constraint optimisation problems.

In order to limit the number of communicated messages, our key observation is that agents must share information. Algorithms must contain enough centralisation in order to limit communication sufficiently, while facilitating distributed and parallel search. Given that variables are distributed among agents, the real question is how to gather and assign multiple variables among agents in a principled way during the course of distributed problem solving (also emphasised in [Liu and Sycara, 1995]).

Another observation is in the nature of solving optimisation problems. A solution of a

satisfaction problem requires all constraints be satisfied, any solution with a violation can be rejected. When a satisfied solution is found, computation can terminate. Solving optimisation problems is different, there are many solutions with different qualities. When a solution is found, computation must continue to look for other better solutions for completeness of the algorithm. Therefore, optimisation problems are usually more complex than satisfaction problems. Algorithms for optimisation problems should gear towards comparing solutions, rather than rejecting non-solutions as for satisfaction problems.

A constraint optimisation problem is defined, and corresponding focused solutions, that facilitates the comparison of solutions of the same problem with different focuses. A distributed focused decomposition is also defined, which utilises graph-based decomposition that is originated from prior work on Role-graph hierarchies [Soon *et al.*, 2004].

The distributed focused decomposition approach employs the *repeatedly-half* principle to manage complexity by hierarchical distribution. Distributed agents work on different problem sizes with different focuses, essentially exploring different portions of the entire solution space independently without duplicating efforts. Solution qualities, but not solutions, are communicated. This leads to faster termination and better solution quality estimation while fitting nicely into a hierarchical graph-based decomposition. The result is a complete, communication efficient (with linear communication complexity) and computationally efficient algorithm for solving DCOPs.

The new approach is evaluated in two ways. Firstly, we analyse the properties and complexity of our algorithm. Secondly, we compare our approach empirically with existing algorithms. The approach is assessed quantitatively from three perspectives: (i) how efficient is the algorithm in terms of computation time when compared with another DCOP algorithm? (ii) how efficient is the algorithm in terms of communication when compared with another DCOP algorithm? (iii) how efficient is the algorithm in terms of computation time when compared with a centralised approach? The result is encouraging and shows good performance in all three cases.

1.1 Contributions

This thesis contributes to the advancement in distributed constraint processing in multi-agent systems research in the following aspects:

- Firstly, an updated formalism of the DCOP problem that not only specifies the initial

fully distributed setting of variables and constraints among agents, but also allows flexibility in the distribution of variables among agents.

- More importantly, a new algorithm for solving distributed constraint optimisation is introduced. The new algorithm (DistDecomp) utilises graph decomposition and partial ordering of variables to explore solution optimality among distributed agents to obtain guaranteed optimal solutions through a concept of focused search. The distributed algorithm effectively generates good quality sub-solutions for comparison and significantly reduces search space exploration by using effective bounds as heuristics. In addition, communication between agents is efficient, overcoming the problem of communication explosion usually exists in distributed optimisation solving. To the best of our knowledge, the DistDecomp algorithm is the first complete and optimal algorithm for solving distributed constraint optimisation problems with polynomial communication complexity (in relation to both number of messages and message size).

Moreover, there are some additional contributions in the investigation described in this thesis that also worth noting:

- It was found that the *generate and test* strategy often used by constraint satisfaction algorithms to reject non-solutions is often not efficient for solving constraint optimisation problems. Instead, constraint optimisation algorithms frequently perform better using a *generate and compare* strategy that compares qualified solutions. Therefore, simply extending a constraint satisfaction algorithm for constraint optimisation purpose may not necessarily results in good performance.

- Solution-based communication, where sub-solutions are communicated, is identified as an efficiency issue in optimisation solving because checking combinations of sub-solutions is expensive. Quality-based communication facilitates comparison of solutions. Sub-optimal solution qualities can be regarded as effective bounds for optimal solution. Distribution of optimality search is therefore possible for a decomposition-based approach.

- In addition to the *optimality/efficiency trade-off* common in constraint optimisation processing techniques, the *distribution/communication trade-off* has been analysed for distributed constraint processing. The less local knowledge available to an agent, the more communication is required for guaranteeing global satisfiability or optimality.

Increase in an agent's local knowledge about the global problem reduces communication requirement.

- Cooperative problem solving sessions can be organised through a multi-agent communication scheme. It coordinates problem identification and problem distribution and facilitates distributed asynchronous solving. Organising large scale cooperative problem solving sessions for computational difficult problems is facilitated through a simple communication protocol.

1.2 Organisation

The following chapters provide a detailed report on our investigation. Chapter 2 establishes a foundation by discussing constraint optimisation problems and presents a formal definition of distributed constraint optimisation problems, Chapter 3 discusses the problem of distribution and communication, and surveys existing approaches to solving distributed constraint optimisation problems (DCOP). Chapter 4 presents the Focused Decomposition solution. Chapter 5 evaluates the new approach and includes a complexity analysis and experimental results. Chapter 6 discusses the issues and variations of the approach. Finally, Chapter 7 concludes the contributions and discusses directions for future research.

Chapter 2

Distributed Constraint Processing

Constraints are general, flexible, simple and powerful abstractions for real world problems. By modelling computational problems as constraint problems, we can solve them with known proven techniques and fine tuned algorithms. From software engineering perspective, using constraint abstraction facilitates code reuse and reduces the cost of software implementation and maintenance. It has the benefit of clean separation of problem specification and computation of solutions. It also means that domain specific heuristics cannot be applied directly to problem solving algorithms, but the heuristics can be formulated as extra constraints to help the problem solving efforts.

For these reasons, commercial constraint processing systems have enjoyed success in solving combinatorial optimisation problems across many different industries such as banking, insurance, logistics, manufacturing, telecommunication, transportation, etc. These combinatorial problems are inherently complex, and getting solutions efficiently cannot be guaranteed.

When we add distribution in constraint processing, there are two different aspects of distributed constraint processing: (i) distributed constraints, and (ii) distributed processing. When we use the term distributed constraint processing, we mean both, that the *constraints are distributed* and the *processing is distributed*.

Distributed constraints are the result of modelling a problem that is distributed by nature. Distributed processing refers to the problem solving approach that incorporates multiple problem solvers. The problem solvers can operate and cooperate asynchronously but each of them lacks a global picture of the overall problem. The problem solvers are intelligent autonomous agents in a multi-agent system setting. Although we use intelligent

agents as problem solvers, we are not concern about the internal architecture of an agent in this thesis.

Let us consider Mary's overseas conference trip introduced in Chapter 1 (See Figures 1.1 & 1.2) as a distributed constraint problem. Recall that Mary's personal digital assistant (PDA) needs to find a flight that fits in Mary's schedule, with the help from the university's calendar agent and airliners' information agents. Mary's calendar time slots are modelled as event variables, each time slot can be occupied by at most one event. An event has a start time and duration. There are personal events, university events and travel events. If an event overlaps with another event, they are related by a conflict constraint.

After exchanging events information with other agents, Mary's PDA can find a suitable flight for Mary by applying a search algorithm on event variables and conflict constraints. Constraint processing systems not only can solve event scheduling problems, they can also be used to solve many seemingly different problems in various problem domains. For example, storing files in a file system, parking airplanes in an airport, directing water into reservoirs, just to name a few.

Constraints and variables are simple constructs, yet compositions of these simple abstractions are powerful enough for modelling complex problems. In this chapter, the basics of constraint problems are discussed and the DCOP problem is introduced. In Section 2.1, CSP and COP are presented more formally, and discussed from two different representations for constraints, i.e. graphs and relations. Then some common approaches to solving constraint optimisation are discussed. In Section 2.2, the distributed constraint optimisation problem is introduced. Section 2.3 summaries the background we presented.

2.1 Constraint Problems

A constraint problem consists of three things: variables, the potential values of the variables, and the constraints. Solving the problem involves finding a mapping from a set of values, according to some domains, to a set of variables. Typically, a *true* value means the constraint is satisfied, and a *false* value means the constraint is not satisfied, or violated. Constraints can be of varying arity, in terms of the number of variables involved in each constraint.

Definition 1 *Formally, a* constraint problem P *is a tuple* (X, D, C) *such that:*

- ***Variables***, $X = \{x_1, x_2, x_3, \ldots, x_n\}$ *is a set of variables;*
- ***Values***, $D = \{D_1, D_2, D_3, \ldots, D_n \mid D_i = \{d_{i1}, d_{i2}, d_{i3}, \ldots\}\}$ *is a set of domains, each domain* D_i *is a set of possible values,* d_{ij}*, of the corresponding variable* v_i*;*
- ***Constraints***, $C = \{c_1, c_2, c_3, \ldots | c_k = (D_{k1} \times D_{k2} \times D_{k3} \times \cdots \times D_{km}) \longrightarrow \{true, false\}\}$ *is a set of constraints on the values of the variables, each constraint* c_k *is a mapping from a set of values to a boolean. A* true *value means the constraint is satisfied, and a* false *value means the constraint is violated.* m *is the arity of the constraint, for a binary constraint, the arity* $m = 2$*.*

If the variables, x_i, can take constraints, c_i, as values the constraints are higher-order constraints. From a programming perspective, a constraint can be coded as a function with some conditional statements, or as a predicate that specifies the relation of some symbols.

Constraint satisfaction

A constraint satisfaction problem (CSP) is to find an assignment of all variables of a constraint problem that satisfies all constraints. For example, the graph colouring problem can be posed as an NP-complete CSP problem. If a satisfied solution is found, computation can terminate.

Programming satisfaction involves doing constraint checks on variable assignments.

Constraint optimisation

A constraint optimisation problem (COP) is to find an assignment of all variables of a constraint problem that violates the least number of constraints. Computation can terminate only if an optimal solution is found. For example, the optimisation variant of the graph colouring problem can be posed as an NP-hard constraint optimisation problem.

Normally, the problem is posed in terms of an objective function that maps assignments to a value which is sometimes used to evaluate different assignments. However, a more formal definition of the objective function is left until the distributed version of the problem is presented in Section 2.2.3. Let us now consider the problem of representing constraint problems.

2.1.1 Representing Constraint Problems

There are two types of abstractions commonly used for representing constraint problems: graphs and relations.

Graphs

Definition 2 *Formally, a* graph $G = (V, E)$ *contains vertices and edges such that:*

- ***Vertices*** $V = \{v_1, v_2, v_3, \ldots, v_n\}$ *is a set of vertices,*
- ***Edges*** $E = \{e_1, e_2, e_3, \ldots, e_m | e_k = (v_a, v_b, v_c, \ldots)\}$ *is a set of edges. Each edge of a simple graph connects two vertices. Each edge of a hyper-graph connects two or more vertices.*

Each vertex is typically labelled using a unique identifier. A graph can contain cycles, form trees, construct paths. Possibilities for using graphs in distributed computing show great promises. For example, in multi-agent systems research, [Soon, 2005] used graphs to represent agents' intention structures to facilitate teamwork and coordination. Despite the high expressiveness of graph representations, certain graph problems are also infamous for complexity of computation.

To represent a constraint problem as a graph, also called a constraint graph, the vertices are used to represent variables and the edges are used to represent *binary* constraints. To achieve this, the vertices and edges must be *attributed* with variables and constraints. [Hannebauer, 2002a] provides an overview of constraint graphs in constraint problems.

Definition 3 *A* constraint graph $G = (V, E)$ *consists of a set of vertices* $V = \{v_1, \ldots, v_n\}$*, where each vertex* v_i *is attributed with variable* $x_i \in X$ *and a set of edges* $E = \{e_1, e_2, \ldots | e_k = (v_i, v_j)\}$*. Each edge* e_k *is attributed with a binary constraint* $c_k \in C$ *over the variables of two domains* D_i *and* D_j*, where* $c_k = (D_i \times D_j)$*.*

The approach relies on the well-known fact that it is possible to convert any (arbitrary arity m) constraint problem into a binary constraint problem. The binarisation of constraints relies on the introduction of new variables and is based on the idea of introducing a new variable that encapsulates the set of constrained variables. Sometimes called *encapsulated variables*, these new variables have assigned a domain that is a Cartesian product of the domains of the (combined) individual variables.

The possibility to express constraints of higher arity in terms of binary constraints is important from the theoretical point of view as, in an expressive sense, binary CSPs are representative of all CSPs. In addition, note that many problems can often be naturally represented as constraint problems without the need for any transformation.

Relations and predicate logic

Relations are used in predicate logic to specify the relationships of some symbols. Logic programming is useful for formal verification, theorem proving, knowledge representation and reasoning, etc. and it has been found to be convenient for programming constraints.

There are some interesting correlation between theorem proving and constraint processing. For example, finding an interpretation of first order predicate logic is equivalent to finding a solution of a constraint satisfaction problem. Also, finding the best solution of a constraint optimisation problem is the same as finding the best interpretation in first order predicate logic. Since automatic theorem proving is generally based on the idea of proof by refutation, it is ideal for deciding satisfaction problems, but not good at solving optimisation problems.

To represent a constraint problem as relations, predicates are specified as relations of variables to represent constraints.

Interchanging representations

The two different representations are not mutually exclusive and can complement each other. The resource description framework [Klyne and Caroll, 2004] is a specification that uses both graphs and predicates together.

2.1.2 Solving Constraint Optimisation

In this thesis, we are interested in solving constraint optimisation. In the following sections, we will discuss the essentials in solving optimisation problems and present current approaches in solving constraint optimisation.

The fundamental difference between solving satisfaction and solving optimisation is in *solution quality*. In a CSP, all satisfied solutions are equivalent. In a COP, solutions are differentiated by their qualities. The solutions with the best qualities are the *optimal solutions*.

In solving satisfaction, a solver rejects a non-solution if a violation is found, computation can terminate once a satisfied solution is found.

Unlike solving satisfaction, an optimisation problem has many solutions with different qualities. When an optimisation solver finds a solution, it cannot terminate immediately. Instead, it has to continue looking for better solutions. Therefore, a solving strategy that based on rejecting non-solutions is not good enough for efficiency because it requires exhaustively generating solutions. An optimisation solver needs to *generate and compare* a lot of good solutions efficiently and pick out the optimal ones.

Depending on the complexity nature of a problem, generating all possible solutions can be time consuming. Therefore, efficiency usually cannot be guaranteed for algorithms dealing with general optimisation problems.

Search and optimal search

Solving search problems is fundamental in artificial intelligence research. In order to estimate the difficulty of a search problem, we can estimate the size of the search space in terms of the input problem size. For constraint optimisation problems, the search space is the total number of possible solutions. We can estimate the size of the search space in terms of variables and the possible values (domains). For a constraint problem with n variables and each variable's domain size is b, the total number of possible solutions is $S^* = b^n$, b is the branching factor for general search problems. It is clearly exponential with respect to n.

Backtracking

Backtracking is a basic exhaustive search strategy for finding a solution in a search space. It explores the search space one variable at a time. If a solution cannot be found in the current branch, the search *backtracks* to a previous decision stage and explores an alternative.

Backtracking is efficient, i.e. $O(n)$, in memory usage and is good for solving constraint satisfaction. However, backtracking is too time consuming for constraint optimisation, because of its exhaustive nature.

Branch-and-Bound

Branch-and-Bound is a common search strategy for optimisation problems which guarantees optimality and completeness. It keeps a growing ordered priority queue of partial solutions.

Partial solutions, at the head of the priority queue, are expanded until an optimal solution is found. A bound is used to restrict search exploration. It requires heuristics to improve efficiency, and may require exponential memory.

Dynamic Programming

The A* search algorithm is based on Branch-and-Bound and includes heuristics for estimating the overall solution quality of the partial solutions. It incorporates dynamic programming to prevent revisiting previously explored paths.

It is well known that good bounds significantly prune the search space. However, reaching a good bound is not trivial for two reasons: (i) there is lack of information at the beginning of search; and (ii) the effective range of bounds with respect to the overall solution quality is limited. A bound loses its effectiveness quickly as its value moves away from the (real) optimum.

A simple experiment using Branch-and-Bound and A* search confirms this, the results in Table 2.1 demonstrate that only a small range of bound values are effective in practice. The bound needs to be *close* to the optimal solution for the search to be efficient.

We can see in Table 2.1 that A* performs very well with the heuristic operating on a bound close to the optimal solution. In fact, [Pearl, 1983] established a necessary and sufficient condition for maintaining polynomial complexity in A* search. He concluded that polynomial A*, which is optimal and complete, is possible if and only if it is guided by heuristics with logarithmic precision, e.g., $\phi(N) = (logN)^k$.

Local search

Local search is another common search strategy for solving combinatorial problems. The philosophy behind local search is to sacrifice solution quality for efficiency. Since finding optimal solutions is often time consuming and getting a good solution is sufficient for some applications, we can quickly obtain an overall solution regardless of the quality, then gradually improve the quality of the solution.

The gain in efficiency by local search results in the loss of optimality. A solution found can be locally optimal, but is not guaranteed to be a global optimal. The introduction of constraints into local search strategies is often called constraint-based local search.

Bound used	# (expansions)	# (pruned p.soln.)	Queue size at termination
Branch & Bound			
5	6609	6801	6420
6	6609	2286	10935
7	6609	522	12699
8	6609	126	13095
9	6609	0	13221
A* (B&B + Expected Conflicts Heuristic)			
5	1623	2973	276
6	3111	5172	1053
7	4671	6672	2673
8	5877	6390	5367
9	6471	4596	8349
10	6609	1854	11367
11	6609	396	12825
12	6609	66	13155
13	6609	6	13215
14	6609	0	13221
Problem: 15 nodes, 58 edges, 4 colors Solution: optimal cost 5, using 3 colors			

Table 2.1: **Bounds lose effectiveness quickly:** A bound is effective only if it prunes unexplored partial solutions.

Constraint logic programming

Constraint logic programming is specifically designed for programming with constraints. Efficiency in solving satisfaction improves significantly by using constraint propagation and ordered binary decision diagram.

Decomposition and partition

Decomposition and partition based approaches aim to *divide and conquer*. A problem is divided into sub-problems. After the sub-problems are solved, a solution is formed by combining the sub-solutions.

Decomposition is a popular approach for efficiency and for processing large scale problems. A multilevel graph partitioning scheme is proposed in [Karypis and Kumar, 1998], where a graph is coarsened by partitioning into sections to manage complexity and the solution of a coarsened graph is mapped to the original graph. In [Larrosa and Meseguer, 2002], a partition-based scheme is proposed to obtain better lower bounds for Max-CSP problems.

The lower bounds computed depend on the selected partitions. [Gottlob *et al.*, 2000] applied hypertree decomposition to solve CSPs that can identify large tractable classes of constraints. A graph decomposition approach is proposed in [Messmer and Bunke, 2000] that leads to an efficient algorithm for subgraph isomorphism. Decomposition is used in [Slechta, 2005] for parallel processing to improve overall efficiency.

2.2 Distributed Constraint Optimisation Problems

The distributed constraint satisfaction problem (DCSP) provides a formal and general model for a wide range of distributed applications. The DCSP is a decision problem that defines agents, variables and constraints [Yokoo *et al.*, 1998], and seeks to find an overall assignment of all variables that satisfies *all* of the constraints. The distributed constraint optimisation problem (DCOP) is the optimisation variant of the DCSP. The goal of solving DCOP is to find an overall assignment of all variables that violates the *least* number of constraints, sometimes more generally as maximising a global utility function.

2.2.1 Cooperative problem solving

Distributed problem solving is described in [Davis and Smith, 1983] as modelling from human behaviours. A group of human experts cooperate by task-sharing. Each one works alone on subtasks, and occasionally pause to interact with other members.

As the concept of intelligent agent has become popular in artificial intelligence research, problem solving in distributed artificial intelligence has evolved to search algorithms for multi-agent systems [Liu and Sycara, 1994, Yokoo and Ishida, 1999]. Distributed constraint problems are found to be good models for cooperative problem solving. A distributed constraint problem unifies the model of constraint problems in problem solving and the model of cooperation in multi-agent systems [Hannebauer, 2002b].

One central issue of distributed problem solving, pointed out by [Davis and Smith, 1983] is the fundamental conflict between the *complete knowledge* needed to ensure coherence and the *local incomplete knowledge* inherent in the distribution of problem solving effort.

2.2.2 Local and global views

There are two ways to view distributed constraints, the local view and the global view.

The local view regards the distributed constraints as a group of CSPs with inter-CSP constraints. [Solotorevsky and Gudes, 1996] describes it as having two types of different problems. Solving the inter-CSP constraints is called the skeleton problem, and solving the individual CSPs is called the peripheral problem. [Solotorevsky and Gudes, 1996] also suggests that solving the inter-CSP constraints is more expensive than solving the local CSPs, it is because checking inter-CSP constraints is more expensive than checking local constraints. We now understand it as the communication problem between distributed agents. *"Communication is slower than computation"* is an observation stated in [Davis and Smith, 1983]. It is still true today because of the continual increase in computation processing power and the unavoidable overheads in communication protocols.

The global view regards the distributed constraints as a single uniform global problem. Each variable can be considered separately. The argument for this is an agent with a local CSP can employ a logical agent for each of the variables concerned. The global view simplifies the problem definition and is more consistent with the research in constraint processing. Recent developments in distributed constraint processing [Yokoo *et al.*, 1998, Modi *et al.*, 2005, Mailler and Lesser, 2006a, Jung and Tambe, 2005] embrace this view.

2.2.3 A formal definition of DCOP

The distributed constraint optimisation problem (DCOP) is therefore defined according to the global view discussed in Section 2.2.2, consistent with the recent constraint literature.

Definition 4 *A* DCOP problem *is a tuple* (X, D, C, A, f) *such that:*

- ***Variables***, $X = \{x_1, x_2, x_3, \ldots, x_n\}$ *is a set of variables;*
- ***Values***, $D = \{D_1, D_2, D_3, \ldots, D_n | D_i = \{d_{i1}, d_{i2}, d_{i3}, \ldots\}\}$ *is a set of domains, each* D_i *is given as a set of possible values,* d_{ij}*, of its corresponding variable* x_i*;*
- ***Constraints***, $C = \{c_1, c_2, c_3, \ldots | c_k = (D_{k1} \times D_{k2} \times D_{k3} \times \cdots \times D_{km}) \longrightarrow \{true, false\}\}$ *is a set of constraints on the values of the variables, where each constraint* c_k *is a mapping from a set of values to a boolean. A* true *means the constraint is satisfied, and a* false *means the constraint is violated.* m *is the arity of the constraint, for a binary constraint, the arity* $m = 2$*;*

- ***Agents***, *$A = \{a_1, a_2, a_3, \ldots, a_n\}$ is a set of agents, where each agent $a_i \in A$ initially knows (or owns, or is allocated) one variable x_i and its values D_i, and all the constraints that involve D_i. An agent can communicate with another agent related by a common constraint directly;*

- ***Objective***, *$f = (d_1, d_2, d_3, \ldots, d_n) \longrightarrow \{0, 1, 2, \ldots\}$ is an objective function mapping an assignment of variables to an integer value indicating the number of constraints violated (or utility of an agent).*

Distributed constraint optimisation

The goal of solving DCOP is to find an overall assignment of values, d_{ij}, to each variable, x_i, that minimises the objective, f (or maximises the utility).

DCOP and COP are sometimes referred to as distributed partial CSP, distributed maximal CSP, and over-constrained CSP. Although DCOP and DCSP are related problems, we discussed in 2.1.2 that solving optimisation requires a different approach than solving satisfaction.

2.3 Summary

Constraint problems are good models for many real world applications. A constraint problem consists of variables, values, and constraints. There are two abstractions to represent constraint problems: graphs or relations. The two representations are not mutually exclusive and are interchangeable. Constraint problems are computationally complex (frequently either NP-complete or NP-hard), they have exponential solution space with respect to input size. In solving the constraint optimisation problem, a strategy that compares solutions (dynamic programming) is generally more effective than rejecting non-solutions (backtracking).

An overview of common algorithms has been undertaken for solving constraint optimisation. Backtracking is memory efficient and good for solving satisfaction. Branch-and-Bound and A* are memory consuming and good for solving optimisation. Also, effective bounds are essential for good performance. Local search is time efficient but not optimal. Constraint logic programming is efficient for solving satisfaction. Last but not least, decomposition is popular for improving efficiency.

The DCOP problem has also been introduced as a good model for cooperative problem solving. The local view separates a distributed constraint problem into its skeleton problem and peripheral problems. The global view regards a distributed constraint problem as a uniform distribution of variables into agents. Finally, the formal definition of DCOP is given relevant to multi-agent systems.

Chapter 3

Communication in DCOP

In this chapter, the issue of communication in solving distributed constraint optimisation by multiple cooperating agents is investigated.

3.1 Requirements of DCOP

A DCOP algorithm specifies how a group of agents cooperate to pick out an optimal assignment to all variables out of an exponential solution space. There are some important properties that are desirable in designing a DCOP algorithm on a multi-agent system. In descending order of importance, they are:

1. **Solution quality.** An optimal search algorithm guarantees to produce optimal solutions at termination. If the algorithm is not optimal, it is desirable to produce solutions as close to an optimal solution as possible.

2. **Time efficiency.** An algorithm must terminate in a reasonable time. Finding efficient algorithms for computational complex problems is always a challenge for algorithm designers.

3. **Memory efficiency.** Some applications have strict memory requirements. If the computation runs out of memory, it terminates unexpectedly and will not produce a solution even if the algorithm is theoretically sound. This is especially significant if the computation is embedding on devices where memory space is a premium.

4. **Communication efficiency.** Because of the distributed processing nature of multi-agent systems, excessive communication between agents can have an adverse effect on algorithm performance. We will have more discussions on distribution and communication in Section 3.2.

5. **Distribution effectiveness.** Distribution is an important ingredient of multi-agent systems. It has benefits such as concurrency, fault tolerance, increase availability, and so on. However, distribution also increases the overheads in multi-agent systems. The overheads incurred may limit the size of distribution. Therefore, a multi-agent system can enjoy the benefits of distribution only if it manages distribution well. A good example from distributed systems is the domain name system [Mockapetris, 1987]. It was designed to manage distribution well, and it fuelled the growth and success of the Internet.

3.2 Distribution and Communication

Traditional centralised approach in problem solving may be time consuming in finding an optimal solution for a large scale problem, or it may be more time efficient but needs to sacrifice solution quality. Optimality and efficiency can be expected from a centralised approach only for small scale problems.

To obtain optimal solutions efficiently, an attractive strategy is to employ large scale distributed processing. However, distributed processing usually requires extra overheads in coordination and synchronisation in the form of communication messages. The problem is to maximise distribution for efficiency while minimising communication messages in obtaining optimal solutions of large scale constraint problems.

3.2.1 The need for distribution

The exponential complexity of constraint problems we have seen in Section 2.1.2 is the major limitation of a centralised solving approach. It is the cause of the *optimality/efficiency trade-off* in centralised optimal problem solving. In addition, there are concerns that sometimes a centralised approach is infeasible. For example, if the applications have real-time constraints that it needs to make local decisions within a time limit, or if some information is not to be shared. Therefore, we need to explore alternatives so breakthroughs in optimality and efficiency are possible.

An alternative to centralisation is distribution. Thanks to the advancement in computer communication networks, distributed systems are found to be an excellent system architecture for handling large scale problems because of its ability to incorporate extra computation resources on demand. Recent successes in large scale distributed architecture include indexing in search engines and distributed query in peer-to-peer file sharing.

Although we have seen that distribution is good for improving efficiency, guaranteeing overall optimality efficiently in cooperative problem solving under the assumption that each local problem solver has *incomplete knowledge* remains challenging. One underlining issue in guaranteeing overall optimality is the explosion of communication among distributed agents.

3.2.2 Explosion of communication

This problem is largely due to full distribution of variables. When an agent explores the domain of values of its variable, communication messages are generated between agents. Since every agent is typically responsible for a single variable, one change to a value of a variable inevitably results in (at least) one message for the change of value to propagate among agents for overall agreement. Since the solution space, that is the combinations of variable assignments, is exponential, it requires at least exponential communication messages for the agents to agree upon an optimal solution.

Using heuristics is the key to reduce exhaustive exploration in centralised solving. However, heuristics usually rely on the overall problem structures to calculate estimates. Without the knowledge of the global problem, a local solver cannot apply heuristics effectively on local problems.

Constraint propagation can significantly improve the efficiency for satisfaction problems. In a distributed setting, constraint propagation is in the form of communication messages among agents. It is effective in rejecting non-solutions in satisfaction solving, but it still requires evaluating all different solutions in optimisation solving.

One effective way of facilitating solutions comparison is to use upper and lower solution bounds as in [Modi *et al.*, 2003, Modi *et al.*, 2005]. However, as we will see in Section 3.3.8, the explosion of communication in constraint optimisation cannot be avoided because of full distribution of variables.

In summary, a full distribution of variables is expensive in communication for solving constraint optimisation. The high cost in communication limits scalability, and at the same

time reduces the effectiveness of efficiency that we expected from distribution.

Let us review some existing approaches in solving DCOP in the following sections.

3.3 DCOP Algorithms

Depending on the specific problems involved, algorithm designers put emphasis on different requirements. The following provides a short discussion on different approaches of DCOP algorithms.

3.3.1 Centralised solving

One straightforward way that minimises the overheads in managing distribution is the centralised approach in [Solotorevsky and Gudes, 1996] and also in [Yokoo and Hirayama, 2000]. Distributed constraints are collected and solved by a COP algorithm. It is good for problems where distribution is not necessary and a central location has enough computing resources for the problem. Efficiency is usually a concern for large scale problems. In some applications where agents do not want to share information about their variables and constraints, the centralised approach cannot be applied.

3.3.2 Distributed local solving

Asynchronous backtracking [Yokoo and Hirayama, 2000] fully distributes the variables among agents. There is a static ordering of variables. All agents explore its domain and communicate to check for satisfaction. If a violation is found, it is communicated to all relevant agents, the contacted agents then explore an alternative and check for satisfaction. Just like the backtracking approach in centralised solving, asynchronous backtracking is used for solving satisfaction problems.

3.3.3 Agent hierarchies

The static ordering of variables in asynchronous backtracking is a form of agent hierarchy. An agent hierarchy is a common scheme to organise agent communication. Asynchronous Partial Overlay [Mailler and Lesser, 2006a] uses agent priorities. In Asynchronous Distributed Optimisation [Modi *et al.*, 2005] and Distributed Pseudotree Optimisation (DPOP) [Petcu and Faltings, 2005], a hierarchy is constructed based on the constraint graph.

3.3.4 Dynamic variable ordering

The Asynchronous weak commitment search [Yokoo and Hirayama, 2000] introduces dynamic ordering of variables. Dynamic hierarchical ordering of variables gives higher priority to more difficult sub-problems. An asynchronous weak-commitment search restarts whenever ordering of variables is adjusted. It is more efficient than asynchronous backtracking by iteratively applying the *more-difficult-section-first* approach to problem solving. It also uses the min-conflict heuristic for satisfaction checking. In recent work, dynamic variable ordering has also been integrated with hierarchical organisation (see [Davin and Modi, 2006]).

3.3.5 Distributed local search

The Anchor & Ascend agent coordination approach [Liu and Sycara, 1995] uses distributed local search for solving DCOP. Since optimal solutions are not guaranteed, the algorithm introduces the idea of exploiting problem structures to improve solution quality.

3.3.6 Negotiation

The contract net [Davis and Smith, 1983] introduced the concept of negotiation for distributed problem solving. It is good for decentralised dynamic control of distributed problem solvers. Good matches of tasks and solvers can be achieved through bidding and contracting, however, optimal solution is not guaranteed.

Optimal Asynchronous Partial Overlay (OptAPO) [Mailler and Lesser, 2004] partially centralises problem substructures via a technique named cooperative mediation. It reduces communication among agents and maintains distributed problem solving. However, communication still grows more than quadratically in the number of agents for under-constrained problems.

3.3.7 Reconfiguration

Autonomous Dynamic Reconfiguration [Hannebauer, 2002c] presents the concept of reconfiguring agents for constraint optimisation problems. The idea is to change the configuration of agents to adapt to different problems. It formalises the agent melting operation that combines agents, and the agent splitting operation to create new agents from existing ones. It generates much less communication than a distributed setting, but optimal solutions are not guaranteed.

3.3.8 Limiting memory usage

Asynchronous Distributed Optimisation (Adopt) [Modi *et al.*, 2005] aims at restricting memory usage. It performs in bounded memory. It also assumed that agents do not share variable information, only value and cost messages are communicated.

An agent only needs to maintain a partial solution (the *CurrentContext*), the bounds on partial solution costs and the backtrack thresholds. Value assignment messages are communicated top-down an agent hierarchy, according to the constraint graph. Cost feedback messages are communicated bottom-up the hierarchy. Each agent chooses a local value that minimises the solution cost. The cost bounds at the root agent represent the total solution cost. The agents operate asynchronously until the root agent observes that an acceptable solution is reached and terminates. Each agent only keeps one partial solution and the backtrack thresholds are used for faster solution recovery after backtracking.

It can find the optimal solution with limited amount of memory in a fully distributed setting, or terminates early with solution quality guarantees. However, the full distribution of variables and bounded memory usage generates exponential communication among agents.

The cycles-based runtime measurement [Davin and Modi, 2005] aimed at evaluating Adopt's and OptAPO's communication and computation efficiency in a distributed setting. The measurement metric includes the time for constraint checks and communication latency. The result shows that Adopt uses more communication cycles but has a lower computational cost per cycle.

The MB-DPOP algorithm [Petcu and Faltings, 2007] is a memory bounded version of the Distributed Pseudotree Optimisation (DPOP) algorithm, which reduces the communication requirements to polynomial number of messages but still has exponential message size in the worst case.

3.3.9 Communication delay

Communication delays, as a result of both propagation delay and inter-agent processing have a significant impact on the solution of distributed constraint problems. Work by [Bejar *et al.*, 2005, Silaghi and Faltings, 2005] reports on the unexpected effects of communication delays on single process DCSPs in sensor network types of applications.

[Zivan and Meisels, 2006] have developed a simulation model for empirically measuring

the effects of communication delays on performance of distributed constraint satisfaction algorithms. They differentiate *single process* algorithms—those where agents assign values to only one variable at a time—with multi-process algorithms—where agents are in charge of the assignment of *multiple* variables. The results of their study favour multi-process algorithms for DCSPs. The results indicate that the performance of single process DCPSs, such as synchronous backtracking (SBT) [Yokoo and Hirayama, 2000] and asynchronous backtracking (ABT) [Bessiere *et al.*, 2005], are limited in the presence of communication delays. Importantly, this work empirically demonstrates the benefit of assigning multiple variables by empirically testing some initial multi-process algorithms.

The results imply that concurrent searching strategies show great promise for improved computation of distributed constraint solutions, a theme explored and better formalised in this thesis.

3.3.10 Aggregating communication messages

Distributed Pseudotree Optimisation (DPOP) [Petcu and Faltings, 2005, Petcu *et al.*, 2007] seeks to reduce the number of communication among agents.

Agents communicate through a pseudo-tree, which is based on the constraint graph. Utility vectors are communicated through the pseudo-tree. Values are communicated down the tree and utility messages are aggregated into vectors up the tree. Because of the aggregation in utility vectors, the number of messages required is reduced to linear with respect to the number of agents. However, the aggregation also means that the message size can be exponential to the width of the pseudo-tree. The problem of finding the minimum width pseudo-tree is known to be NP-complete.

3.4 Overcoming communication explosion

We can see that guaranteeing solution optimality while maintaining computation and communication efficiency is a major bottleneck in advancing distributed constraint optimisation. Our solution to this problem is based on some key insights.

First of all, a single problem solver can simultaneously achieve both optimality and efficiency for small scale problems.

In addition, agents need to share variable information. [Jung and Tambe, 2005] also recognises the restriction on value-only communication. They conclude that using additional

information in communication can improve performance of DCSP strategies. A number of recent studies on large scale coordination in [Scerri *et al.*, 2006, Davin and Modi, 2005, Mailler, 2005] have also shown that dynamic, partial centralisation is a common scheme for large scale coordination.

An important insight for efficiency is that hierarchical graph decomposition can improve efficiency. It originates from Role-graph coordination strategy [Soon *et al.*, 2004, Soon, 2005] and the recognition of efficient decomposition for subgraph isomorphism detection in [Messmer and Bunke, 2000]. [Yadgar *et al.*, 2003] also shows that using hierarchy has a benefit in reducing communication between agents. Therefore, it is contemplated that hierarchical graph decomposition can uncover bounds for optimal search, and the uncovered bounds can be used for comparing solutions. Our solution is largely based on this insight, however, realizing this insight is not straight forward, we will discuss this in more details in Chapter 4.

3.5 Summary

We identified the five algorithmic properties which are desirable in designing DCOP algorithms for multi-agent systems: (1) solution quality, (2) time efficiency, (3) memory efficiency, (4) communication efficiency, and (5) distribution effectiveness.

We discussed the need for distribution for efficiency and the problem of communication explosion in guaranteeing solution optimality in multi-agent systems.

We reviewed the pros and cons of different approaches of DCOP algorithms, which are: centralised solving, distributed local solving, agent hierarchy, weak commitment, distributed local search, negotiation, reconfiguration, limiting memory usage, communication delay, and aggregating communication messages.

Finally, we presented some key insights in small scale problems, sharing information and hierarchical graph decomposition that ultimately led to our solution for distributed constraint optimisation.

Chapter 4

Focused Decomposition

Given that optimal solutions can be found more efficiently for small scale problems and that cooperative agents can share information, it is contemplated that hierarchical graph decomposition using a sub-graph decomposition scheme similar to [Messmer and Bunke, 2000] can uncover bounds which would lead to efficient optimal graph search algorithms for multi-agent systems. It was soon discovered that two major issues remain to be solved.

A problem can have multiple optimal solutions. Let us briefly describe a simple graph decomposition scheme. Suppose a graph $G = \{G1, G2, H\}$ is decomposed into two sub-graphs $(G1, G2)$, and a set H of edges between the sub-graphs. A graph represents a constraint problem. Now if we have all the optimal solutions of two decomposed sub-problems, can we combine two optimal sub-solutions to form one optimal solution? Trying all combinations of sub-solutions does not seem to be a good idea because of the growing number of combinations as a solution gets larger.

More importantly, combining optimal sub-solutions does *not* necessarily lead to an optimal solution. It is because an optimal solution may come from two sub-optimal sub-solutions due to some decomposed constraints in H do not belong to either decomposed sub-problems. Therefore, combinations of sub-optimal sub-solutions with the decomposed constraints need to be checked. This could be worse than finding all optimal solutions! Nevertheless, the qualities of sub-solutions are good indicators of optimality of a sub-problem. Therefore, we concluded that the qualities of sub-solutions are what we interested.

Another issue is about the bounds that can improve optimal search efficiency. It was suspected that the number of constraints in H after decomposition can be used as a bound for optimal search. It was found that the size of H is almost always a lot bigger than the

effective range of bounds for simple decompositions. It is because of the limited effective range of bounds for optimal search (Refer to Section 2.1.2 for a discussion on effective bounds in solving a COP.). There are polynomial time algorithms for finding a minimal cut of a graph. However, given that two optimal sub-solutions do not necessarily form an optimal solution, the minimal cut found does not translate to a bound for an optimal solution. Therefore, we need other ways to relate the qualities of sub-solutions to an effective bound for an optimal solution.

So there comes the birth of a new decomposition scheme. It is previously described as Distributed Decomposition (DistDecomp) in [Law and Pearce, 2006] as a more general distributed search strategy. In this chapter, the author traces pieces of thought, reconstructs the idea and formulates them coherently as *Focused Decomposition*, emphasising its multi-agent aspect. The name Focused Decomposition carries the picture that each agent focuses on a sub-section of a problem.

In Section 4.1, a constraint optimisation problem with a focus and the corresponding focused solution are defined. Then we discuss how focused solutions can be compared. We define a focused decomposition in Section 4.2, followed by focused graph decomposition in Section 4.3, and then describe the distribution of focuses in Section 4.4. In Section 4.5, we present the strategy for cooperative agents solving DCOP through distributed focused decomposition. Finally, we conclude the chapter with a simple application in event scheduling in Section 4.6.

4.1 Constraint optimisation with a focus

A focus of a constraint optimisation problem simply specifies a sub-section of the constraint optimisation problem that has a higher priority during problem solving. Therefore, it is the same problem with or without a focus. The focused sub-section is solved first before solving the rest of the problem. Different solutions are obtained from solving the same problem with different focuses. Since the solutions obtained are of the same problem, comparisons of the solutions are valid.

Let us use Figure 4.1 as an example constraint graph with 10 nodes and 23 edges. The problem is a graph colouring problem using 3 colours. The optimal cost of this graph is 1.

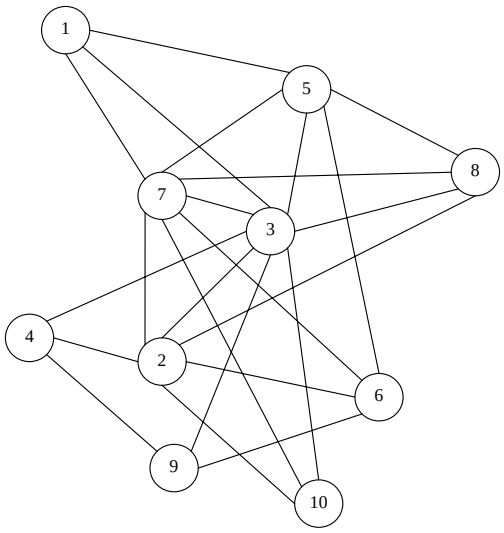

Figure 4.1: **Example constraint graph:** a 10-node 23-edge example constraint graph

4.1.1 A focused optimisation problem

Given a constraint optimisation problem, a focused problem is the same problem with a subset of higher priority variables. For example, Figure 4.2 pictures the example problem with a focus on variable nodes $\{1, 2, 3, 4, 5\}$.

Formally, a constraint optimisation problem with a focus is defined as a COP.

Definition 5 *Given a COP problem* $P = (X, D, C)$*, a* focused COP problem *is defined as* $P_F = \{P|F \subseteq X\}$*, which is the same problem* P *where the focus* F *specifies a subset of higher priority variables.*

In the definition above, X corresponds to the variables, D the values, C the constraints (see Definition 4). Note that the focused COP problem specifies a subset of higher priority variables, F, from the complete set of variables X.

For simplicity, the size of the subset F was chosen to be half of the size of X. Different varieties of focus sizes can be changed easily to adapt to specific needs. Dynamically adjusting the size of a focus that best suits the problem structure can also be experimented for future research.

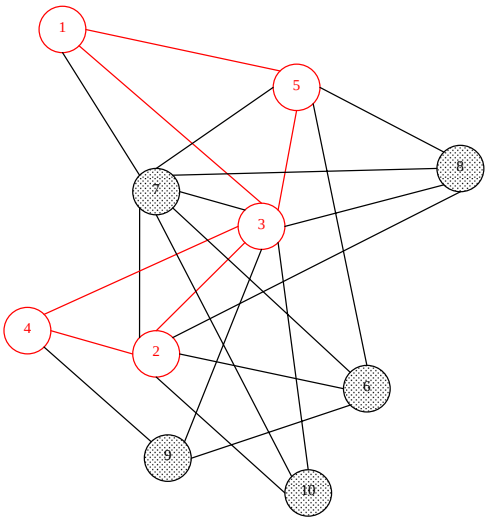

Figure 4.2: **Focused problem:** a focused problem with a focus on nodes $\{1, 2, 3, 4, 5\}$

4.1.2 A focused solution

A solution S_F of the focused problem, P_F, is a solution of the constraint optimisation problem P such that the variables in focus F are optimally assigned.

To find a focused solution, a normal optimal search strategy such as the A* search algorithm needs to separate its search procedure into two stages. An optimal sub-solution is found first by solving the variables in focus. Once an optimal sub-solution is found, the search procedure can reset its internal state and continue to search for a complete solution. The focused solution found at termination includes the optimal sub-solution found in the first stage.

A focused search finishes faster in most cases than a normal search by searching half of the problem size at each stage, since a problem with input size n becomes a problem with two sub-sections with size $n/2$ each.

If the problem requires exponential time to solve in the worst case, the benefit of reducing input size to one-half is significant. Increase in efficiency of course comes at a cost. Breaking an optimal search strategy like A* search into two stages also breaks the optimality property.

Although a focused solution is not guaranteed to be optimal, a good quality solution with values optimally assigned in both halves will emerge. Non-optimal values are only located around the borderline of the two sub-sections.

In short, a focused solution has good overall quality and can be solved much more efficiently for computational complex problems.

4.1.3 Comparison of focused solutions

Given $P = \{X, D, C | X = \{F1, F2\}\}$, we can find a focused solution S_{F1} of P_{F1} having $F1$ variables optimally assigned and also a focused solution S_{F2} of P_{F2} having $F2$ variables optimally assigned. We can compare S_{F1} and S_{F2} by comparing their qualities, that is, the number of constraints violated.

If S_{F1} is a better solution than S_{F2}, and we know that S_{F1} is a good quality solution of P based on the focused search strategy, we can conclude that the optimal solution of P must be at least as good as S_{F1}. Therefore, we can use the quality of S_{F1} as a bound for optimal search of P.

Upper and lower bounds

We define an upper bound of a solution of P as a known solution cost that is higher than the optimal cost. Any solution with cost higher than an upper bound can be discarded. The quality of S_{F1} above is therefore an upper bound for P. A solution cost s with an upper bound b is in the range $0 \leq s \leq b$. The least upper bound is an upper bound that is the same as the optimal solution cost. An effective upper bound is a bound that can be used to discard (partial) solutions during a search operation.

A lower bound of a solution of P is a known solution cost that is common to all possible solutions. A solution must have a cost as least as high as a lower bound, no better solution is possible. The greatest lower bound is a lower bound that is the same as the optimal solution cost. If a lower bound is the same as an upper bound, it must be the case that the optimal solution cost is the same as the two bounds.

In the search of focused solution, we do not consider lower bounds, only upper bounds are used.

4.2 A focused decomposition

A focused decomposition constructs smaller sub-problems with different focuses. Before we define a focused decomposition, let us define a core problem.

Definition 6 *Given a focused problem P_F, a* core problem $CP = \{F, D, C\}$ *of P_F is a constraint sub-problem of P_F where the variables in focus are taken to form a new problem, and the domains and constraints are inherited from P_F.*

Recall that $F \subseteq X$ is a subset of variables, the problem size of CP is smaller than P, that is, half of the size of P_F in our examples. We can now define a focused decomposition of a constraint problem.

Definition 7 *A* focused problem decomposition *of P_F, $\mathcal{D}(P_F) = \{CP_{Fi} | Fi \in F = \{F1, F2, \dots\}\}$ is a set of core problems of P_F each with a different focus.*

For example, Figure 4.3 shows a focused decomposition of the problem in Figure 4.2. Two core problems of the focused problem $P_{F[1-5]}$ are constructed with different focuses as $CP_{F[1-3]}$ and $CP_{F[4-5]}$.

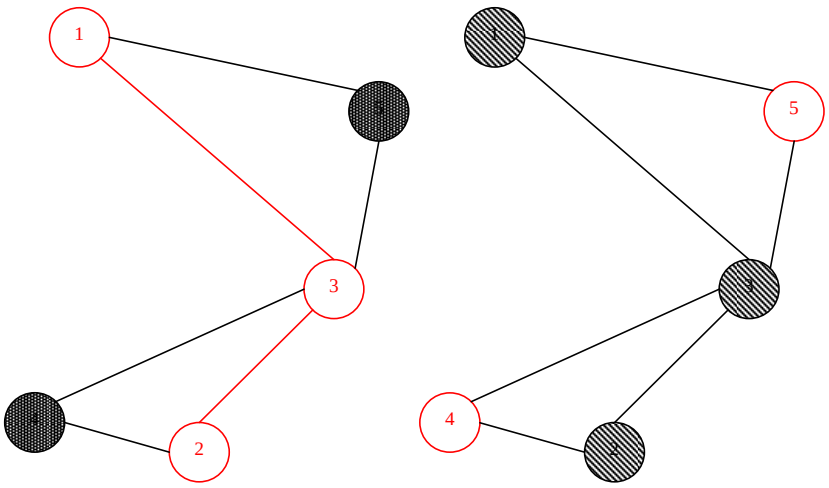

Figure 4.3: **Focused decomposition example:** a focused decomposition of the Figure 4.2 problem: a smaller problem with a focus on $\{1, 2, 3\}$(left), a smaller problem with a focus on $\{4, 5\}$(right).

The focused decomposition described is unique in that multiple instances of the same sub-problem are constructed. Since the same problem is solved with different focuses, multiple good quality solutions can be expected in a much time efficient manner. The multiple good quality solutions can then be compared to obtain a bound that is close to an optimal solution to the core problem of a focused problem. Figure 4.4 shows an

implementation of focused decomposition that is also related to the following Section of distributed focuses.

```
function focused_decomp_at (graph, agent, hierarchy)
    inputs:
        graph, a core problem of a focused problem
        agent, an ID representing an agent
        hierarchy, a hierarchy of agents
    midpoint ← (size_of graph) / 2
    focus1_nodes ← nodes from 1 .. midpoint in graph
    focus2_nodes ← nodes from midpoint+1 .. (size_of graph)
    if (left_child exists in hierarchy(agent))
        graph1 ← graph
        set_focus (graph1, left_child, focus1_nodes)
    if (right_child exists in hierarchy(agent))
        graph2 ← graph
        set_focus (graph2, right_child, focus2_nodes)
    return

function set_focus (graph, agent, focus_nodes)
    FocusedProblem[agent] ← graph
    set_high_priority (FocusedProblem[agent], focus_nodes)
    core_problem ← subgraph (graph, focus_nodes)
    focused_decomp_at (core_problem, agent)
    return

function subgraph (graph, subset_of_nodes) returns graph
    new_graph ← initialize_graph (size_of (subset_of_nodes))
        for (every node i in subset_of_nodes)
            for (every node j in subset_of_nodes)
                if (i == j) continue
                if (i and j are connected in graph)
                    insert_edge i to j in new_graph
    return new_graph
```

Figure 4.4: **Focused decomposition algorithm:** a focused decomposition algorithm for a binary agent hierarchy.

In spite of having defined the focused decomposition in terms of the sets of variables involved in sub-problems, and by way of pseudocode, it remains to formally define the decomposition more precisely in terms of the *constraint graphs* that the variables and constraints of each subproblem attribute.

4.3 Focused graph decomposition

The focused decomposition is now formally defined in terms of graph representation. It relies on two key ideas: decomposition of the constraint problem based on the idea of constraint subgraphs; and ordering over variables of the problem using vertex permutations.

First, the decomposition of constraint graphs is defined. Then, the ordering of variables is defined. Finally, a distributed focused decomposition example is provided in figurative form.

Problem decomposition is generally good for efficiency because of the reduced complexity of sub-problems. Our approach therefore follows the classic *divide-and-conquer* approach of problem solving. Nevertheless, it differs from *divide-and-conquer* in that there is no combining of sub-solutions involved. Instead, the quality of sub-solutions is used to guide further problem solving effort in promising directions.

The new decomposition also differs from the multilevel partitioning approach as in [Karypis and Kumar, 1998], where a graph is coarsened by partitioning into sections to manage complexity and the solution of a coarsened graph is mapped to the original graph before partitioning.

In the spirit of [Messmer and Bunke, 2000], a decomposition is defined as follows, building on the definition of a constraint graph (see Definition 3).

Definition 8 A *constraint subgraph* $G' = (V', E')$ is a subgraph of a constraint graph $G = (V, E)$ iff $V' \subseteq V$, and $E' \subseteq ((v_1, v_2) \in E \rightarrow v_1, v_2 \in V')$.

It follows (based on the definition of the constraint graph) that subgraph G', contains a subset of variables $x_i \in X'$ that attribute vertices $v_i \in V'$ and binary constraints $c_i \in C'$ that attribute edges $(v_i, v_j) \in E'$. Sufficient definitions are now in place to define the *decomposition*.

Definition 9 *Given a set of constraint graphs* $\mathcal{G} = \{G_1, \ldots, G_n\}$, *a* decomposition *of* $\mathcal{G}$, $D(\mathcal{G}) = \{t_1, t_2, \ldots, t_n | t_i = (G_i, G'_i, G''_i)\}$, *is a finite set of tuples* (G, G', G'') *where*

- G, G', *and* G'' *are graphs with* $G' \subset G$, *and* $G'' \subset G$.
- *For each* $G_i \in \mathcal{G}$ *where* $i = 1, 2, \ldots, n$, *there exists a tuple* $(G_i, G'_i, G''_i) \in D(\mathcal{G})$.
- *For each tuple* $(G_i, G'_i, G''_i) \in D(\mathcal{G})$ *there exists (up to) one tuple* $(G_1, G'_1, G''_1) \in D(\mathcal{G})$ *with* $G_i = G_1$.

- *For each tuple* $(G_i, G'_i, G''_i) \in D(\mathcal{G})$
  - *if* G'_i *consists of more than one vertex, then there exists a tuple* $(G_1, G'_1, G''_1) \in D(\mathcal{G})$ *such that* $G'_i = G_1$.
  - *if* G''_i *consists of more than one vertex, then there exists a tuple* $(G_2, G'_2, G''_2) \in D(\mathcal{G})$ *such that* $G''_i = G_2$.
  - *if* G'_i *consists of one vertex, then there exists no tuple* $(G_3, G'_3, G''_3) \in D(\mathcal{G})$ *such that* $G'_i = G_3$.
  - *if* G''_i *consists of one vertex, then there exists no tuple* $(G_4, G'_4, G''_4) \in D(\mathcal{G})$ *such that* $G''_i = G_4$.

Note that the definition of decomposition does not take into account the edges *between* decomposed subgraphs. For example, given graph G there will likely exist some edges H between subgraphs G' and G''. However, as stated at the beginning of this Chapter, taking H into account in bounds computation is not effective in practice due the the potentially large number of edges that can exist between subgraphs. Instead, bounds are computed *separately* for each of the constituent subgraphs.

Now that the decomposition is defined, one more element of the focused decomposition remains: the permutation of the *ordering* of variables involved in focused problems for each decomposed subgraph. The approach broadly follows work on graph decompositions and factorising permutations [Capelle *et al.*, 2002].

Definition 10 *A* vertex permutation*,* σ*, of vertex set* V *is denoted by* $\sigma(1) = v_i, \sigma(2) = v_j, \ldots \sigma(n) = v_n$.

This naturally places an ordering over the variables, denoted $\sigma(1) = x_i, \sigma(2) = x_j, \ldots \sigma(m) = x_n$, that attribute each corresponding vertex according to Definition 3.

In this thesis, the vertex permutations are randomly generated for simplicity. However, there are a number of algorithms for easily generating permutations efficiently in the literature. These include permutations that are known as *factorising* permutations in the sense that each can be considered as an (ordered) factor of another, convenient to the needs of a focused decomposition. [Habib *et al.*, 1999] proposed a relatively easy to understand $O(n + m)$ algorithm for computing a factorising permutation based on a modular decomposition of undirected graphs.

With the graph decomposition and vertex permutation defined, the *focused decomposition* can now be defined as a decomposition of constraint graphs if subgraphs represent different problem focuses.

Definition 11 *A* focused decomposition *of $\mathcal{G}$, $FD(\mathcal{G})$, is a decomposition of $\mathcal{G}$, $D(\mathcal{G})$, where each tuple $(G_i, G_i', G_i'') \in D(\mathcal{G})$ specifies problem focuses. If G_i represents a focused problem P_F, then $\{G_i' = (V_i', E_i'), G_i'' = (V_i'', E_i'')\}$ is $\mathcal{D}(P_F)$, which is a set of core problems of P_F (as determined by Definition 7), each with a different vertex permutation: σ' of vertex set V'; and σ'' of vertex set V''.*

For example, Figure 4.6 shows a focused decomposition initiated by the root agent. Each agent gets a different subgraph (corresponding to focused problem P_F), each with it's own vertex permutation (corresponding to a different variable ordering). In the Figure, assuming agent a_3 is assigned a graph $G = (V, E)$ with permutation σ which is decomposed into two subgraphs $G_1' = (V', E')$ and $G_2' = (V', E')$: then agent a_2 is assigned permutation σ_1' of vertices V' and agent a_4 is assigned a permutation σ_2' of the same vertices V'.

Notice that as a result of the decomposition, for solving any (hierarchically) focused problem P_F each agent focuses on a different core subproblem, $CP_{Fi}|F_i \in F = \{F_1, F_2, \dots\}\}$ where $Fi \subseteq X$ is a subset of variables that attribute subgraphs assigned to agents. Notice also that the problem size of CP_{Fi} is smaller than P_F, that is, half of the size of P_F.

4.4 Distribution of focused subproblems

The agents follow the focused decomposition tree hierarchy for communication. Unlike recent DCOP algorithms Adopt [Modi *et al.*, 2005] and DPOP [Petcu and Faltings, 2005] which require the agent hierarchy conforming to the constraint graph, the agent hierarchy for distributed focuses can be any arbitrary tree structure. This is an advantage for deploying multi-agent systems. We use a binary tree as in Figure 4.5 for our example. The agents communicate through the hierarchy.

Distribution of focuses

To distribute problem focuses, each agent is assigned a focused problem P_F^i from its higher rank agent, and constructs a focused decomposition $\mathcal{D}(P_F^i)$ which is a set of focused core problems, then assigns each decomposed problem to one of its lower rank agents.

The higher the rank of an agent, the bigger problem it gets. Similarly, the lower the rank of an agent, the smaller problem it gets. Every agent gets a different focused problem, essentially each agent is focusing on a different sub-section of the overall problem.

Once an agent gets a focused problem, it can start searching for a focused solution independently. An agent reports the quality of its focused solution to its higher rank agent when its focused search terminates. For example, Figure 4.6 shows a distributed focused decomposition initiated by the root agent. Each agent gets a different focused problem. The red colour indicates a core problem of a focused problem. The shaded variables are the variables not in focus.

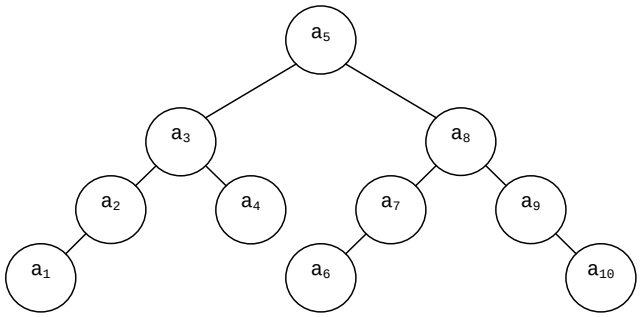

Figure 4.5: **Agent communication hierarchy:** an agent hierarchy showing the communication path for focused decomposition

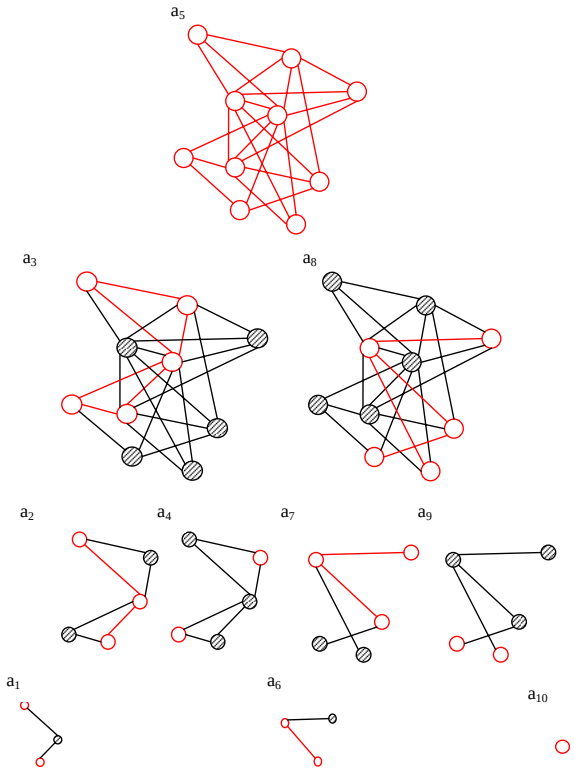

Figure 4.6: **Distributed focused decomposition:** a distributed focused decomposition where each agent gets a different focused problem. Red colour indicates the core problems. Shaded variables are not in focus. For example, assuming agent a_3 is assigned permutation σ of vertices V for graph $G = (V, E)$ which is decomposed into two instances of (the same) subgraph $G'_1 = (V', E')$ and $G'_2 = (V', E')$: then agent a_2 is assigned permutation σ'_1 of vertices V' and agent a_4 is assigned a different permutation σ'_2 of vertices V'. Similarly, agent a_8 assigns two subgraphs $G''_1 = (V'', E'')$ and $G''_2 = (V'', E'')$: then agent a_7 is assigned permutation σ''_1 of vertices V'' and agent a_9 is assigned a different permutation σ''_2 of vertices V''. Notice that as a result of the decomposition, for solving any (hierarchically) focused problem P_F each agent focuses on a different core subproblem, $CP_{Fi}|F_i \in F = \{F_1, F_2, \dots\}\}$ where $Fi \subseteq X$ is a subset of variables that attribute subgraphs assigned to agents. Notice also that the problem size of CP_{Fi} is smaller than P_F, that is, half of the size of P_F.

Efficiency

When an agent receives quality reports, it can compare the core problems of its focused problem and obtain an effective bound for the first stage of its focused search. Every agent gets advice from the quality reports and improve the efficiency of its search. In addition, the lower rank agents finish their jobs faster simply because they got smaller size problems.

As a result, every agent searches through its core problem efficiently with the knowledge of bound information. It does not repeat the exploration of the higher cost search space already explored by other agents down the sub-hierarchy. In addition, all paths with similar costs that other agents did not explore can be ignored too. It finishes the search to obtain a good quality focused solution for its higher rank agent. The effective bounds guide the searches of different problem sizes along the optimal path. Non-optimal solutions are filtered out early in the process when sub-problems are small and easy to solve. Therefore, the informed root agent can conduct an efficient optimal search rather than putting up exhaustive effort.

For example, running our distributed focused decomposition based search on the problem in Table 2.1, the root agent can conduct an efficient A* search because it got a quality report of cost 5, and another quality report of cost 9. Knowing that a good quality solution is found to have cost 5, it starts searching with bound 5. It requires 4,986 less expansions than an exhaustive search and prunes 2,973 partial solutions in the process. Only 276 partial solutions remain in the queue at termination, showing that the exploration of search space is mostly along the optimal path.

We can see that a group of cooperative agents can search through a computational complex problem efficiently through distributed focuses, much more efficient than a single agent can achieve.

Optimality

The optimality of the algorithm is maintained at the root agent. Since the root agent is the highest rank agent, it has the complete big picture of the problem. The root agent is also the one that initiates a focused decomposition in a hierarchy by treating its whole problem as the core problem. In other words, the root agent put its focus on all the variables, $P_F^{root} = \{F, D, C | F = X \in P\}$. Therefore, the root agent can conduct an optimal search with an effective bound on all variables. The optimality guarantee is inherited from the

optimal search algorithm we used, that is A* search, which is optimal and complete.

In our example, the qualities of the focused solutions found by the agents are shown in Table 4.1. The root agent a_5 found an optimal solution with a cost 1.

agent:	focused problem P_F^i	: quality of S_F^i
a_5:	{(1 2 3 4 5 6 7 8 9 10)}	: 1
a_3:	{(1 2 3 4 5) 6 7 8 9 10}	: 3
a_2:	{(1 2 3) 4 5}	: 0
a_4:	{(4 5) 1 2 3}	: 0
a_1:	{(1 2) 3}	: 0
a_8:	{(6 7 8 9 10) 1 2 3 4 5}	: 2
a_7:	{(6 7 8) 9 10}	: 0
a_9:	{(9 10) 6 7 8}	: 0
a_6:	{(6 7) 8}	: 0
a_{10}:	{(10)}	: 0

Table 4.1: **Qualities of focused solutions:** found for a distributed focused decomposition in Figure 4.6. Variables inside of parentheses indicate focuses.

4.5 Solving DCOP with distributed focused decomposition

Given a DCOP as specified in 2.2 that each agent is initialised with one variable, we can solve it using distributed focuses. Previously, a distributed focused decomposition was described that is optimal and efficient. We will see that using distributed focused decomposition to solve DCOP is also communication efficient.

There are four phases in solving DCOP in our approach, namely: (1) Problem identification, (2) Problem distribution, (3) Solution quality report, and (4) Solution propagation. Each phase has different communication requirements. We will describe the purposes of each phase and specify the communication messages needed for each phase.

4.5.1 Problem identification

A group of cooperative agents form a team in a tree hierarchy. As a team member, each cooperative agents are willing to share information. How different agents fit into an hierarchy is not the concern here, it could be pre-defined and static or dynamically fill in by

some election protocols or by invitation from a root agent or available agents just filling up arbitrarily.

The purpose of the problem identification phase is that the root agent at the top of an hierarchy decides that there is a need to solve a DCOP problem. Since the constraint variables are distributed among agents initially, it centralises the information about variables and constraints, then formulates an overall problem so that it can be solved by distributed focuses which is optimal and efficient.

Two types of communication messages are sufficient for this purpose: problem request message and problem response message.

A problem request message is sent by the root agent to all members of a team requesting problem information. The root agent sends one problem request to each member agent. Problem requests are fixed size messages. A problem response message is sent by an agent back to the root agent in response to a problem request containing information about the problem. Problem information related to constraint problems are variables and constraints, represented by one of the representations in Section 2.1.1. Since every agent knows one variable initially, the size of a problem response message depends on the number of constraints related to the variable.

The problem request and response pair also serves as the agreement on initiating a cooperative problem solving session. The agents response to a problem request agree to engage in the cooperative problem solving session. By participation in a cooperative problem solving session, an agent will obtain an optimal assignment to its variable that it cannot easily obtain based on its incomplete local knowledge.

Once the root agent gets the problem responses, it can formulate an overall constraint problem and initiate distributed focused decomposition.

4.5.2 Problem distribution

The second phase of our approach is to distribute decomposed problems to cooperative agents. It is accomplished by focused decomposition.

A problem assignment message is sufficient for this purpose. After an agent constructs a set of focused problems by focused decomposition, it sends one problem assignment message containing a focused problem to each of the agents under its hierarchy.

The size of a problem assignment message varies according to the number of variables and constraints in a focused problem. Note that the sizes of focused problems reduce quickly

down the hierarchy due to the decomposition approach.

When an agent receives a problem assignment. It can start searching for focused solutions while waiting for solution quality reports.

4.5.3 Solution quality report

When an agent finishes searching and got a focused solution. It reports the solution quality in response to a problem assignment.

A solution report message contains the quality of a focused solution. Since an agent which sent a problem assignment knows about the focused problems it assigned, a solution report message does not need to include a copy of the focused problem. Therefore, a solution report message is a fixed size message.

After an agent sends out a solution report message, it can wait for a guaranteed optimal solution.

4.5.4 Solution propagation

When the root agent completes its searching, an optimal solution to the overall problem is found. It informs all member agents about the optimal solution.

A solution assignment message is the final message the root agent sends to all other agents. It contains an optimal assignment of the variable in the problem response message. Since there is only one variable in a problem response message, a solution assignment message is fixed in size.

A solution assignment message also serves as the message to indicate termination of the cooperative problem solving session. The problem request, problem response, and solution assignment together act as the mechanism for session establishment and termination. The specific connection establishment and the communication medium between two agents depends on the transport layer protocol an application chooses to use and the medium access of a communication network. A cooperative problem solving session does not need to concern about it.

4.5.5 Communication in solving DCOP

Figure 4.7 shows a sequence diagram to demonstrate the communication protocol used in solving DCOP by distributed focused decomposition.

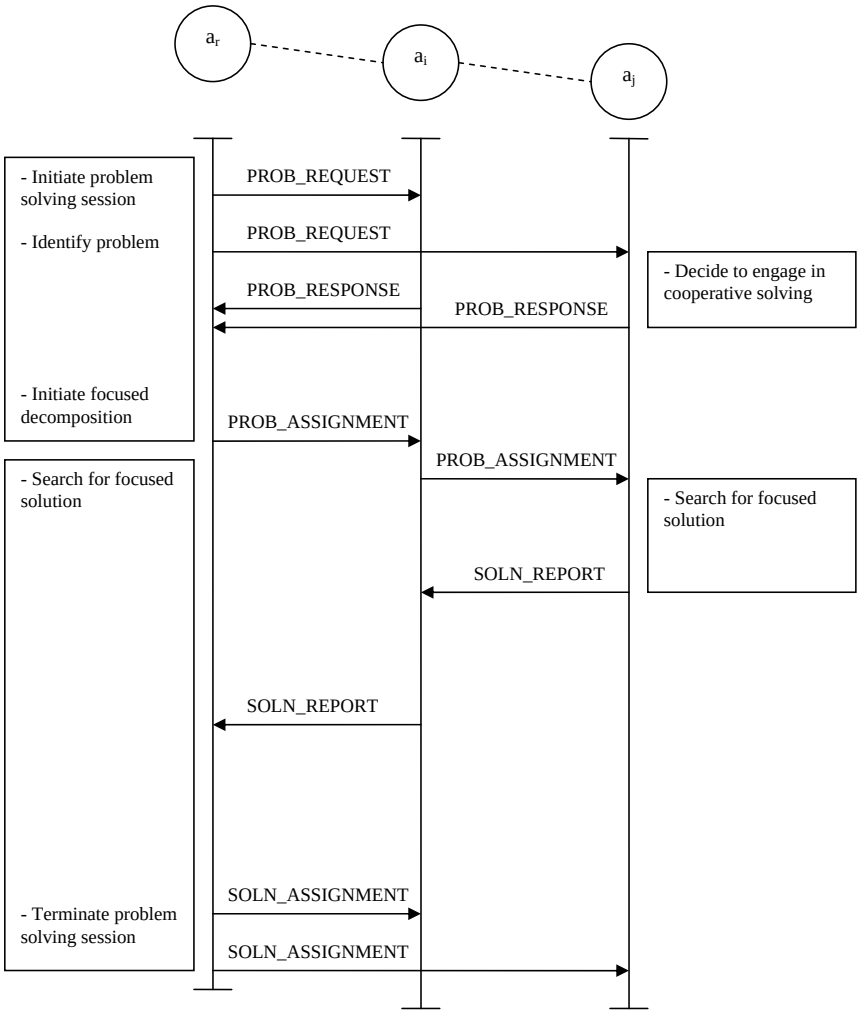

Figure 4.7: **Communication protocol:** the communication protocol is shown for solving DCOP by distributed focuses for three agents a_R, a_I and a_J. The protocol is shown based on problem request, problem response, problem assignment, solution quality report, and solution assignment.

The focused decomposition approach in solving DCOP achieves linear communication complexity, i.e. $O(N)$, in terms of number of messages required with respect to the number of variables in a constraint problem. As a matter of fact, it requires exactly $5 \times (N-1)$ total number of messages for an entire cooperative problem solving session. As in a simple scenario with 3 agents in Figure 4.7, 10 messages are necessary and sufficient for an optimal assignment to all variables. For our 10-variable example, 45 messages are enough compared to a possible exponential number $2^{10} = 1024$!

In terms of message size, 3 out of 5 different types of messages are fixed in size. Only problem response and problem assignment messages are variable size messages. The size of a problem response message depends on the number of constraints related to a variable, which is the degree of a node in a constraint graph. The size of a problem assignment message depends on the size of a focused problem. The maximum size of a focused problem, also indicating the maximum size of any messages, is in the complexity $O(X + C)$ where X are the variables and C are the constraints, which is the size of representing a constraint graph.

Therefore, we conclude that the distributed focuses approach for solving DCOP we presented is communication efficient, with linear communication complexity in terms of number of messages and linear in message size with respect to the size of a constraint graph.

4.6 An application in event scheduling

Recall in the introduction that Mary travels to attend a conference. She wants to participate in as many events as possible.

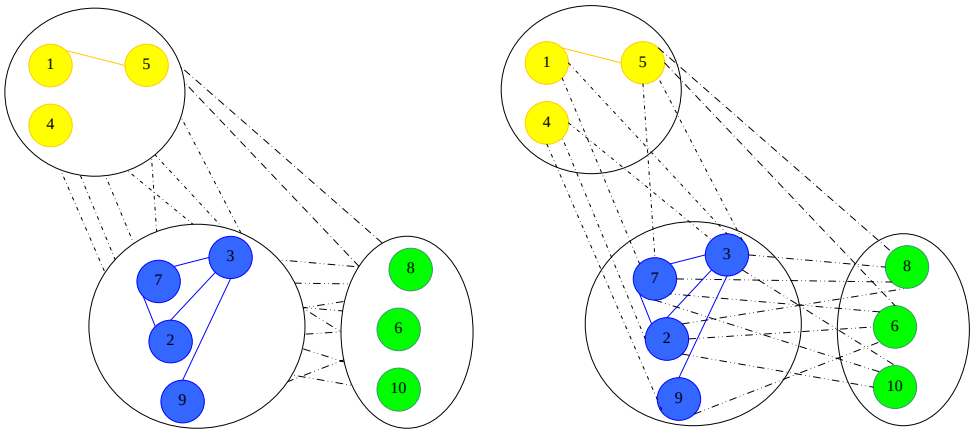

Figure 4.8: **Constraints in event scheduling:** the green events on the right side of the constraint graph represent personal events without conflicts. The blue events in the middle represent conference events. There are a lot of conflicts among conference events. The yellow events on the left side represent local events with less conflicts than the conference events.(left) A global view shows more constraints across groups of events. A focused decomposition distributed search strategy can be employed to find the least conflicting event schedules.(right)

In Figure 4.8(L), the green events on the right side represent her personal events. There are no conflicts among her personal events. The blue events in the middle represent the conference events. There are some sessions conflicting with each other. The yellow events on the left side represent local events with less conflicts than the conference events.

A global view of all the events in Figure 4.8(R) shows more conflicts across different groups of events. Redrawing results in our example problem as in Figure 4.1. Therefore, it can be solved by a focused decomposition distributed search as described in previous sections.

Chapter 5

Evaluation

The results of the evaluation is published in [Law and Pearce, 2006] as the DistDecomp algorithm. DistDecomp is the distributed focused decomposition mechanism presented in Section 4.4, which is part of our multi-agent cooperation strategy proposed in this thesis.

The evaluation is updated to fit the multi-agent cooperation theme of this thesis. The experimental results of DistDecomp can be used for evaluation of the multi-agent cooperation strategy proposed in this thesis because the performance of distributed focused decomposition is the major factor in the improvement of both computation and communication efficiency.

Section 5.1 presents the experimental results of DistDecomp, which stands for distributed decomposition or distributed focused decomposition using the newly established Focused Decomposition formalism in Section 4.2. Section 5.2 analyses the complexity of our approach with respect to the five DCOP algorithm requirements in Section 3.1.

5.1 Experimental comparison with Adopt

In this section we compare DistDecomp with Adopt [Modi *et al.*, 2005]. In our experiments we compared how efficient DistDecomp is, both in terms of computation and communication costs. We also compared the algorithm with a centralised approach where decomposition and communication are not required.

In keeping with the Adopt experiments, we implemented a multi-threaded simulation of multiple agents for our algorithm and obtained a copy of the ADOPT implementation from its authors for comparison. We utilised the same MaxSAT graph colouring problem

instances as Adopt. It is known that all planar graphs are 4-colourable and finding a 2-colour assignment is the same as the bijection problem, which can be solved in polynomial time. Of course, finding the existence of a 3-colour assignment of general graphs is NP-complete, and finding the optimal assignment (MaxSAT) of general graphs is NP-hard. There are 25 different instances of graph with the same properties. A graph instance can be identified by the number of vertices $|V|$, the number of edges $|E|$, and the number of colours used to solve the problem. In our experiments, $|E|$ is 3 times $|V|$, 3 colours are used as the domain of the colouring problem. The graph instances used were sourced directly from [Modi *et al.*, 2005] and are summarised in Table 5.1. The optimal cost of a graph instance is the number of edges connecting two vertices with the same colour (constraint violated) of an optimal solution. The average optimal cost is from 3 to 4 for these instances.

$\|V\|$	$\|E\|$	# colors	Optimal cost (avg.)
10	30	3	3.28
12	36	3	3.24
14	42	3	3.44
16	48	3	3.56
18	54	3	3.76
20	60	3	3.6

Table 5.1: **Graph colouring problem:** instances of the graph colouring problem

5.1.1 Performance of computation

Figure 5.1 shows the performance in execution time. We can see that the rate of increase is much faster in Adopt than in DistDecomp when problem size increases.

The better time efficiency can be explained by the effect of problem decomposition, where distributed focused decomposition efficiently discovers effective bounds both recursively (by focused decomposition) and in parallel (through distributed focuses) contributing to significant reduction in searching overall. The difference in performance for larger problem sizes indicates the potential for DistDecomp to scale up to larger problem instances.

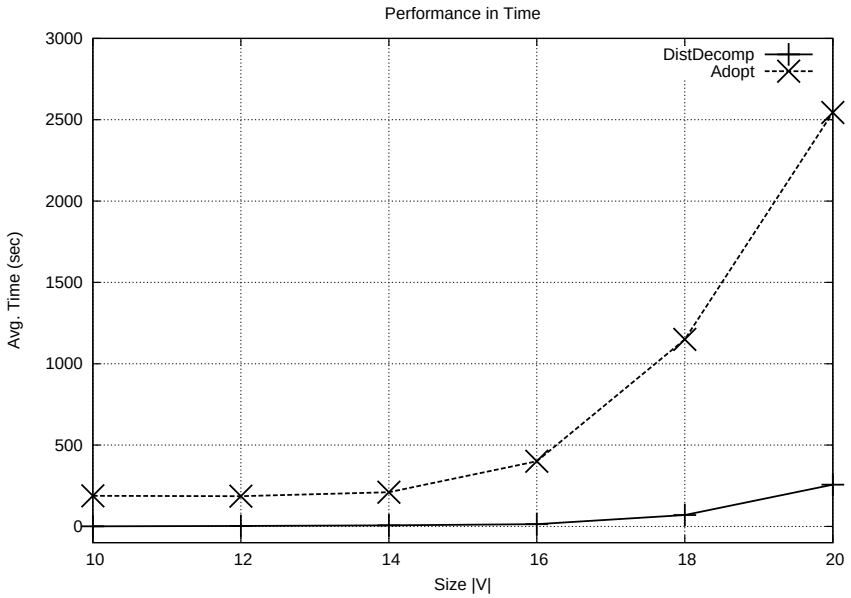

Figure 5.1: **Computation comparison:** Adopt vs. DistDecomp

5.1.2 Cost of communication

Figure 5.2 shows the growth of communication messages required for the Adopt algorithm, while DistDecomp only requires a fixed number of communication messages. The number of messages required is always $N - 1$ when there are N agents in the hierarchy. The communication messages in DistDecomp correspond to the solution quality report messages in our multi-agent cooperation strategy.

The number of messages sent as in Figure 5.2 is related to the performance in time in Figure 5.1. When communication latency is high, the extra time required for communication adds to the overall solution time. This can be a big advantage for DistDecomp when the agents are operating in an environment where communication is expensive or communication links are unreliable or the latency of communication medium is high. The overall performance of DistDecomp is unaffected by communication cost.

5.1.3 Performance in distribution

It is interesting to note that DistDecomp outperforms a centralised A* search. Figure 5.3 shows the average time required for a single A* search compared with DistDecomp.

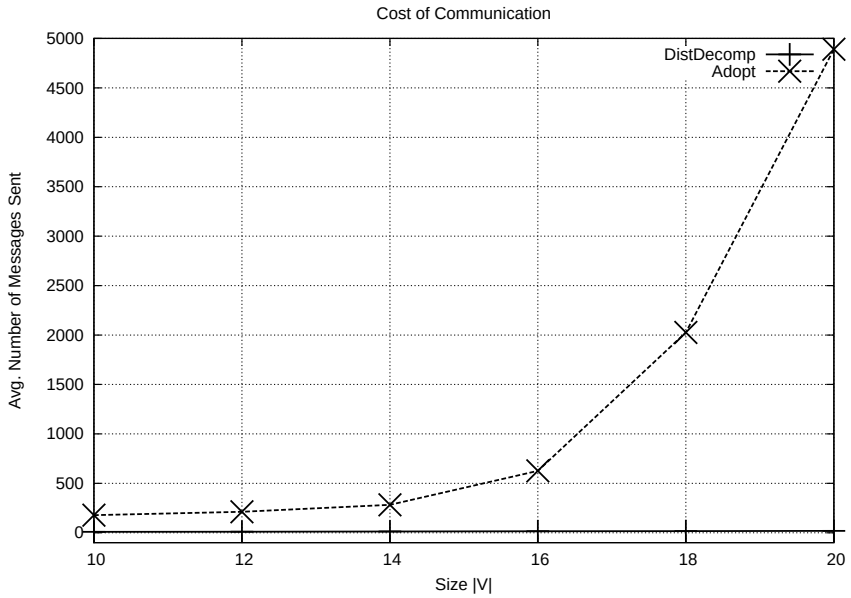

Figure 5.2: **Communication comparison:** Adopt vs. DistDecomp

In DistDecomp, the root agent's search operation dominates the overall search time because it has the biggest problem to solve. The root agent can use an effective bound once it receives solution quality reports. Effective bounds information is not available in the centralised case. Therefore, the result in Figure 5.3 reflects the performance benefit of effective bounds in DistDecomp.

DistDecomp also produces smoother growth curve. It is because of the distribution of problem focuses. Optimality of different sub-sections of a problem is explored in different agents. Therefore, difficult and time consuming sections of the search space are identified and skipped early down the hierarchy. Local structural difficulties can slow down A* search significantly without an effective bound.

In summary, our experiments indicate that DistDecomp is significantly more efficient computationally than Adopt, in searching for an optimal assignment of the MaxSAT graph colouring problems.

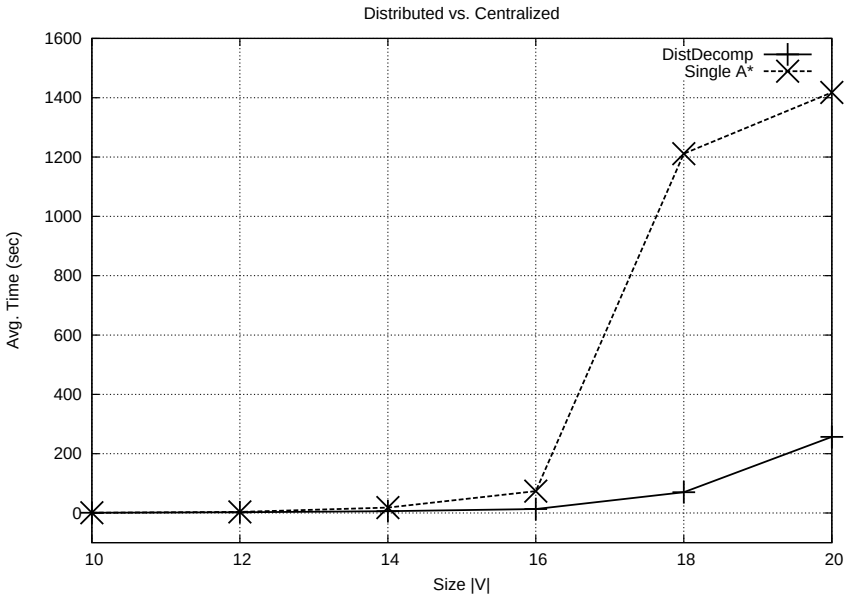

Figure 5.3: **Computation comparison:** DistDecomp vs. centralised solving in A* search

5.2 Algorithmic analysis

We analyse our DCOP algorithm with respect to the five desirable requirements presented in Section 3.1.

5.2.1 Solution quality

As described in Section 4.4, an optimal solution is guaranteed at termination by the root agent which initialises distributed focused decomposition.

It is also possible for the distributed focuses mechanism to provide solution quality guarantee with reduced computation time. When the root agent receives a quality report, which contains a bound, the optimal solution would be in the range $[0, bound]$. If the quality guarantee we want is greater than or equal to the bound, we already have an acceptable solution in one of the agents which reports the bound. It is because a quality report indicates the quality of a focused solution, and a focused solution is also a good quality solution. On the other hand, if the quality guarantee we want is less than the bound, the root agent can search for an optimal solution in the range $[quality\_guarantee, bound]$.

5.2.2 Time efficiency

There are three attributes in the distributed focused decomposition algorithm that contribute to the improvement in efficiency. First, focused decomposition reduces problem sizes. Smaller problem sizes simply require less computation time, especially when we are dealing with computational complex problems. Second, a focused search employed by agents further reduces a problem into sub-sections. Searching on 2 sub-sections with size $N/2$ each is faster than searching on a single problem with size N, because of the exponential growth of search space. Finally and most importantly, discovering effective bounds improves the search performance significantly in every level of an hierarchy. The performance benefit of effective bounds is demonstrated by the experimental result versus a centralised approach in Section 5.1.

5.2.3 Memory efficiency

Similar to time efficiency, a decomposition based approach reduces problem sizes, and also reduces the search space to explore. A* search stores its partial solution exploration in memory. The memory requirement of a focused search is roughly half of what is required by a normal A* search. This is because after the first stage of optimal search is completed, memory already used in keeping partial solutions can be recovered before starting the second stage search. The effective bounds discovered also reduce the requirement of memory space significantly, as evidenced by the reduced size of the partial solution priority queue at termination in Table 2.1.

5.2.4 Communication efficiency

The Communication efficiency aspects of multi-agent cooperation strategy are discussed in Section 4.5. To solve DCOP with distributed focused decomposition, our multi-agent cooperation strategy requires exactly $5 \times (N - 1)$, i.e. $O(N)$, in number of communication messages with respect to the overall problem size. The size of communication messages is bounded by the size of a problem representation.

5.2.5 Distribution effectiveness

The distribution effectiveness of a DCOP algorithm is related to the distribution/communication trade-off in Section 3.2. Although problem centralisation happens in the problem identification phase (Section 4.5), our distributed focused decomposition mechanism distributes problem focuses among agents. Each agent explores a different sub-section of the overall problem for optimality.

The performance improvements, in both time efficiency and memory efficiency, are the result of distributed focused decomposition. It can be therefore concluded that the distributed focuses mechanism is effective in distributing problem sub-sections for efficiency.

5.3 Summary

We have seen that DistDecomp terminates faster than Adopt and also faster than a single A* search. Adopt generates exponential number of communication messages, while DistDecomp requires linear number of messages.

The analysis of distributed focused decomposition shows that our proposed solution to DCOP addresses all five requirements, it guarantees solution optimality, improves time efficiency, requires less memory than a normal optimal search, is communication efficient, and effectively utilises distribution.

Chapter 6

Unsuitable scenarios and Possible improvement

Agents with private local knowledge

Our approach relies on cooperative agents that are willing to share information. Therefore, it is not suitable for some multi-agent systems in an open environment where agents would like to keep local knowledge private.

Also, it is assumed that agents participating in a problem solving session cooperate, trust one another, and give no false information.

Agent failures

In our agent hierarchy, the root agent has special responsibilities such as problem identification, distribution initiation, and termination detection. It is important that the root agent functions throughout the entire problem solving session. Therefore, the root agent becomes the single point of failure, similar to other centralised approaches.

Special configurations such as backup agents, redundant agents, root agent re-election, etc. need to be considered if it is applied to multi-agent systems that require fault-tolerant.

Sessions and connections

The distributed focuses approach to solving DCOPs includes the concept of a problem solving session. A session is a temporary organisation of a group of participating agents. It primarily deals with agents, but not communication between agents. In other words, a

session contains agents only. A connection is used to refer to the communication between two agents. Communication messages are sent and received through a connection. The root agent assumes the duty of managing a cooperative problem solving session. A multi-agent application can have multiple concurrent sessions. Each session has multiple agents. Each agent has multiple connections with other agents. An agent can join multiple sessions at the same time.

Idle agents

In a hierarchy, agents with smaller problems finish searching faster. In the time after an agent finishes its search and before a solution assignment from the root agent is received, it becomes idle. Depending on application requirements, an idle agent can obtain another problem assignment from the same problem solving session, or it can join different sessions to utilise computation resources.

Solving satisfaction

The focused decomposition based approach can be modified to solve satisfaction problems. After distribution of focuses, an agent can apply a rejecting non-solution style of search strategy such as backtracking for satisfaction. An agent checks for constraint violations of the higher variables in a problem focus first, and the rest of the variables later. When a violation is found, it can immediately stops and report the violation. In this way, a group of agents can decide unsatisfiable problems faster, but it may not be better than a basic backtracking search for satisfiable problems.

Cooperation with different search strategies

Since each agent solves a focused problem independently and only the quality of a solution is communicated, different agents in the same problem solving session can use different search strategies. For example, since a focused solution may not be optimal and optimality is only guaranteed at the root agent. Other agents down the hierarchy can use an efficient local search strategy to obtain a solution, the root agent uses an optimal A* search with a bound found by local search. It may further improve overall efficiency for large complex problems, assuming local search is faster than A* search for most of the cases.

Variations of agent hierarchies

The agent hierarchy can be arbitrary and does not need to be a binary tree. An agent can construct multiple different focused problems for multiple agents under its sub-hierarchy. It could be useful to have more agents focusing on sub-sections with higher density of constraints.

Dynamic variants

A dynamic variant is easily facilitated by allocating decomposed constraint graphs to agents at run time based on previous work on Role-graph matching that utilises polynomial time bipartite matching of sub-problems (or roles) to agents [Soon *et al.*, 2004]. In this variant, when an agent finishes it is allocated another subgraph to solve, by adapting multi-stage search. Another variant has the parent assign a different sub-problem whenever it receives a quality report from a child. Since agents associate different bounds to different sub-problems, they can dynamically allocate children to explore different sub-problems. Some work into this approach has also been investigated by [Mailler and Lesser, 2006b], although the potential of DistDecomp's polynomial communication requirements is promising.

Chapter 7

Conclusions and Future work

The thesis presents a novel decomposition algorithm for searching and solving the distributed constraint optimisation problem.

In particular, the formalism of the DCOP definition has been updated so that it facilitates flexibility in variable distribution among agents, identified a *generate and compare* approach for solving optimisation problems that is different from the *generate and test* approach for solving satisfaction problems, analysed the trade-off between distribution and communication in distributed constraint processing, formulated a decomposition-based search algorithm involving distribution of optimality search, defined formally a focused problem, a focused solution, a core problem and a focused decomposition utilising graph decomposition and vertex permutations for constraint problems, and introduced a multi-agent cooperation strategy including a communication protocol for initiating and terminating cooperative problem solving sessions.

A DCOP can be solved by cooperative agents in four phases: (1) problem identification, (2) problem distribution by focused decomposition, (3) solution quality report to refine solution bounds, and (4) solution propagation. The distributed focused decomposition mechanism guarantees solution optimality, improves time efficiency, reduces memory requirement, is communication efficient, and distributes optimality search effectively.

To the best of our knowledge, it is first complete and optimal algorithm for solving distributed constraint optimisation problems with polynomial communication complexity (in relation to both number of messages and message size).

Future research work

There are two area of related research works that may worth further investigations:

(1) Dynamic problem solving: In this thesis, we are dealing with distributed constraint problems that are static. A lot of real time applications require optimal solutions to dynamically changing constraints. Continually maintaining optimality of distributed constraints is also desirable for agents situated in dynamic environments. To handle dynamic problems, problem update messages can be introduced into the cooperative problem solving session.

(2) Co-relation between focus size and solution optimality: It appears that there is co-relation between the size of a focus and the solution optimality. It may be possible to analyse the optimality of a problem by solving with varying sizes of focuses and predict the optimal cost, so that solution can be assigned without rigorous solving.

Bibliography

[Bejar *et al.*, 2005] R. Bejar, C. Domshlak, C. Fernandez, K. Gomes, B. Krishanmachari, B. Selman, and M. Valls. Sensor netowkrs and distributed CSP: communication, computation and complexity. *Artif. Intell.*, 161(1-2):117–148, 2005.

[Bessiere *et al.*, 2005] C. Bessiere, A. Maestre, I. Brito, and P. Meseguer. Asynchrnous backtracking without adding links: A new member in the abt family. *Artif. Intell.*, 161(1-2):7–24, 2005.

[Capelle *et al.*, 2002] Christian Capelle, Michel Habib, and Fabien de Montgolfier. Graph decompositions and factorizing permutations. *Discrete Mathematics and Theoretical Computer Science*, 5:55–70, 2002.

[Davin and Modi, 2005] John Davin and Pragnesh Jay Modi. Impact of problem centralization in distributed constraint optimization algorithms. In *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1057–1063. ACM, 2005.

[Davin and Modi, 2006] John Davin and Pragnesh Jay Modi. Hierarchical variable ordering for distributed constraint optimization. In *Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1433–1435. IEEE Computer Society, 2006.

[Davis and Smith, 1983] Randall Davis and Reid G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20:63–109, 1983.

[Genesereth *et al.*, 2005] Michael Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the AAAI competition. *AI Magazine*, 26:62–72, 2005.

[Gottlob *et al.*, 2000] Georg Gottlob, Nicola Leone, and Francesco Scarcello. A comparison of structural CSP decomposition methods. *Artificial Intelligence*, 124:243–282, 2000.

[Habib *et al.*, 1999] Michel Habib, Christophe Paul, and Laurent Viennot. Partition refinement techniques: An interesting algorithmic tool kit. *Int. J. Found. Comput. Sci.*, 10(2):147–170, 1999.

[Hannebauer, 2002a] Markus Hannebauer. *Autonomous Dynamic Reconfiguration in Multi-Agent Systems: Improving the Quality and Efficiency of Collaborative Problem Sovling*, chapter 3. Distributed Constraint Problems–A Model for Collaborative Problem Solving, pages 27–57. LNAI 2427. Springer-Verlag, 2002.

[Hannebauer, 2002b] Markus Hannebauer. *Autonomous Dynamic Reconfiguration in Multi-Agent Systems: Improving the Quality and Efficiency of Collaborative Problem Sovling*, chapter 2. Basics of Collaborative Problem Solving, pages 9–24. LNAI 2427. Springer-Verlag, 2002.

[Hannebauer, 2002c] Markus Hannebauer. *Autonomous Dynamic Reconfiguration in Multi-Agent Systems: Improving the Quality and Efficiency of Collaborative Problem Sovling*, chapter 4. Autonomous Dynamic Reconfiguration–Improving Collaborative Problem Solving, pages 59–94. LNAI 2427. Springer-Verlag, 2002.

[Jung and Tambe, 2005] Hyuckchul Jung and Milind Tambe. On communication in solving distributed constraint satisfaction problems. In Michael Pechoucek, Paolo Petta, and László Zsolt Varga, editors, *Multi-Agent Systems and Applications IV*, LNCS 3690, pages 418–429. Springer-Varlag, 2005.

[Karypis and Kumar, 1998] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[Klyne and Caroll, 2004] Graham Klyne and Jeremy J. Caroll. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. World Wide Web Consortium, February 2004. *http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/*.

[Larrosa and Meseguer, 2002] Javier Larrosa and Pedro Meseguer. Partition-based lower bound for Max-CSP. *Constraints*, 7:407–419, 2002.

[Law and Pearce, 2006] Terence H.-W. Law and Adrian R. Pearce. A multi-stage graph decomposition algorithm for distributed constraint optimisation. In *Proceedings of IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT)*, pages 506–511. IEEE Computer Society, 2006.

[Liu and Sycara, 1994] JyiShane Liu and Katia P. Sycara. Distributed problem solving through coordination in a society of agents. In *Proceedings of the 13th International Workshop on Distributed Artificial Intelligence*, 1994.

[Liu and Sycara, 1995] JyiShane Liu and Katia P. Sycara. Exploiting problem structure for distributed constraint optimization. In *Proceedings of the First International Conference on Multiagent Systems*, pages 246–253, 1995.

[Mailler and Lesser, 2004] Roger Mailler and Victor Lesser. Solving distributed constraint optimization problem using cooperative mediation. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 438–445. IEEE Computer Society, 2004.

[Mailler and Lesser, 2006a] Roger Mailler and Victor R. Lesser. Asynchronous partial overlay: A new algorithm for solving distributed constraint satisfaction problems. *Journal of Artificial Intelligence Research (JAIR)*, 25:529–576, 2006.

[Mailler and Lesser, 2006b] Roger Mailler and Victor R. Lesser. A cooperative mediation-based protocol for dynamic distributed resource allocation. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, 36(1):80–91, 2006.

[Mailler, 2005] Roger Mailler. Comparing two approaches to dynamic, distributed constraint satisfaction. In *Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1049–1056. IEEE Computer Society, 2005.

[Messmer and Bunke, 2000] Bruno T. Messmer and Horst Bunke. Efficient subgraph isomorphism detection: a decomposition approach. *IEEE Transactions on Knowledge and Data Engineering*, 12(2):307–323, 2000.

[Mockapetris, 1987] P. Mockapetris. *RFC 1034: Domain Names – Concepts and Facilities.* The Internet Engineering Task Force, November 1987. IETF Standard.

[Modi *et al.*, 2003] Pragnesh Jay Modi, Wei-Min Shen, and Milind Tambe. An asynchronous complete method for distributed constraint optimization. In *Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 161–176. IEEE Computer Society, 2003.

[Modi *et al.*, 2005] Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161:149–180, 2005.

[Newell *et al.*, 1959] Allen Newell, J. C. Shaw, and Herbert A. Simon. Report on a general problem-solving program. In *Proceedings of the International Conference on Information Processing*, pages 256–264, 1959.

[Pearl, 1983] Judea Pearl. Knowledge versus search: A quantitative analysis using A*. *Artificial Intelligence*, 20:1–13, 1983.

[Petcu and Faltings, 2005] Adrian Petcu and Boi Faltings. A scalable method for multiagent constraint optimization. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 266–271, 2005.

[Petcu and Faltings, 2007] Adrian Petcu and Boi Faltings. MB-DPOP a new memory-bounded algorithm for distributed optimization. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI) (to appear)*, 2007.

[Petcu *et al.*, 2007] Adrian Petcu, Boi Faltings, and Roger Mailler. PC-DPOP a new partial centralization algorithm for distributed optimization. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI) (to appear)*, 2007.

[Scerri *et al.*, 2006] Paul Scerri, Régis Vincent, and Roger Mailler. Comparing three approaches to large-scale coordination. In Paul Scerri, Régis Vincent, and Roger Mailler, editors, *Coordination of Large-Scale Multiagent Systems*, pages 53–71. Springer, 2006.

[Silaghi and Faltings, 2005] M. C. Silaghi and B. Faltings. Asynchronous aggregation and consistency in distributed constraint satisfaction. *Artif. Intell.*, 161(1-2):25–54, 2005.

[Slechta, 2005] Petr Slechta. Decomposition and parallelization of multi-resource timetabling problems. In Edmund Burke and Michael Trick, editors, *Practice and Theory of Automated Timetabling V*, LNCS 3616, pages 177–189. Springer-Verlag, 2005.

[Solotorevsky and Gudes, 1996] Gadi Solotorevsky and Ehud Gudes. Algorithms for solving distributed constraint satisfaction problems (DCSPs). In *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS)*, pages 191–198. AAAI Press, 1996.

[Soon *et al.*, 2004] Susannah Soon, Adrian Pearce, and Max Noble. Adaptive teamwork coordination using graph matching over hierarchical intentional structures. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 294–301. IEEE Computer Society, 2004.

[Soon, 2005] Susannah Soon. *Multi-agent Teamwork Coordination: A Graph-Based Intention Recognition Approach.* PhD thesis, Department of Computer Science and Software Engineering, The University of Melbourne, December 2005.

[Yadgar *et al.*, 2003] Osher Yadgar, Sarit Kraus, and Jr. Charles L. Ortiz. Hierarchical information combination in large-scale multiagent resource management. In Marc-Philippe Huget, editor, *Communication in Multiagent Systems*, LNAI 2650, pages 129–145. Springer-Verlag, 2003.

[Yokoo and Hirayama, 2000] Makoto Yokoo and Katsutoshi Hirayama. Algorithms for distributed constraint satisfaction: A review. *Autonomous Agents and Multi-Agent Systems*, 3(2):198–212, 2000.

[Yokoo and Ishida, 1999] Makoto Yokoo and Toru Ishida. Search algorithms for agents. In Gerhard Weiss, editor, *Multiagent systems: a modern approach to distributed artificial intelligence*, chapter 4, pages 165–199. The MIT Press, 1st edition, 1999.

[Yokoo *et al.*, 1998] 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*, 10(5):673–685, 1998.

[Zivan and Meisels, 2006] Roie Zivan and Amnon Meisels. Message delay and DisCSP search algorithms. *Ann. Math. Artif. Intell.*, 46:415–439, 2006.