CONFLICT RESOLUTION STRATEGIES AND THEIR PERFORMANCE MODELS FOR LARGE-SCALE MULTIAGENT SYSTEMS

by

Hyuckchul Jung

A Dissertation Presented to the

FACULTY OF THE GRADUATE SCHOOL

UNIVERSITY OF SOUTHERN CALIFORNIA

In Partial Fulfillment of the

Requirements for the Degree

DOCTOR OF PHILOSOPHY

(COMPUTER SCIENCE)

December 2003

Dedication

This thesis is dedicated to my loved family...

Acknowledgements

After long years of formal school education, I won the highest degree from one of the top universities in my field. Recalling the past, I found that, without the support of many people, it would have not been possible for me to make this great achievement.

First of all, I would like to express my deepest gratitude to my advisor, Milind Tambe. As a researcher, he has taught me passion, patience, devotion, and thoroughness which are the necessity for a true researcher. As a teacher, he has given me endless care and encouragement which helped me overcome a lot of difficulties throughout my Ph.D. program. Furthermore, as a human being, he has shown wit and humor (sometimes).

I am very grateful to my thesis committee members who deserve many thanks for their sincere advice and hearty guidance. I was very lucky to have some of the best researchers in my research area on my thesis committee. Paul Rosenbloom pointed out what I had accepted as a matter of course, and made me think of my work in new perspectives. Ed Durfee spent enormous amount of time in discussing my thesis topics and helped me see a big picture in them. Guarav Sukhatme always made himself available to provide help and technical advice for my research. Finally, Dan O'Leary always encouraged me with his cheerful comments and provided guidance from non-CS perspective.

While Tony Barrett at JPL was not in my thesis committee, he provided me with a lot of pointers based on his practical experience and gave me good and thorough comments on my thesis. He also helped me improve my writing in English and gave me good suggestion for my thesis presentation. I also owe a lot to Brad Clement at JPL who helped me clarify what were the assumptions in my thesis. It was a great privilege to communicate with Makoto Yokoo at NTT in Japan when I started to work on DCSP. As a leading researcher in the field, he gave me insights into DCSP and provided me with kind comments on my work. I wish to give special thanks to Jeff Bradshaw and James Allen at IHMC for their understanding and support when I had difficulties in completing my thesis as scheduled.

I would like to thank my colleagues who were and have been at USC and ISI. In particular, I wish to thank Gal Kaminka, Paul Scerri, David Pynadath, Jay Modi, Ranjit Nair, Nathan Schurr, and Praveen Paruchuri for their time and effort to improve my presentation. My grateful thanks also go to my Korean friends at USC who helped me settle down when I first came to US and were always with me at difficult times as good friends. Just being with them gave me great comfort.

Finally, I would like to thank my family. Whichever decision I made, my parents have always supported me and sacrificed themselves for my success. I know that there is no way for me to compensate for my indebtedness to them. I have always sought advice from my sister whose counsel was a great help to me at every decision. My loved wife, Jaeyeon, is a great gift from God. When I had the most difficult time in my Ph.D. program, she was there and, with her help, I could cope with the situation making this great achievement. I thank her for everything and love her not because she has given me help but because she is just here on earth with me.

The research in this thesis was sponsored by NASA Jet Propulsion Laboratory subcontract “Continual Coherent Team Planning”. The earlier work for cooperative strategies was funded by DARPA ITO award number F30602-99-2-0507.

Contents

List of Figures

List of Tables

Abstract

Distributed, collaborative agents are promising to play an important role in large-scale multiagent applications, such as distributed sensors and distributed spacecraft. Since no single agent can have complete global knowledge in such large scale applications, conflicts are inevitable even among collaborative agents over shared resources, plans, or tasks. Fast conflict resolution techniques are required in many multiagent systems under soft or hard time constraints. In resolving conflicts, we focus on the approaches based on DCSP (distributed constraint satisfaction problems), a major paradigm in multiagent conflict resolution. We aim to speed up conflict resolution convergence via developing efficient DCSP strategies.

We focus on multiagent systems characterized by agents that are collaborative, homogeneous, arranged in regular networks, and relying on local communication (found in many multiagent applications). This thesis provides the followings major contributions. First, we develop novel DCSP strategies that significantly speed up conflict resolution convergence. The novel strategies are based on the extra communication of local information between neighboring agents. We formalize a set of DCSP strategies which exploit the extra communication: in selecting a new choice of actions, plans, or resources to resolve conflicts, each agent takes into account how much flexibility is given to neighboring agents. Second, we provide a new run-time model for performance measurement of DCSP strategies since a popular existing DCSP performance metric

does not consider the extra communication overhead. The run-time model enables us to evaluate the strategy performance in various computing and networking environments. Third, the analysis of message processing and communication overhead of the novel strategies shows that such overhead caused by the novel strategy is not overwhelming. Thus, despite extra communication, the novel strategies indeed show big speedups in a significant range of problems (particularly for harder problems). Fourth, we provide categorization of problem settings with big speedups by the novel strategies Finally, to select the right strategy in a given domain, we develop performance modeling techniques based on distributed POMDP (Partially Observable Markov Decision Process) based model where scalability issue is addressed with a new decomposition technique.

Chapter 1

Introduction

Distributed, collaborative agents play an important role in large-scale multiagent applications [Durfee, 1991; Grosz, 1996; Lesser, 1999]. Examples of such applications include distributed sensor networks [Lesser *et al.*, 2003], distributed spacecraft [Barrett, 1999], distributed disaster rescue applications [Kitano *et al.*, 1999], agent assisted human organizations [Scerri *et al.*, 2001], cooperative teams of robots [Parker, 1993], and virtual environments for training [Rickel and Johnson, 1997]. In these multiagent applications, collaborative agents must coordinate their plans and allocate adequate resources to carry out the plans [Durfee, 2001; Liu and Sycara, 1996; Neiman *et al.*, 1994].

While such applications require agents to be collaborative, conflicts are inevitable even among collaborative agents over shared resources, joint plans, or tasks, since no single agent can have complete global knowledge in such large-scale applications [Müller and Dieng, 2000; Tessier *et al.*, 2000]. In distributed, dynamic and complex environments, centralized conflict resolution is often impractical because of computational and communication bottleneck (in particular for a large-scale multiagent system), the vulnerability of system failure, and security risks. Therefore, agents need to resolve conflicts in a distributed manner without global knowledge. Distributed conflict resolution is thus a fundamental challenge in multiagent systems [Liu *et al.*, 1998; Müller and Dieng, 2000; Tessier *et al.*, 2000; von Martial, 1991].

In many multiagent systems under soft and hard time constraints, fast conflict resolution is required. For instance, in distributed sensor networks, agents must resolve conflicts over shared sensors as quickly as possible since a delay in conflict resolution may lead to a situation where no targets are tracked before moving out of sensor range [Lesser *et al.*, 2003]. Distributed spacecraft is another example where fast conflict resolution in planning and scheduling is required. A constellation of spacecraft flies in a formation investigating three dimensional structure of the phenomena within the Earth's magnetosphere. Given unpredictable phenomena that change rapidly within the magnetosphere and last for only a few seconds, spacecraft must quickly resolve conflicts in sequencing their sensing activities [Angelopoulos and Panetta, 1998]. We will discuss these domains in detail in Chapter 2 of this thesis.

In resolving conflicts, we focus on the approaches based on DCSP (Distributed Constraint Satisfaction Problems) [Yokoo, 2000]. (For readers unfamiliar with DCSP, a formal definition is introduced in Section 3.1.1.) DCSP-based approach is a major technique in multiagent conflict resolution: it provides rich foundation for the representation of conflict resolution problems and there exist highly efficient baseline algorithms [Liu and Sycara, 1993; Mammen and Lesser, 1998; Modi *et al.*, 2003; Sathi and Fox, 1989; Silaghi *et al.*, 2000; Yokoo and Hirayama, 1998; Zhang and Wittenburg, 2002]. Market-based conflict resolution approaches have been also applied to resolving conflicts among collaborative agents [Hunsberger and Grosz, 2000; Walsh and Wellman, 1998], and while the key results of this thesis bear upon those approaches as well (see Chapter 6: Related work), we will not explicitly target them in this thesis.

While DCSP is a major paradigm in distributed conflict resolution [Yokoo, 2000], existing techniques do not provide fast enough conflict resolution for large-scale multiagent systems, given real-time constraint. For instance, in sensor networks, even the best published DCSP algorithm, ABT/AWC [Yokoo and Hirayama, 1998], cannot resolve conflicts in real-time [Scerri *et al.*, 2003; Zhang *et al.*, 2003]. Scerri et al. reported that, while working on the sensor network domain, they had to augment ABT algorithm with ad-hoc reactive layer for fast conflict resolution [Scerri *et al.*, 2003]. Other researchers have also developed approximate algorithms highly compromising on quality [Lesser *et al.*, 2003].

This thesis focuses on fast conflict resolution techniques (based on DCSP) for multiagent systems. To this end, there are two key areas of contributions in this thesis which naturally divide the thesis into two parts: part 1 (Section 1.1) aims to develop novel conflict resolution strategies and part 2 (Section 1.2) aims to build a formal model to analyze the performance of the strategies.

1.1 Part I: Cooperative Conflict Resolution Strategies

In part 1 of this thesis, we focus on novel conflict resolution strategies that improve the speed of conflict resolution convergence over the current best DCSP technique (by exploiting the communication of local constraints). Such improvement is critical since fast conflict resolution is required in multiagent applications under hard and soft real-time constraint. Section 1.1.1 provides the problem statement of part 1, and Section 1.1.2 introduces our approach to the problem described in Section 1.1.1, and show the results.

1.1.1 Problem Statement of Part 1

This section formally defines the type of conflicts that we focus on and provides the actual problem statement of part 1 in detail. We also illustrate the restriction that is applied to our approach and application domains, thus outlining the scope of our empirical investigation.

Conflict Type and Conflict Resolution

Even in collaborative multiagent systems, since no single agent has accurate and complete global knowledge, it is inevitable that agents enter into conflicts over actions, plans, or resources (that they select) [Müller and Dieng, 2000; Tessier *et al.*, 2000]. In this thesis, we focus on such conflicts over agents' individual action/plan/resource choices which can be characterized as follows:

- Given that there exists a set of agents $\mathcal{A} = \{A_1, ..., A_n\}$ and each agent $A_i \in \mathcal{A}$ has a set (S_i) of action/plan/resource choices, A_i's choice c_i from S_i may *conflict* with another agent's choice (c_j) or a set of other agents' choices ($\{c_j, \cdots, c_k\}$).
  - E.g., in sensor networks, each sensor agent (A_i) has a set (S_i) of resource choices, radar sectors, to scan for a target. Assume that sensor agents aim to find non-overlapping sectors for maximizing sensor coverage. When an agent (A_i) selects a sector ($c_i \in S_i$) while another agent (A_j) selects its own radar sector ($c_j \in S_j$), c_i may conflict with c_j if c_i and c_j are scanning the same area.

Note that we do not focus on other types of conflicts such as contradictory goals or beliefs [Liu *et al.*, 1998]. In this thesis, given the type of conflicts described above, the resolution of the conflicts is formally defined as follows:

- Conflict resolution: Given a set of agents $\mathcal{A} = \{A_1, ..., A_n\}$ and a set of action/plan/resource choices selected by the agents $\mathcal{C} = \{c_1, ..., c_n\}$, if there is any conflict in $\mathcal{C}$, agents find a new set of choices $\mathcal{C}'$ such that there exists no conflict in $\mathcal{C}'$. That is, *conflict resolution* requires *all* conflicts to be resolved.

Goal of Part I

We aim to develop algorithmic techniques that *significantly* speed up DCSP-based conflict resolution in *large-scale* multiagent systems. These multiagent systems are characterized by agents which are:

- Collaborative: Agents collaborate with each other to achieve a shared goal (e.g., conflict resolution) throughout their operational period.
- Homogeneous: Agents are identical with the same capabilities.
- Arranged in regular networks: In real applications such as sensor networks [Lesser *et al.*, 2003], distributed spacecraft [Angelopoulos and Panetta, 1998], and micro-air-vehicles for surveillance [Gordon *et al.*, 1999], agents are often arranged in regular networks.
- Relying on local communication between neighboring agents: Local communication is the type of communication in our domains of interest. For instance, in sensor networks, agents indeed communicate only with neighboring agents. Some benefits from local communication are energy saving and the increase in message transmission reliability [Pottie and Kaiser, 2000; Ganesan *et al.*, 2002].

Significant speedup implies that our approach resolves conflicts faster than AWC technique (the best published DCSP approach [Yokoo, 2000]) by an order of magnitude difference in time to solution. This speedup must be observed over a significant range of problems (in particular for harder problems).

- Solution test: Any algorithmic technique that achieves such significant speedup without centralization (that is, in resolving conflicts, each agent selects its own value).

1.1.2 Summary of Approach and Results

A major characteristic of most DCSP-based approaches for multiagent conflict resolution is that they have focused on *minimal communication*: agents communicate only their intended values for the objects on which they need to agree. While the value selection is based on each agent's local knowledge and local situation, agents do not communicate such information. Instead, only values (minimum information required to be communicated for conflict resolution) are communicated. The major assumptions for minimal communication in the DCSP-based approaches are three fold [Yokoo, 2000]:

1. Security/privacy issues in revealing local information: Competitive agents or non-fully-cooperative agents (that make a temporary coalition) may not want to reveal private information.

2. Knowledge transformation: For heterogeneous agents, if one agent has complicated internal constraint (represented with its own knowledge), communicating such local information needs some translation of one's knowledge into an exchangeable format or other agents must have a capability of interpreting it.

3. Communication overhead: For additional information exchange, increased number of messages and larger message size are required.

In this thesis, we challenge these assumptions for minimal communication. As large-scale systems based on such minimal communication get developed and novel conflict resolution algorithms are developed for the domains of interest in this thesis, it is critical to re-examine the commitment to minimal communication that is the foundation of DCSP. The reason for going beyond minimal communication of values is based on the hypothesis that *"exploiting extra communication of local information can significantly increase the speed of conflict resolution in some domains"*.

Indeed, it is feasible that, by unnecessarily subscribing to such minimal communication, researchers may be forced to compromise on correctness or quality of solutions; and/or forced to develop unnecessarily complex algorithms. In the worst case, entire application arenas may remain out of reach. For instance, since finding a complete solution can be extremely slow in large-scale systems, approximate methods have been proposed as a compromise in DCSP-based approaches which rely on minimal communication [Lemaitre and Verfaillie, 1997; Modi *et al.*, 2003]. However, strategies which exploit extra communication could speed up conflict resolution (as shown in this thesis) and make complete methods more feasible in practice.

Questioning minimal communication does not automatically imply centralizing all computation. There is a big space between current methods with minimal communication and fully centralized approaches, and this thesis has taken a key step in exploring that space. Thus, we remain faithful to our assumption of distributed conflict resolution.

Going back to the assumptions regarding minimal communication, note that, in this thesis, assumptions (1) and (2) are not applicable:

1. Security/privacy issue: Collaborative agents have no reason to hide information from other agents.

2. Knowledge transformation issue: Many of the domains we focus on involve homogeneous agents, which nullifies the knowledge transformation issue. Also,

recent advances in the infrastructure for collaborative agents may enable heterogeneous agents to seamlessly communicate with each other using uniform proxies [Tambe *et al.*, 2000], mitigating the knowledge transformation issue.

Finally, we question the one remaining assumption of communication overhead (the third assumption above) that has conspired towards minimal communication. There can be trade-offs in time to solution due to communicating extra local information:

- *Increase*: Extra communication of local information can increase individual agents' computation and communication cost since they need to process and transmit additional information, thus increasing time to solution.
- *Decrease*: If extra communication help agents resolve conflicts quickly (while computation and communication cost may initially increase), overall time to solution (required until a solution is found) may decrease.

The analysis of such trade-offs is an empirical question and various aspects of different types of domains need to be considered in the analysis (e.g., communication or local computation cost). This empirical analysis is at the heart of part 1 of this thesis. Chapter 4 provides detailed results and analysis of the experimentation.

Based on the discussion above, we propose to develop fast conflict resolution techniques that exploit the extra communication of local information between neighboring agents. In particular, our techniques focus on how an agent selects a choice of actions, plans, or resources to resolve a given conflict (referred as *distributed conflict resolution strategy* below). Here, we present the following terminologies regarding the techniques:

- *Distributed conflict resolution strategy* (*strategy* for short): A heuristic function ($\mathcal{F}_i$) by which an agent (A_i) selects a choice (c_i) from a set (S_i) of actions, plans, or resources available to A_i in resolving conflicts that involve A_i.

  - E.g, min-conflict strategy in which, given conflicts, an agent (A_i) selects a choice (c_i) from a set (S_i) of actions, plans, or resources that minimizes the number of conflicts with others' choices known to A_i [Minton *et al.*, 1992].
  - A strategy can have a substantial impact on the time for conflict resolution: if each agent is able to select a choice that has the least possibility of conflicting with others' choices, there is a potentially good chance for agents to rapidly resolve all the conflicts.

- *Local information*: Information only locally known to an agent A_i that affects its choice (c_i) of actions/plans/resources from S_i.
  - E.g., in sensor networks, a sensor agent may have a limited set of available radar sector combinations since one of its radar sectors is out of order. But, the limited set is unknown to a neighboring agents.
  - We limit the extra communication to the local information defined above: information irrelevant to the restriction on agents' choices is not communicated.

We envision that *strategies* based on the extra communication of local information can enable fast conflict resolution. However, in the multiagent literature, such *strategies* remain largely uninvestigated. We need to formalize *strategies* in a general framework so that they can be applied in different problem settings, and investigate their performance and trade-offs to selectively apply them for a given domain.

We formulate different conflict resolution strategies, varying the level of cooperativeness to neighboring agents. For instance, when distributed agents must resolve conflicts over shared resources such as shared sensors, they could select a strategy that offers maximum possible resources to most constrained agents, or a strategy that distributes resources equally among all agents requiring such resources, and so on. A key measure that enables agents to explicitly reason about their cooperativeness towards neighboring agents is the flexibility (number of available action/plan/resource choices) given to neighboring agents.

With regard to the issue of "which level of non-local awareness should agents have in communicating local information?", we focus on a specific level where agents get local information only from their neighboring agents because of the following reasons:

- The approach in this thesis has relation to a popular centralized CSP technique, the *least-constraining-value* heuristic, which chooses a value that rules out the smallest number of values in variables connected to the current variable by constraints [Haralick and Elliot, 1980]. We investigate whether a DCSP version of the popular centralized CSP technique (enhanced by the extra communication of local information) can improve performance of DCSP-based conflict resolution. [1] In the DCSP literature, this investigation of value selection techniques based on the extra communication is the first in the DCSP literature.

- Appendix A shows the comparison between the thesis approach and an approach based on global awareness (an extreme case of non-local awareness) in terms of constraint checks and message sizes per agent. We avoid strategies with global awareness since they lead to significant increase in message sizes and constraint checks per agent. While strategies with more non-local awareness than

[1]Note that our approach is not just a simple mapping of the *least-constraining-value* heuristic onto the DCSP framework. With strategies (defined in Chapter 3), agents explicitly reason about which agents to consider most with respect to the constrainedness given towards neighboring agents.

the approach in this thesis (but much less non-local awareness than global awareness) can be feasible, this thesis focuses on a specific level of non-local awareness described above, opening the space of different strategies based on non-local awareness and showing the potential/feasibility of such strategies. Note that, even with a fixed level of non-local awareness, there is a large problem space to investigate as described in Section 4.3.1.

While previous work has investigated different types of conflict resolution techniques [Decker and Lesser, 1995; Prasad and Lesser, 1997; von Martial, 1991; Zhang and Lesser, 2002], to the best of our knowledge, this is the first detailed systematic investigation of collaborative strategies in varying the degree of cooperativeness towards neighboring agents in a large scale multiagent system. We illustrate the presence of multiple cooperative conflict resolution strategies and systematically investigate the performance of these strategies in conflict resolution convergence.

We also develop a run-time model to measure the performance of strategies since our approach is based on the extra communication of local information which can increase the computation and communication cost. The major performance metric used in DCSP, *cycles* (explained in detail in Section 4.1), does not take into account the overhead from increased message number and size due to the extra communication. [2] Therefore, we propose a run-time model which takes into account computation and computation cost in different computing/networking environments. Chapter 4 provides the performance analysis of strategies based on the run-time model.

[2]*Cycles* has been used as a key measurement of DCSP strategy performance [Bessière *et al.*, 2001; Fernàndez *et al.*, 2003; Modi, 2003; Silaghi *et al.*, ; Yokoo and Hirayama, 1998; Zhang *et al.*, 2003] since the amount of local computation and communication that each agent solves mostly remains same in the most of previous DCSP approaches: the difference is in the protocol for passing values and controlling backtracking. However, *cycles* could be an inappropriate measurement when there exists a big variation in message size/number among strategies and such variation is not ignorable in a given computing/networking environments.

In the empirical investigation for the performance and trade-offs of the strategies, there is an exponential search space as we consider different types of strategies. Thus, we set a boundary in the investigation as follows:

- Homogeneous strategy (a homogeneous strategy implies that, given a conflict resolution problem, all the agents use an identical strategy for conflict resolution): We investigate homogeneous strategies as a key step to verify the hypothesis that exploiting extra communication of local information can enable fast conflict resolution. Indeed, conflict resolution strategies that exploit extra communication show performance improvement over the min-conflict strategy which does not assume local information communication (Chapter 4). Building on our investigation, heterogeneous strategies could help further improve the performance of conflict resolution.

Even with the above assumption, we have a large problem space for systematic investigation (with 208,845 experiments in 351 different problem settings) as explained in Section 4.3.1. The experimentation in this thesis is the most extensive systematic investigation of DCSP strategies with extra communication of local information. The following is the result of the experimentation:

1. Strategies based on extra communication can indeed speed up the conflict resolution convergence in a significant range of problem settings, particularly for harder problems with more than an order of magnitude difference in run-time (compared with the best published DCSP technique, AWC [Yokoo and Hirayama, 1998]).

    - The overhead from increased message processing and communication with extra communication is not overwhelming, and do not cause serious degradation in conflict resolution convergence.

2. It is not always the case that more information communication leads to improved performance (in some cases, strategies with extra communication perform worse than a strategy that does not communicate local information).

3. The strategy that provides maximum flexibility to all neighboring agents is not always the best.

4. No single strategy dominates the other strategies across all domains (a strategy that performs the best in one domain setting can be an order of magnitude slower than another strategy in a different setting).

In particular, the fourth result implies that, for fast conflict resolution convergence, agents are required to adopt the right strategy in a given domain. Furthermore, based on systematic experimentation, we provide categorization of domains where high speedups can be achieved by the strategies with extra communication (formally defined in Chapter 3). Given a domain, such categorization can provide a guidance for whether to apply our approach or not.

1.2 Part II: Performance Models for Conflict Resolution Strategies

In part 2 of this thesis, we focus on techniques for performance modeling of conflict resolution strategies that can help select the right strategy in a given domain. Given the variable nature of multiagent application domains and a wide variety of conflict resolution strategies, a key challenging question is how to select the strategy which will show the best performance in a given domain. Performance modeling of conflict resolution strategies could help predict the right strategy to adopt for a given domain, leading to maximum speedup in conflict resolution convergence. Unfortunately, performance modeling has not received significant attention in the mainstream multiagent research community: although within subcommunities such as mobile agents, performance modeling has been considerably investigated [Rana, 2000].

In this thesis, we provide formal performance models for conflict resolution strategies to select the right strategy in a given large scale multiagent domain. Section 1.2.1 provides the actual problem statement of part 2, and Section 1.2.2 describes our approach for the performance modeling problem.

1.2.1 Problem Statement of Part II

We aim to develop a formal model by which we can compare conflict resolution strategies in multiagent systems by predicting their performance. The multiagent systems are characterized by agents which are [3]:

- Collaborative
- Homogeneous
- Arranged in regular networks
- Relying on local communication

As an initial approach to performance modeling for conflict resolution strategies, the goal of part 2 is not to provide a formal model which computes exact real run-time of conflict resolution strategies. Instead, for the purpose of selecting the right strategy in

[3]Note that the properties of agents are the same with those of agents assumed in the problem statement of part 1 of this thesis

a given domain, we aim to provide a formal model which distinguishes better performing strategies with worse performing strategies. Note that we focus on homogeneous strategies in this investigation as in part 1.

- Solution test: Any formal model that distinguishes better performing strategies with worse performing strategies in a given domain with high correlation between experimental results (e.g., *cycles*) and analytical results from the model.

1.2.2 Summary of Approach and Results

Our performance models are based on recent research in distributed POMDPs (partially observable Markov decision processes) and MDPs (Markov decision processes) that has begun to provides key tools in modeling the performance of multiagent systems [Bernstein *et al.*, 2000; Boutilier, 1999; Pynadath and Tambe, 2002; Xuan and Lesser, 2002]. However, there are at least two major problems in applying such distributed POMDP and MDP based models for performance modeling. First, while previous work has focused on modeling communication strategies within very small numbers of agents [Pynadath and Tambe, 2002], we are interested in strategy performance analysis for large-scale multiagent systems. Second, techniques to apply such models to investigate conflict resolution strategies have not been investigated.

To model the performance of conflict resolution strategies, we provide formal models based on MTDP (Markov Team Decision Problem) [Pynadath and Tambe, 2002], a distributed POMDP model for multiagent analysis, although we could use other distributed POMDP based models such as DEC-POMDP (decentralized POMDP) [Bernstein *et al.*, 2000], POIPSG (Partially Observable Identical Payoff Stochastic Game) [Peshkin *et al.*, 2000] and Xuan-Lesser framework [Xuan *et al.*, 2001]. We illustrate how conflict resolution strategies can be modeled in the MTDP. A conflict resolution strategy is mapped onto a policy in the MTDP, and the mapped policy is evaluated to predict the performance of the strategy. The MTDP provides tools for varying key domain parameters to compare the performance of different conflict resolution strategies.

A key difficulty in the MTDP-based performance modeling for a large-scale multiagent system is the combinatorial explosion of the problem space. To address this scale-up issue, we have developed a new technique based on small-scale models (called *"building blocks"*) that represent the local interaction among a small group of agents. While MDP and POMDP decomposition techniques have been introduced in the literature [Dean and Lin, 1995; Hauskrecht *et al.*, 1998; Parr, 1998; Pineau *et al.*, 2001], the novelty of our building-block-based technique is three fold:

1. Rather than exploiting geometric clusters within a domain (suitable in a single-agent POMDP), building-blocks exploit the multiagent nature of the domain — the decomposition is based on clustering agents with close interactions with each other.

2. We are focused on evaluating policies rather than searching for optimal policies.

3. Observability conditions within a building block can be exploited to further reduce the complexity of policy evaluation.

The building block based approach enables efficiency in computation (by significant reduction of search space) and reusability of building blocks in different domains which have some commonalities. We investigate several ways to combine building blocks for performance prediction of a larger-scale multiagent system, and present the performance analysis results. Combining building blocks by taking into account their interaction - via agents acting as docking points between building blocks - performs the best in predicting the performance of conflict resolution strategies: better performing strategies can be distinguished from worse performing strategies with statistical significance.

1.3 Contribution

In part 1, for fast conflict resolution in multiagent systems, we provide novel strategies that exploit extra communication of local information. Our approach breaks the barrier of conventional DCSP algorithms which focus on minimizing communication, and opens a big space of approaches to solving DCSP. Through systematic experimentation (which is the most extensive empirical investigation of DCSP strategies with extra communication), we show that the novel strategies can significantly increase the speedup of conflict resolution convergence in a significant range of problems (in particular, for harder problems). We also presents a categorization of problem settings where our approach provides big speedups, providing a guidance for whether to apply our approach or not given a domain.

In addition to developing the novel strategies, we provide a run-time model to evaluate the performance of the strategies since an existing popular performance measurement does not take into account the overhead from extra communication in our approach. As the first analytical model for the performance measurement in DCSP which takes into account the overhead (e.g., increase message size/number) from extra communication, the run-time model could provide a useful method for performance measurement to the DCSP community. Furthermore, as shown in Chapter 4, the run-time model provides interesting results in different computing/networking environments: we can easily simulate different computing/networking environments by changing parameters for message processing/communication overhead in the model.

Part 2 addresses the performance modeling problem for conflict resolution strategies which remains largely uninvestigated in the multiagent literature. We provide a distributed POMDP-based model by which we can distinguish better performing strategies with worse performing strategies in a given domain. We addresses the scalability issue by introducing small scale models (called building blocks) and presenting several methods of combining the building blocks. We also show that, under certain circumstances, the evaluation of a distributed POMDP policy can be reduced into the evaluation of a Markov chain.

In summary, the research in this thesis makes the following key contributions:

- Novel conflict resolution strategies that significantly increase the speed of conflict resolution convergence (in particular for harder problems).
- Run-time model for performance measurement of strategies that takes into account message processing/communication overhead from extra communication of local information.
- Analysis of message processing/communication overhead which shows that the overhead from the novel strategy is not overwhelming. Thus, despite extra communication, the novel strategies can improve the speed of conflict resolution convergence.
- Categorization of problem settings where the novel conflict resolution strategies in this thesis provide high speedups.
- Distributed POMDP-based performance model for conflict resolution strategies
  - Addressing scalability issue with small-scale models
  - Investigation of different composition methods for the small-scale models

The research in this thesis has implications in many different settings. Coordination or conflict resolution strategies based on resource flexibility have been investigated in multi-linked negotiation [Zhang and Lesser, 2002], distributed multiagent planning [Decker and Lesser, 1995; Prasad and Lesser, 1997; von Martial, 1991], and centralized planning/scheduling [Nareyek, 2001; Sadeh and Fox, 1996; Smith, 1994]. The conflict resolution strategies with extra communication of local information can be applied to such applications for faster conflict resolution convergence. The new run-time model presented in this thesis could provide a useful method for performance measurement to the DCSP research community (as the first run-time model which takes into account message processing/communication overhead). Furthermore, our approach in modeling the performance of conflict resolution strategies points the way to new tools for strategy analysis and performance modeling in multiagent systems in general.

1.4 Organization of This Thesis

This thesis is organized as follows. Chapter 2 presents motivating examples and background. Chapter 3 defines novel conflict resolution strategies and presents how the strategies can be embedded in DCSP framework. Chapter 4 provides an existing performance measurement of DCSP and the run-time model for strategy performance measurement (that takes into account the overhead of extra communication of local information). Chapter 4 also presents experimental settings in detail and shows the results of systematic investigation. Chapter 5 provides details in performance modeling based on distributed POMDP and addresses the scale-up issue by introducing small-scale models (called “building-blocks”). Chapter 6 presents related work in the CSP/DCSP and the distributed POMDP literature. Finally, Chapter 7 concludes with future works.

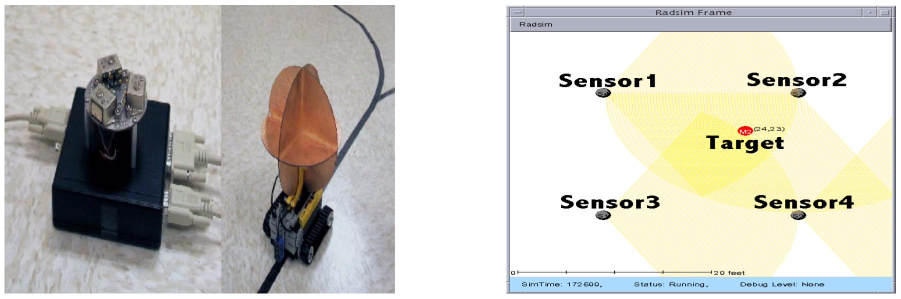
Chapter 2

Motivating Domains

This chapter outlines three key domains that motivate our work, and provides key illustrative examples. The first domain is a distributed sensor domain. This domain consists of multiple stationary sensors and targets moving through their sensing range (Figure 2.1.a and Figure 2.1.b illustrate the real hardware and simulator screen, respectively). Each sensor is equipped with a Doppler radar with three sectors. At a given time, at most one sector of a sensor can be activated. While the activation of sensors must be coordinated to track targets, there are some key difficulties in such tracking. First, in order to accurately track a target, at least three sensors must concurrently turn on overlapping sectors. This allows the target's position to be triangulated. Second, to minimize power consumption, sensors need to be periodically turned off. Third, sensor readings may be noisy and false detections may occur. Finally, the situation is dynamic as targets move through the sensing range [1].

Tracker agents control multiple sensors in a region to coordinate the sensing activities of multiple sensors, and collect sensing information to track targets in the region (Figure 2.2). When a sensor detects a target, it notifies the target detection to a tracker agent associated with the sensor. Then, the tracker agent assigns a set of three sensors to track the detected target. For instance, in Figure 2.2, when sensor 4 detects target 1 in its sector 0 (shaded region in Figure 2.2), it notifies tracker 1 of target 1. Then, tracker

[1]The simulation testbed was created for DARPA's ANTS program [Lesser *et al.*, 2003]



(a) sensor(left) and target(right) (b) simulator (top-down view)

Figure 2.1: A distributed sensor domain

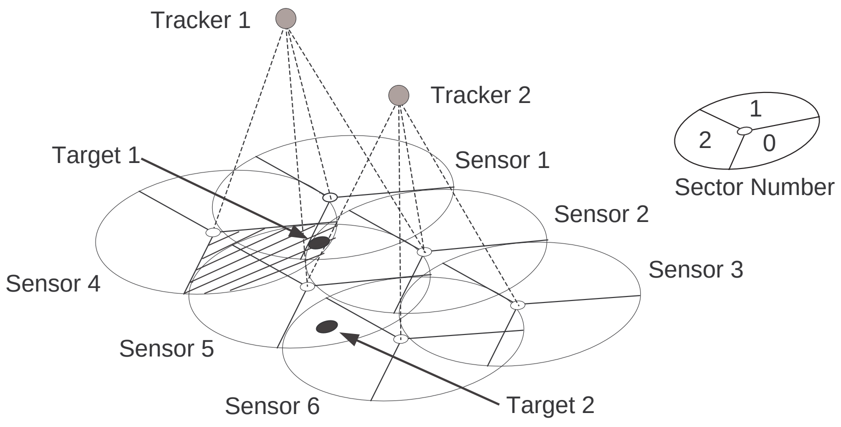


Figure 2.2: sensor sectors

1 assigns sensor 1, sensor 4, and sensor 5 to the task of tracking target 1: distance information from the three sensors will enable tracker 1 to triangulate the position of target 1. Tracker agent can physically reside in any sensor which it controls.

Given multiple targets, tracker agents may have contention over resources (e.g., shared sensors). For instance, a sensor may be required to track multiple targets at the same time by two different tracker agents. As illustrated in Figure 2.2, given target 1 and target 2, sensor 5 may be requested to turn on sector 0 and sector 1 by tracker 1 and tracker 2 respectively. Since only one sector can be activated at a given time, the resource choice of tracker 1 conflicts with that of tracker 2. Therefore, to track multiple targets, a team of tracker agents must resolve such conflicts in selecting their choice of resources. If there is a delay in conflict resolution, the system performance will significantly degrade since targets may not be tracked at all before moving out of sensing range. Therefore, it is critical for agents to make the right decision to resolve conflicts over their sensors.

In a collaborative setting, an agent can make an efficient decision if the local information of neighboring agents is available to the agent. For example, if tracker 1 in Figure 2.2 is notified of target 1 by sensor 4, it may assign sensor 1, sensor 4, and sensor 5 to track target 1: in this assignment, tracker 1 also specifies radar sectors to turn on (e.g., sensor 1's sector 0, sensor 4's sector 0, and sensor 5's sector 1). In particular, it may be the case that sensor 5 is already assigned to track target 2. If tracker 1 had the knowledge of its neighboring tracker 2's local situation (sensor 5's sector 0 is already activated to track another target), tracker 1 would not assign sensor 5 to track target 1. Instead, tracker 1 could assign another set of sensors, sensor 1, sensor 2, and sensor 4, if sensor 2 is available. Here, tracker 1's resource allocation that takes into account its neighboring agents' local information can avoid possible conflicts with the neighbors.

The second application domain is distributed spacecraft domain [2]. NASA is increasingly interested in multi-platform space missions [Angelopoulos and Panetta, 1998]. Pathfinder has its rover (Sojourner), Cassini has its Huygens lander, and Cluster II has 4 spacecraft for multi-point magnetosphere plasma measurements. This trend is expected to continue to progressively larger fleets. For example, one proposed mission involves 44 to 104 spacecraft flying in formation to measure global phenomena within the Earth's magnetosphere [Mettler and Milman, 1996]. In this mission, since the science goal is

[2]This project was supported by NASA JPL subcontract "Continual Coherent Team Planning"

to collect 3 dimensional structure of the phenomenon, any missing data from a single spacecraft can degrade the quality of the whole data. Therefore, the spacecraft agents must coordinate their sensing activities to collect data simultaneously. Here, a key difficulty is that the location of such phenomenon is not known in advance, which requires agents to quickly resolve conflicts given an opportunity to observe the phenomenon. Furthermore, each spacecraft agent has its own restrictions from limited battery power, a specified small time window for science data communication, etc. If each agent can make a choice of action (e.g., sensor activation) that allows more action choices to its neighboring agents, it will make easier for the neighbors to schedule their local activities or resolve future conflicts with others.

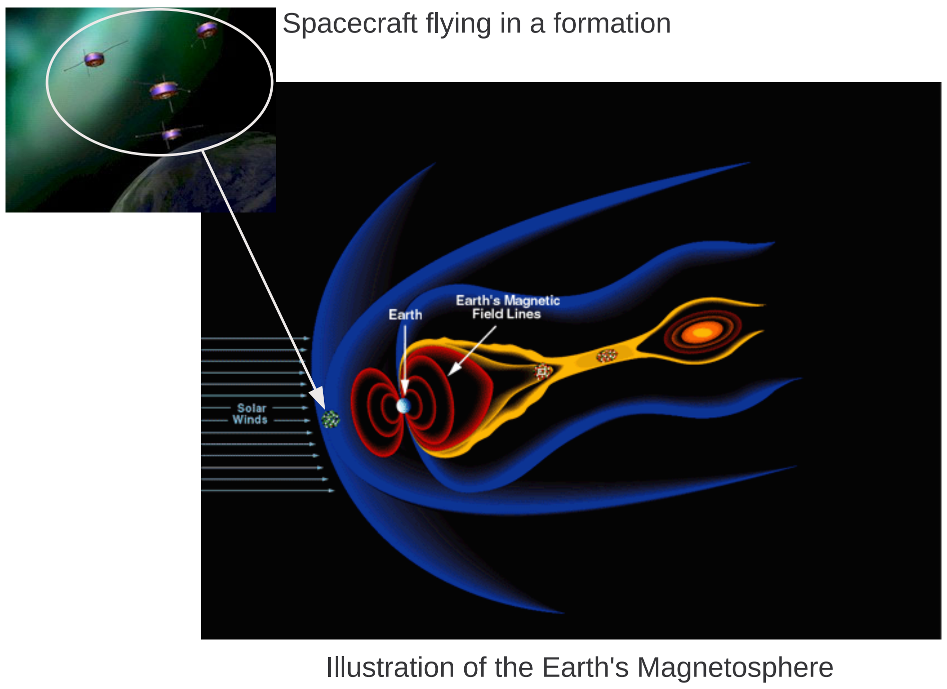

Spacecraft flying in a formation
Earth
Earth's Magnetic Field Lines
Solar Winds
Illustration of the Earth's Magnetosphere

Figure 2.3: Distributed spacecraft domain

The third application domain is helicopter pilot agents in battlefield simulation. Figure 2.4 is a snapshot of ModSAF (*Modular Semi-Automated Forces*) [Calder *et al.*, 1993]

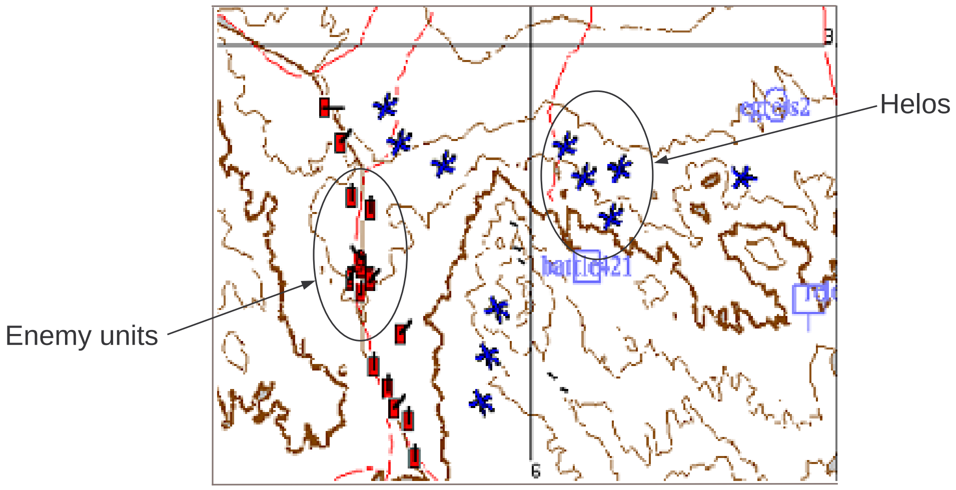


Figure 2.4: Helicopter combat simulation

simulator for pilot agents. One example of conflicts that arise in a team of simulated pilot agents involves allocating firing positions for the team. Individual pilots in a helicopter team attack the enemy from firing positions. Each firing position must enable a pilot to shoot at enemies while protecting the pilot from enemy fire. In addition, a pilot's firing position is constrained by the firing positions of the others. Two firing positions are in conflict if they are within one kilometer of each other. Therefore, the pilot agents must position themselves on a field while ensuring safe distance. Here, a pilot agent's selection of resource (firing position) that takes into account the other agents' local situation will also reduce possible future conflicts. For instance, if an agent knows that enemy exists near a neighboring agent, the agent would not ask its neighbor to move towards the enemy.

The above applications illustrate the importance of efficient conflict resolution techniques. They also show that an agent's choice of actions/plans/resources based on its neighbors' local situations may improve the performance in conflict resolution convergence. Here, it is useful to understand that, in this paper, we focus on distributed approach to resolve conflicts as opposed to a centralized approach, where a single agent gathers all information to provide a solution. Indeed, in many applications, such a centralized approach could prove problematic. First, this approach introduces a central point of failure, so that there is no fault tolerance. Second, centralization of all information could be a significant security risk, open to actual physical or cyber-attacks, particularly in hostile adversarial environments. Third, centralization requires all agents to accept a central authority, which may not always be feasible. Finally, a centralized agent could be a significant computational and communication bottleneck. Specifically, in domains such as distributed sensors, conflict resolution must continually occur among all agents for readjustment of the sensors. Centralization would require all sensors to continuously communicate their local information to the centralized agent which can be a significant bottleneck given scale-up to thousands of agents. Furthermore, given limited communication ability, agents may be unable to directly communicate with a central agent: introducing many hops for message transmissions may cause delays. In contrast, a distributed system provides fault tolerance, reduces the security risk, avoids a central authority and prevents a centralized communication/computational bottleneck.

Part I

Cooperative Conflict Resolution Strategies

Chapter 3

Fast Convergence Strategies

As shown in Chapter 2, fast conflict resolution is critical in many real world multiagent applications particularly under real-time constraints. In this section, we provide novel conflict resolution strategies to enable fast conflict resolution convergence. As mentioned earlier, the strategies are built on DCSP (Distributed Constraint Satisfaction Problem) framework which is a major paradigm for conflict resolution in collaborative multiagent systems and provides highly efficient base algorithms. Thus, our strategies are improving the performance of state of the art conflict resolution algorithms. We first introduce DCSP and formalize novel strategies based on the local cooperativeness towards neighboring agents. We also systematically investigate the performance of the local cooperativeness based strategies (defined below).

3.1 Formal Framework to Illustrate Conflict Resolution Strategies

DCSP techniques have been used for coordination and conflict resolution in many multiagent applications such as distributed sensor networks [Modi *et al.*, 2001]. DCSP provides an abstract formal framework to model multiagent conflict resolution: agents are modeled as variables and the restrictions on agents are mapped onto intra- or inter-agent

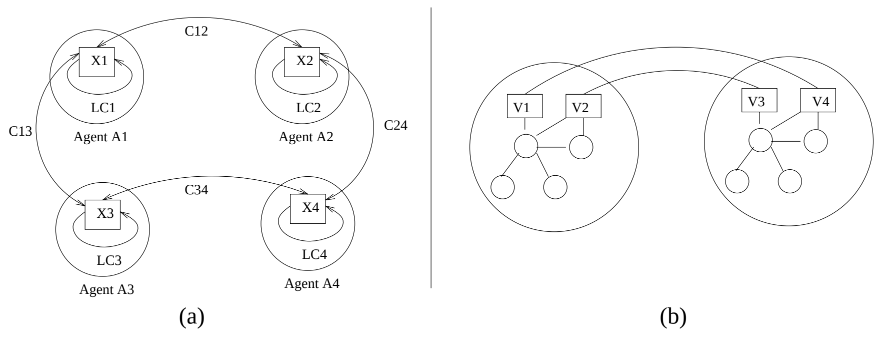


Figure 3.1: Model of agents in DCSP

constraints. Rich body of knowledge on constraint representation and efficient algorithms already exist and the nature of many DCSP algorithms fits into our assumption that no centralized control is required.

3.1.1 Distributed Constraint Satisfaction Problems (DCSP)

A Constraint Satisfaction Problem (CSP) is commonly defined by a set of n variables, X = $\{x_1$, ..., $x_n\}$, each element associated with value domains D_1, ..., D_n respectively, and a set of k constraints, Γ = $\{C_1$, ..., $C_k\}$. A solution in CSP is the value assignment for the variables which satisfies all the constraints in Γ. A DCSP is a CSP in which variables and constraints are distributed among multiple agents [Bessière *et al.*, 2001; Silaghi *et al.*, 2000; Yokoo *et al.*, 1998; Yokoo and Hirayama, 1998]. Formally, there is a set of m agents, Ag = $\{A_1$, ..., $A_m\}$. Each variable (x_i) belongs to an agent A_j. There are two types of constraints based on whether variables in a constraint belong to a single agent or not:

- For a constraint $C_r \in \Gamma$, if all the variables in C_r belong to a single agent $A_j \in Ag$, it is called a *local constraint*.
- For a constraint $C_r \in \Gamma$, if variables in C_r belong to different agents in Ag, it is called an *external constraint*.

Figure 3.1.a illustrates an example of a DCSP: each agent A_i (denoted by a big circle) has a local constraint LC_i and there is an external constraint C_{ij} between A_i and A_j. As illustrated in Figure 3.1.b, each agent can have multiple variables. Here, the squares labeled v1, v2, v3, and v4 are the variables whose values are externally known, and the small circles and the links between them are locally known variables and constraints. There is no limitation on the number of local/external constraints for each agent. Solving a DCSP requires that agents not only satisfy their local constraints, but also communicate with other agents to satisfy external constraints. Note that DCSP is not concerned with speeding up centralized CSP via parallelization; rather, it assumes the problem is originally distributed among agents.

3.1.2 Mapping Multiagent Conflict Resolution onto DCSP

Given the DCSP framework, we can model a mutliagent conflict resolution problem as a DCSP in terms of variables, values and constraints. Depending on the problem, the definition of variables, values, and constraints can vary. For instance, in distributed resource allocation, tasks can be modeled as variables whose values are actions or resources to complete the tasks. Constraints do not allow simultaneous selection of exclusive actions or resources. In contrast, in distributed planning, variables model operators and the values of the variables can be constants that represent time. Constraints between variables will be precedence relationship that enforces the ordering of operators. The solution of a multiagent conflict resolution is a DCSP instance where the selected values of variables satisfy all the constraints defined in the DCSP mapping.

Here, we discuss this mapping in the distributed sensor network domain described in Chapter 2 both as a basis for further discussion and as a concrete illustration. Each tracker agent is modeled as a variable whose values are radar sector combinations of sensors controlled by the tracker plus a special value NT (no target). For instance, if we let $sector^i_j$ be the sector j of sensor i, the domain of a variable representing tracker 1 in Figure 2.2 will be $\{< sector^1_0, sector^2_2, sector^4_0 >, < sector^1_0, sector^2_2, sector^5_1 >, < sector^1_0, sector^4_0, sector^5_1 >, < sector^2_2, sector^4_0, sector^5_1 >, NT\}$. There exist external binary constraints between variables and unary constraints for each variable as follows:

- External constraint: two tracker agents cannot share any sensor in their values.
- Local constraint: if a tracker agent is notified of target detection by any associated sensor, the tracker's value cannot be NT.

Note that this model is an initial mapping of the distributed sensor networks. We can easily extend this mapping to problems involving more resources (e.g., power, bandwidth, etc.) with multiple variables per agent. For instance, if we include power in the mapping, a tracker agent could be modeled with five variables: one variable (S) for sector combinations and four variables (B_i) for the batteries of the four sensors controlled by the tracker. B_i's value indicates the power level of a sensor. Between S and B_i, there exists an internal binary constraint as follows:

- If B_i's value is less than a threshold θ, S's value cannot include sensor i's sector.

Henceforth, for simplification, we will assume that each agent has exactly one variable and the constraints between variables are binary. In our initial investigations of conflict resolution strategy, we found it sufficient to model each agent as having only one variable and a single node constraint as illustrated in Figure 3.1.a. Henceforth, agents and variables in our description are used interchangeably since each agent has only one variable in the current mapping. Here, the local constraint LC_i often involves very complex computation, and there is no limitation on the number of external constraints C_{ij} for each agent.

3.1.3 Asynchronous Weak Commitment (AWC) DCSP Algorithm

In this thesis, we use Asynchronous Weak Commitment (AWC) search algorithm (one of the best published DCSP algorithms [Yokoo and Hirayama, 1998]) as a basis to build our novel strategies for conflict resolution. Thus, our strategies introduced below are improving the performance of state of the art conflict resolution algorithms. In the AWC approach, agents asynchronously assign values to their variables from available domain values, and communicate the values to neighboring agents with shared constraints. Each variable has a non-negative integer priority that changes dynamically during search. A variable is consistent if its value does not violate any constraints with higher priority variables. A solution is a value assignment in which every variable is consistent.

When the value of an agent's variable is not consistent with the values of its higher priority neighbor agents' variables, there can be two cases: (i) a *good* case where there exists a consistent value in the variable's domain; (ii) a *nogood* case that lacks a consistent value. In the good case with one or more value choices available, an agent selects a value that minimizes the number of conflicts with lower priority agents. On the other hand, in the nogood case, an agent increases its priority to *max+1*, where *max* is the highest priority of its neighboring agents, and selects a new value that minimizes the number of conflicts with all of its neighboring agents. This priority increase forces previously higher agents to select new values. Agents avoid infinite cycles of selecting non-solution values by saving the *nogood* situations.

3.2 Cooperative Strategies

While AWC is one of the most efficient DCSP algorithms, it is focused on minimal communication between agents. Each agent communicate only their value assignment when there is a value change in its variables. To enable fast conflict resolution convergence in large scale multiagent applications under real-time constraints, we need to improve the performance of existing DCSP techniques. In enhancing the performance of AWC algorithm, it is hypothesized that some extra local communication can significantly increase the speedup of conflict resolution convergence. This section introduces novel conflict resolution strategies that vary the degree of local cooperativeness towards neighboring agents (which is enabled by local communication of agents' internal constraints). We formally define the local cooperativeness based on the notion of flexibility and introduce the new strategies varying the degree of flexibility towards neighboring agents.

3.2.1 Local Cooperativeness

In AWC, agents relies on *min-conflict* strategy [Minton *et al.*, 1990] for conflict resolution. That is, when an agent's choice of actions or plans conflicts with other agents' choices, the agent changes its current choice into one that minimizes the number of conflicts with other agents' currently assigned choices. However, this *min-conflict* strategy does not take neighboring agents' local constraints into account. Instead, it only considers the conflicts between an agent's action/plan/resource choice and neighboring agents' choices. Even if an agent selects a choice that resolves current constraint violation, it can have another constraint violation since neighboring agents may have to change their values because of other constraints. By considering neighboring agents' local constraints, an agent can generate a more locally cooperative response, potentially leading to faster conflict resolution convergence. The concept of local cooperativeness goes beyond merely satisfying constraints of neighboring agents to accelerate convergence. That is, an agent A_i cooperates with a neighbor agent A_j by selecting a value for its variable that not only satisfies the constraint with A_j, but also maximizes A_j's flexibility (choice of values). Then A_j has more value choices that satisfy A_j's local constraints and other external constraints with its neighboring agents, which can lead to faster convergence.

Such cooperative value selection is enabled by legal value communication. By receiving neighboring agents' local constraint induced legal values, an agent can compute how much flexibility is given to its neighbors by a value in its domain. For instance, agent A_j might have a domain D_j. After applying its local constraints LC_j, A_j discovers its revised domain $D'_j \subset D_j$, and communicates D'_j to its neighboring agent A_i. [1] After receiving D'_j, by propagating the external constraint C_{ij} between A_i and A_j, A_i changes its domain D_i into D'_i. This constraint propagation (enabled by local constraint communication) eliminates incompatible values with A_j from A_i's domain, which may lead to improved conflict resolution convergence.

However, for faster conflict resolution, agent A_i can reason explicitly about its available values' impact on A_j and generate a more locally cooperative response to A_j, i.e., select a value from D'_i that gives more flexibility to A_j (more value choices in D'_j). To elaborate on this cooperative value selection, we define the notion of *local cooperativeness*. For this definition, let A_i be an agent with domain D_i, and a set of A_i's neighboring agents is denoted as N_i.

- **Definition 1**: For a value $v \in D_i$ and a set of agents $N_i^{sub} \subseteq N_i$, *flexibility function* is defined as $f^{\oplus}(v, N_i^{sub}) = \oplus(c(v, A_j))$ where

    1. $A_j \in N_i^{sub}$
    2. $c(v, A_j)$ is the number of values of A_j that are consistent with v

[1] Alternatively, if A_i is aware of A_j's domain D_j, A_j sends its local constraint LC_j to A_i. Then, A_i could infer D'_j by applying LC_j to D_j.

3. $\oplus$, referred to as a *flexibility base*, can be sum, min, max, $product$, $weightedsum$, etc.

- **Definition 2**: For a value v of X_i, *local cooperativeness* of v is defined as $f^{\oplus}(v, N_i)$. That is, the *local cooperativeness* of v measures how much flexibility (choice of values) is given to all of A_i's neighbors by v.

As an example of the flexibility function, suppose agent A_1 has two neighboring agents A_2 and A_3, where a value v leaves 70 consistent values to A_2 and 40 to A_3 while another value v' leaves 50 consistent values for A_2 and 49 to A_3. Now, assuming that $\oplus$ is set as sum and values are ranked based on flexibility, an agent will prefer v to v' since v is more locally cooperative than v': $f^{sum}(v, \{A_2, A_3\})$ = 110 and $f^{sum}(v', \{A_2, A_3\})$ = 99. However, if $\oplus$ is set to min, the preference will change since $f^{min}(v, \{A_2, A_3\})$ = 40 and $f^{min}(v', \{A_2, A_3\})$ = 49. Thus, the value choice based on *local cooperativeness* is influenced by *flexibility base*. In the next section, using the notion of *flexibility function* and *local cooperativeness*, we define new strategies which consider how much flexibility (choice of values) is given to other agents by a selected value. The effect of *flexibility base* on the performance of conflict resolution strategies (defined below) is investigated in the subsequent section.

3.2.2 Cooperativeness-Based Strategies

Based on the definitions of flexibility function and local cooperativeness, we define cooperative strategies by varying the degree of cooperativeness towards neighboring agents. Given multiple neighbors, agents provide different degrees of cooperativeness to different groups of the neighbors. For each agent A_i, its neighboring agents (denoted by N_i) are grouped into higher and lower agents based on their priorities defined in DCSP algorithms. While we illustrate the cooperative strategies based on dynamic priorities in AWC framework, the strategy definition in this section can be applied to other DCSP frameworks. To define the cooperative strategies, let N_i^{high} (N_i^{low}) be the subset of N_i (a set of all neighboring agents) such that, for every $A_j \in \mathrm{N}_i^{high}$ (N_i^{low}), the priority of A_j is higher (lower) than the priority of A_i.

- $S_{min-conflict}$: Each agent A_i selects a value based on min-conflict heuristic (the original strategy in the AWC algorithm);
- S_{high}: Each agent A_i attempts to give maximum flexibility towards its higher priority neighbors by selecting a value v that maximizes $f^{\oplus}(v, N_i^{high})$;
- S_{low}: Each agent A_i attempts to give maximum flexibility towards its lower priority neighbors by selecting a value v that maximizes $f^{\oplus}(v, N_i^{low})$;
- S_{all}: Each agent A_i selects a value v that maximizes $f^{\oplus}(v, N_i)$, i.e. max flexibility to all neighbors.

We now define cooperativeness relation among these strategies based on the cooperativeness of values they select.

- **Definition 3**: For two different strategies S_α and S_β defined on a flexibility base $\oplus$, S_α is *more cooperative* than S_β iff the following two conditions are satisfied:
  1. for all A_i, X_i, and v_α, $v_\beta \in D_i$ such that v_α and v_β are selected by S_α and S_β respectively, $f^{\oplus}(v_\alpha, N_i) \geq f^{\oplus}(v_\beta, N_i)$
  2. for some A_i, when $f^{\oplus}(v_\alpha, N_i) \neq f^{\oplus}(v_\beta, N_i)$, $f^{\oplus}(v_\alpha, N_i) > f^{\oplus}(v_\beta, N_i)$.
- **Theorem 1**: For any given flexibility base $\oplus$, the strategy S_{all} is *maximally locally cooperative* strategy in the *good* case, i.e., for any other strategy S_{other} on the same

flexibility base $\oplus$, S_{all} is more cooperative than S_{other}.

Proof: By contradiction. Assume that S_{other} is more cooperative given the flexibility base $\oplus$. For A_i, v_{all} is selected by S_{all} and v_{other} by S_{other} such that if $f^{\oplus}(v_{all}, N_i) \neq f^{\oplus}(v_{other}, N_i)$, then $f^{\oplus}(v_{all}, N_i) < f^{\oplus}(v_{other}, N_i)$. However, by the definition of S_{all}, v_{other} would be selected by S_{all} instead of v_{all}. A contradiction.

By theorem 1, S_{all} is more cooperative than the other strategies S_{high}, S_{low}, and $S_{min-conflict}$ for the *good* case. The theorem also applies to the *nogood* case: the computation in the *nogood* case is identical to the *good* case except that the domains values to consider are different (details appear in Figure 3.2). Therefore, S_{all} is also the most locally cooperative strategy in the *nogood* case. Note that both S$_{high}$ and S$_{low}$ have trade-offs. For instance, S$_{high}$ may leave very little or no choice to an agent's neighbors in N$_i^{low}$, making it impossible for them to select any value for their variables. S$_{low}$ has a reverse effect. S$_{min-conflict}$ also has trade-offs because it does not consider neighbors' flexibility.

Based on the ideas introduced above, we can create different strategy combinations for the *good* and *nogood* cases: there are sixteen strategy combinations for each flexibility base because the four strategies defined above can be applied to both the *good* and *nogood* cases. Since, we will only consider strategy combinations, henceforth, we will refer to them as strategies for short. Note that all the strategies are enhanced with local constraint communication and propagation as described in Section 3.2.1. Here, three exemplar strategies are listed:

- $S_{min-conflict} - S_{min-conflict}$: This is the original AWC strategy. Min-conflict heuristic is used for the good and nogood cases.

- $S_{low} - S_{high}$: For the *good* case, an agent is most locally cooperative towards its lower priority neighbor agents by using S_{low}. (Note that the selected value doesn't violate the constraints with higher neighbors). On the contrary, for the nogood situations, an agent attempts to be most locally cooperative towards its higher priority neighbors by using S_{high}.

- $S_{all} - S_{all}$: In both the *good* and the *nogood* cases, an agent uses S$_{all}$ which is to select a value that maximizes flexibility of all neighboring agents.

Figure 3.2 describes how a cooperative strategy can be incorporated in AWC [Yokoo and Hirayama, 1998] framework. In Figure 3.2, *check_agent_view* is a procedure performed by an agent A_i receiving its neighbors' current value assignments and legal value sets. *Check_agent_view* procedure checks the consistency of A_i's current value assignment (v_i) with other agents' values (agent_view) and selects a new value if it violates any constraint with higher agents. For Figure 3.2, let's assume that A_i selects $S_\alpha - S_\beta$ strategy such that α, $\beta \in$ {high, low, all, min-conflict}. In the new_cooperative_value procedure (Figure 3.2), S_{high}, S_{low}, and S_{all} use min-conflict heuristic to break ties among the values with the same max flexibility.

Among the cooperative strategies described above, $S_{all} - S_{all}$ is the most cooperative strategy because it is maximally locally cooperative to neighboring agents in both *good* and *nogood* cases. Figure 3.3 shows a partial order over the cooperativeness of 16 different strategies. A higher strategy is more locally cooperative than a lower one. In general, the strategies at the same level are not comparable to each other such as $S_{low} - S_{high}$ and $S_{high} - S_{low}$. However, strategies such as $S_{min-conflict} - S_{min-conflict}$ were not originally defined with the notion of cooperativeness as defined in this section; and could thus be considered less cooperative than a strategy such as $S_{low} - S_{high}$ that attempts to be explicitly cooperative to neighboring agents.

Given: for each agent $A_j \in N_i$ (a set of A_i's neighbors),

- Current value assignment (v_j)
- Legal value set (D'_j) under A_j's local constraint LC_j

```
Procedure check_agent_view

 1. Revise $A_i$'s domain by constraint propagation: $D_i$ changes into $D'_i$;
 2. If there is any constraint ($C_{ij}$) violation with higher priority neighbors or $A_i$'s
    current value assignment $v_i$ is not included in $D'_i$;
    (a) Find a value set $D_{co} \subseteq D'_i$ whose values satisfy constraints with higher neigh-
        bors;
    (b) If $D_{co} \neq \emptyset$    // good case
          • new_cooperative_value($\alpha$, $D_{co}$);
    (c) Else    // nogood case (no consistent value in $D'_i$)
          • Record and communicate nogood;
          • $X_i$'s priority = max of neighbors' priorities + 1;
          • new_cooperative_value($\beta$, $D'_i$)
    (d) If there exists a change for $A_i$, communicate it to neighbor agents;

Procedure new_cooperative_value (Input: strategy $\sigma$, domain $\Delta \subseteq D'_i$)

 • If $\sigma \equiv min\text{-}conflict$,
     1. select $v_{new} \in \Delta$ where $v_{new}$ minimizes the number of constraint violation
        with lower priority agents;
 • Else ($\sigma \in$ {high, low, all})
     1. $\Omega$ = $N_i^\sigma$ (Here, $N_i^{all} \equiv N_i$);
     2. For each value $v \in \Delta$, $v$'s flexibility = $f^{\oplus}(v, \Omega)$;
     3. Find $v \in \Delta$ which has max flexibility;
          – Apply min-conflict heuristic to break ties;
     4. Set the selected $v$ to $v_{new}$;
 • Change $A_i$'s current value assignment into $v_{new}$;
```

Figure 3.2: Cooperative strategy $S_\alpha - S_\beta$ performed by agent A_i in AWC framework

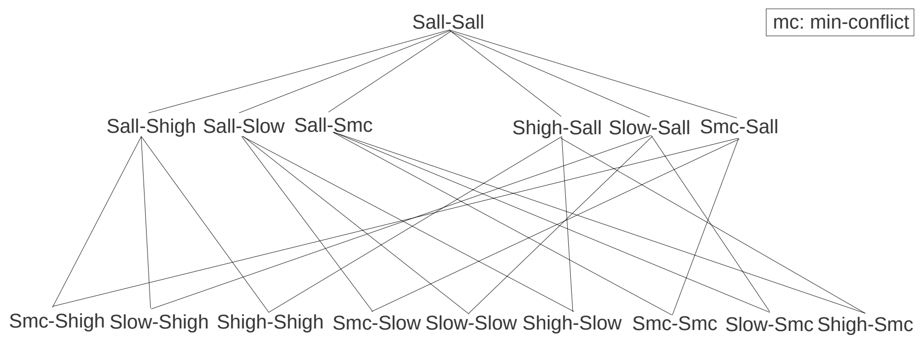


Figure 3.3: Cooperativeness relationship: the higher, the more locally cooperative

Chapter 4

Strategy Performance Measurements

This chapter provides detailed information about the metrics to evaluate the performance of DCSP strategies. Section 4.1 describes the existing method used in the DCSP literature. Section 4.2 presents a new analytical method which overcomes the shortcomings of the existing method (introduced in Section 4.1). Section 4.3 provides the analytical results for the performance of the locally cooperative strategies (defined in Section 3.2.2).

4.1 Existing Method

Since it has been practically difficult to access a real large-scale distributed system (with hundreds of nodes in DCSP), the standard methodology in the field [Bessière *et al.*, 2001; Fernàndez *et al.*, 2003; Modi, 2003; Silaghi *et al.*, ; Yokoo and Hirayama, 1998; Zhang *et al.*, 2003] is to implement a synchronized distributed system which is a model of distributed system where every agent synchronously performs the following steps (which is called a *cycle*):

1. Agents receives all the messages sent to them in the previous cycle.

2. Agents resolve conflicts, if any, and determine which message to send.

3. Agents send messages to neighboring agents.

In the absence of a true distributed implementation, the benefit of the synchronized distributed setting is the ease of simulation on a single machine: while an agent resolves conflicts, other agents wait and all the messages are communicated at the same time after all the agents perform their own local computation to resolve conflicts, which provides an effect of a synchrony. Given such a synchronized distributed system, it is difficult to directly measure the run-time for real distributed conflict resolution. However, in the literature, as a compromise, researchers have used hardware independent metrics such as *cycles* and *constraint checks* defined below. Note that hardware independent evaluation is important since it enables researchers to compare their approaches without re-running others' approaches in their systems. The cycles and constraint checks are defined as follows:

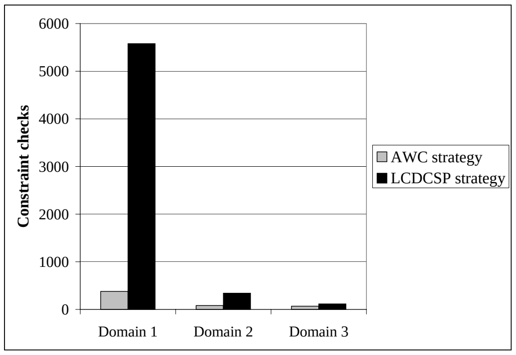
- *Cycles*: The number of cycles until a solution is found. As the number of cycles increases, agents' communication is increased since agents communicate with others at each cycle. Note that total time for conflict resolution is expected to be proportional to *cycles* [Yokoo and Hirayama, 1998]: the more cycles, the more time to find a solution. If computation time is extremely fast, then the total time will be dominated by *cycles*.

- *Constraint checks*: The total number of the maximum number of constraint checks at each cycle until a solution is found. More specifically, at each cycle, we identify a bottleneck agent which performs the most constraint checks, and sum up all those maximum numbers of constraint checks over all cycles. Note that the bottleneck agent may vary at each cycle. In a synchronized distributed system, while the local computation of constraint checks can vary among agents, the whole amount of computation at each cycle is dominated by the bottleneck agents. Thus, we use this measurement as a main indicator for the time consumed for computation until a solution is found. If communication delay is insignificant, the total time for conflict resolution will be dominated by *constraint checks*.

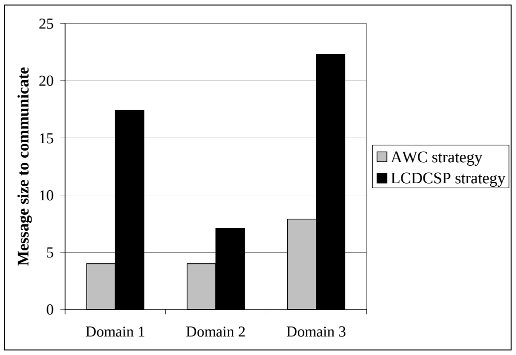
In the DCSP research community, *cycles* is used as a major metric for performance evaluation since the amount of local computation and communication that each agent solves mostly remains same in the most of previous DCSP approaches [Bessière *et al.*, 2001; Fernàndez *et al.*, 2003; Modi, 2003; Silaghi *et al.*, ; Yokoo and Hirayama, 1998; Zhang *et al.*, 2003]: the difference is in the protocol for passing values and controlling backtracking. Furthermore, it is often assumed that the local computation time is insignificant. However, there are two shortcomings for *cycles* as follows:

- Local computation overhead: For the hardware with limited computing power, the time for local computation may not be ignored, and there can be a variation in local computation which depends on the number of constraint checks (as shown in Figure 4.1-(a)).
- Message communication overhead: While *cycles* assumes that uniform time is taken at each communication phase, the time for message communication often depends on the size/number of messages to communicate. Figure 4.1-(b) shows that there exists a difference between AWC strategy and LCDCSP strategies in message size which may affect the communication time.

Despite these shortcomings, it is still important to view results in terms of cycles for two reasons. One is a practical reason. This is the technique of measurement used in the DCSP community, and and it is important to present results using cycles to make the results accessible and acceptable by that community. Second, as shown in Section 4.3.3, under certain conditions, *cycles* could be a reasonable approximations for measurement.



(a) Constraint checks



(b) Message size

Figure 4.1: Difference between AWC strategy and LCDCSP strategy in constraint checks and message size per cycle for different domains (based on empirical results in Section 4.3)

[1] The next section presents an analytical model for run-time measurement which takes into account various message processing/communication overhead in different computing/networking environments.

Recently, the issue of run-time measurement (including the validity of *cycles*) was discussed as a key topic in the panel discussion at the IJCAI 2003 workshop on Distributed Constraint Reasoning. There was an intense debate over how to measure run-time performance. Many DCSP researchers who participated in the panel discussion were not able to provide a convincing measurement method (that can be universally accepted by the DCSP research community). How to devise accurate, yet feasible and practical, evaluation metrics that generalize across domains remains as an open question which may require extensive efforts and novel techniques.

[1]Furthermore, for a certain distributed system where communication is based on a synchronized clock, *cycles* provides a reasonable performance measurement. For instance, agents in some sensornet systems communicate only at specific times with a fixed interval to save energy for communication [Elson, 2003]. If the interval is t seconds, the total run-time will be $cycles \times t$ seconds.

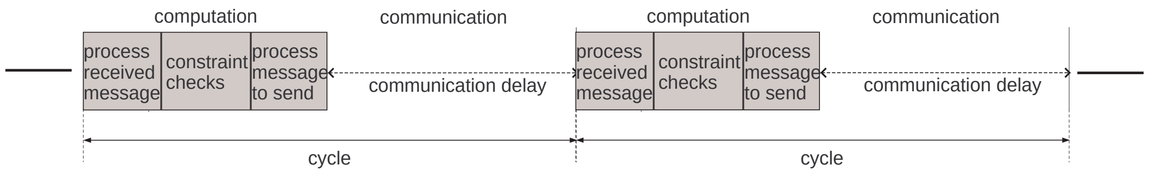


Figure 4.2: Model of runtime

4.2 Analytical Model for Run-time

As shown in Figure 4.2, the local computation processed by an agent at each cycle consists of processing received messages, performing constraint checks, and determining which message to send for its neighbors. The run-time taken by an agent for a cycle is the sum of the local computation time and the communication time for the agent's outgoing messages. In this thesis, I present an analytical model that enable the estimation of run-time from the data collected from the experimentation on a synchronized distributed system. The following terms are defined for the model:

- n_i^k: number of incoming messages for an agent i at cycle k
- $s_i^k(j)$: size of j^{th} incoming message for an agent i at cycle k
- $\mathcal{I}$(l): computation time to process one incoming message (whose size is l)
- c_i^k: number of constraint checks by an agent i at cycle k
- t: computation time to perform a single constraint check
- o_i^k: number of outgoing message for an agent i at cycle k
- u_i^k: size of an outgoing message for an agent i at cycle k
- $\mathcal{O}(m)$: computation time to process an outgoing message (whose size is m)
- $\mathcal{T}(d)$: communication time to transmit an outgoing message (whose size is d)

In a synchronous distributed system, at each cycle, agents synchronously start their local computation and message communication at the same time. Thus, the run-time for a cycle is dominated by an agent which requires maximum time for its local computation and message communication.

- *Run-time for a cycle* k ($\mathcal{R}(k)$) = $max_{i \in Ag}$(*local computation time of an agent i at cycle k* + *communication time of an agent i at cycle k*) where Ag is a set of agents.

The local computation time and the communication time of an agent i at cycle k are given by the following equations:

- *Local computation time of an agent i at cycle k* ($\mathcal{L}_i^k$) = $\sum_{j=1}^{n_i^k}(\mathcal{I}(s_i^k(j)))$ (time for processing received messages) + $c_i^k \times t$ (time for performing constraint checks) + $o_i^k \times \mathcal{O}(u_i^k)$ (time for processing outgoing messages) [2]

- *Communication time of an agent i at cycle k* ($\mathcal{C}_i^k$) = $o_i^k \times \mathcal{T}(u_i^k)$ (time for transmitting a message whose size is u_i^k for o_i^k times)

Finally, the total run-time is given by summing up the run-time ($\mathcal{R}(k)$) for each cycle:

- *Total run-time* = $\sum_{k=1}^{K}(\mathcal{R}(k))$ where K is the number of cycles until a solution is found.

While the above performance model aims to provide a metric which takes into account message processing/communication overhead (based on message size/number),

[2] If an agent uses a broadcasting method to transmit its message to multiple neighbors, $o_i^k = 1$. Otherwise (an agent sends the same message separately to each neighbor), $o_i^k =$ *the number of neighbors of an agent i*. In this thesis, broadcasting is not assumed. Instead, agents sends the same message multiple times for each neighbor as was done in DARPA ANTS testbed hardware (a specific message receiving channel is assigned for each sensor) [Lesser *et al.*, 2003].

it is flexible enough to subsume the existing method of performance measurement (Section 4.1) as follows:

- *Constraint checks* corresponds to the total run-time (defined above) where $t = 1$, $\mathcal{I}(\cdot) = \mathcal{O}(\cdot) = 0$ (message processing cost is zero), and $\mathcal{C}_i^k = 0$ (communication cost is zero).

- *Cycles* corresponds to the total run-time under the assumption that $t = 0$ (the cost for constraint checks is zero), $\mathcal{I}(\cdot) = \mathcal{O}(\cdot) = 0$ (message processing cost is zero), and $\mathcal{C}_i^k = 1$ (a constant communication time which is independent of message size and number).

While there was an attempt to provide an analytical model for DCSP run-time measurement [Modi, 2003], the model takes into account only message number. Therefore, as the first analytical model for the performance measurement in DCSP which takes into account the overhead based on message size and number, the above model could provide a useful method for performance measurement to the DCSP community. Furthermore, as shown below, the model shows interesting results as the parameters for message processing/communication overhead vary.

4.3 Analysis of Locally Cooperative Strategy Performance

Section 4.3.2 provides an overview of the experimental setting and results. Section 4.3.3 presents the impact of the analytical model on the performance improvement by the locally cooperative strategies. Section 4.3.4 shows further analysis for how the performance of the strategies changes in different computing/networking environments. Section 4.3.5 addresses the scalability issue of the locally cooperative strategies. Finally, Section 4.3.6 shows the dominance relation among the strategies, motivating the part 2 of this thesis.

Note that, for the performance analysis, the locally cooperative strategies are embedded as value ordering heuristics in the AWC algorithm. Henceforth, for brevity, the locally cooperative strategy embedded in AWC is referred as *LCDCSP strategy* (Locally Cooperative DCSP strategies). The min-conflict strategy without extra communication of local information (originally used in the AWC algorithm [Yokoo and Hirayama, 1998]) is referred as *AWC strategy*.

4.3.1 Experimental Settings

While we focus on the domain where agents' interaction topology is regular, there can be variations (e.g., problem hardness) in different problem settings that arise within the domain. In this section, we provide various problem settings controlled by several parameters (described below). Systematic changes in the parameters generate a wide variety of problem settings, and enable us to evaluate the performance of the strategies (defined in Chapter 3 of this thesis) and find their communication vs. computation tradeoffs in different situations. Here, parameter selection is motivated by the experimental investigation in the CSP and DCSP literature [Cheeseman *et al.*, 1991; Hirayama *et al.*, 2000; Hogg and Williams, 1994] where problem characteristics (e.g, the hardness of problems) depend on the following:

1. Graph density: Ratio of constraints out of all possible constraints. If there are N variables, it specifies how many constraints are given to a problem out of $N(N-1)/2$ constraints.

2. Constraint tightness: Ratio of allowed value pairs for each constraint. If each variable has K values, it specifies the number of allowed value pairs out of K^2 value pairs.

3. Domain size: The number of values for each variable.

We also add additional parameters based on the properties of the domains of interest. For instance, we make a variation in the percentage of local constraints (explained below) which corresponds to how many targets exist in sensor networks domain.

First, regarding the graph density, since we focus on regular graphs where each agent's relation with its neighboring agents is uniform, we vary the graph density by changing the number of neighboring agents as follows (the description applies to every agent except for boundary agents):

1. Hexagonal topology: Each agent is surrounded by three agents (separated by 120 degrees).

2. Grid topology: Each agent (located in a vertex of square lattice) is surrounded by four neighboring agents (separated by 90 degrees).

3. Triangular topology: Each agent is surrounded by eight neighboring agents (separated by 45 degrees).

The purpose of trying three different regular graphs is to investigate the impact on performance by the degree of connectivity (number of interactions for each agent):

connectivity matters since the computation overhead from extra communication can increase or decrease depending on the number of neighboring agents. Furthermore, the above topologies are applied to real applications such as sensor networks [Lesser *et al.*, 2003] and micro-air-vehicles for surveillance [Gordon *et al.*, 1999].

Second, given a topology (among the three topologies above), we make variations in constraint tightness. Even for the same topology, the constraint tightness has a great impact on the hardness of problems. For instance, if most of value pairs are allowed, agents can easily find a solution since each value has a high possibility of being compatible with others. Note that, for conflict resolution in multiagent systems, there are two types of constraints: external constraints and local constraints as described in Section 3.1.1. Therefore, to analyze the effect from each constraint, we distinguish external constraints from local constraints in defining the constraint tightness. That is, separate constraint tightness is specified as follows:

1. External constraint tightness: Given an external constraint, for a value in an agent's domain, the percentage of compatibility with neighboring agents' values is defined. The percentage varies from 30% to 90% with intervals of 30% (30%, 60%, 90%).

   - Note that 0% case and 100% case are not tried

     - For 0% case, there is no solution: strategies with extra communication can easily find that there is no solution by constraint propagation.
     - For 100% case, problems are trivial (every value assignment is a solution).

2. Local constraint: Given a local constraint, a portion of agents' original domains is not allowed. We make the following two variations in terms of local constraint.

- The percentage of locally constrained agents changes from 0% to 100% (0%, 30%, 60%, 90%, 100%).
- Given a local constraint, the portion of allowed values varies from 25% to 75% with intervals of 25% (25%, 50%, 75%). Note that 0% and 100% are not tried since 0% case gives agents empty domain and 100% case has no effect of having a local constraint.

Third, we make a variation in domain size to investigate how strategies' performance and trade-offs are affected by the domain size. It has been shown that, given the same graph density and constraint tightness, the hardness of problems may vary depending on the number of domain values [Cheeseman *et al.*, 1991].

- Domain size: The number of domain values varies from 10 to 80 (10, 40, 80). While the number may vary depending on domains, the main purpose of this variation is to check how different domain sizes have an impact on the performance and trade-offs of the strategies.

Given the above variations, the total number of settings is 351. [3] For a given problem setting, the performance of strategies is measured on 35 problem instances which are randomly generated by the problem setting (defined with the above parameters): 35

[3]When the ratio of locally constrained agents is 0%, the local constraint compatibility does not matter since no agent has a local constraint. Except for the case with 0% ratio of locally constrained agents, there exist 324 problem settings = 3 topologies $\times$ 3 external constraint compatibilities $\times$ 4 percentages of locally constrained agents (30% $\sim$ 90%) $\times$ 3 local constraint compatibilities $\times$ 3 domain sizes. For the 0% ratio case, there exist 27 problem settings = 3 topologies $\times$ 3 external constraint compatibilities $\times$ 1 percentages of locally constrained agents (30% $\sim$ 90%) $\times$ 3 domain sizes. Therefore, the total number of problem settings is 351 (= 324 + 27).

problem instances are tried for statistical significance. [4] For each problem setting, seventeen strategies (sixteen strategies defined in my thesis plus the original AWC strategy [5]) are tried for each problem instances. Thus, the total number of experimental runs is 208,845 ($= 351 \times 35 \times 17$). [6] Here, the number of agents is 512. Finally, to check the scalability of our conflict resolution strategies, we also vary the number of agents for selected problem settings (e.g., a setting where the min-conflict strategy or a strategy based on extra communication shows the best performance) as follows:

- Number of agents: Given a topology, the number of agents varies from 64 to 1024 (64, 128, 256, 512, 1024). In our domains of interest such as sensor networks and distributed spacecraft, the number of agents in hundreds is considered to be large-scale. For instance, in distributed spacecraft domain, a future large-scale mission assumes 44 $\sim$ 104 spacecraft to investigate the Earth's magnetosphere [Angelopoulos and Panetta, 1998].

4.3.2 Overview of Experimental Results

This section provides an overview of the experimentation proposed in Section 4.3.1, and presents in which problem settings LCDCSP strategies show big speedups.

[4]In general, the sample size of 35 provides statistically significant data with p-value < 0.1 in comparing the results from two different strategies. (from e-Handbook of Statistical Methods published by National Institute of Standards and Technology available at http://www.itl.nist.gov/div898/handbook/). Given a parameter such as the percentage of compatible value pairs (defined by external constraint tightness above), different problem instances can be generated since different set of value pairs can be allowed under the same percentage.

[5]For the sixteen locally cooperative strategies, sum is used as a flexibility base. Note that the original AWC strategy is *min-conflict* strategy without extra communication of local information.

[6]Total run-time of the 208,845 experiments takes 21 days on a Linux machine with 4 Pentium-4 2GHz processors and 4 GBytes memory. Note that, to conduct the experiments within a reasonable amount of time, the number of cycles was limited to 1000 for each run (a run was terminated if this limit exceeded).

cycles(N)	Number of problem settings	Number(percentage) of problem settings for different speedup (K fold)			
		$K < 2$	$2 \leq K < 5$	$5 \leq K < 8$	$K > 8$
$N < 200$	298	221 (74.2%)	72 (24.2%)	1 (0.3%)	4 (1.3%)
$200 \leq N < 500$	17	3 (17.6%)	5 (29.4%)	3 (17.6%)	6 (35.3%)
$N \geq 500$	36	29 (80.5%)	6 (16.7%)	0 (0%)	1 (2.8%)
Total	351	253 (72.1%)	83 (23.6%)	4 (1.1%)	11 (3.1%)

Table 4.1: Overview of problem hardness and speedup by LCDCSP strategies

Overview of Speedup by LCDCSP Strategies

In this section, I first present an initial set of results for the speedup by LCDCSP strategies (Table 4.1). Then, in the latter half of section, I provide further analysis that shows LCDCSP strategies significantly improve performance particularly in harder problems rather than easier ones.

Table 4.1 provides an overview for the variation in problem hardness among the problem settings (defined in Section 4.3.1) and an initial perspective for the speedup achieved by LCDCSP strategies against the AWC strategy. For Table 4.1, given a setting, average *cycles* (until a solution is found) is computed over the 35 problem instances which are randomly generated in the setting. Henceforth, *cycles* for a problem setting indicates the average *cycles* of 35 problem instances in the setting.

The second column in Table 4.1 shows the number of problem settings in different range of *cycles*. The results in the second column shows that there is a significant variation in the problem hardness of the problem settings. [7] 298 problem settings out of the total 351 problem settings (defined in Section 4.3.1) requires less than 200 *cycles*. 17 problem settings (out of 351 settings) takes between 200 and 500 *cycles* and, for

[7]Appendix B.1 presents the detailed information on *cycles* for all the 351 problem settings, and Table B.1 in Appendix B.2 shows detailed distribution of problem hardness.

the remaining 36 problem settings, it takes more than 500 *cycles* on average to find a solution.

Table 4.1 also provides the number (percentage) of problem settings with different speedups achieved by the best LCDCSP strategy in a given setting. Table 4.1 shows that LCDCSP strategies show at least more than two fold speedup in 28% problem settings overall. Furthermore, for a certain group of problem settings (where *cycles* ranges between 200 and 500), more than eight fold speedup is achieved by LDCSP strategies in a significant range of problem settings.

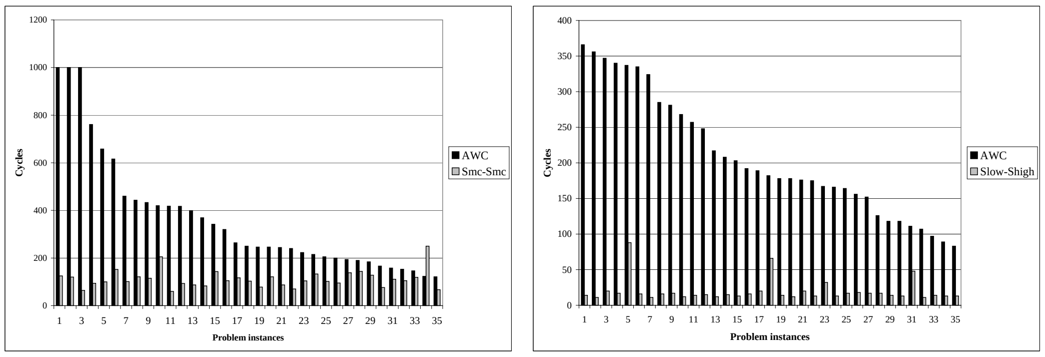
Here, a larger speedup could be shown in the problem settings where *cycles* is more than 500 ($N \geq 500$) and LCDCSP strategies show more than two fold speedup since the cycle limit is set as 1000: AWC strategy takes more than 1000 *cycles* for majority of problem instances in a given setting (while the best LCDCSP strategy in the setting does not exceed the cycle limit). Note that we cannot provide definitive analysis for 37 problem settings in the grey cell of Table 4.1 (where *cycles* is more than 500 ($N \geq 500$) and LCDCSP strategies show less than two fold speedup) since both AWC strategy and LCDCSP strategies do not terminate in 1000 *cycles*. [8]

While Table 4.1 presents an overview of the speedup by LCDCSP strategies, it is useful to investigate the results in Table 4.1 in more detail to understand how speedups change on individual problem instances. Figure 4.3 shows prototypical performance results for individual problem instances. In Figure 4.3, the horizontal axis is the number of problem instances, and the vertical axis plots the number of cycles taken until a solution is found for each problem instance. (Problem instances are ordered by their problem hardness which is decided by the performance of the AWC strategy on the

[8]Table B.2 in Appendix B.2 lists the 29 problem settings.

instances.) Black and grey columns indicate the number of cycles taken by the AWC strategy and the best LCDCSP strategy respectively.

Figure 4.3-(a) & (b) show that the speedup by LCDCSP strategies does not come from only easier problem instances. Surprisingly, the speedup could be higher on harder problem instances than on all the instances in a given problem setting: the speedup for easier instances is lower than the speedup for harder instances.

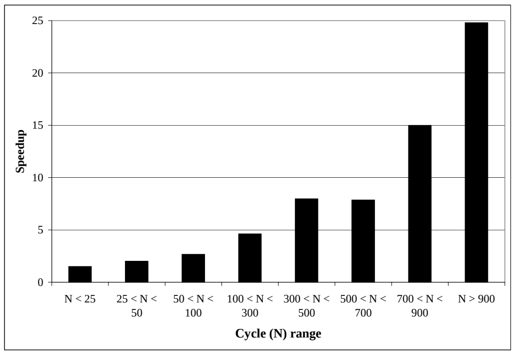


(a) Problem setting with average three fold speedup

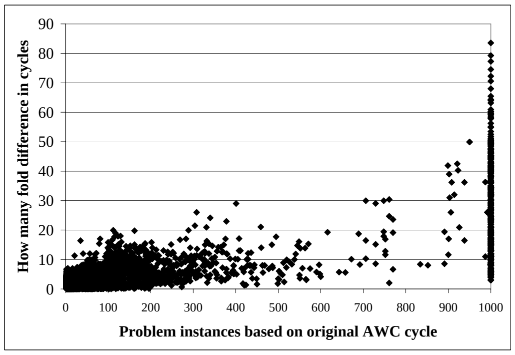
(b) Problem setting with average ten fold speedup

Figure 4.3: Example: Speedup for individual problem instances (S_{mc} means $S_{min-conflict}$ strategy)

Figure 4.4-(a) & (b) provide further analysis for the speedup by LCDCSP strategies in the problem settings where more than two fold speedup is achieved by LCDCSP strategies. For Figure 4.4, problem instances are grouped together (regardless of problem settings) by the performance (*cycles*) of the baseline strategy (the AWC strategy). Figure 4.4-(a) shows how much speedup is achieved by LCDCSP strategies for a set of problem instances with different problem hardness. As shown in Figure 4.4-(a), a big speedup by LCDCSP strategies is achieved when a given problem instance is difficult to solve (taking a large number of *cycles* by the AWC strategy): as the problem instances get more difficult, the speedup increases.



(a) Speedup for problem instances grouped by problem hardness (based on AWC performance)



(b) Speedup for individual problem instances (based on AWC performance)

Figure 4.4: Speedup for problem instances with different problem hardness: Checking whether speedup comes from easier or harder problem instances

To check whether such a speedup comes from only a few exceptional cases (while LCDCSP strategies do not provide performance improvement overall), Figure 4.4-(b) shows the scattered graph of speedups for individual problem instances. The results in Figure 4.4-(b) indicates that LCDCSP strategies show performance improvement for majority of problem instances across different problem hardness: while there is a variation in performance improvement, the speedups do not come from only a few exceptional cases. The results in Table 4.1 and Figure 4.4 show that LCDCSP strategies show performance improvement (with more than two fold speedup) in a significant range of problem settings (28% problem settings overall), and, within each problem setting, LCDCSP strategies provide more than an order of magnitude speedup in particular for harder problem instances.

In Which Problem Setting LCDCSP Strategies Show High Performance Improvement?

While it is difficult to crisply define problem settings where different levels of speedups (based on *average cycles*) are achieved, in this section, I will attempt to provide some categorization of the problem settings where high speedups are achieved by LCDCSP strategies over the AWC strategy in terms of *cycles*. Note that this categorization is not exhaustive, and focuses on problem settings (not on individual problem instances). The result in Figure 4.4 already shows that, when individual problem instances appear to become difficult, LCDCSP strategies outperform the AWC strategies.

To get a broad view of performance improvement by LCDCSP strategies, Figure 4.5 shows the maximum speedup by LCDCSP strategies for a set of problem settings grouped by external constraint compatibility and topology. [9] Each data point (a 3D column) in Figure 4.5 includes multiple settings with different local constraint compatibility, the ratio of locally constrained agents, and domain size under the same external constraint compatibility and topology. The results in Figure 4.5 show the followings:

- When external constraint compatibility is low (e.g., 30%), more than an order of magnitude speedup can be achieved at each topology. The speedup is greater as the graph density decreases (from triangular topology to hexagonal topology).
- With the external constraint compatibility of 60%, a big speedup is shown only in the grid topology. When the graph density is either low (hexagonal topology) and high (triangular topology), LCDCSP strategies do not show such a big speedup.

[9]The maximum speedup for a set of problem settings is the greatest speedup among the speedups in the problem settings. For each problem setting, speedup is based on the average cycles on the 35 problem instances of the setting by the AWC strategy and the best LCDCSP strategy.

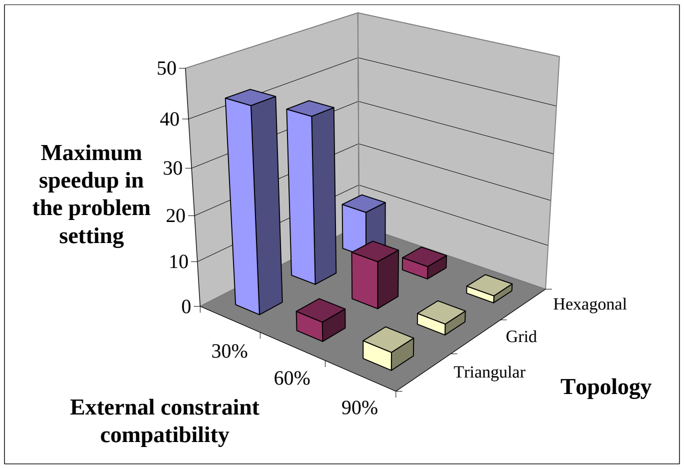


Figure 4.5: Maximum speedup in the problem settings classified by external constraint compatibility and topology

- When external constraint compatibility is 90%, the speedup is relatively small since the problem settings with 90% external constraint compatibility is easier than other settings (taking less than 30 *cycles* in general) so that there is no big difference in *cycles* between the AWC strategy and LCDCSP strategies.

Local constraint compatibility	Domain size	Ratio of locally constrained agents	Speedup at Each Topology		
			Hexagonal	Grid	Triangular
25%	10	30%, 60%, 90%	11	37	44
		others	2	5	3
25%	40	60%, 90%	2	7	1
		others	2	2	1
25%	80	60%, 90%	2	14	4
		others	2	5	3
50%	10, 40, 80	0 ∼ 100%	2	4	3
75%	10, 40, 80	0 ∼ 100%	3	5	4

Table 4.2: Maximum speedup in the problem settings where external constraint compatibility is 30%

Table 4.2 and 4.3 provide further analysis for the problem settings with a big speedup when external constraint compatibility is 30% for each topology and 60% in the grid

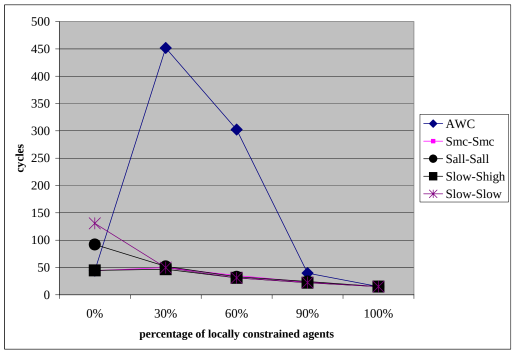
Local constraint compatibility	Domain size	Ratio of locally constrained agents	Speedup
25%	10	$0 \sim 100\%$	2
25%	40	90%	11
		others	3
25%	80	$0 \sim 100\%$	4
50%	10, 40, 80	$0 \sim 100\%$	4
75%	10, 40, 80	$0 \sim 100\%$	4

Table 4.3: Maximum speedup in the problem settings where topology is grid, and external constraint compatibility is 60%

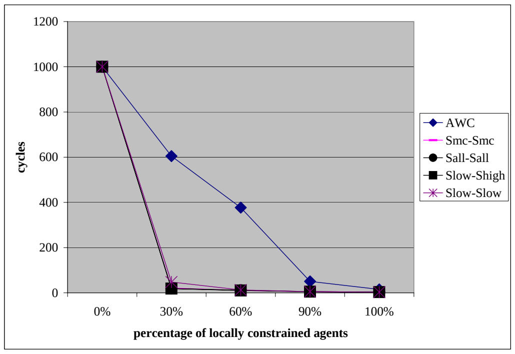
topology. Each cell in Table 4.2 and 4.3 shows the maximum speedup in the subgroup of problem settings classified by local constraint compatibility, domain size, and the ratio of locally constrained agents. While there can be a few exceptions, Table 4.2 and 4.3 provide the following results:

- When external constraint compatibility is low (30%),
  - For each topology, high performance improvement is achieved when local constraint compatibility is low (25%) and domain size is small (10).
    - A big speedup by LCDCSP strategies is shown except for the cases where, in terms of local constraints, agents are either totally unconstrained (0%) or totally constrained (100%).
    - For grid topology, a big speedup is also shown when domain size is large (80), and the ratio of locally constrained agents is moderate (60%) or high (90%). Note that, when all agents are locally constrained (100%), no speedup is shown.
- When external constraint compatibility is moderate (60%) in grid topology,
  - High performance improvement is achieved when local constraint compatibility is low (25%) and domain size is moderate (40).

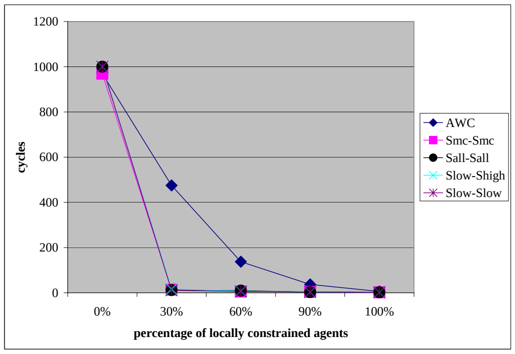
∗ A big speedup by the best locally cooperative strategy is shown when the ratio of locally constrained agents is high (90%). However, note that, when the ratio is 100%, there is no big speedup since all the problems in the setting are easy regardless of strategies to be applied (Figure B.13 in Appendix B.1).



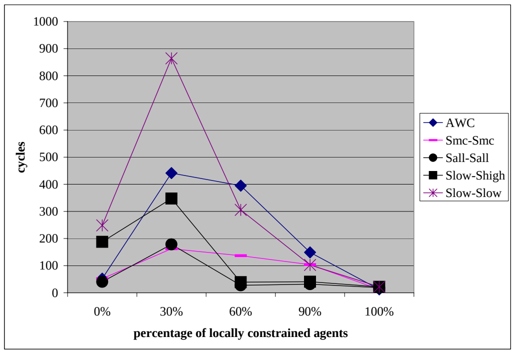
(a) Hexagonal topology; Domain size 10



(b) Grid topology; Domain size 10



(a) Triangular topology; Domain size 10



(b) Grid topology; Domain size 80

Figure 4.6: Example: Problem settings where external constraint compatibility is 30% and local constraint compatibility is 25%

Figure 4.6 and 4.7 show some prototypical cases where an order of magnitude speedup is achieved by LCDCSP strategies (including the problem settings represented

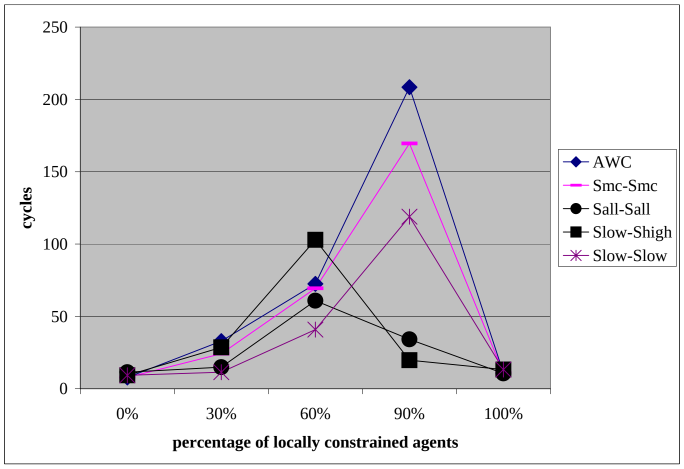


Figure 4.7: Example: Problem settings where topology is grid, external constraint compatibility 60%, local constraint compatibility 25%, and domain size 40

as grey cells in Table 4.2 and 4.3). [10] In Figure 4.6 and 4.7, the horizontal axis indicates the ratio of locally constrained agents and the vertical axis is the number of cycles taken by each strategy. The lower a data point of a strategy appears, the better performance the strategy shows. The results in Figure 4.6 and 4.7 show the followings:

- No single LCDCSP strategy dominates across all problem settings.
  - Therefore, given a problem setting, selecting the right strategy for the setting is essential to maximize the speedup by LCDCSP strategies, which motivates the second part of my thesis (performance modeling). [11]
- The most locally cooperative strategy ($S_{all} - S_{all}$) is not the best.

[10] While all the 16 locally cooperative strategies were tried on the problem settings described in Section 4.3.1, for expository purpose, the results of five locally cooperative strategies (the AWC strategy, $S_{min-conflict} - S_{min-conflict}$, $S_{all} - S_{all}$, $S_{low} - S_{high}$, and $S_{low} - S_{low}$) are shown in Figure 4.6 and 4.7. Note that using these five locally cooperative strategies does not change the conclusion from the work in this thesis.

[11] Section 4.3.6 provides further information about the dominance relation among strategies.

- It is not always the case that more information communication leads to improved performance (in some cases, LCDCSP strategies do not show performance improvement).

4.3.3 Performance in Run-time Analytical Model

In this section, I present how the performance results (e.g., speedup) changes with the analytical run-time model in Section 4.2 compared with the results based on *cycles*. To illustrate which property affects the change in performance by the run-time model, I also provide detailed results for some problem settings where LCDCSP strategies show a big speedup over the AWC strategy (Table 4.4)

	Topology	External constraint compatibility	Local constraint compatibility	Domain size	Ratio of locally constrained agents	*cycles*	
						AWC strategy	Best LCDCSP strategy
1	Hexagonal	30%	25%	10	30%	452	43 (11)
2	Hexagonal	30%	25%	10	60%	302	31 (10)
3	Grid	30%	25%	10	60%	377	10 (37)
4	Grid	30%	25%	80	60%	395	27 (14)
5	Grid	60%	25%	40	90%	208	20 (11)
6	Triangular	30%	25%	10	30%	475	11 (44)

Table 4.4: Problem settings with more than an order of magnitude speedup by LCDCSP strategies based on *cycles*: numbers in parenthesis (the last column) the speedup achieved by the best LCDCSP strategy

While there is a large degree of freedom in selecting the parameters for the run-time model, the parameters specified in this section assumes a realistic domain where message communication overhead dominates local computation cost and message processing overhead is relatively smaller than communication overhead (but cannot be ignored).

[12] In defining the parameters for such a domain, two different properties for message processing and communication overhead are considered as follows:

- Property 1: Message processing/communication overhead mainly depends on the size of messages to process/communicate.
  - A general case where the amount of information to send/receive is proportional to the real-time taken to process/communicate messages.
- Property 2: Message processing/communication overhead mainly depends on the number of messages to process/communicate.
  - Message is processed as a bundle (e.g., the processing time for one bit is same with that for one word), and message communication delay is dominated by message contention for each communication trial (message contention is less related with message size).

Next, we show the performance results (i.e., speedup by LCDCSP strategies) from the run-time model given property 1 and property 2 respectively.

Message Size as a Main Factor for Message Processing & Communication Overhead

For a domain where message size is a main factor for message processing and communication overhead, parameters for the run-time model are set as follows:

- $\mathcal{I}(l) = l \times t \times \alpha$ and $\mathcal{O}(m) = m \times t \times \alpha$

[12]In a prototype networked sensor (e.g., Berkeley Motes [Hill *et al.*, 2000] which has 4 MHz CPU), message processing takes tens or hundreds microseconds for a single bit. Furthermore, for a typical WINS (Wireless Integrated Networked Sensors) architecture, communication rate is assumed to be 1 $\sim$ 100 Kbps [Pottie and Kaiser, 2000]: sending a bit takes milliseconds or hundreds micro seconds without any message contention.

  - Message processing is assumed to be slower than a constraint check by two order of magnitude. To simulate such a difference, α is set as 100 or 1000.

- $\mathcal{T}(d) = d \times t \times \beta$

  - To simulate the situation where communication overhead dominates local computation cost, β is set as 1000 or 10000.

Note that further variation with different α and β is presented in Section 4.3.4. Table 4.5 shows the speedup by the best LCDCSP strategy for the problem settings shown in Table 4.4 given different α and β. Detailed information of the run-time for each problem setting computed from the run-time model is shown in Appendix B.3.1. In Table 4.5, the speedup based on the run-time model for different α and β is less than the speedup based on *cycles*. That is, the performance of LCDCSP strategies based on the run-time model appear to be worse than the performance based on *cycle*.

Case	Speedup by LCDCSP strategies				
	Based on *cycles*	Based on run-time model			
		$\alpha = 100$ $\beta = 1000$	$\alpha = 100$ $\beta = 10000$	$\alpha = 1000$ $\beta = 1000$	$\alpha = 1000$ $\beta = 10000$
1	11	7	7	7	7
2	10	9	9	8	9
3	37	21	21	20	21
4	14	4	7	5	7
5	11	7	8	7	8
6	44	33	33	31	33

Table 4.5: Speedup change in run-time model

The decrease in speedup with the *run-time* model is due to the fact that LCDCSP strategies have larger message size to process/communicate and more constraint checks (for computing flexibility towards neighboring agents) than the AWC strategy. Table 4.6 and 4.7 show the average message size and number of constraint checks for bottleneck agents which affect the *run-time* in the model proposed in Section 4.2 when when $\alpha = 100$ and $\beta = 10000$. The results with other other values of α and β (in Appendix B.4.1) are similar with those shown in Table 4.6 and 4.7.

Note that case 4 in Table 4.6 and 4.7 shows the largest difference between the AWC strategy and the best LCDCSP strategy in both message size and the number of constraint checks. The case 4 is for the problem setting where domain size (80) is larger than other cases (10 or 40). As domain size increases, the difference in message size and constraint checks between the AWC strategy and LCDCSP strategies also increases, leading to significant decrease in speedup for LCDCSP strategies. Furthermore, as graph density increases (i.e., as the number of neighbors increases), the difference in message size and constraint checks between the AWC strategy and LCDCSP increases. For instance, comparing case 1 (hexagonal topology: 3 neighbors) and case 5 (triangular topology: 8 neighbors) in Table 4.6 and 4.7, the high density setting (case 5) has larger difference in message size and constraint checks.

Cases	Average message size to process		Average message size to transmit	
	AWC strategy	Best LCDCSP strategy	AWC strategy	Best LCDCSP strategy
1	4.5	7.4	3.0	4.8
2	4.4	6.8	3.0	4.2
3	6.9	18.3	4.0	11.0
4	5.8	27.0	4.0	17.4
5	5.2	12.5	4.0	7.1
6	14.0	37.2	7.9	22.3

Table 4.6: Message size of AWC and the best LCDCSP strategy when $\alpha = 100$ and $\beta = 10000$

While *cycles* is independent of message size, the *run-time* model takes into account the overhead from the increased message size for LCDCSP strategies. Therefore, the speedup based on the *run-time* model is lower than the speedup based on *cycles*. Table

Cases	Average number of constraint checks	
	AWC strategy	Best LCDCSP strategy
1	28	40
2	24	28
3	30	46
4	379	5579
5	82	341
6	68	115

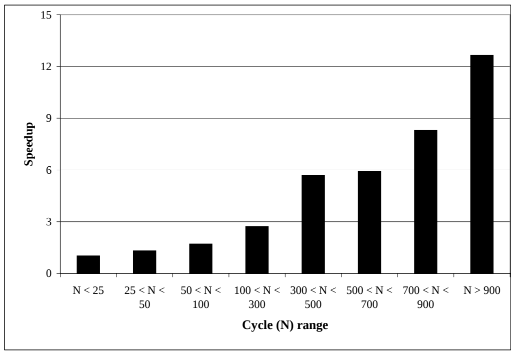
Table 4.7: Number of constraint checks of AWC and the best LCDCSP strategy when $\alpha = 100$ and $\beta = 10000$

cycles(N)	Number(percentage) of problem settings for different speedup (K fold)			
	$K < 2$	$2 \leq K < 5$	$5 \leq K < 8$	$K > 8$
$N < 200$	279 (93.6%)	18 (6.0%)	1 (0.3%)	0 (0%)
$200 \leq N < 500$	5 (29.4%)	4 (23.5%)	4 (23.5%)	4 (23.5%)
$N \geq 500$	29 (80.5%)	6 (16.6%)	0 (0%)	1 (2.7%)

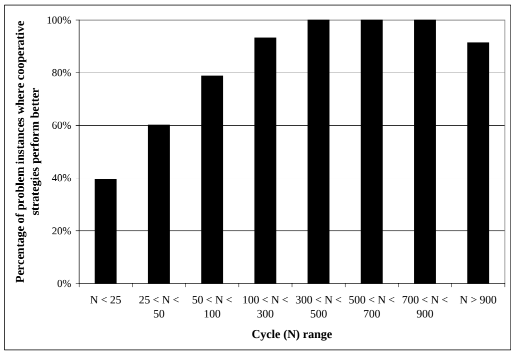
Table 4.8: Speedup by LCDCSP strategies when $\alpha = 100$ and $\beta = 10000$

4.8 shows the number (percentage) of problem settings with different speedups by LCDCSP strategies. For the 298 problem settings where *cycles* is less than 200, compared with the results in Table 4.1, the number (percentage) of problem settings with two or more speedup significantly decreases from 77 (25.8%) to (19) 6.3%. However, for the 61 problem settings with *cycles* more than 200, the decrease in the number (percentage) of problem settings with two or more speedup is relatively small: from 21 (39.6%) to (19) 35.8%. That is, for harder problem settings, LCDCSP strategies has less impact by the message processing/communication overhead.

Figure 4.8 shows how speedups change with this run-time model from the perspective of problem instances. To make a comparison, the problems are grouped by problem hardness (based on AWC performance in *cycles*) as is done in Figure 4.4. While the magnitude of speedup decreases, as problems get harder, big speedups are achieved by LCDCSP strategies and such speedups are shown in the majority of problem instances.



(a) Speedup for problem instances grouped by problem hardness (based on AWC performance)



(b) Percentage of problem instances where LCDCSP strategies perform better than AWC

Figure 4.8: Speedup for problem instances with different problem hardness: Checking whether speedup comes from easier or harder problem instances

Message Number as a Main Factor for Message Processing & Communication Overhead

For a domain where message number is a main factor for message processing and communication overhead, parameters for the run-time model are set as follows (note that message processing & communication time is independent of message size):

- $\mathcal{I}(l) = t \times \alpha$ and $\mathcal{O}(m) = t \times \alpha$
  - Message processing is assumed to be slower than a constraint check by two order of magnitude. To simulate such a difference, α is set as 100 or 1000.
- $\mathcal{T}(d) = t \times \beta$
  - To simulate the situation where communication overhead dominates local computation cost, β is set as 1000 or 10000.

Case	Speedup by LCDCSP strategies				
	Based on *cycles*	Based on run-time model			
		$\alpha = 100$ $\beta = 1000$	$\alpha = 100$ $\beta = 10000$	$\alpha = 1000$ $\beta = 1000$	$\alpha = 1000$ $\beta = 10000$
1	11	9	10	9	10
2	10	10	10	9	10
3	37	37	37	38	37
4	14	6	12	9	13
5	11	10	10	9	10
6	44	46	44	54	47

Table 4.9: Speedup change in run-time model

Table 4.9 shows the speedup by the best LCDCSP strategy for the problem settings shown in Table 4.4 given different α and β. Detailed information of the run-time for each problem setting computed from the run-time model is shown in Appendix B.3.2. In Table 4.9, the speedup based on the run-time model for different α and β is very similar with the speedup based on *cycles* in general. That is, the performance of LCD-CSP strategies based on the run-time model appear to be equal to (or, for some cases, interestingly better than) the performance based on *cycle*.

In contrast, for the run-time model where message size is a main factor for message processing and communication overhead (Table 4.5), the speedup based on the run-time model decreases in most cases (compared with the speedup based on *cycles*). The difference in speedup change between the run-time model with different properties (size-based message processing & communication overhead vs. number-based message processing & communication overhead) is due to the following:

- For a domain where message size is a main factor for message processing and communication overhead,

  - LCDCSP strategies may have larger message size depending on domain size and graph density while the AWC strategy has fixed message size (a selected value).

- For a domain where message number is a main factor for message processing and communication overhead,

  - While incoming message number may vary depending on the number of neighboring agents which have value change in the previous cycle, the number of messages to communicate is fixed as the number of neighbors (that is decided by graph density). As shown in Table 4.10, the difference in incoming message numbers is very small. Therefore, the AWC strategy and LCDCSP strategies are likely to have similar number of messages to process and communicate.

Cases	Average message number to process		Average message number to transmit	
	AWC strategy	Best LCDCSP strategy	AWC strategy	Best LCDCSP strategy
1	4.5	4.7	3.0	3.0
2	4.5	4.8	3.0	3.0
3	6.9	6.5	4.0	4.0
4	5.8	5.4	4.0	4.0
5	5.2	5.7	4.0	4.0
6	14.0	12.7	8.0	8.0

Table 4.10: Message size of AWC and the best LCDCSP strategy when $\alpha = 100$ and $\beta = 10000$

Table 4.10 and 4.11 show the average message number and number of constraint checks for bottleneck agents when when $\alpha = 100$ and $\beta = 10000$. Results with other values of α and β (which are similar with the results in Table 4.10 and 4.11) are presented in Appendix B.4.2.

Cases	Average number of constraint checks	
	AWC strategy	Best LCDCSP strategy
1	28	85
2	24	41
3	30	95
4	380	7743
5	83	488
6	68	235

Table 4.11: Number of constraint checks of AWC and the best LCDCSP strategy when $\alpha = 100$ and $\beta = 10000$

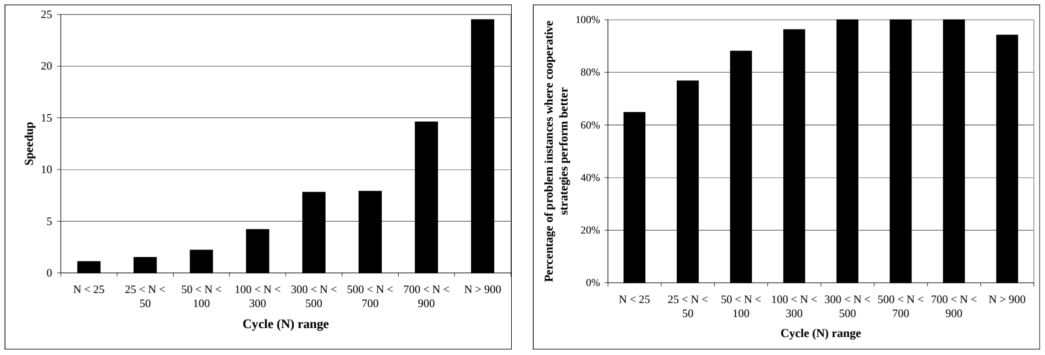
While Table 4.11 shows large difference in constraint checks between the AWC strategy and LCDCSP strategies, Table 4.10 shows that, for each setting, there is little difference in message size for the strategies. Therefore, when the message processing or communication overhead dominates (the difference in *constraint checks* becomes insignificant), the difference between the AWC strategy and LCDCSP strategies depends on the difference in *cycles* because of little difference in message size. Note that a significant speedup decrease (from 14 to 6) occurs only at the problem setting of case 4 in Table 4.9 where there is a big difference in *constraint checks* (case 4 at Table 4.11) and message processing & communication overhead is less dominant ($\alpha = 100$ and $\beta = 1000$).

cycles(N)	Number(percentage) of problem settings for different speedup (K fold)			
	$K < 2$	$2 \leq K < 5$	$5 \leq K < 8$	$K > 8$
$N < 200$	262 (87.9%)	32 (10.7%)	2 (0.7%)	2 (0.7%)
$200 \leq N < 500$	4 (23.5%)	4 (23.5%)	3 (17.6%)	6 (35.3%)
$N \geq 500$	29 (80.6%)	6 (16.7%)	0 (0%)	1 (2.8%)

Table 4.12: Speedup by LCDCSP strategies when $\alpha = 100$ and $\beta = 10000$

Table 4.8 shows the number (percentage) of problem settings with different speedups by LCDCSP strategies. For the 298 problem settings where *cycles* is less than 200, while the number of problem settings with two or more speedup decreases from 77 (25.8%) to 36 (12.1%), the decrease is smaller than the decrease in Table 4.8 (where message processing/communication overhead is based on message size). Furthermore, for the problem settings with *cycles* more than 200, the distribution of problem settings with different speedups is almost similar with the distribution in Table 4.1 (which is based on *cycles*).

Figure 4.9 shows how speedups change with this run-time model from the perspective of problem instances. To make a comparison, the problems are grouped by problem hardness (based on AWC performance in *cycles*) as is done in Figure 4.4. The results in Figure 4.9 are almost same with those in Figure 4.4. That is, for harder instances, LCDCSP strategies show big speedups and such speedups are shown in the majority of problem instances.



(a) Speedup for problem instances grouped by problem hardness (based on AWC performance)

(b) Percentage of problem instances where LCDCSP strategies perform better than AWC

Figure 4.9: Speedup for problem instances with different problem hardness: Checking whether speedup comes from easier or harder problem instances

The results in Table 4.12 and Figure 4.9 show that, for harder problem settings/instances, LCDCSP strategies have little impact by the message processing/communication overhead, which can be explained as follows:

- While *cycles* is independent of message size, the run-time model (which is also independent of message size) depends on a property (message number) for which the AWC strategy and LCDCSP strategies has little difference. The difference in constraint checks becomes insignificant when message processing or communication overhead is dominant.

Therefore, when message processing and communication overhead is mainly decided by message number (not message size) and the message processing/transmission overhead is dominant (the difference in *constraint checks* is not significant), *cycles* can be a reasonable measurement to compare the performance of strategies in particular for harder problem settings.

4.3.4 Performance Variation in Different Computing & Networking Environments

Section 4.3.3 assumes the case where communication overhead dominates the local computation overhead. However, there could be cases with different computing and networking environments. (e.g., communication overhead may be ignored since sensors are equipped with high-bandwidth communication devices). This section provides performance results in such different environments by changing the parameters for message processing overhead and communication overhead.

While the parameter setting is not focused on specific hardware/networking environments, the purpose of this investigation is to check how the speedup (achieved by LCD-CSP strategies) changes in various situations. To simulate various environments with different message processing and communication overhead, the parameters for message processing/communication overhead vary as follows:

- For a domain where message size is a main factor for message processing and communication overhead,
  - $\mathcal{I}(l) = l \times t \times \alpha$ and $\mathcal{O}(m) = m \times t \times \alpha$
    - α varies from 0 to 10000 (α = 0, 0.01, 0.1, 0.5, 1, 5, 10, 100, 1000, 10000)
  - $\mathcal{T}(d) = d \times t \times \beta$
    - β varies from 0 to 10000 (β = 0, 0.01, 0.1, 0.5, 1, 5, 10, 100, 1000, 10000)
- For a domain where message number is a main factor for message processing and communication overhead,
  - $\mathcal{I}(l) = t \times \alpha$ and $\mathcal{O}(m) = t \times \alpha$
    - α varies from 0 to 10000 (α = 0, 0.01, 0.1, 0.5, 1, 5, 10, 100, 1000, 10000)
  - $\mathcal{T}(d) = t \times \beta$
    - β varies from 0 to 10000 (β = 0, 0.01, 0.1, 0.5, 1, 5, 10, 100, 1000, 10000)

When message processing/communication overhead is zero ($\alpha = \beta = 0$), the speedup depends on the constraint checks: communication time is zero and the local computation time for bottleneck agents is decided by how many constraint checks are performed (regardless of how many messages are processed). However, when message processing/communication overhead is non-zero ($\alpha, \beta > 0$), communication and message-processing time is taken into account by the run-time models.

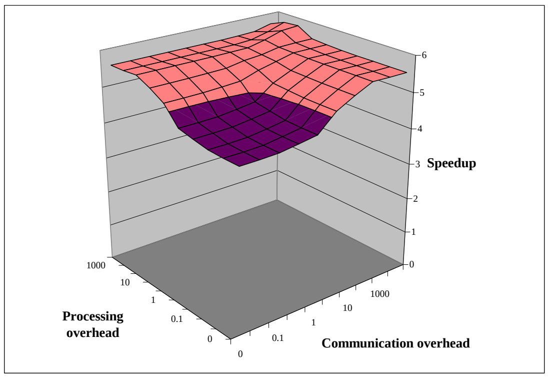
Figure 4.10, 4.11, and 4.12 show the variation in performance results depending on the change of α and β. These three settings (listed below) are prototypical examples that show an interesting phenomenon with the variation of α and β as shown below.

- Figure 4.10: Hexagonal topology where the external constraint compatibility is 30%, the local constraint compatibility is 25%, the domain size is 10, and the ratio of locally constrained agents is 30%.

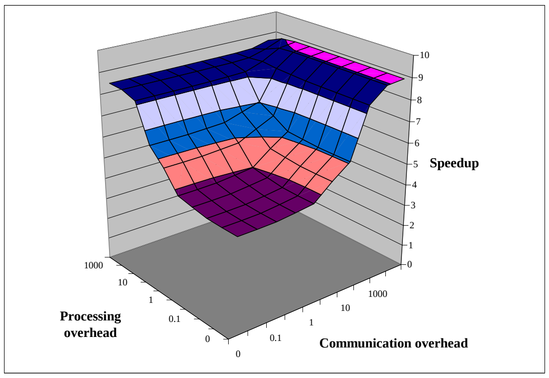
- Figure 4.11: Grid topology where the external constraint compatibility is 60%, the local constraint compatibility is 25%, the domain size is 40, and the ratio of locally constrained agent is 90%.

- Figure 4.12: Triangular topology where the external constraint compatibility is 30%, the local constraint compatibility is 25%, the domain size is 10, and the ratio of locally constrained agents is 90%.

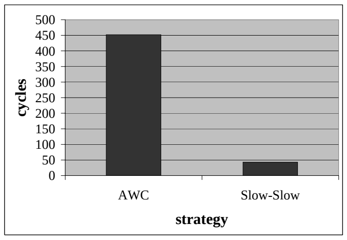
Figure 4.10-(a) shows the variation in the speedup by the best LCDCSP strategy in a given setting based on the run-time model where message processing/communication overhead is based on message size. In contrast, Figure 4.10-(b) is from the run-time model where message processing/communication overhead is based on message number. *Cycles* for both the AWC strategy and the LCDCSP strategy is shown in Figure



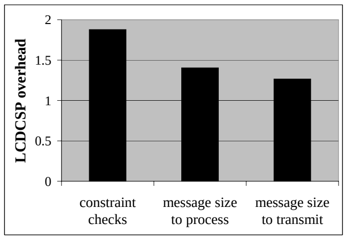
(a) Message processing & communication overhead based on message size



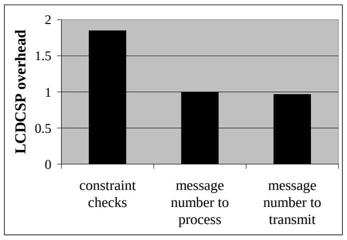
(b) Message processing & communication overhead based on message number



(c) cycles



(d) LCDCSP overhead in message size



(e) LCDCSP overhead in message number

Figure 4.10: Speedup variation: hexagonal topology; external constraint compatibility 30%; local constraint compatibility 25%; domain size 10; Ratio of locally constrained agents 30%

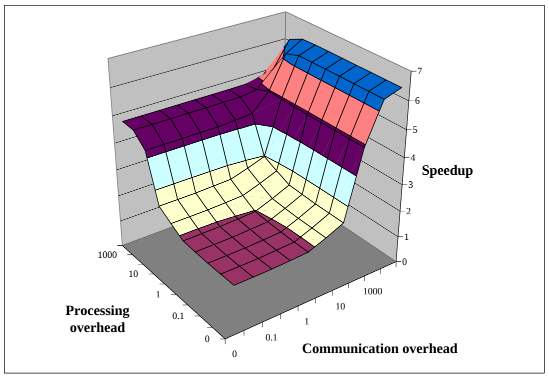
4.10-(c). Figure 4.10-(d) & (e) show the overhead by the LCDCSP strategy (i.e., compared with the AWC strategy, how many fold constraint checks/message-size/message-number per cycle is increased by the LDCSP strategy). [13] Figure 4.10-(a) & (b) show the following results:

[13]Note that the increase in average message size per cycle by the LCDCSP strategy is less than two fold because LCDCSP strategies do not need to communicate large messages at every cycle (only selected values are communicated after constraint propagation ends).

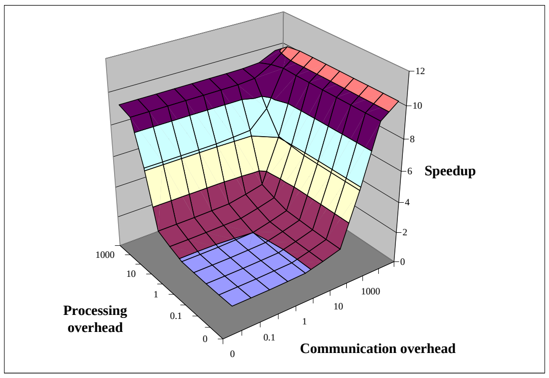
- As message processing or communication overhead increases (as message processing or communication overhead becomes dominant), the speedup by the LCDCSP strategy also increases.
  - In Figure 4.10-(d) & (e), the overhead by the LCDCSP strategy in message size or number is smaller than the overhead in constraint checks. As message processing/communication overhead increases (as the factors with less overhead for the LCDCSP strategy gets dominant), the speedup for the LCDCSP strategy increases. The LCDCSP strategy shows speedup despite of message processing/communication overhead for the LCDCSP strategy because of a large gain in *cycles* (shown in Figure 4.10-(c)).

Note that there is a difference in the magnitude of the speedups depending on runtime models (as shown in Section 4.3.3). While the speedup in Figure 4.10-(a) with the highest message processing overhead ($\alpha = 10000$) and communication overhead ($\beta = 10000$) is about 6 fold, the speedup in Figure 4.10 (b) with the same parameter setting is about 9 fold which is closer to the speedup (11 fold) based on *cycles*. This is because the relative overhead for the LCDCSP strategy in message number shown in Figure 4.10-(e) (compared with constraint-checks overhead) is smaller than the relative overhead in message size shown in Figure 4.10-(d).

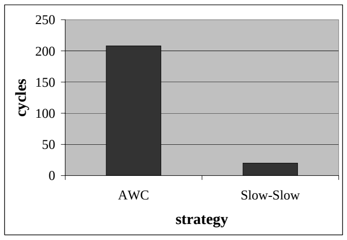
Figure 4.11 also shows the similar results: as message processing or communication overhead increases, the speedup by the best LCDCSP strategy also increases. One thing to notice is that, in Figure 4.11-(d) & (e), the difference between the overhead in constraint checks and the overhead in message size/number is greater than the difference in Figure 4.10-(d) & (e). This is because the domain size (40) of the problem setting for Figure 4.11 is greater than the domain size (10) for Figure 4.10. This difference leads to large variation in speedups depending on message processing/communication



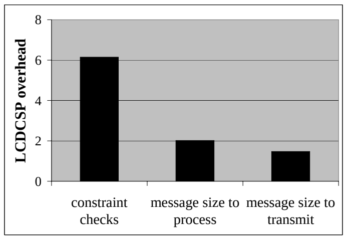
(a) Message processing & communication overhead based on message size



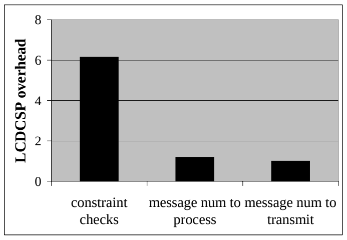
(b) Message processing & communication overhead based on message number



(c) cycles



(d) LCDCSP overhead in message size

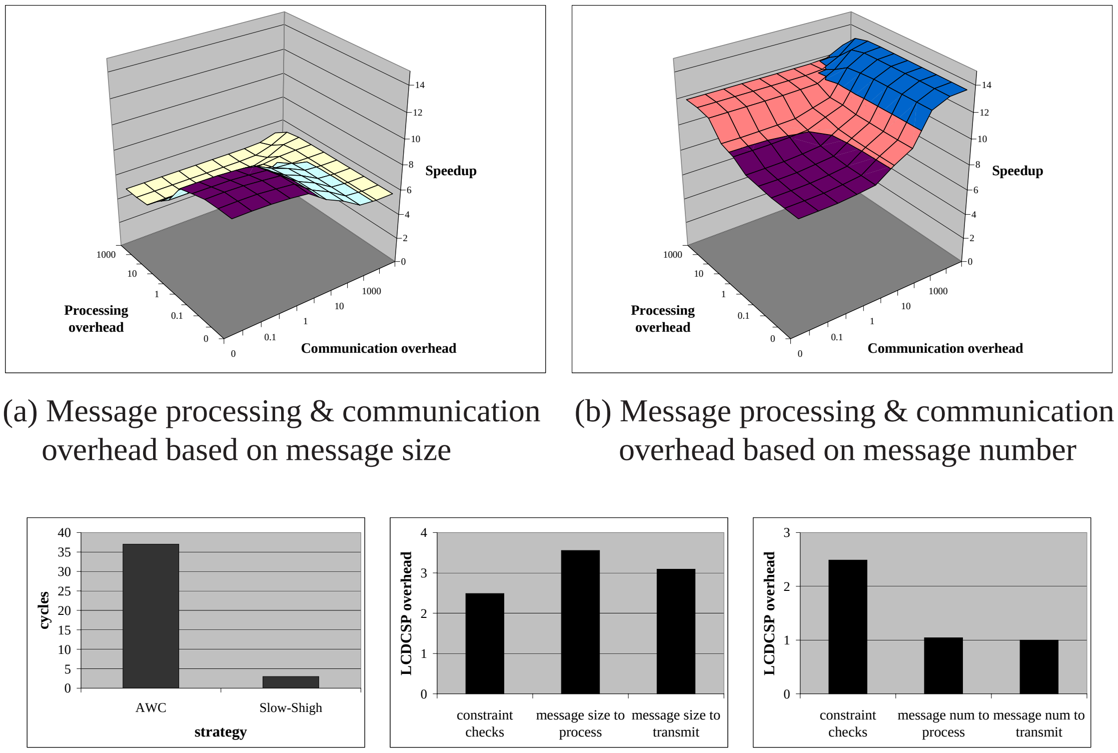


(e) LCDCSP overhead in message number

Figure 4.11: Speedup variation: grid topology; external constraint compatibility 60%; local constraint compatibility 25%; domain size 40; Ratio of locally constrained agents 90%

overhead. For instance, in Figure 4.10-(a), when message processing/communication overhead increases from 0 to 10000, the speedup changes from four-fold to six-fold. In contrast, in Figure 4.11-(a), the speedup changes from two-fold to six-fold.

While Figure 4.10 and 4.11 show that message processing/communication overhead and the speedup by the best LCDCSP strategy is positively correlated, Figure 4.12-(a) shows the opposite phenomenon. In Figure 4.12-(a), as message processing/communication overhead increases, the speedup by the best LCDCSP strategy



(a) Message processing & communication overhead based on message size (b) Message processing & communication overhead based on message number

(c) cycles (d) LCDCSP overhead in message size (e) LCDCSP overhead in message number

Figure 4.12: Speedup variation: triangular topology; external constraint compatibility 30%; local constraint compatibility 25%; domain size 10; Ratio of locally constrained agents 90%

decreases. Figure 4.12-(d) shows that the relative overhead from message size is greater than the overhead from constraint checks.

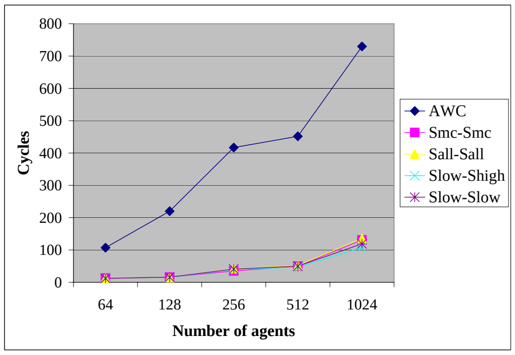
- Since the domain size is small (10) and 90% agents are locally constrained, most of agents has restricted domain (whose size is less than three), leading to smaller number of constraint checks for LCDCSP strategies. In contrast, since the graph density is higher (eight neighboring agents in triangular topology), each agent needs to process/communicate more messages.

However, the run-time model based on message number (Figure 4.12-(b)) is not affected by such a change in message size overhead. As shown in Figure 4.12-(e), the overhead from message number is lower than the overhead from constraint checks. Therefore, as message processing/communication overhead gets dominant, the speedup by the best LCDCSP strategy increases.

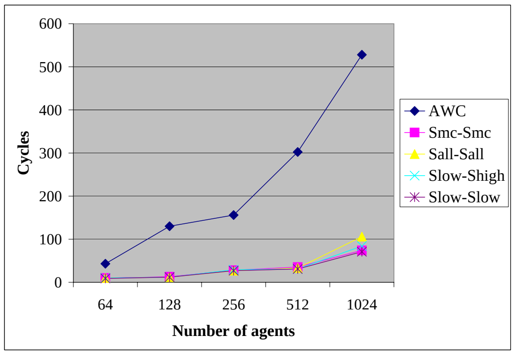
4.3.5 Scalability of LCDCSP Strategies

This section shows how the performance improvement by LCDCSP strategies changes as the number of agents varies. While this section does not provide exhaustive results for LCDCSP performance in the 351 problem settings (defined in Section 4.3.1) with different number of agents, the focus of this section is to get a general prospect for the scalability of LCDCSP strategies in particular for harder problem settings. Figure 4.13 shows the performance of LCDCSP strategies in the problem settings (listed in Table 4.4) where the best LCDCSP strategy provides more than an order of magnitude speedup over the AWC strategy. In Figure 4.13, the horizontal axis is the number of agents and the vertical axis plots the number of cycles taken by each strategy. Each data point is averaged over 35 problem instances randomly generated in a given setting.

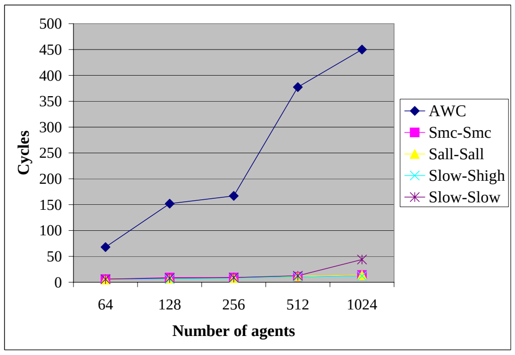
Figure 4.13 shows that the problem hardness increases as the number of agents increases. Overall, when the number of agents is small (e.g., 64), the speedup by the best LCDCSP strategy is not significant since each strategy does not take a large number of cycles to solve given problems instances. However, as the number of agents increases, higher speedups are shown by the best LCDCSP strategy. While there is a variation in speedups depending on problem settings, the speedups by the best LCDCSP strategy in the problem settings with 512 agents are not eliminated at the problem settings with 1024 agents.



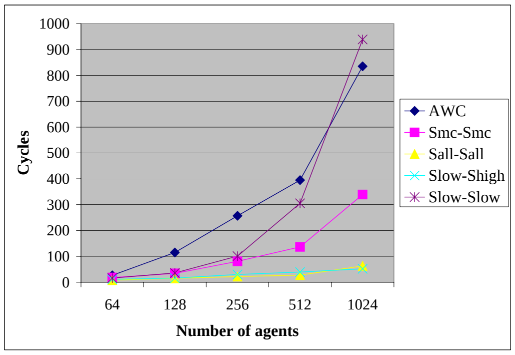
(a) Case 1



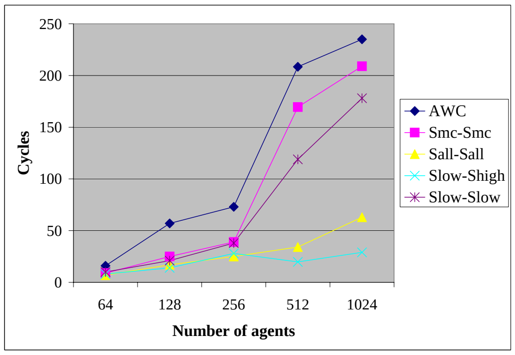
(b) Case 2



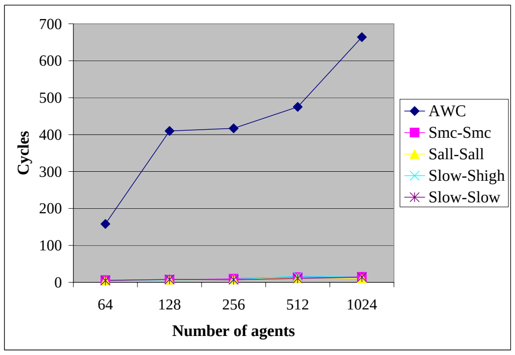
(b) Case 3



(c) Case 4



(d) Case 5



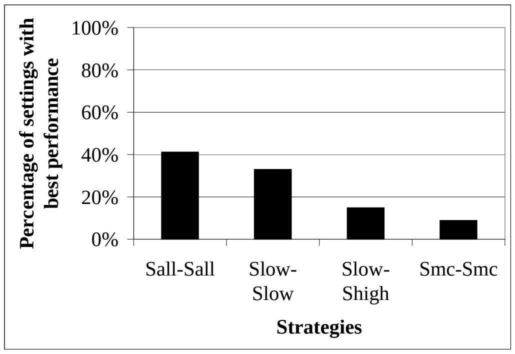
(e) Case 6

Figure 4.13: LCDCSP performance in different number of agents for selected cases (problem settings shown in Table 4.4)

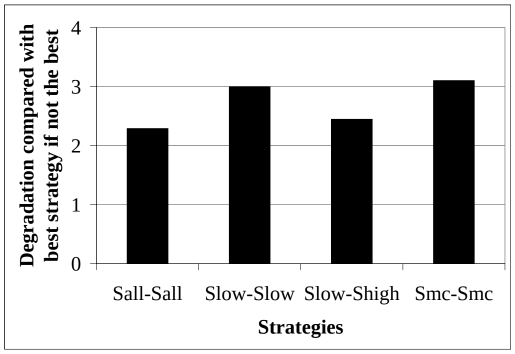
4.3.6 Motivation for Strategy Selection

Figure 4.6 and 4.7 show that no single strategy dominates across different domains. Therefore, for maximum speedup, we need to predict the right strategy in a given domain. However, for strategy selection, there is a question to be answered:

- Is there any strategy which performs best or relatively well with marginal difference between the strategy and the best strategy in a given setting?



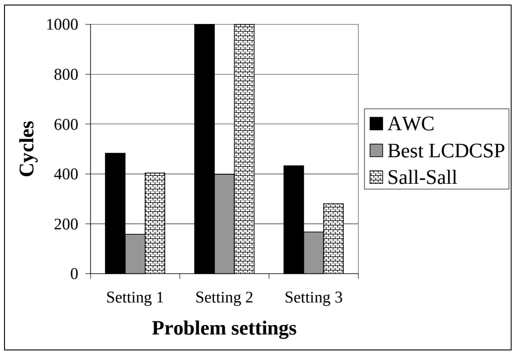
(a) Percentage of problem settings where where each strategy dominates



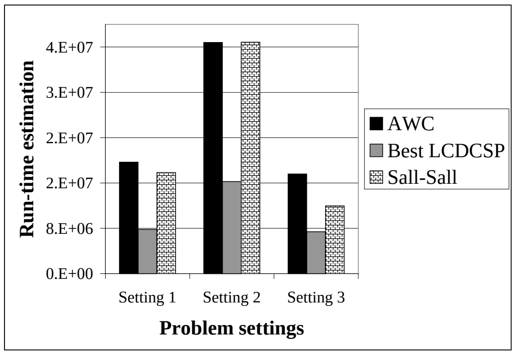
(b) Performance degradation by each strategy when it is not the best

Figure 4.14: Percentage of problem settings where an LCDCSP strategy dominates and how much worse it is if not the best (mc: min-conflict strategy)

If there exists such a strategy, it may not be necessary to predict the right strategy since we can apply the strategy in different problem settings. For instance, an LCDCSP strategy $S_{all} - S_{all}$ performs very well across different problem settings shown in Figure 4.6 and 4.7. Thus, $S_{all} - S_{all}$ could be used for fast conflict resolution in any problem setting described in Section 4.3.1. However, a strategy is not always the best and, when it is not the best strategy, there exists significant difference between the strategy and the best strategy in a given setting. Figure 4.14-(a) shows the percentage of problem settings (out of 351 problem settings) where a strategy dominates, and Figure 4.14-(b)



Cycles



Run-time model

Figure 4.15: Performance comparison between the best strategy and $S_{all} - S_{all}$ in the problem settings where $S_{all} - S_{all}$ is not the best (mc: min-conflict strategy): In run-time model, time unit 1 indicates the time for a single constraint check.

shows how many fold the strategy is worse on average (based on *cycles*) than the best strategy in the problem settings where it is not the best.

Here, $S_{all} - S_{all}$ is the best strategy only for 41% of problem settings. Furthermore, when $S_{all} - S_{all}$ is not best, it shows more than two fold degradation. Figure 4.15 shows the difference between the best strategy and $S_{all} - S_{all}$ in *cycles* (Figure 4.15-(a)) and the run-time estimation with $\alpha = 100$ and $\beta = 10000$ (Figure 4.15-(b)) (the parameters used in Section 4.3.3) for some selected settings where $S_{all} - S_{all}$ is worse.

As shown in Figure 4.15, there is a significant difference (e.g., more than 600 *cycles*) between $S_{all} - S_{all}$ and the best strategy in a given setting. Therefore, to gain maximum speedup with LCDCSP strategies, we need to predict the right LCDCSP strategy in a given problem setting. In the part 2 of this thesis, we will present novel performance modeling techniques for strategy selection.

Part II

Distributed POMDP-based Performance Models for Cooperative Conflict Resolution Strategies

Chapter 5

Performance Analysis

Chapter 4 shows that, given large-scale multiagent systems, predicting the right strategy to adopt in a given domain is essential to maximize the speedup of conflict resolution convergence, and the critical factor for strategy performance is in the long tail part where a small number of conflicts exist (explained in Section 5.1.3). Here, we provide formal models for performance analysis and the mapping of DCSP onto the models, and present the results of performance prediction.

5.1 Distributed POMDP-based Model

As a formal framework for strategy performance analysis, we use a distributed POMDP model called MTDP (Multiagent Team Decision Process) [Pynadath and Tambe, 2002]. The MTDP model has been proposed as a framework for multiagent analysis. Distributed POMDP-based model is an appropriate formal framework to model strategy performance in DCSP since it has distributed agents and the agentView (other agents' communicated values, etc.) in DCSP can be modeled as observations. In DCSP, the exact state of a system is only partially observable to an agent since the information that the agent receives is limited to its neighboring agents. Therefore, there is strong correspondence between DCSP and distributed POMDP as shown in Table 5.1. While we focus on the MTDP model in this paper, other distributed POMDP models such as DEC-POMDP [Bernstein *et al.*, 2000] could be used.

DCSP	Distributed POMDP
Distributed variables	Distributed agents
Communicated values of other agents	Observations
Value ordering strategy (e.g., S_{all})	Action
Fixed strategy combination (e.g., $S_{low} - S_{high}$)	Local policy

Table 5.1: Correspondence between DCSP and distributed POMDP

Here, we illustrate the actual use of the MTDP model in analyzing DCSP strategy performance. The MTDP model provides a tool for varying key domain parameters to compare the performance of different DCSP strategies, and thus select the most appropriate strategy in a given situation. We first briefly introduce the MTDP model. Refer to [Pynadath and Tambe, 2002] for more details.

5.1.1 MTDP model

The MTDP model involves a team of agents operating over a set of world states during a sequence of discrete instances. At each instant, each agent chooses an action to perform and the actions are combined to affect a transition to the next instance's world state. The current state is not fully observed/known and transitions to new world states are probabilistic. Each agent gets its own observations to compute its own beliefs, and the performance of the team is evaluated based on a joint reward function over world states and combined actions.

More formally, an MTDP for a team of agents, α, is a tuple, $< S, A_\alpha, P, \Omega_\alpha, O_\alpha, B_\alpha, R >$. S is a set of world states. $A_\alpha = \prod_{i \in \alpha} A_i$ is a set of combined domain-level actions where A_i is the set of agent i's actions. P controls the effect of agents' actions in a dynamic environment: $P(s, a, s') = Pr(S^{t+1} = s' | S^t = s, A^t_\alpha = a)$. Ω_α is a set of combined observations where Ω_i is the set of observations for agent i. Observation function, O_α specifies a probability distribution over the observations of a team of agents α at a given state after performing a joint action: $O_\alpha = \prod_{i \in \alpha} O_i$ where $O_i(s, a, \omega) = Pr(\Omega^t = \omega | S^t = s, A_\alpha^{t-1} = a)$. The observability for a world state by a team of agents (α) is classified as follows:

- *Collectively partial observability*: A general case where no assumption is made on the observations.
- *Collectively observability*: A unique world state is determined by α's joint observation. $\forall \omega \in \Omega_\alpha, \exists s \in S$ such that $\forall s' \neq s, Pr(\Omega_\alpha^t = \omega | S^t = s') = 0$.
- *Individual observability*: A unique world state is determined by each individual agent's observation. $\forall \omega \in \Omega_i, \exists s \in S$ such that $\forall s' \neq s, Pr(\Omega_i^t = \omega | S^t = s') = 0$.

For each agent, its belief state is derived from the observations. $B_\alpha = \prod_i B_i$ is the set of possible combined belief states where B_i is the set of possible belief state for an agent i. $R : S \times A_\alpha \rightarrow \Re$ is a reward function over states and joint actions. A policy (π_α) in the MTDP model maps agents' belief states to their actions: $\pi_\alpha : B_\alpha \rightarrow A_\alpha$.

5.1.2 Mapping from DCSP to MTDP

In a general mapping, the first question is how to select the right state representation for the MTDP. One typical state representation could be a vector of the values for all the variables in a DCSP. However, this representation leads to a huge state space. For instance, if there are 10 variables (agents) and 10 possible values per variable, the number of states is 10^{10}. To avoid this combinatorial explosion in state space, we use an abstract state representation in the MTDP model. In particular, as described in Chapter 3, each agent's local state can be abstractly characterized as being in a *good* or *nogood* state. *Good state* represents either the case where there is no constraint violation or the case where a constraint violation can be resolved locally (*good* case in Figure 3.2). *Nogood* state represents the *nogood case* in Figure 3.2 where a constraint violation cannot be resolved locally. We use this abstract characterization in our MTDP model. Henceforth, the *good* state and the *nogood* state are denoted by G and N respectively. The initial state of the MTDP is a state where all the agents are in G state since, in DCSP, an agent finds no inconsistency for its initial values until it receives the values of its neighboring agents. The world state is the product of individual agents' states (e.g., $<G$, G, N, G, $G>$ for five agents). There is also a special terminal state in which all the constraint violations are resolved.

In this mapping, the reward function R is considered as a cost function. The joint reward (cost) is proportional to the total number of agents in the N state since we focus on the overall performance, not the performance of individual agents. This reward is used for strategy evaluation based on the fact that the better performing strategy has less chance of forcing agents into the N state than other strategies: as a DCSP strategy performs worse in a given problem setting, more agents will be in the N states.

The locally cooperative strategies (such as S_{low}, S_{high}, S_{all}, and $S_{min-conflict}$ defined in Section 3.2.2) are mapped onto actions for agents in the MTDP model. A fixed DCSP strategy combination (e.g., $S_{low} - S_{high}$) provides a local policy for each agent in the MTDP model, e.g., the $S_{low} - S_{high}$ strategy implies that each agent selects action S_{low} when its local state is G (*good*), and action S_{high} when its local state is N (*nogood*). The state transition in the MTDP model is controlled by the joint action of a team of agents. The transition probabilities can be derived from the simulation on DCSP. Since DCSP strategies are mapped onto MTDP policies, we compare strategies by evaluating their corresponding policies in the MTDP model. Policy evaluation can be done by value iteration method. Note that, while policy evaluation computes values for all states in a MTDP, strategies are compared only with the initial state values under their corresponding policies. Our initial results from policy evaluation in this model match the actual experimental strategy performance results shown before (Chapter 4). Thus, the model could potentially form a basis for predicting strategy performance in a given domain.

In the AWC algorithm, each agent receives observations only about the states of its neighboring agents and its current state as well. Thus, the world is not individually observable but rather it is collectively observable (in the terminology of Pynadath and Tambe [Pynadath and Tambe, 2002]). In an individually observable world where each agent can individually observe the current world state, an MTDP can be reduced to an MDP. Refer to the theorems in [Pynadath and Tambe, 2002] for detailed reduction. In contrast, in a collectively observable world, if each agent can collect observations from all the others by communicating with them, an agent can identify the current world state with combined observations. However, in our investigation, agents only communicates their local constraints only with neighboring agents (note that it is not the case that information is propagated to all neighboring agents using multiple hops). Since agents cannot collect the information of all the other agents, the MTDP we have cannot be reduced to a single MDP. Initially, we assume that the observations from local communication are perfect (without message loss in communication). This assumption can be relaxed in our future work with unreliable communication.

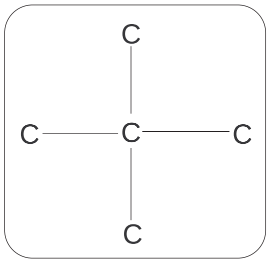
5.1.3 Building Block

While the abstract representation in the mapping above can reduce the problem space, for a large-scale multiagent system, if we were to model belief states of each agent regarding the state of the entire system, the problem space would be enormous even with the abstract representation. For instance, the number of states (including a terminal state) in the MTDP model for the system with 512 agents would be $2^{512}+1$: each agent can be either G or N. To further reduce the combinatorial explosion, we use small-scale models, called *building blocks*. As we show in the rest of this Chapter, this building block based approach enables efficiency in computation (by significant reduction of search space) and reusability of building blocks in different domains.

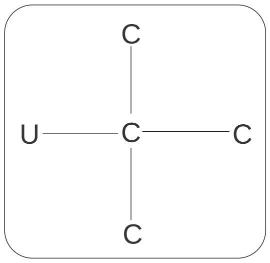
To model the performance of conflict resolution strategies in the experiments (presented in Chapter 3), each building block represents the local situation among five agents in the 2D grid configuration. In the problem domains for the experiments shown in Chapter 4, each agent's local situation depends on whether it has a unary local constraint or not: each agents can be either constrained under a local constraint (C) or unconstrained (U). Figure 5.1 illustrates some exemplar building blocks for the domain used in the experiments. For instance, block 1 (Figure 5.1.a) represents a local situation where all the five agents are constrained (C) while block 2 (Figure 5.1.b) represents a local situation where an agent in the left side is unconstrained (U) but the other four agents are locally constrained (C).

Note that, when the percentage of locally constrained agents is high, most of building blocks would be the block 1 (Figure 5.1.a) and a small portion of building blocks would be like the block 2, block 3, and block 4 (Figure 5.1.b, 5.1.c, and 5.1.d). As the percentage of locally constrained agents decreases, more building blocks include unconstrained agents (U) like block 5 and block 6 (Figure 5.1.e and 5.1.f).

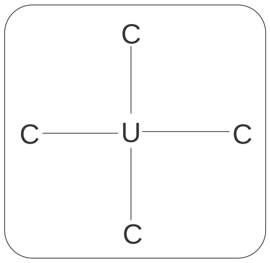
In each building block, as shown in Figure 5.2, a middle agent (A_3) is surrounded by four neighboring agents (A_1, A_2, A_4, A_5). Thus, the state of a building block can be represented as a tuple of local states $<s_1, s_2, s_3, s_4, s_5>$ (e.g., $<G, G, G, G, G>$ if all the five agents are in the *good* (G) state). There are totally 33 states (including a



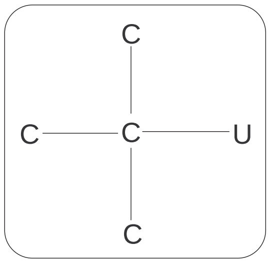
(a) Block 1



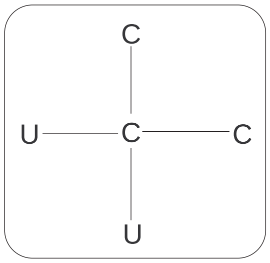
(b) Block 2



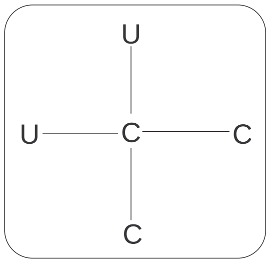
(c) Block 3



(d) Block 4



(e) Block 5



(f) Block 6

Figure 5.1: Building block example

terminal state) in a building block of the MTDP model (e.g., $<G, G, G, G, G>$, $<G, G, G, G, N>$, $<G, G, G, N, G>$, etc), and the initial state of a building block is $<G, G, G, G, G>$. Here, agents' actions will cause a transition from one state to another. For instance, if agents are in a state $<G, G, G, G, G>$ (Figure 5.2-a) and all the agents choose the action S_{high}, there is a certain transition probability that the next state will be $<G, G, N, G, G>$ (Figure 5.2.b) when only the third agent is forced into a *nogood* (N) state. However, there may be a transition to $<G, G, G, N, G>$ (Figure 5.2.c) if only the fourth agent enters into the N state.

The novelty of our building block decomposition technique is three fold, compared with the MDP and POMDP decomposition techniques in the literature [Dean and Lin,

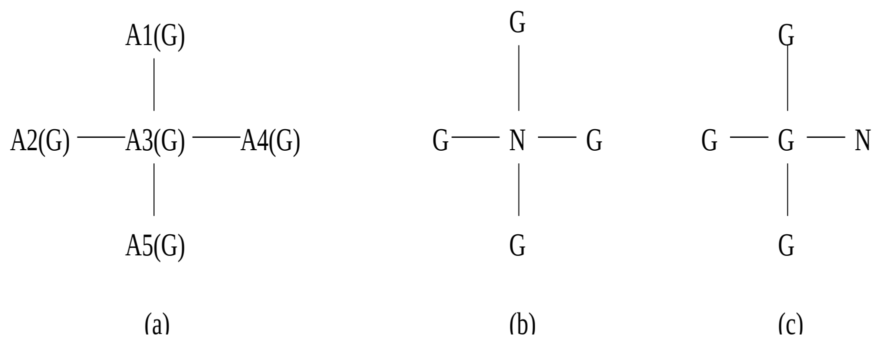


Figure 5.2: Example of states in a building block

1995; Hauskrecht *et al.*, 1998; Parr, 1998; Pineau *et al.*, 2001]: First, rather than exploiting geometric clusters within a domain (suitable in a single-agent POMDP), building-blocks exploit the multiagent nature of the domain — the decomposition is based on clustering agents with close interactions with each other; Second, wee are focused on evaluating policies rather than searching for optimal policies; Third, observability conditions within a building block can be exploited to further reduce the complexity of policy evaluation (proved in Section 5.2.1). Furthermore, building blocks can be reused in different domains which have some commonalities.

Finally, one may argue that these small-scale models are not sufficient for the performance analysis of the whole system since they represent only local situations. However, as seen in the long tail distribution shown in Figure 5.3, 5.4, 5.5, and 5.6 (in the following subsection), the key factor in determining the performance of strategies is in the long tail where only a small number of agents are in conflicts. Therefore, the performance of strategies are strongly related to the local situation where a conflict may or may not be resolved depending on the local agents' actions. That is, without using a model for the whole system, small scale models for local interaction can be sufficient for performance analysis. More importantly, composition techniques as introduced below show how larger scale models can be developed.

Long tail distribution in Convergence

The key factor in determining the performance of strategies is in the long tail where only a small number of agents are in conflicts. Figure 5.3 shows the number of average conflicts at each cycle for two different strategies, $S_{low} - S_{high}$ and $S_{min-conflict} - S_{min-conflict}$, with sum as their flexibility base for problem instances where 90% of agents are locally constrained in the *high flexibility setting*. Note that the average speedup difference in *cycles* in the problem setting was 7-fold. As shown in Figure 5.3, in the beginning of conflict resolution, both strategies show similar performance in resolving conflicts.

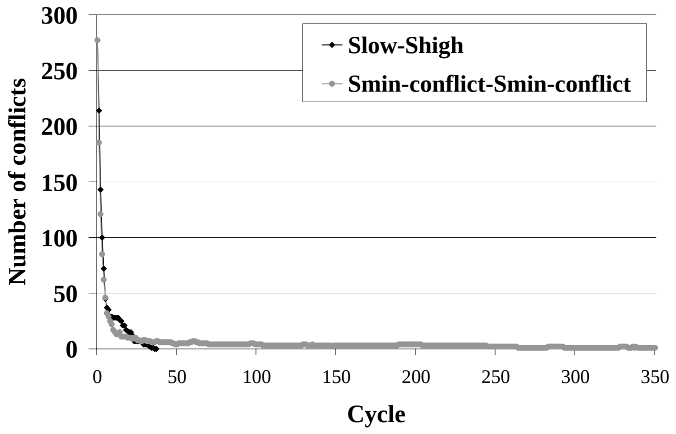


Figure 5.3: Long-tail distribution example for the case with 90% locally constrained agents in high flexibility setting

However, the performance difference appears in the long tail part. While $S_{low} - S_{high}$ quickly solves a given problem, $S_{min-conflict} - S_{min-conflict}$ has a long tail with a small number of conflicts remaining unresolved. Figure 5.4 also shows such long tail distribution for two strategies $S_{low} - S_{high}$ and $S_{min-conflict} - S_{min-conflict}$ given problem instances with 90% locally constrained agents in the *low flexibility setting*: on average, they showed more than 10-fold difference in conflict resolution performance.

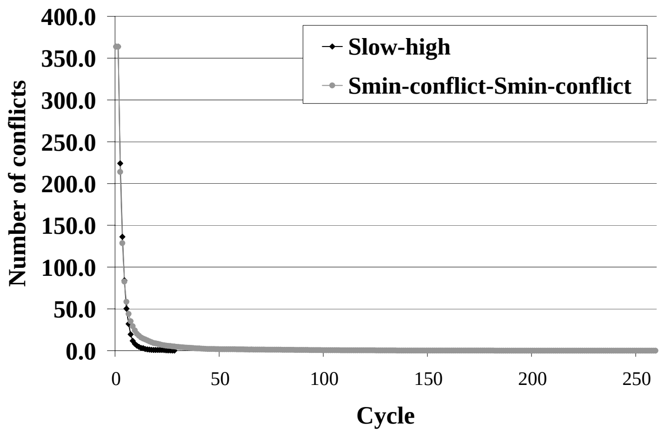


Figure 5.4: Long-tail distribution example for the case with 90% locally constrained agents in low flexibility setting

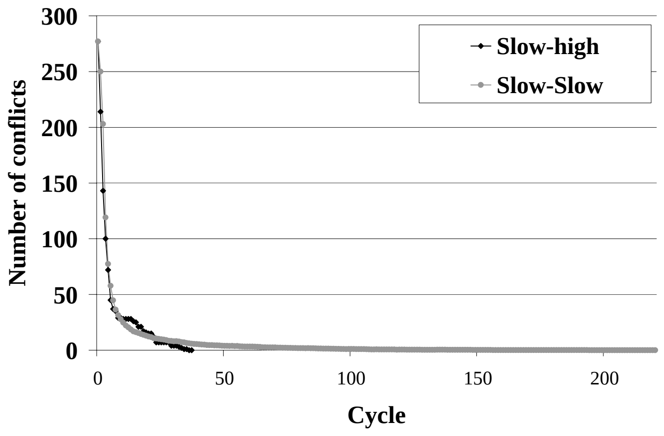


Figure 5.5: Long-tail distribution example for the case with 10% locally constrained agents in high flexibility setting

Figure 5.5 compares another pair of strategies, $S_{low} - S_{high}$ and $S_{low} - S_{low}$, (that show a 6 fold average speedup) in the same setting with Figure 5.3. Figure 5.6 also shows the long tail distribution for $S_{high} - S_{high}$ and $S_{min-conflict} - S_{min-conflict}$ when 10% agents are locally constrained in the *low flexibility setting*. This type of long tail distribution has been also reported in many constraint satisfaction problems [Gomes *et al.*, 2000].

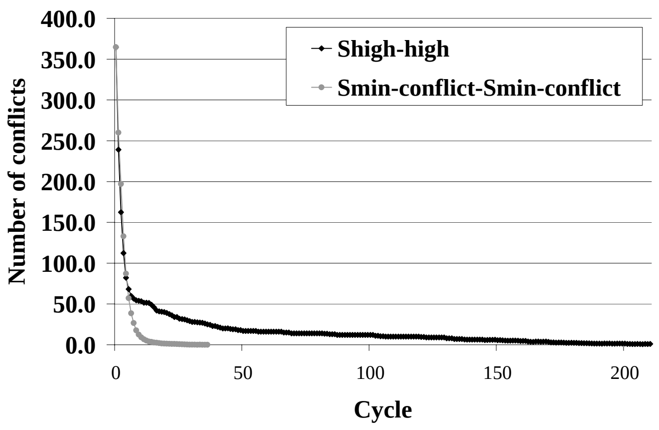


Figure 5.6: Long-tail distribution example for the case with 10% locally constrained agents in low flexibility setting

5.1.4 Building Block Composition for Performance Analysis

While the building blocks are the basis for performance analysis, we need to deal with multiple building blocks that can exist in a given domain. Each building block has a different impact on conflict resolution convergence. It is expected that the local interaction in a single building block does not totally determine the performance of strategies, but the interactions between building blocks have a great impact on the strategy performance. In this section, we propose four methods of building block composition to evaluate MTDP policies (mapping of DCSP strategies) as follows:

- **Single block**: For a given domain, a single building block is selected to represent the domain. For instance, when 90% of agents are locally constrained, block 2 (Figure 5.1.b) is selected as a representative block since most of agents are locally constrained (C) in the problem setting. For the case of 60% locally constrained agents, block 6 (Figure 5.1.f) is selected instead of block 1. The performance of a strategy is evaluated based on the value of an initial state ($<G, G, G, G, G>$) for a single representative building block.

- **Simple sum**: For a given domain which has multiple building blocks, we compute the value of each building block's initial state. Performance evaluation of a policy is based on the summation of the initial states' values of the multiple building blocks in the domain. Individual building blocks are selected to mirror the local constrainedness ratio. For instance, in the case of 90% locally constrained agents, block 1 (Figure 5.1.a), block 2 (Figure 5.1.b), block 3 (Figure 5.1.c), and block 4 (Figure 5.1.d) are selected: the local constrainedness ratio in these building blocks varies from 100% to 80%, but it is as close to 90% as possible. Thus, when 90% of agents are locally constrained agents, block 1, 2, 3, and 4 in Figure 5.1 are chosen and strategies are compared with the sum of initial state values for the selected four building blocks (block 1, 2, 3, and 4).

- **Weighted sum**: Given multiple building blocks, for each building block, we compute the ratio of the building block in the domain and the value of its initial state. Thus, in contrast with "simple sum" method, it is the combination of building blocks that should mirror the ratio in a given domain. Performance evaluation of a policy is based on the weighted sum of initial states' values where the weight is the ratio of each building block in a given domain. The detailed procedure is shown in Figure 5.7. For instance, in the case of 90% locally constrained agents, block 1, 2, 3, and 4 (four building blocks selected above) have weights 0.4, 0.2, 0.2, and 0.2 respectively.

- **Interaction**: Given a set of building blocks, their local interaction influences the starting state of neighboring building blocks since neighboring building blocks may share the same agent. For instance, in a 2D grid configuration, two building blocks can be connected via an agent which, henceforth, is called a docking point as shown in Figure 5.8. The joint action in a building block may lead to a specific

```
Input: for a given domain,
  – joint policy: $\pi$
  – building blocks in the domain: $B_1 \ldots B_k$
  – weight for each building block: $\omega_1 \ldots \omega_k$

Output:
  – policy evaluation value ($eval$) for the given joint policy $\pi$
    /* this will be used to compare strategy performance */

Procedure compute_weighted_sum
  1. $eval$ = 0 /* initialize policy evaluation value */
  2. for $i$ = 1 to $k$ do
       – do value iteration for $B_i$ under policy $\pi$ /* compute values for every state */
       – assign the initial state value to $v_i$
  3. for $i$ = 1 to $k$ do
       – $eval = eval + \omega_i \times v_i$
  4. return $eval$
```

Figure 5.7: Weighted sum method

state that influences the initial state of its neighboring agent. As illustrated in Figure 5.9, block 2 (Figure 5.1.b) and block 3 (Figure 5.1.c) may interact side by side so that the rightmost C agent of block 2 shares the same state with the leftmost C agent of block 3. The joint action under a given policy in block 2 may force an agent in a docking point to enter into the *nogood* (N) state, and let block 3's starting state have N in its leftmost agent. With the *interaction* between these two building blocks, the policy of block 2 influences the probability that the initial local state of the leftmost C agent of block 3 starts in N state. Without such *interaction*, it would always start in the G state.

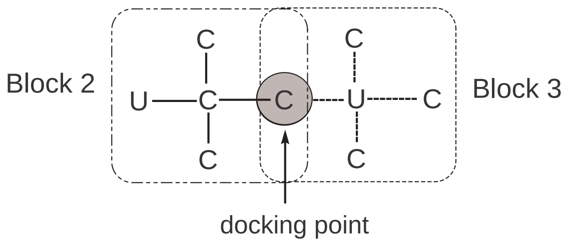


Figure 5.8: Docking point between two building blocks

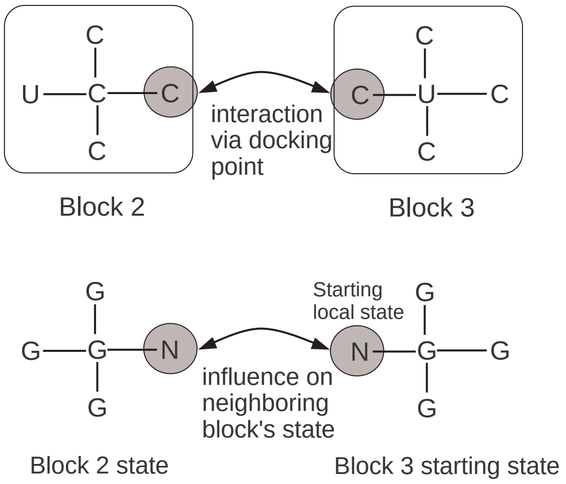


Figure 5.9: Interaction between building blocks

In contrast, the previous two methods (*sum* and *weighted sum*) are based on the value of the initial state $<G, G, G, G, G>$ of each building block without taking interactions between building blocks into account. However, since building blocks interact with each other, their starting state is not always $<G, G, G, G, G>$. As shown in the long tail distribution of constraint violation (Figure 5.3), the performance difference depends on how a conflict is resolved in a building block or propagated to another building block, and how the propagated constraint violation is resolved in the neighboring building block also has an impact on the performance.

Figure 5.10 shows the detailed procedure of the *interaction* method. Here, we assume that we have building blocks in an arrangement of one block (a *trigger* building block) in the center with other k blocks surrounding it: the *trigger* building block is assumed to have initial constraint violation. For instance, to analyze the conflict resolution strategies in our experimental setting with 2D grid configuration, a *trigger* building block is surrounded by four neighboring blocks to create an effect of a grid.

Note that, for the *interaction* method, we don't have arbitrary degrees of freedom. Not every combination of building blocks is feasible since the building blocks must actually *dock* at the proper points. For instance, block 1 (Figure 5.1.a) cannot be directly connected to block 2 (Figure 5.1.b) at the U node as illustrated in Figure 5.11. This is because the docking point does not match: the block 1 only has C nodes, and has no matching U node. The combination also has to maintain the percentage of locally constrained agents. For instance, when 90% of agents are locally constrained, the ratio of C agents in a building block combination must be 90%.

As we change from the first method (*single block*) to the fourth method (*interaction*), we gradually increase the complexity of composition. The accuracy of the performance prediction with those methods is presented in the next section. Here, note that the focus of our building block composition is not on computing the optimal policy, but it remains on matching the long-tailed phenomenon shown in Figure 5.3. Thus, our interactions essentially imply that neighboring blocks affect each other in terms of the values of the policies being evaluated, but we are not computing optimal policies that cross building block boundaries.

```
Input: for a given domain,
  – a trigger building block $B_0$ and its neighboring building blocks: $B_1$, $B_2$, $\cdots$, $B_k$
  – joint policy: $\pi$

Output:
  – policy evaluation value ($eval$) for the given joint policy $\pi$
    /* this will be used to compare strategy performance */

Procedure interaction
  1. $eval$ = 0 /* initialize policy evaluation value for interaction method */
  2. for $i$ = 0 to $k$ do
       – do value iteration for $B_i$ under policy $\pi$      $\cdots\cdots\cdots$   $(\star)$
         /* compute values of all states in $B_i$ */
  3. $eval$ = $B_0$'s initial state value /* $B_0$ is a trigger building block */
  4. for $i$ = 1 to $k$ do
       – compute the probability that $B_0$'s joint action forces the agent in a docking
         point with $B_i$ to enter into the nogood ($N$) state
       – assign the probability to $p_i$
  5. for $i$ = 1 to $k$ do
       – for building block $B_i$, find a state $s$ where a docking point agent is in $N$ state
         and other agents are in $G$ state ($B_i$'s starting state forced by $B_0$)
       – assign the value of $s$ to $v_i$
         /* values of all $B_i$'s states (including $s$) are already computed at the value
         iteration step above (marked with $\star$)*/
       – eval = eval + $p_i \times v_i$
  6. return $eval$
```

Figure 5.10: Interaction method

5.2 Performance Prediction

To check whether the MTDP based model can effectively predict the performance of strategies, four methods of building block composition (defined in Chapter 5.1.4) are applied for performance analysis, and the performance evaluation results from the

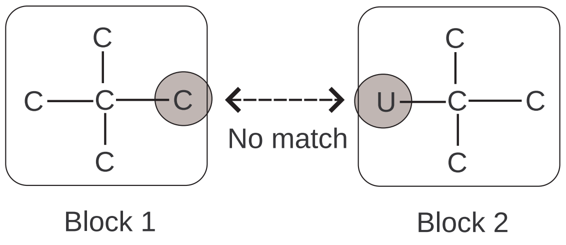


Figure 5.11: Building block docking points

MTDP model are compared with the real experimental results presented in Chapter 3. Before showing the performance analysis results, we present the complexity of DCSP strategy performance evaluation. The complexity analysis presented below shows that observability conditions within a building block can be exploited to further reduce the complexity of policy evaluation, which makes our building block based approach more useful in practice by saving computation overhead.

5.2.1 Complexity of Performance Evaluation

The policy evaluation for a finite horizon MTDP is a computationally expensive problem since the action is indexed by observation history. In the worst case, the computational complexity of evaluating a single policy is $O((|S||\Omega|)^T)$: $|S|$ and $|\Omega|$ are the number of states and observations respectively. However, in the environment where a given policy is based only on agents' local states, we prove that evaluating the policy becomes the evaluation of a time homogeneous Markov chain, which leads to low computation overhead. To elaborate on the proof, we first introduce the following definitions, followed by a theorem that proves the MTDP reduction to a Markov chain:

- **Markov Chain** ([Puterman, 1994]): Let $\{X_n \mid n = 0, 1, \cdots\}$ be a sequence of random variables which assume values in a discrete (finite or countable) state space S. The sequence is called a Markov Chain if $P\{X_{n+1} = j_{n+1} \mid X_n = j_n, X_{n-1} = j_{n-1}, \cdots, X_0 = j_0\} = P\{X_{n+1} = j_{n+1} \mid X_n = j_n\}$ for $n \geq 1$ and $j_k \in S, 0 \leq k \leq n$.

- **Time Homogeneous Markov Chain** ([Puterman, 1994]): A Markov Chain is homogeneous if, for $n \geq 1$ and $\forall p, q \in S$, $P\{X_{n+1} = p \mid X_n = q\}$ does not depend on n.

- **Local state** LS_i

  - $LS_i = \Xi_{i1} \times \Xi_{i2} \times \cdots \times \Xi_{im_i}$: the subset of features of the world state that affect the observation of an agent i.
  - A world state $s \in S$ in MTDP is $\bigcup_{i \in \alpha} ls_i$ where $ls_i \in LS_i$.

- **Local observability**: each individual's observation uniquely determines its local state.

  - $\forall \omega \in \Omega_i, \exists ls' \in LS_i$ such that $ls' \neq ls$, $Pr(\Omega_i^t = \omega \mid LS_i^t = ls') = 0$.

Based on the above definitions, we show the proof that, under certain assumptions, an MTDP reduces to a Markov chain, leading to significant computation cost savings.

- **Theorem 2**: Given a MTDP $< S, A_\alpha, P, \Omega_\alpha, O_\alpha, B_\alpha, R >$, given a fixed policy, the MTDP reduces to a time homogeneous Markov chain with S if the following assumptions hold:

  - Assumption 1: the environment is locally observable;
  - Assumption 2: the domain level policy π_{iA} is a function of i's current local state ($ls_i \in LS_i$) only.

– **Proof sketch**: Assume a sequence of random variables for states S, $\{X_0, X_1, \cdots\}$. First, we show that the conditional probability distribution of X_{k+1} depends on only X_k: that is, $P\{X_{k+1} = j_{k+1} \mid X_k = j_k, X_{k-1} = j_{k-1}, \cdots, X_0 = j_0\} = P\{X_{k+1} = j_{k+1} \mid X_k = j_k\}$ where $p, q, j_m \in S, 0 \leq m \leq k-1$.

Given that the world is Markovian, the probability distribution of X_{k+1} can be computed by $P\{X_{k+1} = j_{k+1} \mid X_k = j_k, a\}$ where the a is the joint action such that $a = \bigcup_{i \in \alpha} a_i$ where a_i is agent i's individual action selected at state j_{k+1}.

Note that the action a_i is selected by i's domain-level policy $\pi_{iA} : B_i \rightarrow A$. At state j_k, i's belief state is its observation history $< \Omega_i^0, \Omega_i^1, \cdots, \Omega_i^k >$ where Ω_i^m is i's observation at the m^{th} state.

Since $a_i = \pi_{iA}(< \Omega_i^0, \Omega_i^1, \cdots, \Omega_i^k >)$,

$P\{X_{k+1} = j_{k+1} \mid X_k = j_k, a\}$

$= P\{X_{k+1} = j_{k+1} \mid X_k = j_k, \bigcup_{i \in \alpha} a_i\}$

$= P\{X_{k+1} = j_{k+1} \mid X_k = j_k, \bigcup_{i \in \alpha} \pi_{iA}(< \Omega_i^0, \Omega_i^1, \cdots, \Omega_i^k >)\}$

Because of assumption 1 (local state uniquely decided by observation) and 2 (action decided by current local state only), $\pi_{iA}(< \Omega_i^0, \Omega_i^1, \cdots, \Omega_i^k >) = \pi_{iA}(ls_i^k)$ where ls_i^k is i's local state at state j_k that is uniquely decided by Ω_i^k.

Here, the above expression is transformed to

$P\{X_{k+1} = j_{k+1} \mid X_k = j_k, \bigcup_{i \in \alpha} \pi_{iA}(ls_i^k)\}$

$= P\{X_{k+1} = j_{k+1} \mid X_k = j_k, \bigcup_{i \in \alpha} ls_i^k\}$ since the policy π_{iA} is fixed

$= P\{X_{k+1} = j_{k+1} \mid X_k = j_k\}$ because $\bigcup_{i \in \alpha} ls_i^k = j_k$

Therefore, the conditional probability distribution of X_{k+1} depends on only X_k

Now, to prove that the Markov chain is time homogeneous, we show that the transition between X_{k+1} and X_k is independent of the index k.

$\forall p, q \in S$ and $\forall k, m \in \mathbf{N}$, $P\{X_{k+1} = p \mid X_k = q\} = P\{X_{m+1} = p \mid X_m = q\} = P(q, a, p)$ where P is defined in the MTDP and a is decided by state q as shown above.

Therefore, the Markov chain is time homogeneous, and the reduction from an MTDP to a time homogeneous Markov chain is proved.

The above theorem shows that the performance evaluation of the DCSP strategies defined in Chapter 3 can be reduced to the evaluation of a Markov chain because the DCSP strategies and the environment for them satisfy the required assumptions for the above theorem 2 as follows:

- DCSP strategies are based on only local state (whether an agent is in the *good* (G) or the *nogood* (N) state).

  - E.g., $S_{low} - S_{high}$ indicates that S_{low} strategy is applied for G state and S_{high} strategy for N state.

- The environment is locally observable since each agent can find its own state based on its value and communicated values (observation) of neighboring agents.

Thus, performance evaluation of the DCSP strategies can be done in $O(|S|^3)$ where $|S|$ is the number of states in a given MTDP since their corresponding policies can be evaluated by the same method of value determination for MDP: value determination in MDP can be done in $O(|S|^3)$ by solving a system of linear equations where the number of variables is the number of states [Littman *et al.*, 1995].

5.2.2 Analysis of Performance Prediction

While there can be various problem domains, in this initial investigation, we focus on the following problem setting:

- Grid topology with domain size of 40 and local constraint compatibility of 25%

  1. External constraint compatibility of 60% (henceforth, this setting is referred as *high flexibility setting*)
  2. External constraint compatibility of 30% (henceforth, this setting is referred as *low flexibility setting*)

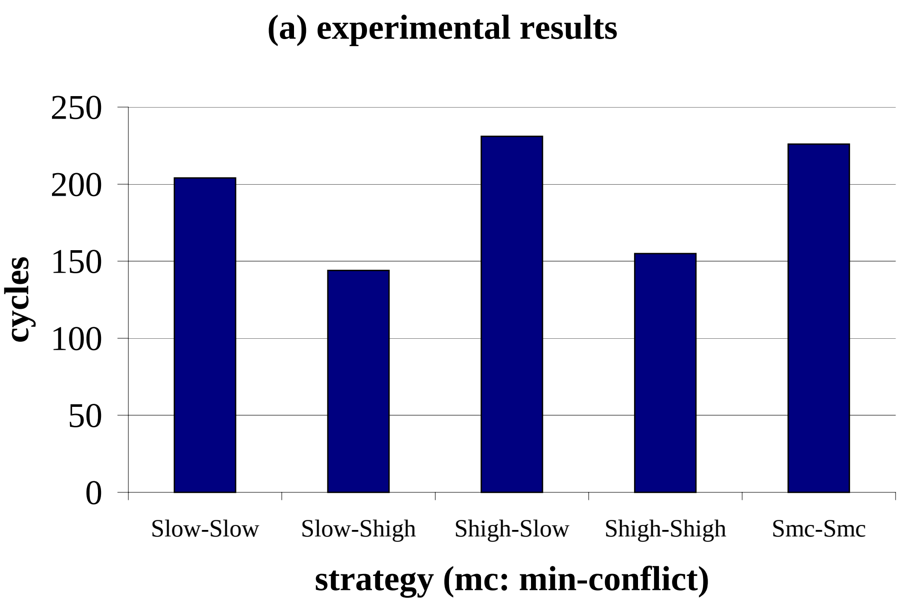
For both the *high flexibility setting* and the *low flexibility setting*, we focus on two special cases where the percentages of locally constrained agents are 60% and 90% respectively [1]. Among the possible strategies defined in Chapter 3, we focus on the strategies with

[1]Note that we do not perform analysis in the 0% case since there is no significant performance difference in the case

sum as a flexibility base that were selected in Chapter 4 for expository purpose: $S_{low} - S_{low}$, $S_{low} - S_{high}$, $S_{high} - S_{low}$, $S_{high} - S_{high}$, and $S_{min-conflict} - S_{min-conflict}$. Note that all of these strategies are enhanced with extra communication of local information. Thus, we compare four building block composition methods described in Section 5.1.4 for the above five strategies in the following four cases:

- Case 1: when 60% agents are locally constrained in the *low flexibility setting*.
- Case 2: when 90% agents are locally constrained in the *low flexibility setting*.
- Case 3: when 60% agents are locally constrained in the *high flexibility setting*.
- Case 4: when 90% agents are locally constrained in the *high flexibility setting*.

Figure 5.12 shows the results when 60% agents are locally constrained in the *low flexibility setting*. Figure 5.12-(a) (top) shows the experimental results in the case and Figure 5.12-(b) (bottom) shows the performance evaluation from the MTDP model using the *interaction* composition method. In the performance evaluation, the lower value means the better performance (indicating that less number of cycles will be taken until a solution is found). Figure 5.12-(a) and (b) show that the performance prediction results match to the real experimental results in the problem setting considered here: the patterns of column graphs in Figure 5.13-(a) and (b) correspond to each other. That is, the strategies ($S_{low} - S_{high}$ and $S_{high} - S_{high}$) that showed better performance in the experiments are expected to perform better than the others according to the performance prediction. Figure 5.13, 5.14, and 5.15 also show the performance prediction results in different domains using the *interaction* method of building block composition. The performance prediction results from the *interaction* method show that the MTDP based model evaluation can distinguish better performing strategies from worse performing ones in different problem domains.



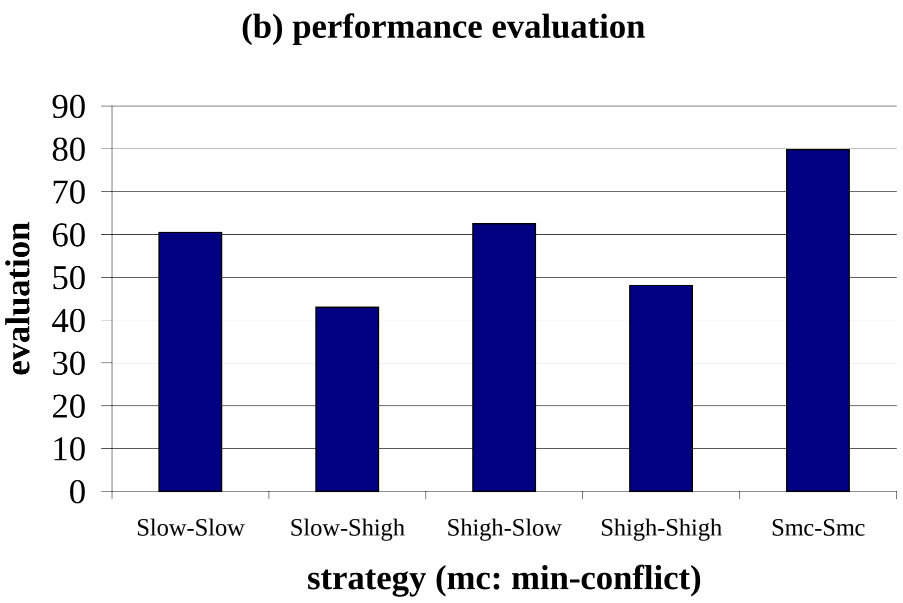
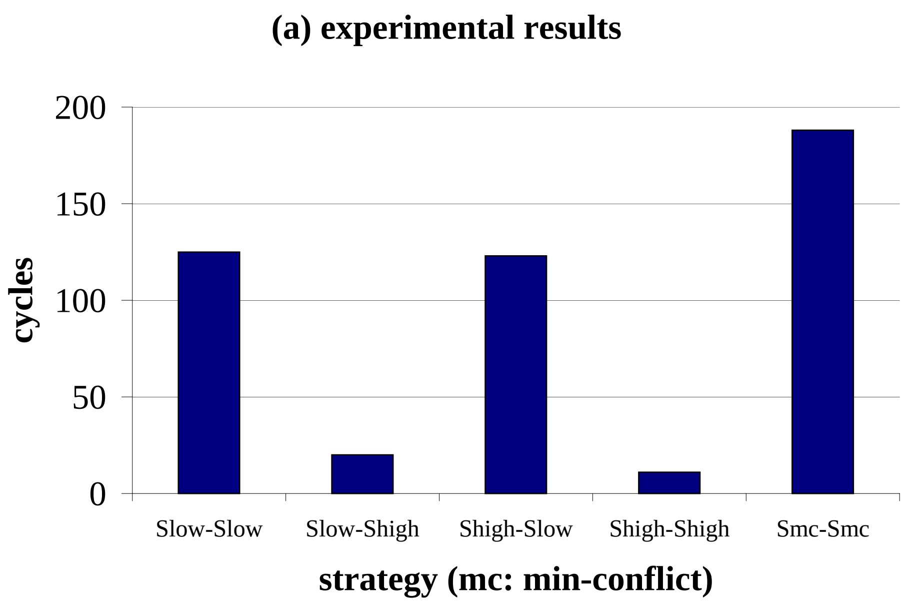


Figure 5.12: Performance prediction in low flexibility setting with 60% locally constrained agents using *interaction* method



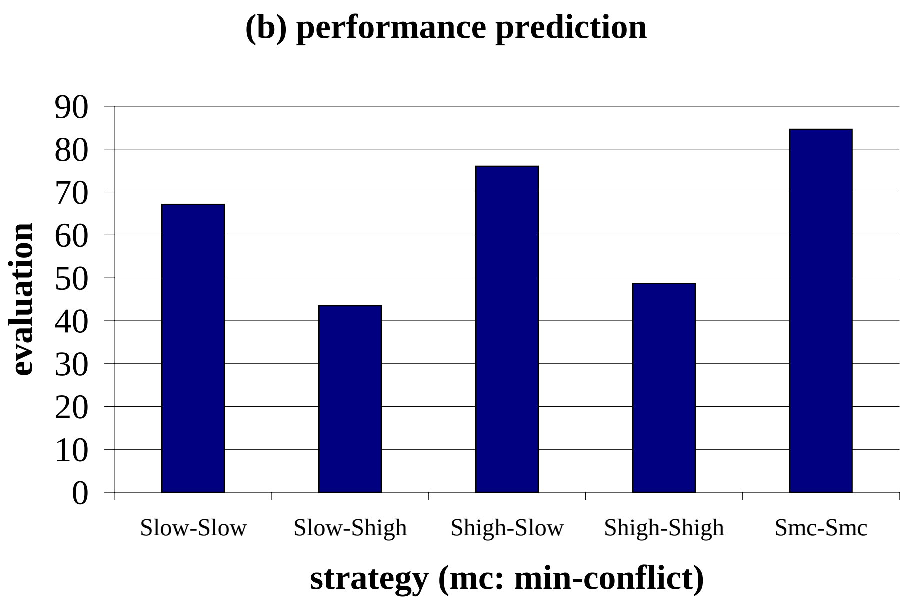


Figure 5.13: Performance prediction in low flexibility setting with 90% locally constrained agents using interaction method

	low flexibility setting with 60% locally constrained agents	low flexibility setting with 90% locally constrained agents	high flexibility setting with 60% locally constrained agents	high flexibility setting with 90% locally constrained agents
Single block	0.57	0.80	0.74	0.97
Simple sum	0.69	0.96	0.40	0.89
Weighted sum	0.70	0.96	0.38	0.89
Interaction	0.88	0.97	0.79	0.96

Table 5.2: Correlation between experimental results and performance evaluation for four cooperative strategies and min-conflict strategy

	low flexibility setting with 60% locally constrained agents	low flexibility setting with 90% locally constrained agents	high flexibility setting with 60% locally constrained agents	high flexibility setting with 90% locally constrained agents
Single block	0.67	0.67	0.63	0.68
Simple sum	0.91	0.97	-0.26	0.68
Weighted sum	0.92	0.97	-0.32	0.68
Interaction	0.98	0.95	0.80	0.74

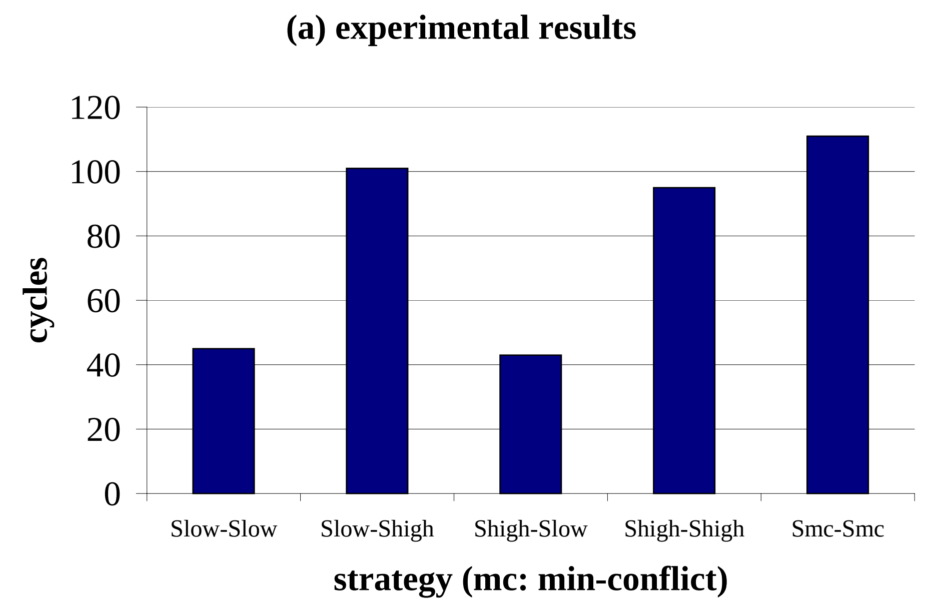
Table 5.3: Correlation between experimental results and performance evaluation for only cooperative strategies without min-conflict strategy

While the *interaction* method shows matching results, a question remains to be answered for whether the other composition methods of building blocks can provide the same power of performance prediction as the *interaction* method does. Figure 5.16, 5.17, 5.18, and 5.19 show the policy evaluation results from four different methods (*single block*, *simple sum*, *weighted sum*, and *interaction*) in the four different cases. Table 5.2 shows the correlation between the experimental results and the performance evaluation for the five strategies to compare (four cooperative strategies and the min-conflict strategy). Table 5.3 shows the correlation between the experimental results and the performance evaluation for four *locally cooperative* strategies (to check how well each method can distinguish the *locally cooperative* strategies.

Note that the correlation measures how much positive relationship exists between the number of cycles (experimental results) and the evaluation values from the MTDP based model. While high correlation does not directly indicate that strategies' ranking in the experimental results will match to the ranking in evaluation values, it implies that the evaluation value can clearly distinguish better performing strategies from worse performing strategies. For instance, with a high correlation value, two strategies that perform better than the others but show a very little difference between them may not have a big difference in their evaluation values. However, the two better performing strategies will be distinguished from other worse performing strategies in terms of evaluation value from the MTDP-based model.

Table 5.2 shows that *interaction* method has higher correlation values than other methods. While the *single block* method shows a little better correlation in one case, methods other than the *interaction* method are far from matching to the real experimental results. Note that both *simple sum* and *weighted sum* methods show worse correlations in two cases. Furthermore, according to table 5.3 (that shows the correlation between *locally cooperative* strategies), both *simple sum* and *weighted sum* methods show negative correlation in one case, which means that they are not reliable in predicting the strategy performance. Only the composition method that considers the interaction between building blocks shows the best prediction results.

These results illustrate that the MTDP model can be used to predict the right strategy to apply in a given situation (possibly with less computation overhead). That is, given a new domain, agents can analyze different strategies with the simple MTDP model and select the right strategy for the new domain without running a significant number



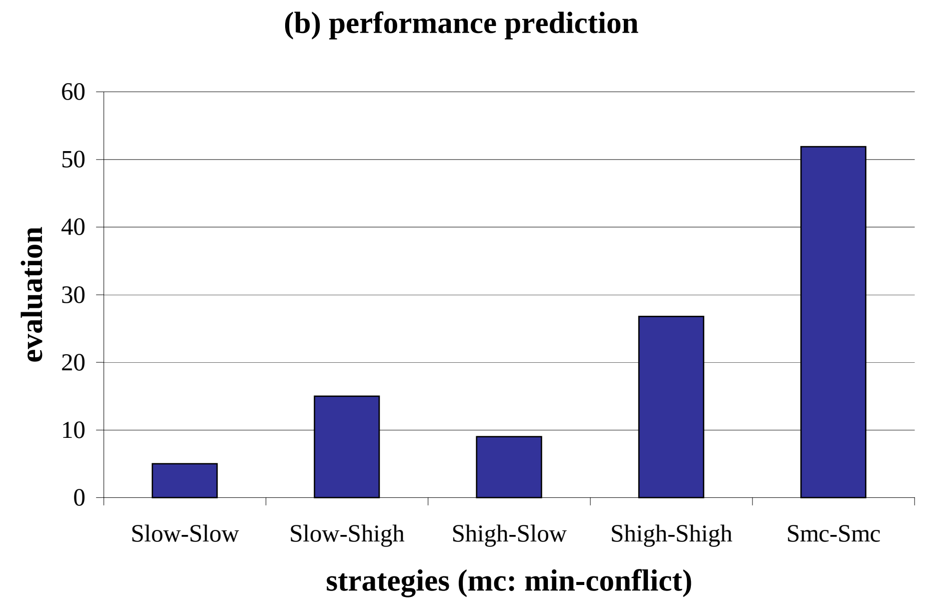
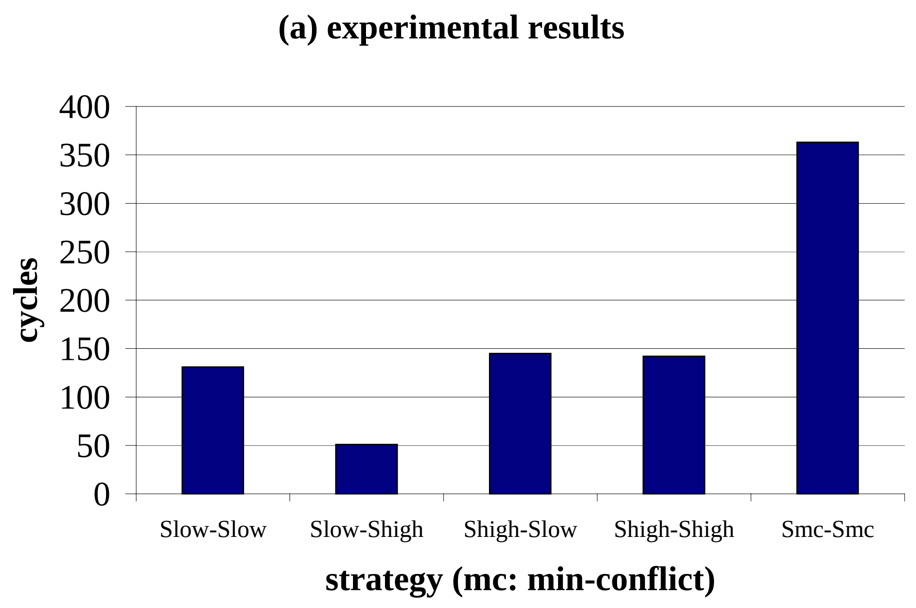


Figure 5.14: Performance prediction in high flexibility setting with 60% locally constrained agents



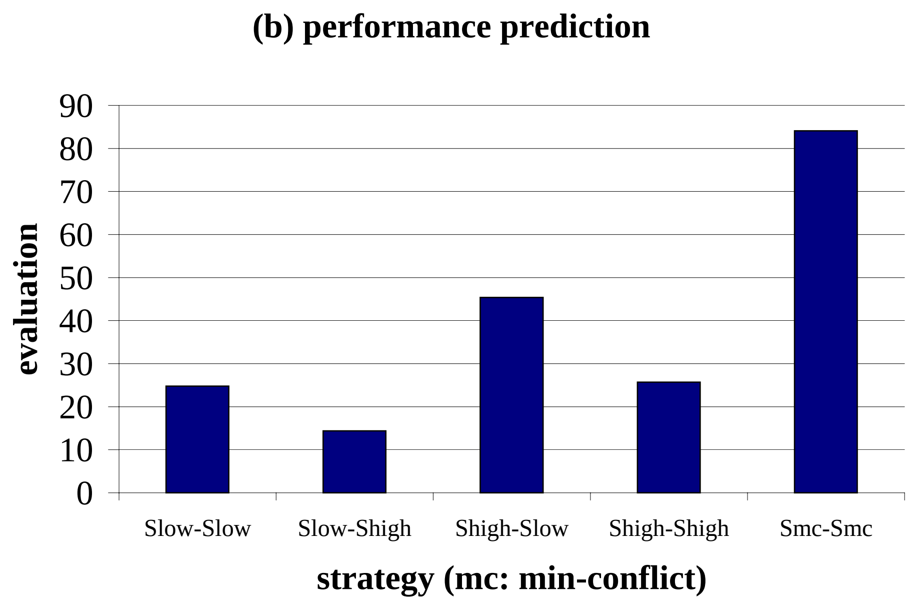


Figure 5.15: Performance prediction in high flexibility setting with 90% locally constrained agents

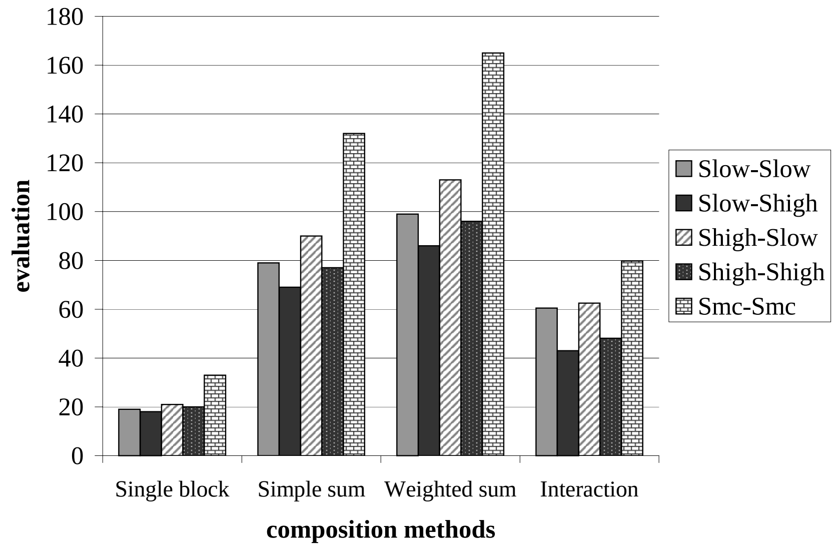


Figure 5.16: Composition method comparison in low flexibility setting with 60% locally constrained agents (mc: min-conflict)

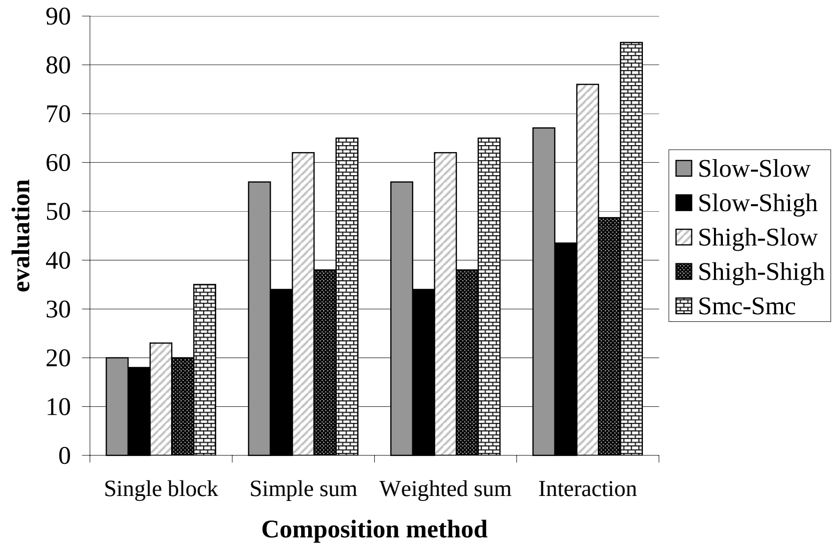


Figure 5.17: Composition method comparison in low flexibility setting with 90% locally constrained agents (mc: min-conflict)

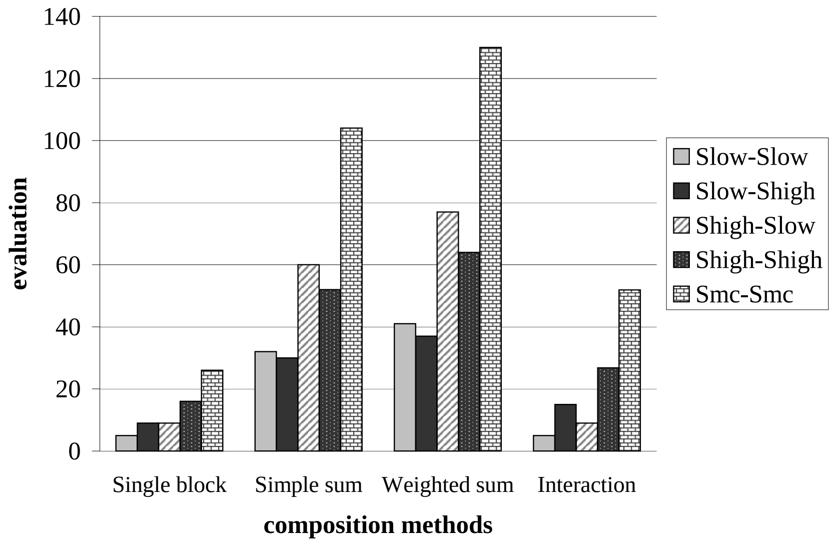


Figure 5.18: Composition method comparison in high flexibility setting with 60% locally constrained agents (mc: min-conflict)

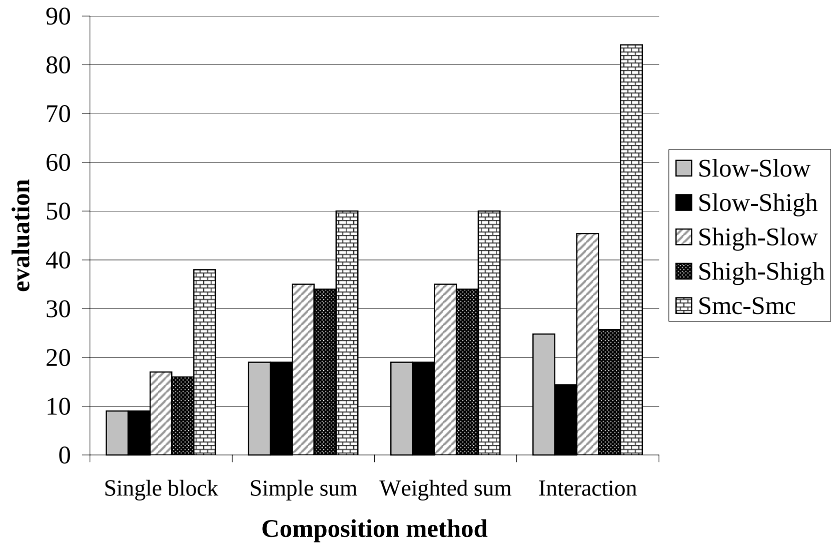


Figure 5.19: Composition method comparison in high flexibility setting with 90% locally constrained agents (mc: min-conflict)

of problem instances for each strategy. Furthermore, this approach will enable agents to flexibly adapt their strategies to changing circumstances. More generally, this result indicates a promising direction for performance analysis in DCSP, and potentially other multiagent systems.

5.2.3 Efficiency of Building Block Based Approach

	low flexibility setting with 60% locally constrained agents	low flexibility setting with 90% locally constrained agents	high flexibility setting with 60% locally constrained agents	high flexibility setting with 90% locally constrained agents
Building block	731.0	736.0	659.0	650.0
Running test cases	13676.4	6072.0	6285.6	9552.0

Table 5.4: Comparison of runtime (sec) to select the best strategy

In this section, we show the efficiency of building block based approach by comparing the runtime of the following two methods in a given domain:

- *Interaction* method
- Running 30 sample problem instances [2]

Note that this is not to compare the possible benefit (e.g., time saving) from using the *interaction* method in a real system after performance analysis since we do not assume future usage of a selected strategy (e.g., how long or how many times the system will operate using the selected strategy). Instead, we focus on the computation overhead to select the best strategy using the two approaches. In Table 5.4, the second row shows the

[2]30 is the minimum number of instances for statistical significance in general.

runtime (sec) of the *interaction* method to find the best strategy among the five strategies (selected for expository purpose in the previous section) in four different settings: $S_{low} - S_{low}$, $S_{low} - S_{high}$, $S_{high} - S_{low}$, $S_{high} - S_{high}$, and $S_{min-conflict} - S_{min-conflict}$.

	low flexibility setting with 60% locally constrained agents	low flexibility setting with 90% locally constrained agents	high flexibility setting with 60% locally constrained agents	high flexibility setting with 90% locally constrained agents
Time (sec)	682.0	736.0	659.0	650.0
Correlation	0.80	0.60	0.26	0.39

Table 5.5: Time and accuracy results with small-scale test runs

The third row in Table 5.4 shows the runtime of running each strategy on 30 sample problem instances. Runtime comparison clearly shows that the building block based approach significantly saves the computation overhead in selecting the best strategy: there exists more than 10 fold difference which may grow as the number of agents or strategies to consider increases. Here, a question to be answered is as follows:

- Can small-scale test runs provide accurate performance prediction with significantly less time?

In Table 5.5, the second row shows the computation overhead of small-scale test runs (running 30 sample problem instances where the number of agents is 64), and the third row shows the correlation between the original experimental results and the (experimental) results in the small-scale test runs.

While the computation overhead in Table 5.5 is similar with the overhead of building block approach (interaction method) in Table 5.4, the correlation which indicates the accuracy of performance prediction is very low overall. problem settings. That is, small-scale test runs cannot provide accurate performance prediction. Therefore, the distributed POMDP based model with building block approach can be of practical use in predicting the best performing strategy given a domain.

Chapter 6

Related Work

In this chapter, we will present related works on multiagent conflict resolution, performance estimation, and decomposition for MDP and POMDP, and discuss how they are related to my thesis work.

6.1 Multiagent Conflict Resolution Techniques

Researchers have investigated various techniques for multiagent conflict resolution such as constraint satisfaction [Liu and Sycara, 1993; Sadeh and Fox, 1996; Sathi and Fox, 1989], argumentation-based negotiation [Chu-Carroll and Carberry, 1995; Sycara, 1988], and auction [Walsh and Wellman, 1998; Hunsberger and Grosz, 2000]. In this section, before presenting previous works in those areas, we first introduce DCSP research in general.

6.1.1 General DCSP Techniques

Yokoo, Durfee, Ishida, and Kuwabara pioneered the area of asynchronous DCSP algorithms by developing *asynchronous backtracking* (ABT) algorithm which is a distributed, asynchronous version of a backtracking algorithms [Yokoo *et al.*, 1992]. In the ABT algorithm, the priority of variables/agents is determined and each agent communicates its tentative value assignment to neighboring agents. An agent changes its assignment if its current value assignment is not consistent with the assignment of higher priority agents. If there exists no consistent value, an agent generates a new constraint (called *nogood*), and communicates it to a higher priority agent: thus, the higher priority agent changes its value.

Yokoo developed the *asynchronous weak-commitment* algorithm in which the priority order of agents changes dynamically during search [Yokoo, 1995]. When an agent cannot find a consistent value with higher priority agents, the agent comes to have the highest priority. As a result, when agents make non-solution value assignments, they can revise them without exhaustive search. Later, Yokoo and Hirayama extended the asynchronous weak-commitment algorithm to deal with multiple variables per agent [Yokoo and Hirayama, 1998].

Inspired by a breakout algorithm [Morris, 1993] in centralized CSP, Yokoo and Hirayama developed the *distributed breakout algorithm* which is not a complete algorithm but improves the performance over the *asynchronous backtracking* algorithm in critically difficult graph coloring problems [Yokoo and Hirayama, 1996]. In the algorithm, each agent tries to minimize the number of constraint violation by exchanging its current value and the possible amount of its improvement (decrease of the violation number) among neighboring agents. Only the agent that can maximally reduce the violation number is given the right to change its value. Recently, Zhang and Wittenburg re-investigated this distributed break algorithm [Zhang and Wittenburg, 2002]. They showed that, for an acyclic graph, the distributed breakout algorithm is complete. However, in the worst case, the algorithm never terminates for cyclic graphs. They proposed stochastic approach to overcome the problem in cyclic graphs.

In the above approaches, when a deadend is detected, a new constraint is created and communicated among agents. A different approach which does not require such new constraint creation is the *distributed backtracking* algorithm [Hamadi *et al.*, 1998].

In the distributed backtracking algorithm, agents maintain partial ordering by which each agent knows which agents are its children or parents in the ordering. When no value satisfies the constraints with children agents, an agent notifies its lowest parent that tries another value. Bessière et al. developed a dynamic version of this distributed backtracking algorithm where the ordering of agents change dynamically [Bessière *et al.*, 2001].

The DCSP algorithms described above assume the case where access to variables is restricted only to agents that own the variable, but the information about constraints are shared among relevant agents. In contrast, Silaghi et al. developed a DCSP algorithm (called *asynchronous aggregation* search) for the case where constraints are private but variables can be manipulated by any agent [Silaghi *et al.*, 2000]. In the asynchronous aggregation search, the basic search mechanism is based on asynchronous backtracking, but agents exchange information about aggregated valuations for variable combination (instead of individual values) by using Cartesian product representation.

Our *locally cooperative* strategies (described in Chapter 3) can be seen as *value ordering heuristics* in DCSP. By incorporating extra local constraint communication, the *locally cooperative* strategies could increase the performance of the above DCSP algorithms other than the distributed backtracking algorithm which have fixed value ordering and the asynchronous aggregation search where local constraints are private.

6.1.2 DCSP Agent Ordering

While our *locally cooperative* strategies are akin to value ordering heuristics in DCSP, in the DCSP literature, value ordering heuristics have been largely uninvestigated. Instead, some researchers have developed techniques for agent ordering as follows.

Collin et al. first investigated the feasibility of DCSP and showed that ordering of agents is necessary to converge in DCSP [Collin *et al.*, 1991]. In the asynchronous commitment algorithm by Yokoo [Yokoo, 1995], agents ordering changes dynamically based on the principle of “constrained agent first”: where an agent cannot find a consistent value, it becomes the highest agent.

Armstrong and Durfee investigated various heuristics to determine efficient agent ordering [Armstrong and Durfee, 1997]. They showed that the best agent ordering is based on the cumulative difficulty of finding assignments to agents’ local variables, and less costly heuristics sometimes perform better than others depending on constraint structure. They also applied genetic algorithm to learn weights for combined heuristics, which led to the best performance.

Silaghi et al. proposed a general heuristic for agent ordering in asynchronous distributed search [Silaghi *et al.*, 2001]: agents propose orders by communicating ordering messages and resolve inconsistency in the orders with history (number of past conflicts for variables). Since our investigation of the *locally cooperative* strategies is based on the reasoning about how to vary the degree of cooperativeness to different agents. Agent ordering is a key base for such variation in cooperativeness. Combining efficient value ordering strategies and agent ordering techniques will further enhance the performance of DCSP based conflict resolution.

6.1.3 CSP-based Conflict Resolution Approach

Sathi and Fox applied constraint satisfaction approach to resolve conflicts in resource allocation [Sathi and Fox, 1989]. Agents’ objectives are represented as constraints together with their associated utilities. When a conflict occurs, agents modify their individual solutions or constraints until a joint compromise is reached. Three operators (composition, reconfiguration and relaxation) are used to modify the current solutions or constraints. However, in the modification, these operators do not take other agents' local constraints into account as our *locally cooperative* strategies do.

Sadeh and Fox also applied constraint satisfaction techniques to resolve conflicts in job shop scheduling problems which are known to be NP-complete [Sadeh and Fox, 1996]. To reduce the size of search space for the problem, they investigated the method to select the order in which variables are instantiated and values are tried for each variable. Their value ordering exploits domain knowledge in computing the probability that a solution will have resource contention in the future. While our strategies defined in Section 3 can be seen as local information-based value ordering heuristics in DCSP (Distributed Constraint Satisfaction Problems), the heuristic by Sadeh and Fox requires global information and has been applied only to centralized problem solving.

Liu and Sycara developed DCSP-based conflict resolution mechanism for distributed scheduling problems [Liu and Sycara, 1993; 1996]. In the mechanism, each resource is assigned to a resource agent which enforces capacity constraints on resources, and each job is assigned to an order agent which maintains temporal constraints on jobs. They developed value ordering heuristics to decide which activities to move for conflict resolution: the order agent selects a new activity based on the cost of replacing activities and the resource agent shifts activities to minimize the amount of time shift. While their approach improved performance over the min-conflict heuristic in small scale scheduling tasks, agent's decision is based on local information and the coordination between agents (e.g., decision for which agent to revise its activities) is managed by a central agent which has only contention ratio information of the agents, which does not guarantee completeness. In contrast, our cooperative strategies are focused on improving the performance of conflict resolution convergence to a complete solution in large scale applications.

Darr et al. investigated interval-based DCSP in concurrent engineering design problems. However, they focused on maintaining consistency which reduces design space by eliminating non-solution space. While they also investigated decomposibility technique for backtracking-free search, their investigation focuses on maintaining consistency by communicating legal intervals for agents' domains [D'Ambrosio *et al.*, 1996; Darr and Birmingham, 1994]. In contrast, we focus on the strategy to exploit the communicated information. Furthermore, their experimentation is limited only into a small-scale elevator configuration problem.

6.1.4 Learning Coordination Strategies

Significant works in multiagent learning are focused on learning to select the right coordination strategy [Matos *et al.*, 1998; Prasad and Lesser, 1997; Excelente-Toledo and Jennings, 2002]. Prasad and Lesser developed a learning algorithm by which agents learn to choose appropriate, situation specific strategies from a set of available strategies [Prasad and Lesser, 1997]. In the learning phase, agents store meta-level information about a given situation and the performance for each strategy. After learning is complete, agents select a strategy based on the similarity between the current situation and past cases. The strategies in their work focus on the type of information exchange (e.g., whether an agent local information is communicated to all agents or previously committed agents) rather than the reasoning about which action or plan to choose.

Excelente-Toledo and Jennings investigated reinforcement learning techniques for multiagent coordination [Excelente-Toledo and Jennings, 2002]. They applied Q-learning, a reinforcement technique, since it does not require an exact model of the environment. In their approach, agents learn which coordination mechanism to choose given a certain task which requires cooperation from others. Here, the coordination mechanisms are in an abstract form (a tuple of cost and time) without explicit description of which actions are involved in a strategy.

While the above learning approaches were applied to cooperative agents, Matos, Sierra, and Jennings used a genetic algorithm approach for agents to learn which strategy to use in a competitive environment [Matos *et al.*, 1998]. Here, a strategy is a function to determine how agents change their values to negotiate in a market. Agents learn which strategy (function) to use in a given situation (e.g., a specific type of seller or buyer).

The goal of learning strategies is related to our goal of choosing the right strategy. However, one key difference is that the learning work focuses on enabling each agent to select a strategy. Instead, our focus is on a complementary goal of trying to predict the overall performance of the entire multiagent system given a set of strategies. Furthermore, the learning approach has a scalability issue. In a large scale system, learning may not converge (within an appropriate time) in particular when agents learn simultaneously.

6.1.5 Other Conflict Resolution Approaches

Sycara developed an argumentation-based negotiation system called PERSUADER, a program to resolve labor disputes [Sycara, 1988]. Negotiation is performed to resolve goal conflicts through compromise which involves cycles of proposal generation and goal relaxation. Case-based reasoning and preference analysis is used to generate arguments. Agents' utilities associated with each of the conflicting goals are used to rank possible compromises without knowing private information of the other agents.

While Sycara's negotiation system is for non-cooperative domains, Chu-Carroll and Carberry also investigated dialogue based negotiation system in a collaborative environment [Chu-Carroll and Carberry, 1995]. In their system, agents reason about which information to share to resolve goal conflicts under uncertainty. While these negotiation systems were successfully applied for conflict resolution in both cooperative and non-cooperative systems, such negotiation systems have been used for small size applications (often with two agents) and have not been applied to large scale systems.

Jung and Tambe also developed an argumentation-based negotiation system, called CONSA, for multiagent conflict resolution in cooperative environments [Tambe and Jung, 1999]. CONSA (COllaborative Negotiation System based on Argumentation) is focused on exploiting the benefits of argumentation in a team setting where collaborative agents do not need to hide information. In their approach, conflict resolution is cast as a team problem, so that the teamwork knowledge developed in the multiagent literature [Tambe, 1997] can be exploited. They provided novel argumentation strategies in a team setting such as improving the quality of teammate's arguments by providing more concrete justifications.

Walsh and Wellman proposed market-based approach for decentralized scheduling which used bidding mechanism to allocate resources [Walsh and Wellman, 1998]. Self-interested bidders and a central auctioneer continue the cycle of revising bids until no agent chooses to revise its bid. While market-based approach is inherently for competitive environment with self-interested agents, it has been also applied for cooperative agents.

Hunsberger and Grosz applied combinatorial auction mechanism for collaborative planning [Hunsberger and Grosz, 2000]. Multiple agents bid for a group activity considering their commitments to their own tasks and new opportunities for their team.

Given conflicting multiple bids, a central agent computes the best combination of the bids. These market-based approaches require minimal communication overhead (only bid communication) and agents' decision making is only based on each agent's local information because of privacy. In contrast, our work is based on local communication of agents' private constraints which increases speedup in conflict resolution convergence without significantly increasing the communication overhead.

Chia, Neiman, and Lesser investigated agent coordination issues in a distributed dynamic scheduling system for airport ground service [Chia *et al.*, 1998]. They investigated the lack of coordination among agents that can exist even when agents have complete global information about resources. It was shown that, when agents are not able to model the global resource state and possible activities of other agents, scheduling cannot be done effectively. They classified two types of actions, poaching and distraction, that result from such lack of coordination, and provided remedial methods to address the problems.

Vaughan, Støy, Sukhatme, and Mataric investigated conflict resolution among multiagents on embedded systems [Vaughan *et al.*, 2000]. They demonstrated the utility of an aggressive competition to reduce interference and increase efficiency in a multi-robot system. Given a resource conflict, each robot compares its own level of aggression with that displayed by others; if smaller, it yield the resource. In a realistic multi-robot transport task, this aggressive signaling strategy showed better performance than previous anti-interference techniques.

6.2 Performance Estimation in Constraint Satisfaction Problems

Researchers have investigated performance prediction for different heuristics in centralized constraint satisfaction problems. There are two approaches in the investigation. First, they estimate the size of search space for a given heuristic [Allen and Minton, 1996; Knuth, 1975; Lobjois and Lemaitre, 1998]. Second, probabilities for decreasing or increasing the number of conflicts are computed to compare different heuristics [Minton *et al.*, 1992; Musick and Russell, 1992].

6.2.1 Cost Estimation by Estimating Search Space Size

Allen and Minton provided a sampling method to select the best heuristic by estimating the cost of different heuristics in a given domain [Allen and Minton, 1996]. By running different heuristics for a very short period of time, they estimated the number of constraint checks for a node and the branching factor of a search tree. Using the estimation, they computed the total expected number of constraint checks which corresponds to the time until a solution is found in centralized constraint satisfaction problems.

Lobjois and Lemaitre developed a performance prediction method for branch and bound algorithm [Lobjois and Lemaitre, 1998]. Their method is based on Knuth's sampling method which estimates the efficiency of a backtracking program on a particular instance by iteratively generating random paths in a search tree [Knuth, 1975]. In applying Knuth's method for branch and bound algorithm, they fixed an upperbound which changes in a real search, leading to inaccuracy in some cases. While these approaches have been applied to centralized CSP, they are not feasible in DCSP since multiple agents asynchronously explore the search space for a given problem instance. [1]

6.2.2 Probabilistic Analysis of Heuristics

Minton et al. provided a mathematical analysis for when min-conflict heuristic performs better than random value selection [Minton *et al.*, 1992]. They showed that, given a certain number of constraint violation (d), the probability of decreasing (or increasing) the violation number by min-conflict heuristic, $P_{d \to d-1}$ ($P_{d \to d+1}$), depends on the violation number d. The ratio $P_{d \to d-1}/P_{d \to d+1}$ for a heuristic provides a useful indication of the heuristic performance: the greater, the better performance. Their analysis indicates that min-conflict heuristic performs well in sparsely-connected graphs.

Musick and Russell developed a method to predict the total amount of time taken by the min-conflict heuristic [Musick and Russell, 1992]. They built a Markov Chain where state is represented with the number of constraint violation. Transition (whether violation number increases or decreases by one) probabilities between states were based on the mathematical analysis by Minton et al [Minton *et al.*, 1992]. Using the Markov chain, they computed the expected number of steps (value changes) given a certain number of current violations.

These investigations have contributed to theoretical foundation for heuristic performance analysis in centralized CSP. However, there has been no theoretical investigation for such performance analysis in DCSP literature, and the applicability of the above approaches is limited due to their simplifying assumptions such as (i) each agent has the same domain and external constraints, (ii) there is only one unique solution, and (iii)

[1]Note that DCSP is not dividing a search space into subproblems. Thus, we cannot apply such an approach that divides a problem and adds the estimated cost for each subproblem.

only one variable changes its value at one time. In particular, the third assumption does not apply to DCSP since, in most advanced DCSP algorithms including AWC, agents change their variables' values simultaneously.

6.3 MDP and POMDP Decomposition

In the MDP and POMDP literature, researchers have investigated decomposition techniques to deal with a large state space which is exponential in the number of state variables [Dean and Lin, 1995; Hauskrecht *et al.*, 1998; Parr, 1998; Pineau *et al.*, 2001]. Basic approach is to divide a given problem (with a large state space) into subproblems, solve the subproblems independently, and merge the solution of each subproblem. Often, the merged solutions is an approximation to a real optimal solution.

Dean and Lin investigated methods that decompose global planning problems into a number of local problems and combine the local solution to generate a global solution [Dean and Lin, 1995]. They presented algorithms that decompose planning problems into smaller problems given an arbitrary partition of the state space. One issue is how to compute the value of a state which may transit into another subproblem for certain actions (for a local problem, the state transition and cost in the other local problems are not known). They applied an iterative approximation technique to converge to an optimal solution.

Parr presented two approaches to decomposing and solving large Markov decision problems, a complete decoupling method and a partial decoupling method [Parr, 1998]. In the complete decoupling method, a policy cache is created for each region. For every possible outer space state, there exists a near optimal policy (called ϵ-optimal policy) in the cache, and the combination of those near optimal policy is guaranteed to produce a global policy with a certain error bound. In the partial coupling method, to remedy the large computational overhead of the complete coupling method, it starts from imperfect policies that may not be ϵ-optimal, but the policy cache is updated to reduce errors when the Bellman error exceeds for each state.

While the above approaches exploit state decomposition, Hauskrecht et al. proposed a hierarchical model with macro-actions (action abstraction) [Hauskrecht *et al.*, 1998]. Inspired by the idea of planning with macro-actions by Precup, Sutton, and Singh [Precup *et al.*, 1998], they defined an abstract MDP model by treating macro-actions as local policies in certain regions and restricting states only to boundary states of regions in the original space. Optimal macro-actions are computed with this abstract MDP model. They also showed the reusability of macro-actions in different domains which have some commonalities (e.g., maze with similar structure).

Compared with hierarchical decomposition works on MDP, there has been little work on extending it to POMDP. Recently, Pineau et al. provided a hierarchical approach to reduce problem space in POMDP by partitioning action space into specialized groups of related actions [Pineau *et al.*, 2001]. In their application domain (mobile robotic assistant), the space of actions is naturally decomposed into a hierarchy of actions depending on the actions' applicability in different situations: for instance, navigation (e.g., move, turn, etc.) and interaction (e.g., speak, display, etc.) Based on the action hierarchy, a POMDP is transformed into a collection of smaller POMDPs: each sub-POMDP independently learns a policy with only relevant actions.

While the above works are related to our building block based approach in composition subproblem solutions, we are really interested in applying these composition techniques for performance modeling, not finding an optimized policy. For instance, our techniques are heavily influenced by the need to capture the long-tailed phenomena in conflict resolution.

Chapter 7

Conclusion and Future Work

The research in this dissertation is motivated by practical concerns in collaborative multiagent systems. Distributed, collaborative agents are promising to play an important role in large-scale multiagent applications such as distributed sensors, distributed spacecraft, and robot teams for surveillance. In such large scale systems, conflicts are inevitable even among collaborative agents over shared resources, joint plans, or task assignments since no single agent can have complete knowledge of the world. An agent's action/plans/resource choice based on local information conflicts with other agents' choices which are also based on limited information of their own.

Since agents need to resolve conflicts in a distributed manner without global knowledge (because of drawbacks in centralized conflict resolution such as computation/communication bottle neck, fault tolerance, etc.), conflict resolution is a fundamental challenge in multiagent systems. As shown in Chapter 2, fast conflict resolution is required in many multiagent applications since the delay in conflict resolution can lead to significant system performance degradation. In this chapter, we will review our major techniques for fast conflict resolution and conclusions presented in previous chapters of this thesis. Remaining questions for future work and possible approaches to deal with them will follow.

7.1 Locally Cooperative Strategies

Given a large group of agents, selecting the right action, plan, or resource to resolve conflicts, i.e., selecting the right conflict resolution strategy, can have a significant impact on their performance (e.g., speed of conflict resolution convergence to a complete solution). Our fist contribution in this thesis is the development of novel conflict resolution strategies based on the notion of *local cooperativeness*. The new strategies (defined in Chapter 3) significantly increase the speedup of conflict resolution convergence, compared with an original state-of-the-art DCSP conflict resolution algorithm. As noted in Chapter 1, they are based on the following hypothesis:

- "*While most constraint-based or market-based approaches for conflict resolution have focused on minimal communication, in a cooperative team setting, some extra local communication can significantly increase the speedup of conflict resolution convergence.*"

Communication of local information enables agents to take into account the situation of neighboring agents in selecting their choice of actions, plans, or resources. *Local cooperativeness* is measured by how much flexibility (number of action/plan/resource choices) is given to neighboring agents. We formalized different *locally cooperative* strategies by varying the degree of cooperativeness to neighboring agents and implemented the strategies in DCSP framework.

For the investigation of strategy performance, we develop a run-time model since an existing popular performance measurement does not take into account the overhead from extra communication in our approach. The issue of run-time measurement is a key topic in DCSP research community and the run-time model presented in this thesis is the first approach that takes into account the overhead from increased message size and number with extra communication. Systematic experimental investigation shows the following results: (from the tests in 351 problem settings with 208,845 experimental runs)

- Strategies based on extra communication can indeed speed up the conflict resolution convergence in a significant range of problem settings, particularly for harder problems with more than an order of magnitude difference in run-time (compared with the best published DCSP technique, AWC [Yokoo and Hirayama, 1998]).
    - The overhead from increased message processing and communication with extra communication is not overwhelming, and do not cause serious degradation in conflict resolution convergence.
- It is not always the case that more information communication leads to improved performance (in some cases, strategies with extra communication perform worse than a strategy that does not communicate local information).
- The strategy that provides maximum flexibility to all neighboring agents is not always the best.
- No single strategy dominates the other strategies across all domains (a strategy that performs the best in one domain setting can be an order of magnitude slower than another strategy in a different setting).

Furthermore, based on systematic experimentation, we provide categorization of domains where high speedups can be achieved by the strategies with extra communication. Given a domain, such categorization can provide a guidance for whether to apply our approach or not.

The research in this thesis has implications in many different settings. Coordination or conflict resolution strategies based on resource flexibility have been investigated in multi-linked negotiation [Zhang and Lesser, 2002], distributed multiagent planning [Decker and Lesser, 1995; Prasad and Lesser, 1997; von Martial, 1991], and centralized planning/scheduling [Nareyek, 2001; Sadeh and Fox, 1996; Smith, 1994]. While flexibility in such application does not distinguish agents, the conflict resolution strategies with different degrees of cooperativeness (presented in Chapter 3) can be applied to the above applications for faster convergence. Furthermore, the flexibility definition is general enough to deal with more complex types of constraints (e.g., valued constraint) so that flexibility can make a distinction between values that satisfy different constraints.

7.2 Distributed POMDP based Performance Models for Conflict Resolution Strategies

To gain the maximum speedup in conflict resolution convergence, we need to predict the right strategy to apply in a given domain. While performance models for conflict resolution strategies could aid in selecting the right strategy, such models remain largely uninvestigated in the multiagent literature. Researchers often rely on running sample problems in a testbed to compare the performance of their strategies, which is very costly in large scale systems.

As the second contribution of this thesis, we provide innovative performance modeling approach by developing formal models based on a distributed POMDP (partially observable Markov decision process) framework such as MTDP [Pynadath and Tambe, 2002]. A conflict resolution strategy is mapped onto an MTDP policy, and strategies are compared by evaluating their corresponding policies. In applying the distributed

POMDP-based model to analyze strategy performance in large-scale systems, the most significant obstacle is the *scalability*: the search space explodes combinatorially as the number of states increases.

To address scale-up issues, we used small-scale models (called *building blocks*) that represent the local interaction among a small group of agents. We investigated several ways to combine the building blocks, and compared with performance prediction results with real experimental results. Combining building blocks by taking into account their interaction - via agents acting as docking points between building blocks - perform the best in predicting the performance of conflict resolution strategies: better performing strategies can be distinguished from worse performing strategies with statistical significance. Compared with previous works on MDP and POMDP decomposition, the novelty of our building block decomposition technique is three fold:

- Rather than exploiting geometric clusters within a domain (suitable in a single-agent POMDP), building-blocks exploit the multiagent nature of the domain — the decomposition is based on clustering agents with close interactions with each other.

- We are focused on evaluating policies rather than searching for optimal policies.

- Observability conditions within a building block can be exploited to further reduce the complexity of policy evaluation.

Our approach in modeling the performance of conflict resolution strategies points the way to new tools for strategy analysis and performance modeling in multiagent systems in general.

7.3 Future Work

Given the complex and dynamic nature of multiagent systems, there exists a variety of challenging problems to explore in multiagent conflict resolution as follows:

1. **Uncertainty**: In many applications (particularly in harsh or hostile environments), the information that agents receive has uncertainty since communication or sensing is not always perfect. For instances, in the distributed sensor domain (described in Chapter 2), sensor may have a noise in detecting a target or the communication between sensors can be interrupted by enemy's jamming signals. In the CSP and DCSP literature, probabilistic approach has been used to deal with this uncertainty. Recently, Scerri et al. applied probabilistic representation for tasks (targets to track) in distributed sensor domain [Scerri *et al.*, 2003]: agents order tasks based on their probabilities and assign resources to most probable tasks. We are interested in how to make agents reason about flexibility to neighboring agents given uncertainty in neighbors' local constraints as well as its own ones.

2. **Extension in performance modeling**: In the DCSP literature, the impact on performance by communication has been studied by several researchers. Mammen and Lesser investigated how the timing of transmitting subproblem solutions between agents has an impact on performance solving efficiency [Mammen and Lesser, 1998]. Fernàndez et al. also modeled the impact of communication delay on the performance of DCSP algorithms [Fernàndez *et al.*, 2003]. In our initial performance modeling (Chapter 5), agents instantly communicate their partial solution and agents' observation is assumed to be perfect. The decision of when to communicate can be modeled by adding communication action as was done

by Pynadath and Tambe in their helicopter transportation domain [Pynadath and Tambe, 2002]. Communication delay or message loss can be also modeled with observation functions. With additional actions and observation functions, we plan to model conflict resolution strategies in more complex environments in our future study.

3. **Optimization**: In our experiments in Chapter 3, we did not assume over-constrained problems where all the constraints cannot be satisfied at the same time. However, in many multiagent applications, there could be over-constrained situations. For instance, in the distributed sensor domain (described in Chapter 2), a team of sensor agents may not be able to track all the targets if the number of targets exceeds its tracking capacity. In the DCSP literature, some researchers have investigated constraint optimization problem as follows: Yokoo and Hirayama have studied maximizing the number of satisfied constraints [Hirayama and Yokoo, 1997] and satisfying maximal higher order constraints given a hierarchy of constraints [Hirayama and Yokoo, 2000]. Recently, Modi et al. developed a complete asynchronous algorithm which is based on Yokoo's Asynchronous Backtracking algorithm [Modi *et al.*, 2003]. Lemaitre and Verfaillie also presented an incomplete method for valued DCSP [Lemaitre and Verfaillie, 1997].

   While all the previous works on distributed constraint optimization above are based on static agent ordering and value ordering, efficient value ordering could enhance the speed of finding an optimal solution. For instance, in exploiting branch and bound approach, finding an upper bound early can significantly save search space. The communication of local constraints could enable agents to reason about probable upper bound without exhaustive search for all the domain values. Furthermore, in the case of constraint relaxation, communicating local constraints with their valuations may help to decide which local constraints to relax.

4. **Dynamism**: Constraints in multiagent applications can vary with time. For instance, in the spacecraft domain (described in Chapter 2), constraints may dynamically change as scientific requests change or unexpected events occur. If a device fails or shows less performance than expected (e.g., battery power exhausted earlier than expected), it may add additional constraints that are not initially taken into account. On the contrary, some requirements may be deleted as some investigations are completed. Then, given the dynamism of constraints, plans or task assignments computed before may not be a solution any more. In contrast, previously abandoned plans or assignments can be considered as a new solution in the case of constraint relaxation. This dynamism can be formalized within the framework of dynamic CSP and DCSP [Mittal and Falkenhainer, 1990; Verfaillie and Schiex, 1994; Modi *et al.*, 2001]. We plan to investigate the performance of different *local cooperativeness* based strategies in dynamically changing environments where both external and local constraints are added or deleted unexpectedly.

In summary, this thesis addresses multiagent conflict resolution problem which is a fundamental challenge in large scale multiagent applications. We provided novel strategies to enhance the performance of conflict resolution convergence and developed a distributed POMDP based models to select the right strategy in a given domain. This thesis also points out several open issues for future investigation.

Reference List

[Allen and Minton, 1996] J. Allen and S. Minton. Selecting the right heuristic algorithm: Runtime performance predictors. In *Proceedings of Canadian Artificial Intelligence Conference*, 1996.

[Angelopoulos and Panetta, 1998] V. Angelopoulos and P.V. Panetta, editors. *Science Closure and Enabling Technologies for Constellation Class Missions*. University of California, Berkeley, 1998.

[Armstrong and Durfee, 1997] A. Armstrong and E.H. Durfee. Dynamic prioritization of complex agents in distributed constraint satisfaction problems. In *Proceedings of International Joint Conference on Artificial Intelligence*, 1997.

[Barrett, 1999] A. Barrett. Autonomy architectures for a constellation of spacecraft. In *Proceedings of International Symposium on Artificial Intelligence Robotics and Automation in Space*, 1999.

[Bernstein *et al.*, 2000] D. S. Bernstein, S. Zilberstein, and N. Immerman. The complexity of decentralized control of mdps. In *Proceedings of the International Conference on Uncertainty in Artificial Intelligence*, 2000.

[Bessière *et al.*, 2001] C. Bessière, A. Maestro, and P. Meseguer. Distributed dynamic backtracking. In *Proceedings of the International Workshop on Distributed Constraint Reasoning*, 2001.

[Boutilier, 1999] Craig Boutilier. Sequential optimality and coordination in multiagent systems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1999.

[Calder *et al.*, 1993] R. B. Calder, J. E. Smith, A. J. Courtemanche, J. M. F. Mar, and A. Z. Ceranowicz. Modsaf behavior simulation and control. In *Proceedings of the Conference on Computer Generated Forces and Behavioral Representation*, 1993.

[Cheeseman *et al.*, 1991] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1991.

[Chia *et al.*, 1998] M. Chia, D. Neiman, and V. Lesser. Poaching and distraction in asynchronous agent activities. In *Proceedings of the International Conference on Multi-Agent Systems*, 1998.

[Chu-Carroll and Carberry, 1995] J. Chu-Carroll and S. Carberry. Generating information-sharing subdialogues in expert-user consultation. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1995.

[Collin *et al.*, 1991] Z. Collin, R. Dechter, and S. Katz. On the feasibility of distributed coonstraint satisfaction. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1991.

[D'Ambrosio *et al.*, 1996] J. G. D'Ambrosio, T. Darr, and W. P. Birmingham. Hierarchical concurrent engineering in a multiagent framework. *Cocurrent Engineering: Research and Applications Journal*, 4, 1996.

[Darr and Birmingham, 1994] T. P. Darr and W. P. Birmingham. An attribute-space an attribute-space representation and algorithm for concurrent engineering. Technical Report CSE-TR-221-94, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, October 1994.

[Dean and Lin, 1995] T. Dean and S. Lin. Decomposition techniques for planning in stochastic domains. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1995.

[Decker and Lesser, 1995] K. Decker and V. Lesser. Designing a family of coordination algorithms. In *Proceedings of the International Conference on Multi-Agent Systems*, 1995.

[Durfee, 1991] E. H. Durfee. The distributed artificial intelligence melting pot. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6), 1991.

[Durfee, 2001] E. H. Durfee. Distributed problem solving and planning. *Lecture Notes in Computer Science*, 2086, 2001.

[Elson, 2003] J. Elson. *Time Synchronization in Wireless Sensor Networks*. PhD thesis, School of Engineering, University of California, Los Angeles, 2003.

[Excelente-Toledo and Jennings, 2002] C. Excelente-Toledo and N. Jennings. Learning to select a coordination mechanism. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 2002.

[Fernàndez *et al.*, 2003] C. Fernàndez, R. Béjar, B. Krishnarnachari, and C. Gomes. Communication and computation in distributed csp algorithms. In *Proceedings of International Conference on Principles and Practice of Constraint Programming*, 2003.

[Ganesan *et al.*, 2002] D. Ganesan, D. Estrin, and J. Heidemann. Dimensions: Why do we need a new data handling architecture for sensor networks. In *Proceedings of the First Workshop on Hot Topics in Networks (Hotnets-I)*, 2002.

[Gomes *et al.*, 2000] C. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tailed phenomenon in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24, 2000.

[Gordon *et al.*, 1999] D. Gordon, W. Spears, O. Sokolsky, and I. Lee. Distributed spatial control, global monitoring and steering of mobile physical agents. In *Proceedings of IEEE International Conference on Information, Intelligence, and Systems*, 1999.

[Grosz, 1996] B. Grosz. Collaborating systems. *AI magazine*, 17(2), 1996.

[Hamadi *et al.*, 1998] Youssef Hamadi, Christian Bessière, and Joël Quinqueton. Backtracking in distributed constraint networks. In *Proceedings of the European Conference on Artificial Intelligence*, 1998.

[Haralick and Elliot, 1980] R. M. Haralick and G. L. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[Hauskrecht *et al.*, 1998] M. Hauskrecht, N. Meuleau, L. P. Kaelbling, T. Dean, and C. Boutilier. Hierarchical solution of Markov decision processes using macro-actions. In *Proceedings of the International Conference on Uncertainty in Artificial Intelligence*, 1998.

[Hill *et al.*, 2000] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, 2000.

[Hirayama and Yokoo, 1997] K. Hirayama and M. Yokoo. Distributed partial constraint satisfaction problem. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, 1997.

[Hirayama and Yokoo, 2000] K. Hirayama and M. Yokoo. An approach to over-constrained distributed constraint satisfaction problems: Distributed hierarchical constraint satisfaction. In *Proceedings of the International Conference on Multi-Agent Systems*, July 2000.

[Hirayama *et al.*, 2000] K. Hirayama, M. Yokoo, and K. Sycara. The phase transition in distributed constraint satisfaction problems: Fist results. In *Proceedings of the International Workshop on Distributed Constraint Reasoning*, 2000.

[Hogg and Williams, 1994] T. Hogg and C. P. Williams. The hardest constraint problems: A double phase transition. *Artificial Intelligence*, 69:359–377, 1994.

[Hunsberger and Grosz, 2000] L. Hunsberger and B. Grosz. A combinatorial auction for collaborative planning. In *Proceedings of the International Conference on Multi-Agent Systems*, 2000.

[Kitano *et al.*, 1999] H. Kitano, S. Tadokoro, I. Noda, H. Matsubara, T. Takahashi, A. Shinjou, and S. Shimada. Robocup rescue: search and rescue in large-scale disaster as a domain for autonomous agents research. In *Proceedings of the IEEE International Conference on System, Man, and Cybernetics*, 1999.

[Knuth, 1975] D. E. Knuth. Estimating the efficiency of backtrack programs. *Mathematics of Computation*, 29, 1975.

[Lemaitre and Verfaillie, 1997] M. Lemaitre and G. Verfaillie. A incomplete method for solving distributed valued constraint satisfaction problems. In *Proceedings of the AAAI Workshop on Constraints and Agents*, 1997.

[Lesser *et al.*, 2003] V. Lesser, C. Ortiz, and M. Tambe, editors. *Distributed Sensor Networks: a Multiagent Perspective*. Kluwer Publishing, 2003.

[Lesser, 1999] V. Lesser. Cooperative multiagent systems: A personal view of the state of the art. *IEEE Transactions on Knowledge and Data Engineering*, 11(1), January 1999.

[Littman *et al.*, 1995] M. Littman, T. Dean, and L. P. Kaelbling. On the complexity of solving markov decision problems. In *Proceedings of the International Conference on Uncertainty in Artificial Intelligence*, 1995.

[Liu and Sycara, 1993] J. S. Liu and K. Sycara. Distributed constraint satisfaction through constraint partition and coordinated reaction. In *Proceedings of the International Workshop on Distributed Artificial Intelligence*, 1993.

[Liu and Sycara, 1996] J. Liu and K. Sycara. Multiagent coordination in tightly coupled task scheduling. In *Proceedings of the International Conference on Multi-Agent Systems*, 1996.

[Liu *et al.*, 1998] T. H. Liu, A. Goel, C. E. Martin, and K. S. Barber. Classification and representation of conflict in multi-agents systems. Technical Report TR98-UT-LIPS-AGENTS-01, The Laboratory for Intelligent Processes and Systems, University of Texas at Austin, 1998.

[Lobjois and Lemaitre, 1998] L. Lobjois and M. Lemaitre. Branch and bound algorithm selection by performance prediction. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, 1998.

[Mammen and Lesser, 1998] Dorothy L. Mammen and Victor R. Lesser. Problem structure and subproblem sharing in multi-agent systems. In *Proc. of the Intl. Conf. on Multi-Agent Systems*, 1998.

[Matos *et al.*, 1998] N. Matos, C. Sierra, and N.R. Jennings. Determining successful negotiation strategies: An evolutionary approach. In *Proceedings of the International Conference on Multi-Agent Systems*, 1998.

[Mettler and Milman, 1996] E. Mettler and M. Milman. Space interferometer constellation: Formation maneuvering and control architecture. In *Proceedings of the SPIE International Symposium on Optical Science, Engineering, and Instrumentation*, 1996.

[Minton *et al.*, 1990] S. Minton, M. D. Johnston, A. Philips, and P. Laird. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proceedings of the National Conference on Artificial Intelligence*, 1990.

[Minton *et al.*, 1992] S. Minton, M. Johnston, A. Philips, and P. Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992.

[Mittal and Falkenhainer, 1990] S. Mittal and B. Falkenhainer. Dynamic constraint satisfaction problems. In *Proceedings of the National Conference on Artificial Intelligence*, 1990.

[Modi *et al.*, 2001] P. Modi, H. Jung, M. Tambe, W. Shen, and S. Kulkarni. A dynamic distributed constraint satisfaction approach to resource allocation. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, 2001.

[Modi *et al.*, 2003] P. Modi, W. Shen, M. Tambe, and M. Yokoo. An asynchronous complete method for distributed constraint optimization. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 2003.

[Modi, 2003] P. Modi. *Distributed Constraint Optimization and its Application*. PhD thesis, School of Engineering, University of Southern California, 2003.

[Morris, 1993] P. Morris. The breakout method for escaping from local minima. In *Proceedings of the National Conference on Artificial Intelligence*, 1993.

[Müller and Dieng, 2000] H. J. Müller and R. Dieng, editors. *Computational Conflicts - Conflict Modeling for Distributed Artificial Intelligent Systems*. Springer Verlag Publishers, 2000.

[Musick and Russell, 1992] R. Musick and S. Russell. How long it will take? In *Proceedings of the National Conference on Artificial Intelligence*, 1992.

[Nareyek, 2001] A. Nareyek. Local-search heuristics for generative planning. In *Proceedings of the Workshop on AI in Planning, Scheduling, Configuration and Design*, 2001.

[Neiman *et al.*, 1994] D. Neiman, David Hildum, V. R. Lesser, and T. W. Sandholm. Exploiting meta-level information in a distributed scheduling system. In *Proceedings of the National Conference on Artificial Intelligence*, 1994.

[Parker, 1993] L. E. Parker. Designing control laws for cooperative agent teams. In *Proceedings of the IEEE Robotics and Automation Conference*, 1993.

[Parr, 1998] R. Parr. Flexibile decomposition algorithms for weakly coupled Markov decision problems. In *Proceedings of the International Conference on Uncertainty in Artificial Intelligence*, 1998.

[Peshkin *et al.*, 2000] L. Peshkin, K. Kim, N. Meuleau, and L. P. Kaelbling. Learning to cooperate via policy search. In *Proceedings of the International Conference on Uncertainty in Artificial Intelligence*, 2000.

[Pineau *et al.*, 2001] J. Pineau, N. Roy, and S. Thrun. A hierarchical approach to POMDP planning and execution. In *Proceedings of the ICML Workshop on Hierarchy and Memory in Reinforcement Learning*, 2001.

[Pottie and Kaiser, 2000] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5), 2000.

[Prasad and Lesser, 1997] N. Prasad and V. Lesser. The use of meta-level information in learning situation-specific coordination. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1997.

[Precup *et al.*, 1998] D. Precup, R. S. Sutton, and S. Singh. Theoretical results on reinforcement learning with temporally abstract behaviors. In *Proceedings of the Advances in Neural Information Processing Systems*, 1998.

[Puterman, 1994] M. L. Puterman. *Markov Decision Processes*. John Wiley & Sons, 1994.

[Pynadath and Tambe, 2002] D. Pynadath and M. Tambe. The communicative multiagent team decision problem: analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, 2002.

[Rana, 2000] O. Rana. Performance management of mobile agent systems. In *Proceedings of the International Conference on Autonomous Agents*, 2000.

[Rickel and Johnson, 1997] J. Rickel and W. L. Johnson. Pedagogical agents for immersive training environments. In *Proceedings of the International Conference on Autonomous Agents*, 1997.

[Sadeh and Fox, 1996] N. M. Sadeh and M. S. Fox. Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem. *Artificial Intelligence*, 86(1), 1996.

[Sathi and Fox, 1989] A. Sathi and M. S. Fox. Constraint-directed negotiation of resource reallocations. In L. Gasser and M. N. Huhns, editors, *Distributed Artificial Intelligence (Vol. II)*. Kaufmann, San Mateo, CA, 1989.

[Scerri *et al.*, 2001] P. Scerri, D. Pynadath, and M. Tambe. Adjustable autonomy in real-world multi-agent environments. In *Proceedings of the International Conference on Autonomous Agents*, 2001.

[Scerri *et al.*, 2003] P. Scerri, P. Modi, W. Shen, and M. Tambe. Are multiagent algorithms relevant for robotics applications? a case study of distributed constraint algorithms. In *ACM Symposium on Applied Computing*, 2003.

[Silaghi *et al.*,] M. Silaghi, D. Sam-Haroud, and B. Faltings. Consistency maintenance for abt.

[Silaghi *et al.*, 2000] M. Silaghi, D. Sam-Haroud, and B. Faltings. Asynchronous search with aggregations. In *Proceedings of the National Conference on Artificial Intelligence*, 2000.

[Silaghi *et al.*, 2001] M. Silaghi, D. Sam-Haroud, and B. Faltings. Polynomial-space and complete multiply asynchronous search with abstraction. In *Proceedings of the International Workshop on Distributed Constraint Reasoning*, 2001.

[Smith, 1994] S. Smith. Opis: A methodology and architecture for reactive scheduling. In *Intelligent Scheduling*. Morgan Kaufman, San Francisco, 1994.

[Sycara, 1988] K. Sycara. Resolving goal conflicts via negotiation. In *Proceedings of the National Conference on Artificial Intelligence*, 1988.

[Tambe and Jung, 1999] M. Tambe and H. Jung. The benefits of arguing in a team. *AI Magazine*, 20(4), 1999.

[Tambe *et al.*, 2000] M. Tambe, D. Pynadath, N. Chauvat, A. Das, and G. Kaminka. Adaptive agent integration architectures for heterogeneous team members. In *Proceedings of the International Conference on Multi-Agent Systems*, 2000.

[Tambe, 1997] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

[Tessier *et al.*, 2000] C. Tessier, L. Chaudron, and H. J. Muller, editors. *Conflicting Agents*. Kluwer Academic Publishers, 2000.

[Vaughan *et al.*, 2000] R. T. Vaughan, K. Stoy, G. S. Sukhatme, and M. J. Mataric. Go ahead, make my day: Robot conflict resolution by aggressive competition. In *Proceedings of the International Conference on Simulation of Adaptive Behavior*, 2000.

[Verfaillie and Schiex, 1994] G. Verfaillie and T. Schiex. Solution reuse in dynamic constraint satisfaction problems. In *Proceedings of the National Conference on Artificial Intelligence*, 1994.

[von Martial, 1991] F. von Martial. Coordinating plans of autonomous agents. *Lecture Notes in Computer Science*, 610, 1991.

[Walsh and Wellman, 1998] W. Walsh and M. Wellman. A market protocol for decentralized task allocation. In *Proceedings of the International Conference on Multi-Agent Systems*, 1998.

[Xuan and Lesser, 2002] P. Xuan and V. Lesser. Multi-agent policies: from centralized ones to decentralized ones. In *Proceedings of the International Conference on Autonomous Agents*, 2002.

[Xuan *et al.*, 2001] P. Xuan, V. Lesser, and S. Zilberstein. Communication decisions in multi-agent cooperation: Model and experiments. In *Proceedings of the International Conference on Autonomous Agents*, 2001.

[Yokoo and Hirayama, 1996] M. Yokoo and K. Hirayama. Distributed breakout algorithm for solving distributed constraint satisfaction problems. In *Proceedings of the International Conference on Multi-Agent Systems*, 1996.

[Yokoo and Hirayama, 1998] M. Yokoo and K. Hirayama. Distributed constraint satisfaction algorithm for complex local problems. In *Proceedings of the International Conference on Multi-Agent Systems*, 1998.

[Yokoo *et al.*, 1992] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. Distributed constraint satisfaction for formalizing distributed problem solving. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*, 1992.

[Yokoo *et al.*, 1998] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):673–685, 1998.

[Yokoo, 1995] M. Yokoo. Asynchronous weak-commitment search for solving distributed constraint satisfaction problems. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, 1995.

[Yokoo, 2000] M. Yokoo. *Distributed Constraint Satisfaction: Foundations of Cooperation in Multi-Agent Systems*. Springer-Verlag, 2000.

[Zhang and Lesser, 2002] X. Zhang and V. Lesser. Multi-linked negotiation in multi-agent system. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 2002.

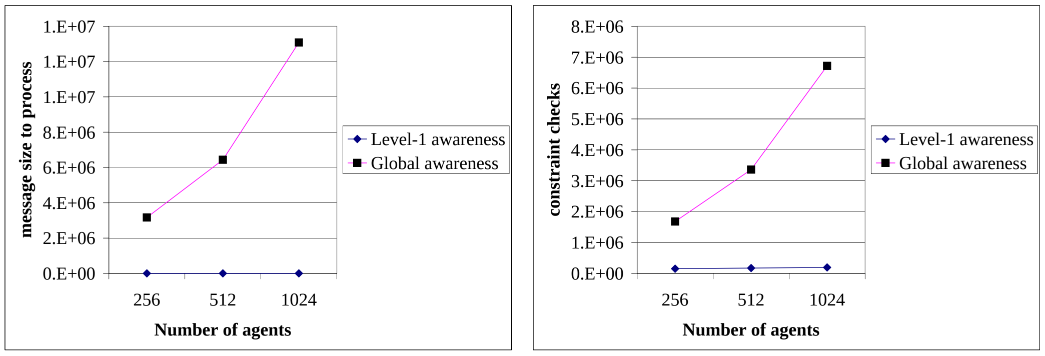
[Zhang and Wittenburg, 2002] W. Zhang and L. Wittenburg. Distributed breakout revisited. In *Proceedings of the National Conference on Artificial Intelligence*, 2002.

[Zhang *et al.*, 2003] W. Zhang, G. Wang, Z. Xing, and L. Wittenburg. Low overhead algorithms for distributed constraint problems in sensor networks. In V. Lesser, C. Ortiz, and M. Tambe, editors, *Distributed Sensor Networks: a Multiagent Perspective*. Kluwer Publishing, 2003.

Appendix A

Comparison between the DCSP Approach in This Thesis and Globally Aware DCSP Approach

In the wide spectrum of different levels of non-local awareness, this thesis focuses on a fixed level of non-local awareness where each agent collects local information only from its neighboring agents. For brevity, this level of non-local awareness (which is the level of awareness applied in this thesis) is referred as level-1 awareness. The advantage of level-1 awareness can be shown in comparing it with one extreme case (in the spectrum of different non-local awareness levels) where each agent has global awareness (the local information of all the agents in a given multiagent system).



(a) Message size to process (b) Constraint checks

Figure A.1: Message size and constraint checks per agent: Assume that each agent (whose domain size is 40) has four neighbors (in 2D grid topology).

Figure A.1 shows message sizes to process and the number of constraints checks per agent for level-1 awareness and global awareness until a solution is found, varying the number of agents. [1] As shown in Figure A.1-(a) and (b), the message sizes and the constraint checks per agent exponentially grow in the case of global awareness while level-1 awareness does not cause such an exponential increase.

In domains of interest where agents have limited computing power (and a bounded energy source), maintaining global awareness has disadvantages since extra hardware (e.g., an additional memory device) may be required to process a large number of messages and the huge number of constraint checks can cause high energy consumption.

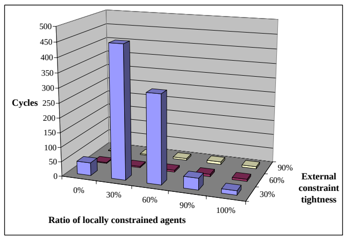
[1] For level-1 awareness, the message sizes and constraint checks are based on real experimentation (test runs on 35 problem instance with the strategies with level-1 awareness proposed in this thesis). In contrast, for global awareness, the message sizes are computed by $N \times D + (2N - A - B) \times D^2$ where N (= A $\times$ B) is the number of agents and D is a domain size: $N \times D$ is the size of domain information for all agents and $(2N - A - B) \times D^2$ is the size of (binary) constraint information. Constraint checks for global awareness are estimated by the time complexity of an arc-consistency technique. Since an advanced arc-consistency technique has time complexity of $O(ND)^2$, the number of constraint checks for the global awareness case is assumed to be $c \times O(ND)^2$ where $c = 1$.

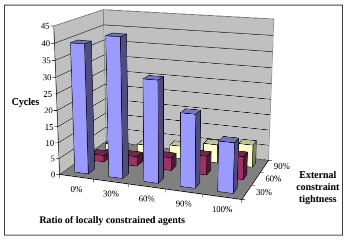
Appendix B

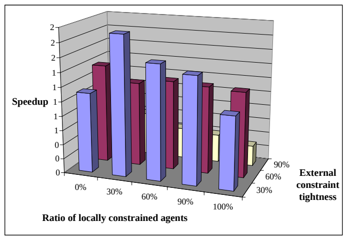
Experimental Results

This appendix presents the results with all the 16 strategies in the experimental setting described in Chapter 3 (Section 3.2.3). Only six strategies were shown for expository purpose in Chapter 3. Each graph is followed by the data table for the graph. Note that the conclusions presented in Chapter 3 (shown below) were based on the results with all 16 strategies (not just on the results from six selected strategies):

B.1 Detailed Peformance Results

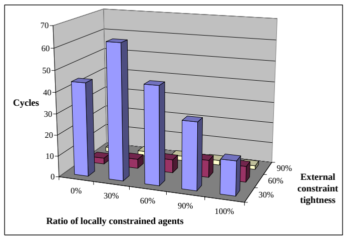


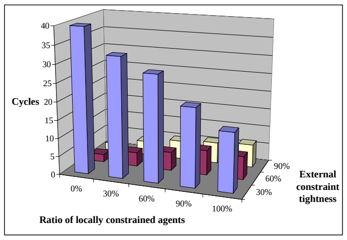


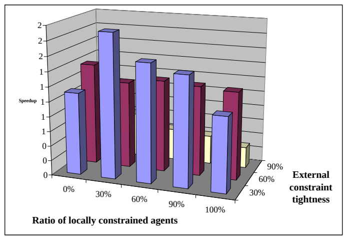


(a) AWC strategy (b) Best LCDCSP strategy (c) Speedup

Figure B.1: Hexagonal topology; local constraint compatibility 25%; domain size 10

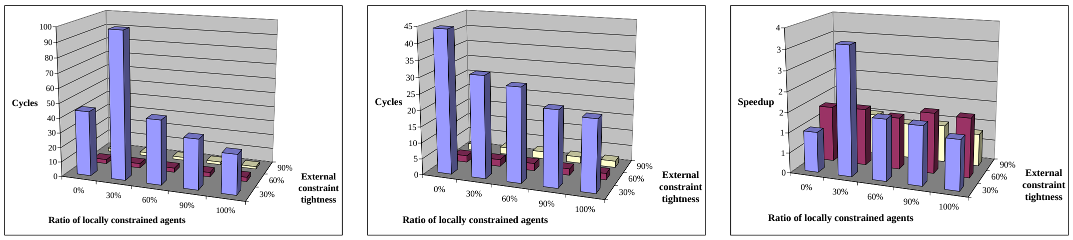






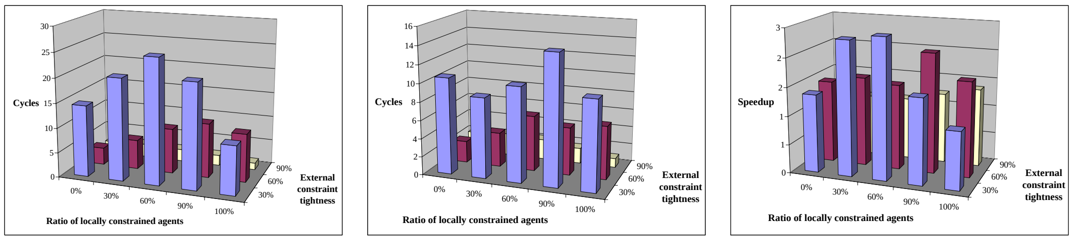
(a) AWC strategy (b) Best LCDCSP strategy (c) Speedup

Figure B.2: Hexagonal topology; local constraint compatibility 50%; domain size 10



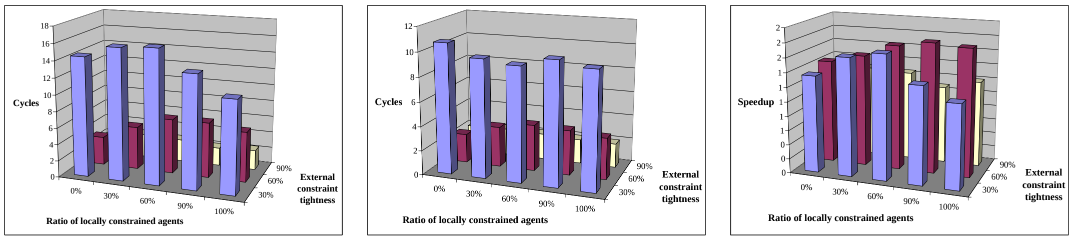
(a) AWC strategy (b) Best LCDCSP strategy (c) Speedup

Figure B.3: Hexagonal topology; local constraint compatibility 75%; domain size 10



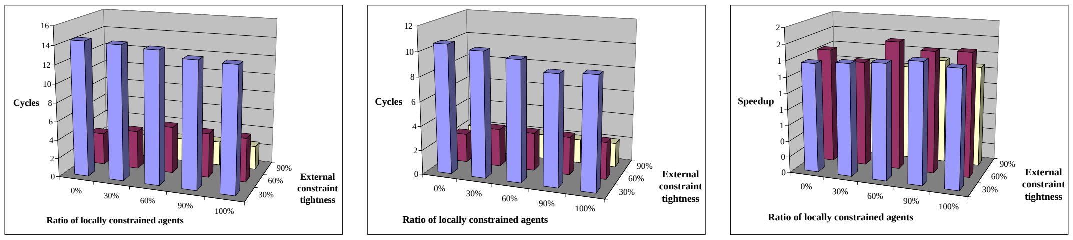
(a) AWC strategy (b) Best LCDCSP strategy (c) Speedup

Figure B.4: Hexagonal topology; local constraint compatibility 25%; domain size 40



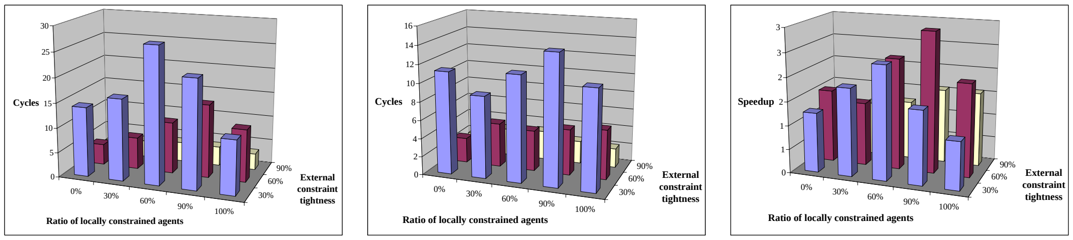
(a) AWC strategy (b) Best LCDCSP strategy (c) Speedup

Figure B.5: Hexagonal topology; local constraint compatibility 50%; domain size 40



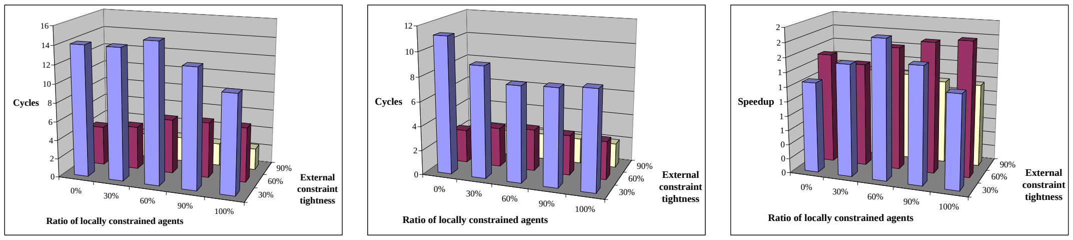
(a) AWC strategy (b) Best LCDCSP strategy (c) Speedup

Figure B.6: Hexagonal topology; local constraint compatibility 75%; domain size 40



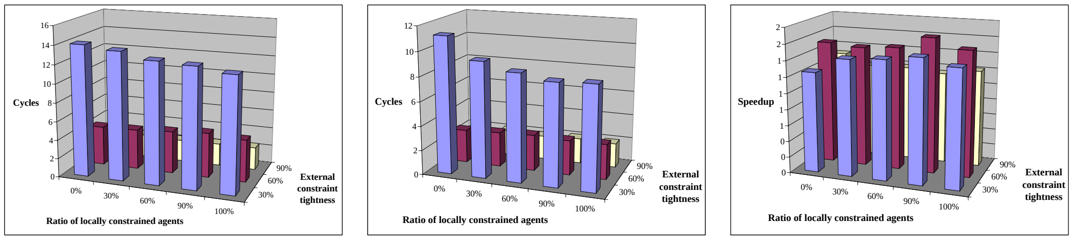
(a) AWC strategy (b) Best LCDCSP strategy (c) Speedup

Figure B.7: Hexagonal topology; local constraint compatibility 25%; domain size 80



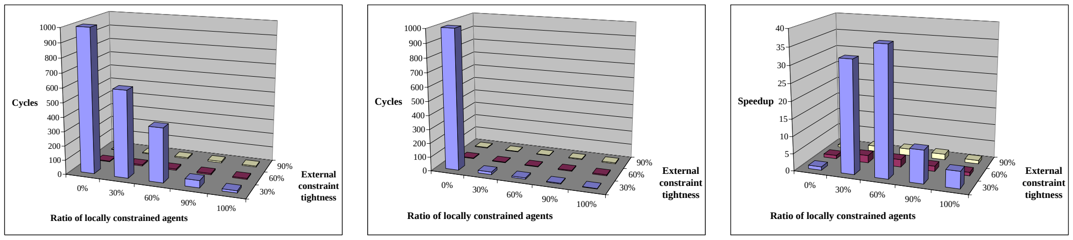
(a) AWC strategy (b) Best LCDCSP strategy (c) Speedup

Figure B.8: Hexagonal topology; local constraint compatibility 50%; domain size 80



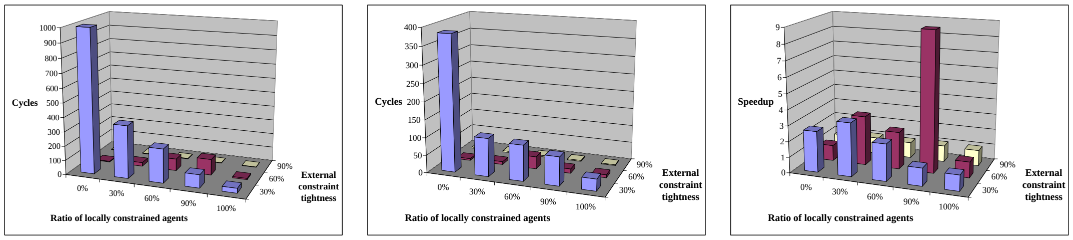
(a) AWC strategy (b) Best LCDCSP strategy (c) Speedup

Figure B.9: Hexagonal topology; local constraint compatibility 75%; domain size 80



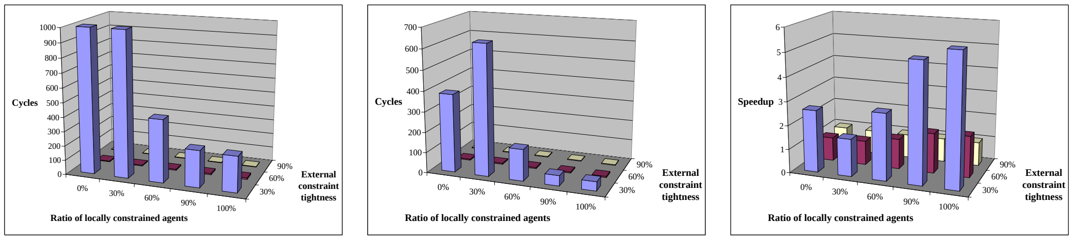
(a) AWC strategy (b) Best LCDCSP strategy (c) Speedup

Figure B.10: Grid topology; local constraint compatibility 25%; domain size 10



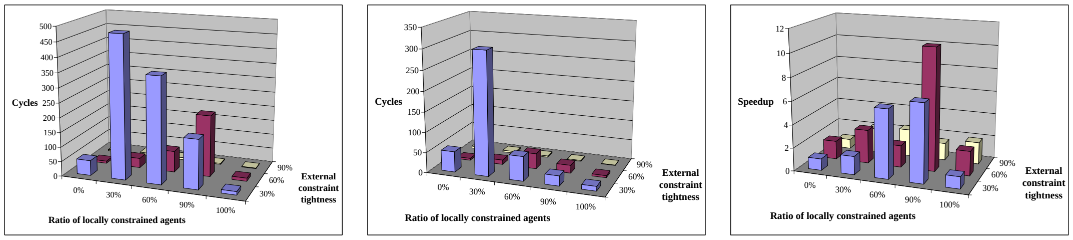
(a) AWC strategy (b) Best LCDCSP strategy (c) Speedup

Figure B.11: Grid topology; local constraint compatibility 50%; domain size 10



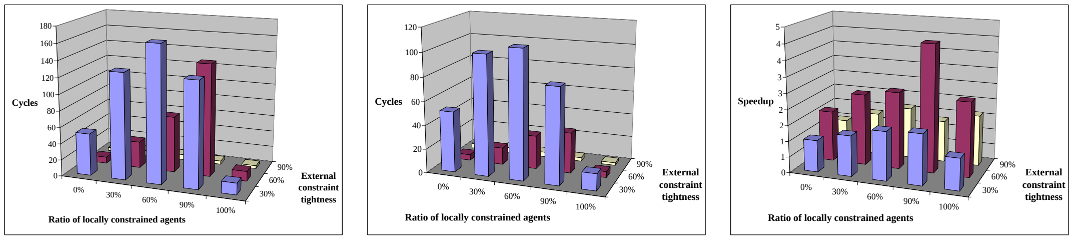
(a) AWC strategy (b) Best LCDCSP strategy (c) Speedup

Figure B.12: Grid topology; local constraint compatibility 75%; domain size 10



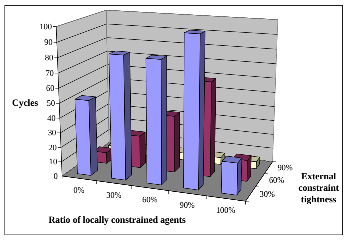
(a) AWC strategy (b) Best LCDCSP strategy (c) Speedup

Figure B.13: Grid topology; local constraint compatibility 25%; domain size 40

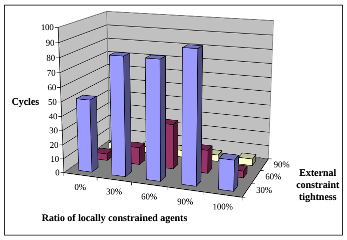


(a) AWC strategy (b) Best LCDCSP strategy (c) Speedup

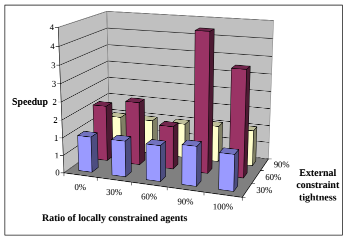
Figure B.14: Grid topology; local constraint compatibility 50%; domain size 40



(a) AWC strategy

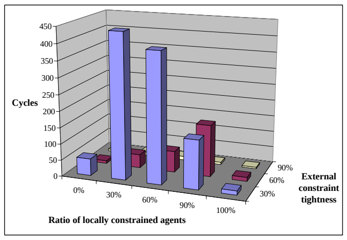


(b) Best LCDCSP strategy

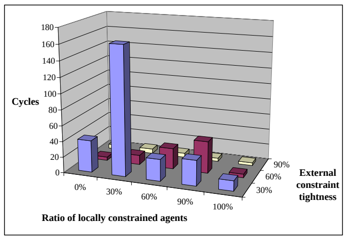


(c) Speedup

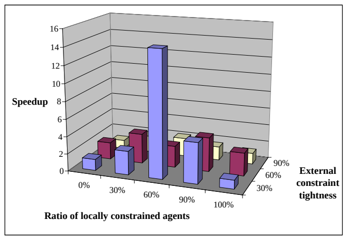
Figure B.15: Grid topology; local constraint compatibility 75%; domain size 40



(a) AWC strategy

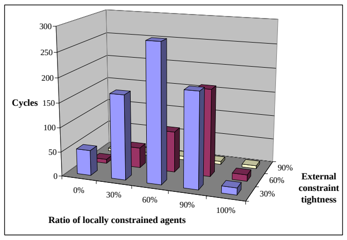


(b) Best LCDCSP strategy

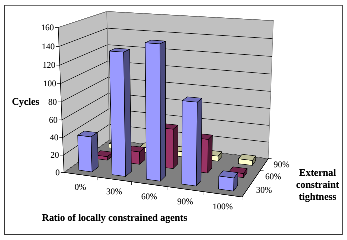


(c) Speedup

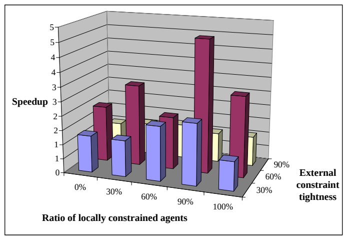
Figure B.16: Grid topology; local constraint compatibility 25%; domain size 80



(a) AWC strategy

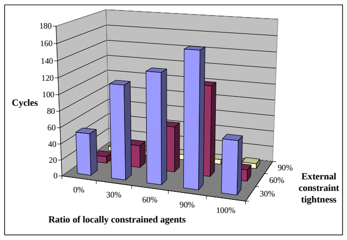


(b) Best LCDCSP strategy

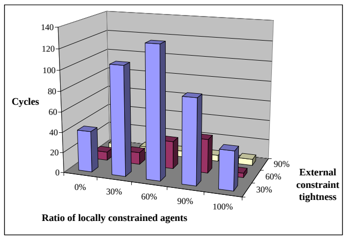


(c) Speedup

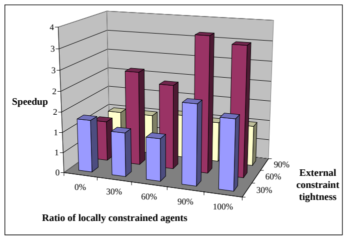
Figure B.17: Grid topology; local constraint compatibility 50%; domain size 80



(a) AWC strategy

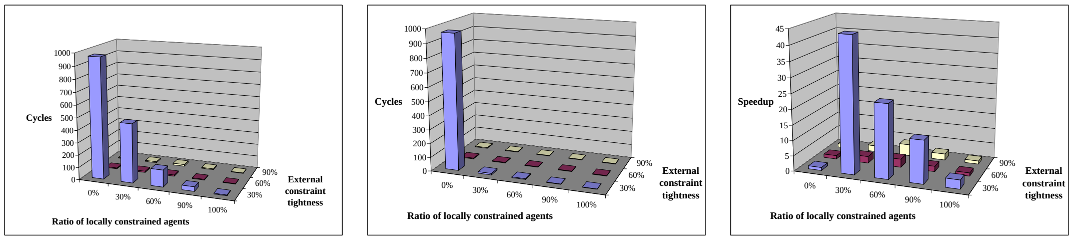


(b) Best LCDCSP strategy



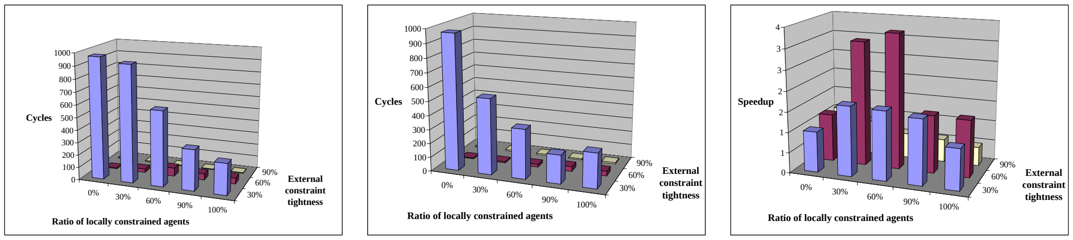
(c) Speedup

Figure B.18: Grid topology; local constraint compatibility 75%; domain size 80



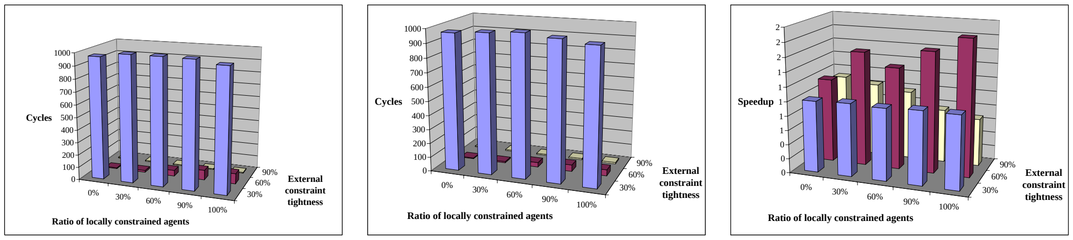
(a) AWC strategy (b) Best LCDCSP strategy (c) Speedup

Figure B.19: Triangular topology; local constraint compatibility 25%; domain size 10



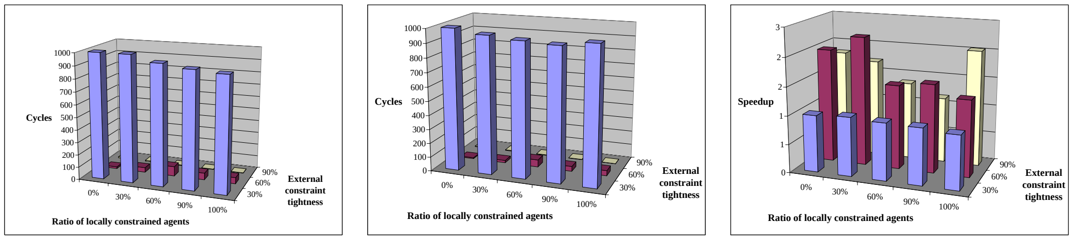
(a) AWC strategy (b) Best LCDCSP strategy (c) Speedup

Figure B.20: Triangular topology; local constraint compatibility 50%; domain size 10



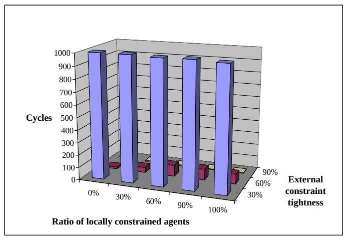
(a) AWC strategy (b) Best LCDCSP strategy (c) Speedup

Figure B.21: Triangular topology; local constraint compatibility 75%; domain size 10

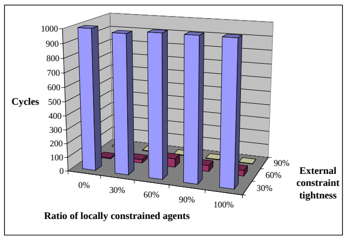


(a) AWC strategy (b) Best LCDCSP strategy (c) Speedup

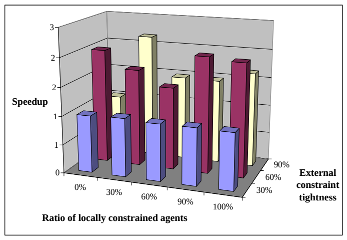
Figure B.22: Triangular topology; local constraint compatibility 25%; domain size 40



(a) AWC strategy

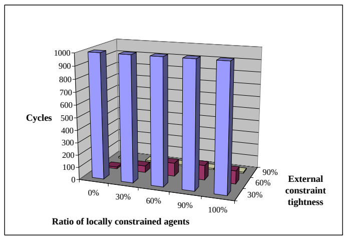


(b) Best LCDCSP strategy

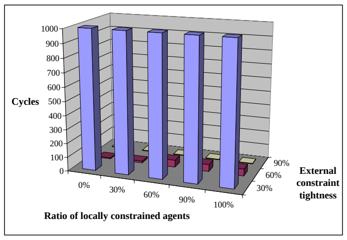


(c) Speedup

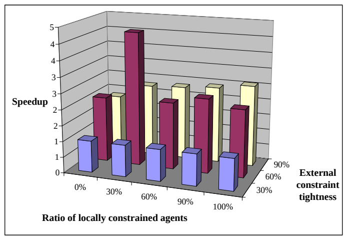
Figure B.23: Triangular topology; local constraint compatibility 50%; domain size 40



(a) AWC strategy

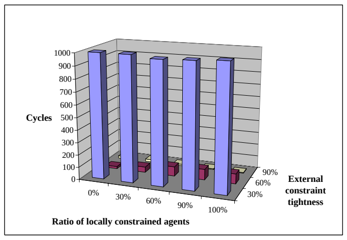


(b) Best LCDCSP strategy

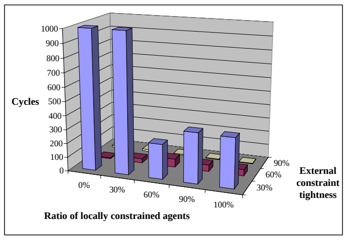


(c) Speedup

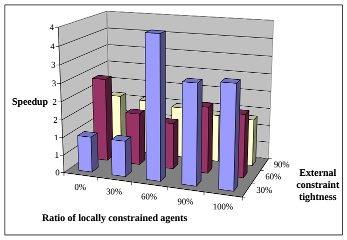
Figure B.24: Triangular topology; local constraint compatibility 75%; domain size 40



(a) AWC strategy

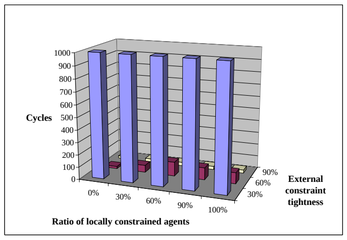


(b) Best LCDCSP strategy

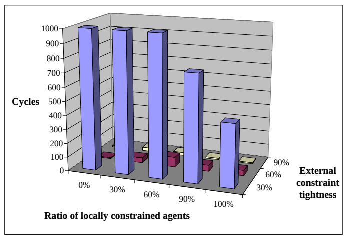


(c) Speedup

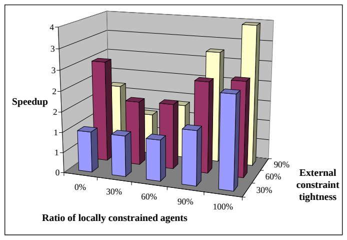
Figure B.25: Triangular topology; local constraint compatibility 25%; domain size 80



(a) AWC strategy

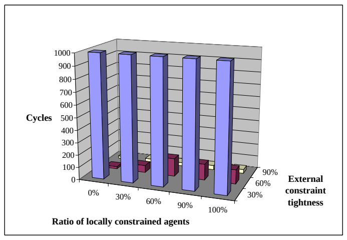


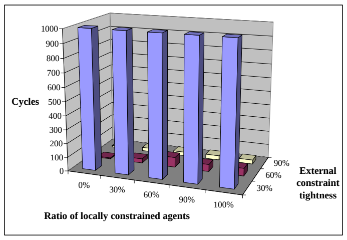
(b) Best LCDCSP strategy

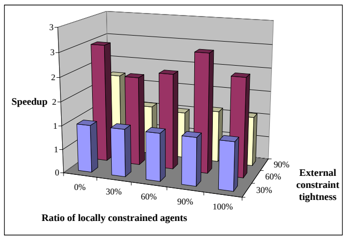


(c) Speedup

Figure B.26: Triangular topology; local constraint compatibility 50%; domain size 80







(a) AWC strategy (b) Best LCDCSP strategy (c) Speedup

Figure B.27: Triangular topology; local constraint compatibility 75%; domain size 80

B.2 Detailed Problem Hardness and Speedup by LCD-CSP Strategies

Cycles (N)	Number of problem settings
$N < 50$	237
$50 \leq N < 100$	38
$100 \leq N < 200$	23
$200 \leq N < 300$	6
$300 \leq N < 400$	6
$400 \leq N < 500$	5
$500 \leq N < 600$	2
$600 \leq N < 700$	
$700 \leq N < 800$	
$800 \leq N < 900$	
$N \geq 900$	34

Table B.1: Distribution of problem hardness

Topology	External constraint compatibility	Local constraint compatibility	Domain size	Ratio of locally constrained agents
Grid	30%	25%	10	10%
		75%	10	30%
Triangular	30%	25%	10	0%
			40	0% $\sim$ 100%
			80	0% $\sim$ 30%
		50%	10	0% $\sim$ 60%
			40	0% $\sim$ 100%
			80	0% $\sim$ 90%
		75%	10	0% $\sim$ 100%
			40	0% $\sim$ 100%
			80	0% $\sim$ 100%

Table B.2: Non-definitive problem settings where both AWC and LCDCSP srategies exceed the cycle limit

cycles (N)	# of settings in the cycle range	# of settings for different speedup (***K***) by locally cooperative strategies								
		$K < 2$	$2 \le K < 3$	$3 \le K < 4$	$4 \le K < 5$	$5 \le K < 6$	$6 \le K < 7$	$7 \le K < 8$	$8 \le K < 9$	$9 \le K < 10$
$N < 50$	113	108	5							
$50 \le N < 100$	2	1		1						
$100 \le N < 200$										
$200 \le N < 300$										
$300 \le N < 400$	1									1
$400 \le N < 500$	1									
$500 \le N < 600$										
$600 \le N < 700$										
$700 \le N < 800$										
$800 \le N < 900$										
$N \ge 900$										

(a) Hexagonal Topology

cycles (N)	# of settings	# of settings for different speedup (***K***) by locally cooperative strategies								
		$K < 2$	$2 \le K < 3$	$3 \le K < 4$	$4 \le K < 5$	$5 \le K < 6$	$6 \le K < 7$	$7 \le K < 8$	$8 \le K < 9$	$9 \le K < 10$
$N < 50$	66	45	16	4	1					
$50 \le N < 100$	19	13	4	1						1
$100 \le N < 200$	15	7	1	2	3		1		1	
$200 \le N < 300$	5	1	1			2				
$300 \le N < 400$	4			1		1				
$400 \le N < 500$	3		3							
$500 \le N < 600$										
$600 \le N < 700$	1									
$700 \le N < 800$										
$800 \le N < 900$										
$N \ge 900$	4	2	2							

(b) Grid Topology

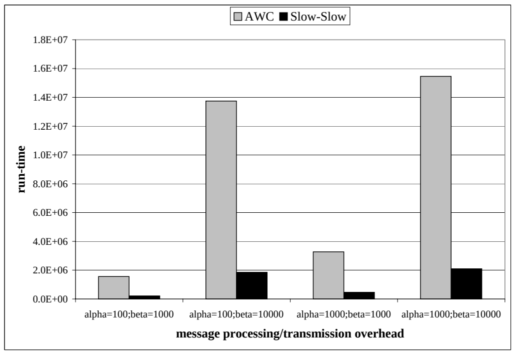
cycles (N)	# of settings	# of settings for different speedup (***K***) by locally cooperative strategies								
		$K < 2$	$2 \le K < 3$	$3 \le K < 4$	$4 \le K < 5$	$5 \le K < 6$	$6 \le K < 7$	$7 \le K < 8$	$8 \le K < 9$	$9 \le K < 10$
$N < 50$	58	36	19	3						
$50 \le N < 100$	17	11	4	1	1					
$100 \le N < 200$	8	1	6							
$200 \le N < 300$	1	1								
$300 \le N < 400$	1	1								
$400 \le N < 500$	1									
$500 \le N < 600$	1	1								
$600 \le N < 700$										
$700 \le N < 800$										
$800 \le N < 900$										
$N \ge 900$	30	26	3	1						

(c) Triangular topology

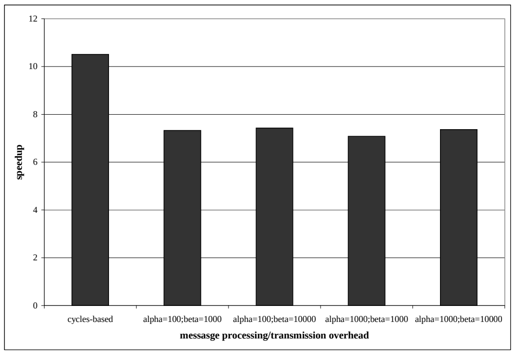
Figure B.28: Number of problem settings for different speedup

B.3 Run-time and Speedup for the Exemplar Problem Settings with High Performance Improvement

B.3.1 Run-time and Speedup When Message Size is a Major Factor for Message Processing/Communication Overhead

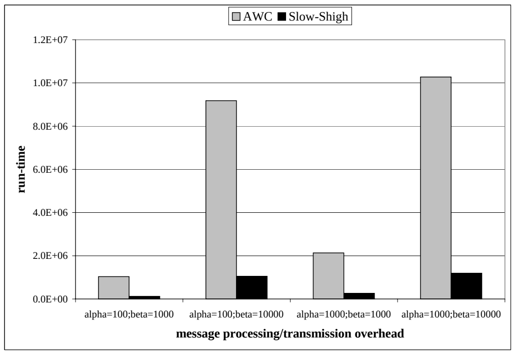


(a) Run-time

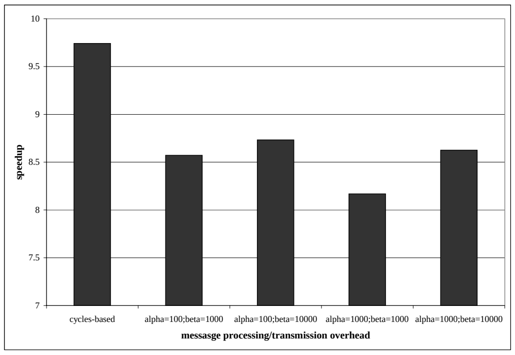


(b) Speedup

Figure B.29: Run-time and speedup for AWC strategy and the best LCDCSP strategy with different message processing/communication overhead: hexagonal topology; external constraint compatibility 30%; local constraint compatibility 25%; domain size 10; Ratio of locally constrained agents 30% (case 1 in Table 4.4)

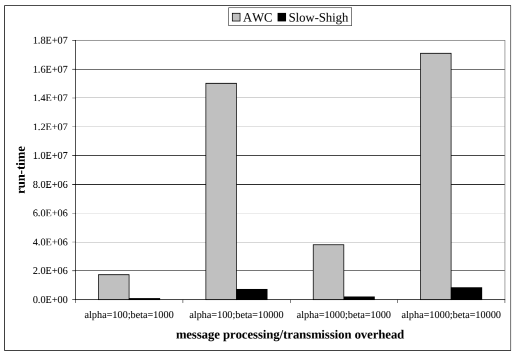


(a) Run-time

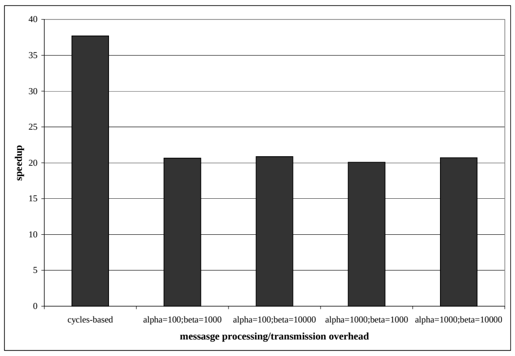


(b) Speedup

Figure B.30: Run-time and speedup for AWC strategy and the best LCDCSP strategy with different message processing/communication overhead: hexagonal topology; external constraint compatibility 30%; local constraint compatibility 25%; domain size 10; Ratio of locally constrained agents 60% (case 2 in Table 4.4)

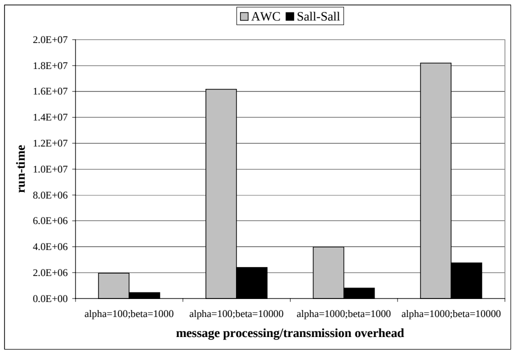


(a) Run-time

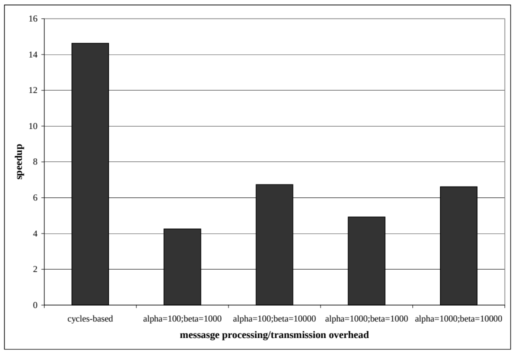


(b) Speedup

Figure B.31: Run-time and speedup for AWC strategy and the best LCDCSP strategy with different message processing/communication overhead: grid topology; external constraint compatibility 30%; local constraint compatibility 25%; domain size 10; Ratio of locally constrained agents 60% (case 3 in Table 4.4)

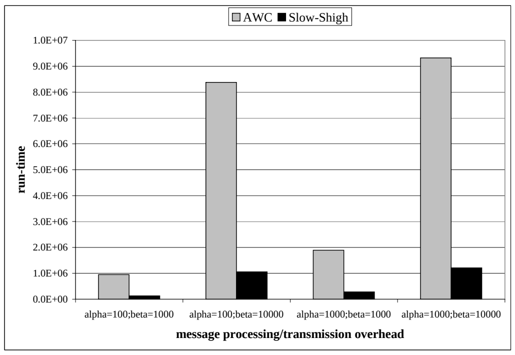


(a) Run-time

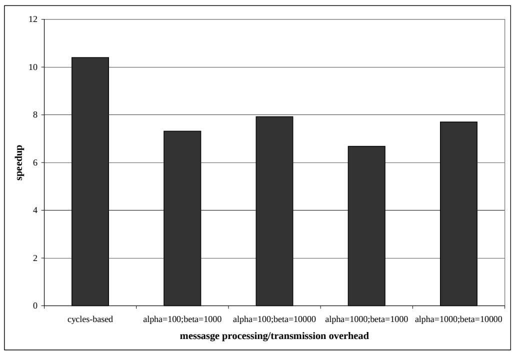


(b) Speedup

Figure B.32: Run-time and speedup for AWC strategy and the best LCDCSP strategy with different message processing/communication overhead: grid topology; external constraint compatibility 30%; local constraint compatibility 25%; domain size 80; Ratio of locally constrained agents 60% (case 4 in Table 4.4)

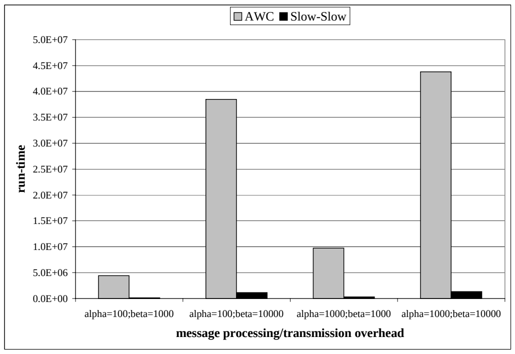


(a) Run-time

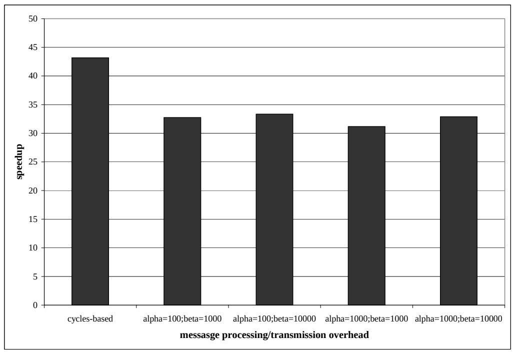


(b) Speedup

Figure B.33: Run-time and speedup for AWC strategy and the best LCDCSP strategy with different message processing/communication overhead: grid topology; external constraint compatibility 60%; local constraint compatibility 25%; domain size 40; Ratio of locally constrained agents 90% (case 5 in Table 4.4)



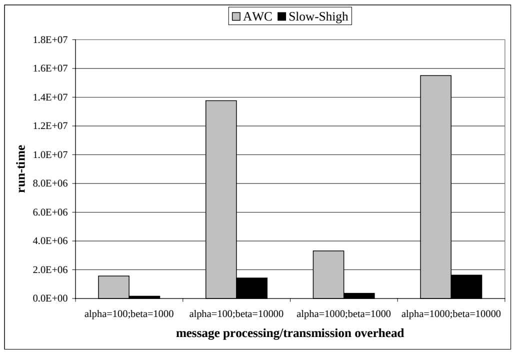
(a) Run-time



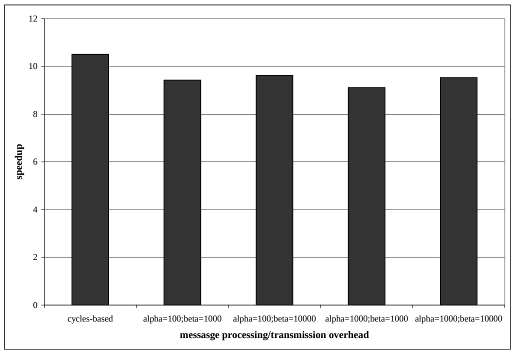
(b) Speedup

Figure B.34: Run-time and speedup for AWC strategy and the best LCDCSP strategy with different message processing/communication overhead: triangular topology; external constraint compatibility 30%; local constraint compatibility 25%; domain size 10; Ratio of locally constrained agents 30% (case 6 in Table 4.4)

B.3.2 Run-time and Speedup When Message Number is a Major Factor for Message Processing/Communication Overhead

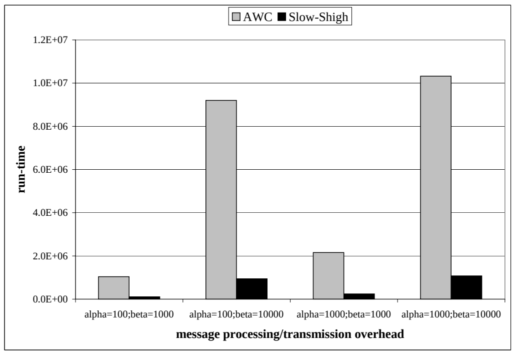


(a) Run-time

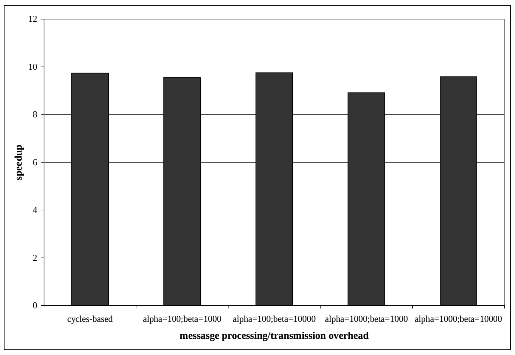


(b) Speedup

Figure B.35: Run-time and speedup for AWC strategy and the best LCDCSP strategy with different message processing/communication overhead: hexagonal topology; external constraint compatibility 30%; local constraint compatibility 25%; domain size 10; Ratio of locally constrained agents 30% (case 1 in Table 4.4)

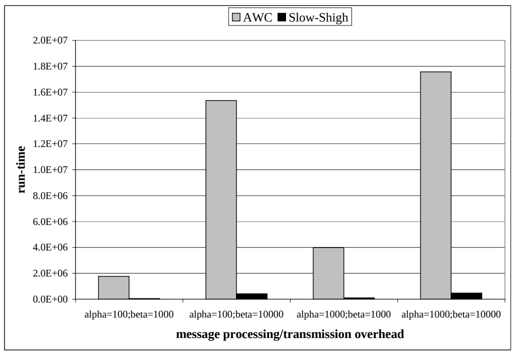


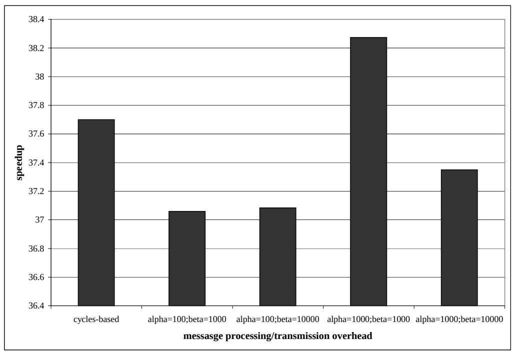
(a) Run-time



(b) Speedup

Figure B.36: Run-time and speedup for AWC strategy and the best LCDCSP strategy with different message processing/communication overhead: hexagonal topology; external constraint compatibility 30%; local constraint compatibility 25%; domain size 10; Ratio of locally constrained agents 60% (case 2 in Table 4.4)

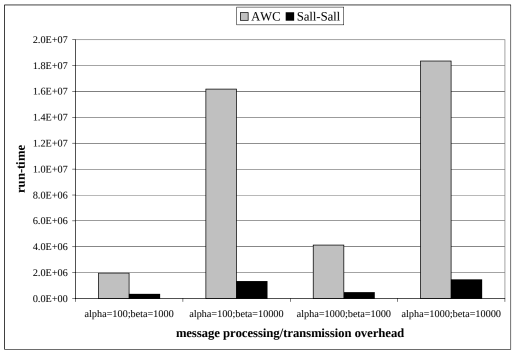


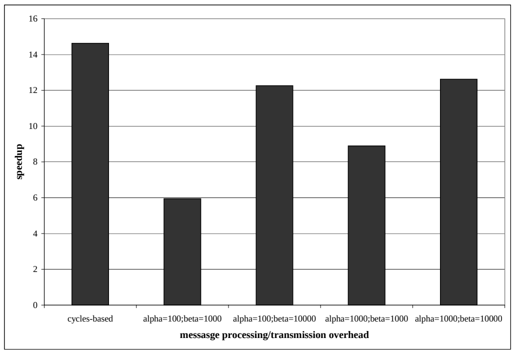


(a) Run-time

(b) Speedup

Figure B.37: Run-time and speedup for AWC strategy and the best LCDCSP strategy with different message processing/communication overhead: grid topology; external constraint compatibility 30%; local constraint compatibility 25%; domain size 10; Ratio of locally constrained agents 60% (case 3 in Table 4.4)

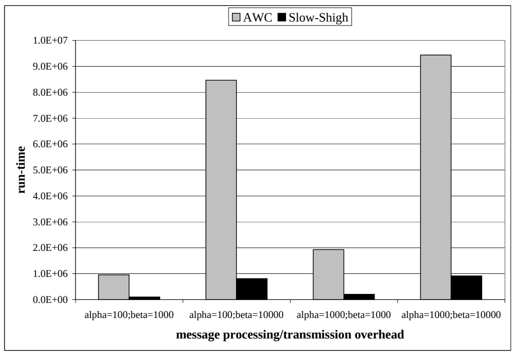


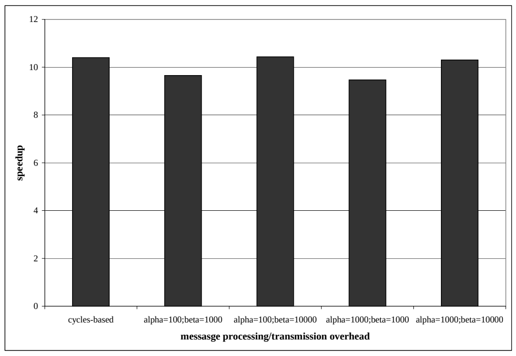


(a) Run-time

(b) Speedup

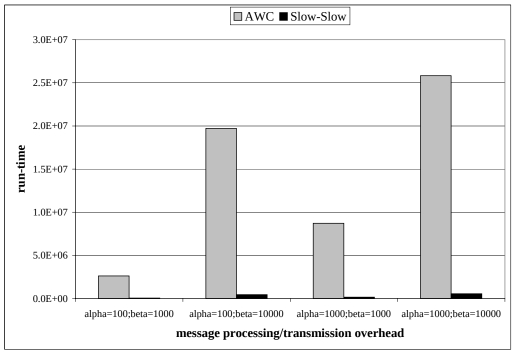
Figure B.38: Run-time and speedup for AWC strategy and the best LCDCSP strategy with different message processing/communication overhead: grid topology; external constraint compatibility 30%; local constraint compatibility 25%; domain size 80; Ratio of locally constrained agents 60% (case 4 in Table 4.4)

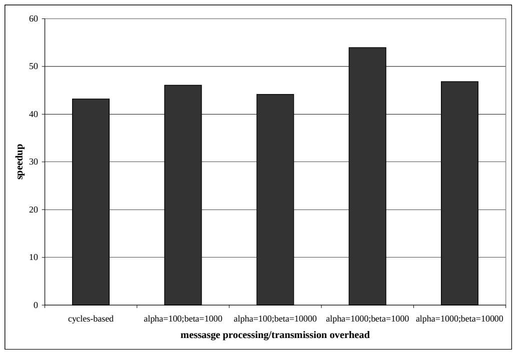




(a) Run-time (b) Speedup

Figure B.39: Run-time and speedup for AWC strategy and the best LCDCSP strategy with different message processing/communication overhead: grid topology; external constraint compatibility 60%; local constraint compatibility 25%; domain size 40; Ratio of locally constrained agents 90% (case 5 in Table 4.4)





(a) Run-time (b) Speedup

Figure B.40: Run-time and speedup for AWC strategy and the best LCDCSP strategy with different message processing/communication overhead: triangular topology; external constraint compatibility 30%; local constraint compatibility 25%; domain size 10; Ratio of locally constrained agents 30% (case 6 in Table 4.4)

B.4 Performance Measurements in Bottleneck Agents for Selected Problem Settings

B.4.1 Message Size and Constraint Checks of Bottleneck Agents

Cases	Average message size to process		Average message size to transmit	
	AWC strategy	Best locally cooperative strategy	AWC strategy	Best locally cooperative strategy
1	4.5	7.4	3.0	4.8
2	4.4	6.8	3.0	4.2
3	6.9	18.3	4.0	11
4	5.8	27.2	4.0	17.6
5	5.1	12.5	4.0	7.1
6	14.0	37.2	7.9	22.3

Table B.3: Average message size of AWC and the best LCDCSP strategy when $\alpha = 100$ and $\beta = 1000$

Cases	Average number of constraint checks	
	AWC strategy	Best locally cooperative strategy
1	28	40
2	24	28
3	30	46
4	379	5849
5	81	341
6	68	115

Table B.4: Average number of constraint checks of AWC and the best LCDCSP strategy when $\alpha = 100$ and $\beta = 1000$

cycles(N)	Number(percentage) of problem settings for different speedup (K fold)			
	$K < 2$	$2 \leq K < 5$	$5 \leq K < 8$	$K > 8$
$N < 200$	333 (96.2%)	12 (3.5%)	1 (0.3%)	0 (0%)
$200 \leq N < 500$	5 (29.4%)	5 (29.4%)	4 (23.5%)	3 (17.6%)
$N \geq 500$	37 (84.1%)	6 (13.6%)	0 (0%)	1 (2.3%)

Table B.5: Distribution of problem hardness and speedup by locally cooperative strategies when $\alpha = 100$ and $\beta = 1000$

Cases	Average message size to process		Average message size to transmit	
	AWC strategy	Best locally cooperative strategy	AWC strategy	Best locally cooperative strategy
1	4.5	7.4	3.0	4.8
2	4.4	6.8	3.0	4.2
3	6.9	18.3	4.0	11.0
4	5.8	27.0	4.0	17.4
5	5.2	12.5	4.0	7.1
6	14.0	37.2	7.9	22.3

Table B.6: Average message size of AWC and the best LCDCSP strategy when $\alpha = 100$ and $\beta = 10000$

Cases	Average number of constraint checks	
	AWC strategy	Best locally cooperative strategy
1	28	40
2	24	28
3	30	46
4	379	5579
5	82	341
6	68	115

Table B.7: Average number of constraint checks of AWC and the best LCDCSP strategy when $\alpha = 100$ and $\beta = 10000$

cycles(N)	Number(percentage) of problem settings for different speedup (K fold)			
	$K < 2$	$2 \leq K < 5$	$5 \leq K < 8$	$K > 8$
$N < 200$	327 (94.5%)	18 (5.2%)	1 (0.3%)	0 (0%)
$200 \leq N < 500$	5 (29.4%)	4 (23.5%)	4 (23.5%)	4 (23.5%)
$N \geq 500$	37 (84.1%)	6 (13.6%)	0 (0%)	1 (2.3%)

Table B.8: Distribution of problem hardness and speedup by locally cooperative strategies when $\alpha = 100$ and $\beta = 10000$

Cases	Average message size to process		Average message size to transmit	
	AWC strategy	Best locally cooperative strategy	AWC strategy	Best locally cooperative strategy
1	4.5	7.4	3.0	4.8
2	4.4	6.8	3.0	4.2
3	6.9	18.3	4.0	11.0
4	6.2	27.1	4.0	17.1
5	5.2	13.4	4.0	7.0
6	14.0	37.2	7.9	22.2

Table B.9: Average message size of AWC and the best LCDCSP strategy when $\alpha = 100$ and $\beta = 10000$

Cases	Average number of constraint checks	
	AWC strategy	Best locally cooperative strategy
1	28	40
2	24	27
3	30	45
4	291	5732
5	72	215
6	68	114

Table B.10: Average number of constraint checks of AWC and the best LCDCSP strategy when $\alpha = 1000$ and $\beta = 1000$

cycles(N)	Number(percentage) of problem settings for different speedup (K fold)			
	$K < 2$	$2 \leq K < 5$	$5 \leq K < 8$	$K > 8$
$N < 200$	332 (96.9%)	13 (3.8%)	1 (0.3%)	0 (0%)
$200 \leq N < 500$	5 (29.4%)	5 (29.4%)	5 (29.4%)	2 (11.8%)
$N \geq 500$	37 (84.1%)	6 (13.6%)	0 (0%)	1 (2.3%)

Table B.11: Distribution of problem hardness and speedup by locally cooperative strategies when $\alpha = 1000$ and $\beta = 1000$

Cases	Average message size to process		Average message size to transmit	
	AWC strategy	Best locally cooperative strategy	AWC strategy	Best locally cooperative strategy
1	4.5	7.4	3.0	4.8
2	4.4	6.8	3.0	4.2
3	6.9	18.3	4.0	11.0
4	6.2	26.9	4.0	17.1
5	5.2	13.3	4.0	7.1
6	14.0	37.2	7.9	22.3

Table B.12: Average message size of AWC and the best LCDCSP strategy when $\alpha = 1000$ and $\beta = 10000$

Cases	Average number of constraint checks	
	AWC strategy	Best locally cooperative strategy
1	28	39
2	24	27
3	30	45
4	292	5483
5	72	195
6	68	115

Table B.13: Average number of constraint checks of AWC and the best LCDCSP strategy when $\alpha = 1000$ and $\beta = 10000$

cycles(N)	Number(percentage) of problem settings for different speedup (K fold)			
	$K < 2$	$2 \leq K < 5$	$5 \leq K < 8$	$K > 8$
$N < 200$	327 (94.5%)	18 (5.2%)	1 (0.3%)	0 (0%)
$200 \leq N < 500$	5 (29.4%)	4 (23.5%)	5 (29.4%)	3 (17.6%)
$N \geq 500$	37 (84.1%)	6 (13.6%)	0 (0%)	1 (2.3%)

Table B.14: Distribution of problem hardness and speedup by locally cooperative strategies when $\alpha = 1000$ and $\beta = 10000$

B.4.2 Message Number and Constraint Checks of Bottleneck Agents

Cases	Average message number to process		Average message number to transmit	
	AWC strategy	Best locally cooperative strategy	AWC strategy	Best locally cooperative strategy
1	4.5	4.7	3.0	3.0
2	4.4	4.8	3.0	3.0
3	6.9	6.5	4.0	4.0
4	5.8	5.4	4.0	4.0
5	5.2	5.7	4.0	4.0
6	14.0	12.7	8.0	8.0

Table B.15: Average message size of AWC and the best LCDCSP strategy when $\alpha = 100$ and $\beta = 1000$

Cases	Average number of constraint checks	
	AWC strategy	Best locally cooperative strategy
1	28	85
2	24	41
3	30	95
4	380	7743
5	83	488
6	68	235

Table B.16: Average number of constraint checks of AWC and the best LCDCSP strategy when $\alpha = 100$ and $\beta = 1000$

cycles(N)	Number(percentage) of problem settings for different speedup (K fold)			
	$K < 2$	$2 \le K < 5$	$5 \le K < 8$	$K > 8$
$N < 200$	318 (92.4%)	23 (6.7%)	1 (0.3%)	2 (0.6%)
$200 \le N < 500$	4 (23.5%)	4 (23.5%)	4 (23.5%)	5 (29.4%)
$N \ge 500$	37 (84.1%)	6 (13.6%)	0 (0%)	1 (2.3%)

Table B.17: Distribution of problem hardness and speedup by locally cooperative strategies when $\alpha = 100$ and $\beta = 1000$

Cases	Average message number to process		Average message number to transmit	
	AWC strategy	Best locally cooperative strategy	AWC strategy	Best locally cooperative strategy
1	4.5	4.7	3.0	3.0
2	4.5	4.8	3.0	3.0
3	6.9	6.5	4.0	4.0
4	5.8	5.4	4.0	4.0
5	5.2	5.7	4.0	4.0
6	14.0	12.7	8.0	8.0

Table B.18: Average message size of AWC and the best LCDCSP strategy when $\alpha = 100$ and $\beta = 10000$

Cases	Average number of constraint checks	
	AWC strategy	Best locally cooperative strategy
1	28	85
2	24	41
3	30	95
4	380	7743
5	83	488
6	68	235

Table B.19: Average number of constraint checks of AWC and the best LCDCSP strategy when $\alpha = 100$ and $\beta = 10000$

cycles(N)	Number(percentage) of problem settings for different speedup (K fold)			
	$K < 2$	$2 \leq K < 5$	$5 \leq K < 8$	$K > 8$
$N < 200$	308 (89.5%)	32 (9.3%)	2 (0.6%)	2 (0.6%)
$200 \leq N < 500$	4 (23.5%)	4 (23.5%)	3 (17.6%)	6 (35.3%)
$N \geq 500$	37 (84.1%)	6 (13.6%)	0 (0%)	1 (2.3%)

Table B.20: Distribution of problem hardness and speedup by locally cooperative strategies when $\alpha = 100$ and $\beta = 10000$

Cases	Average message number to process		Average message number to transmit	
	AWC strategy	Best locally cooperative strategy	AWC strategy	Best locally cooperative strategy
1	4.5	4.7	3.0	3.0
2	4.5	4.8	3.0	3.0
3	6.9	6.5	4.0	4.0
4	5.8	5.4	4.0	4.0
5	5.2	5.7	4.0	4.0
6	14.0	12.7	8.0	8.0

Table B.21: Average message size of AWC and the best LCDCSP strategy when $\alpha = 1000$ and $\beta = 1000$

Cases	Average number of constraint checks	
	AWC strategy	Best locally cooperative strategy
1	28	85
2	24	41
3	30	95
4	380	7743
5	83	488
6	68	235

Table B.22: Average number of constraint checks of AWC and the best LCDCSP strategy when $\alpha = 1000$ and $\beta = 1000$

cycles(N)	Number(percentage) of problem settings for different speedup (K fold)			
	$K < 2$	$2 \leq K < 5$	$5 \leq K < 8$	$K > 8$
$N < 200$	318 (92.4%)	23 (6.7%)	1 (0.3%)	2 (0.6%)
$200 \leq N < 500$	4 (23.5%)	4 (23.5%)	3 (17.6%)	6 (35.3%)
$N \geq 500$	37 (84.1%)	6 (13.6%)	0 (0%)	1 (2.3%)

Table B.23: Distribution of problem hardness and speedup by locally cooperative strategies when $\alpha = 1000$ and $\beta = 1000$

Cases	Average message number to process		Average message number to transmit	
	AWC strategy	Best locally cooperative strategy	AWC strategy	Best locally cooperative strategy
1	4.5	4.7	3.0	3.0
2	4.5	4.8	3.0	3.0
3	6.9	6.5	4.0	4.0
4	5.8	5.4	4.0	4.0
5	5.2	5.7	4.0	4.0
6	14.0	12.7	8.0	8.0

Table B.24: Average message size of AWC and the best LCDCSP strategy when $\alpha = 1000$ and $\beta = 10000$

Cases	Average number of constraint checks	
	AWC strategy	Best locally cooperative strategy
1	28	85
2	24	41
3	30	95
4	380	7743
5	83	488
6	68	235

Table B.25: Average number of constraint checks of AWC and the best LCDCSP strategy when $\alpha = 1000$ and $\beta = 10000$

cycles(N)	Number(percentage) of problem settings for different speedup (K fold)			
	$K < 2$	$2 \leq K < 5$	$5 \leq K < 8$	$K > 8$
$N < 200$	308 (89.5%)	32 (9.3%)	2 (0.6%)	2 (0.6%)
$200 \leq N < 500$	4 (23.5%)	4 (23.5%)	3 (17.6%)	6 (35.3%)
$N \geq 500$	37 (84.1%)	6 (13.6%)	0 (0%)	1 (2.3%)

Table B.26: Distribution of problem hardness and speedup by locally cooperative strategies when $\alpha = 1000$ and $\beta = 10000$

B.5 Performance Variation in Different Computing & Networking Environments

B.5.1 When Message Size is a Major Factor for Message Processing & Communication Overhead

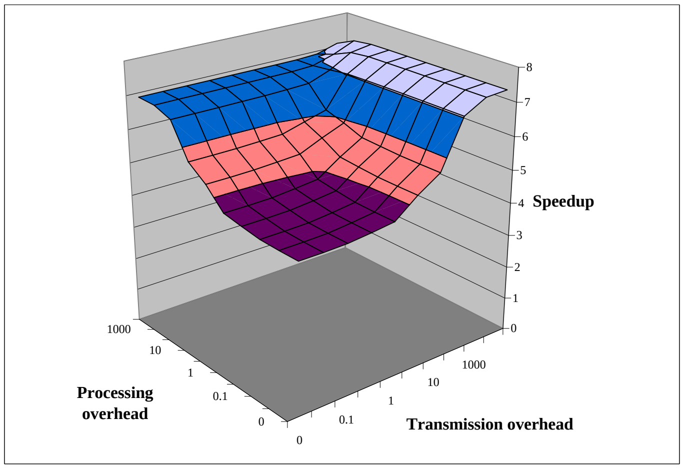


Figure B.41: Speedup variation: hexagonal topology; external constraint compatibility 30%; local constraint compatibility 25%; domain size 10; Ratio of locally constrained agents 30%

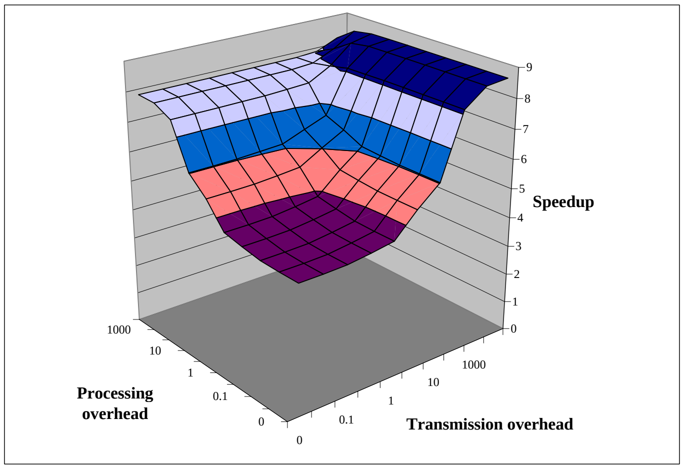


Figure B.42: Speedup variation: hexagonal topology; external constraint compatibility 30%; local constraint compatibility 25%; domain size 10; Ratio of locally constrained agents 60%

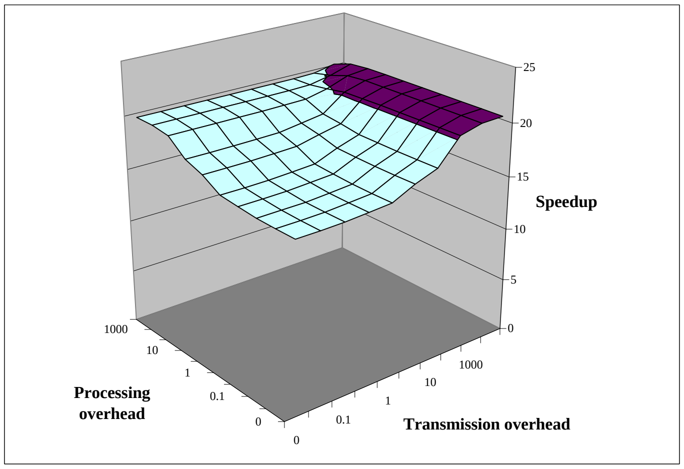


Figure B.43: Speedup variation: grid topology; external constraint compatibility 30%; local constraint compatibility 25%; domain size 10; Ratio of locally constrained agents 60%

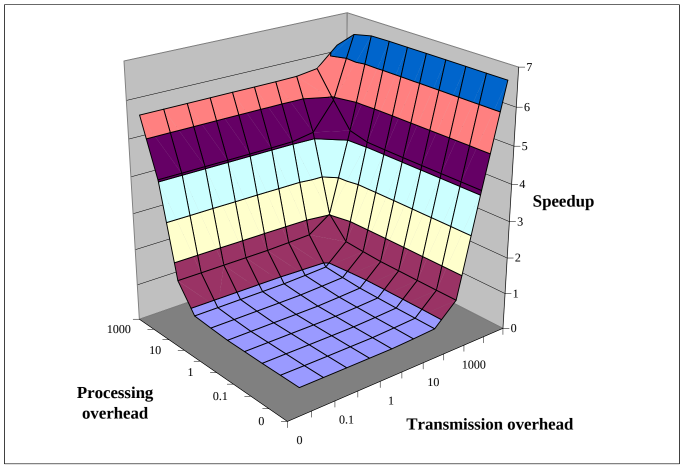


Figure B.44: Speedup variation: grid topology; external constraint compatibility 30%; local constraint compatibility 25%; domain size 80; Ratio of locally constrained agents 60%

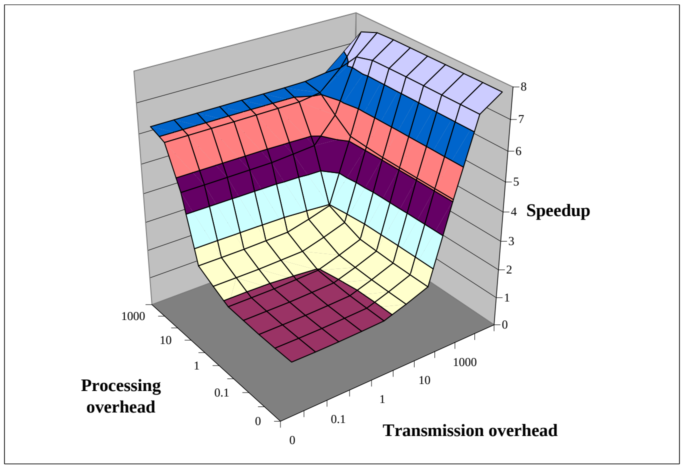


Figure B.45: Speedup variation: grid topology; external constraint compatibility 60%; local constraint compatibility 25%; domain size 40; Ratio of locally constrained agents 90%

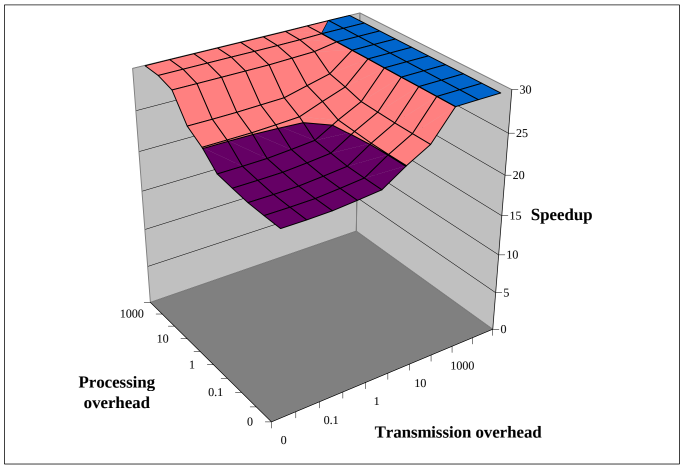


Figure B.46: Speedup variation: triangular topology; external constraint compatibility 30%; local constraint compatibility 25%; domain size 10; Ratio of locally constrained agents 30%

B.5.2 When Message Number is a Major Factor for Message Processing & Communication Overhead

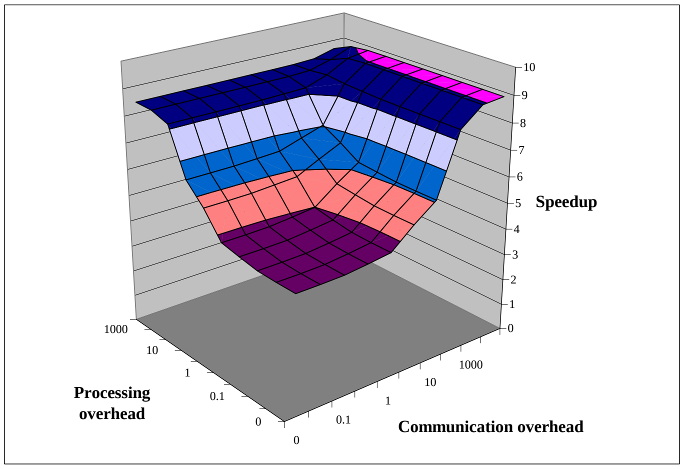


Figure B.47: Speedup variation: hexagonal topology; external constraint compatibility 30%; local constraint compatibility 25%; domain size 10; Ratio of locally constrained agents 30%

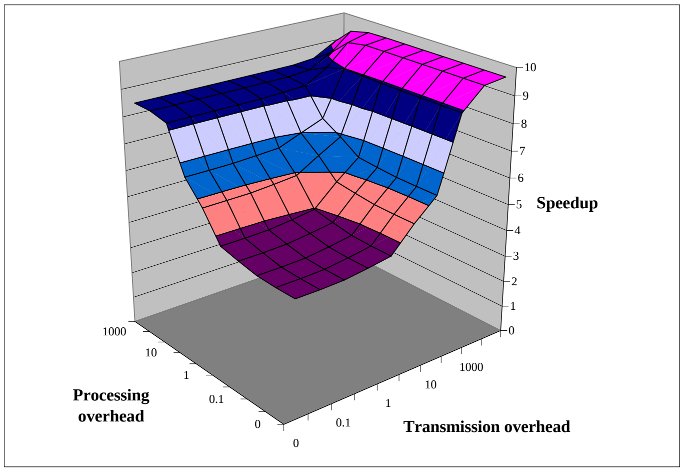


Figure B.48: Speedup variation: hexagonal topology; external constraint compatibility 30%; local constraint compatibility 25%; domain size 10; Ratio of locally constrained agents 60%

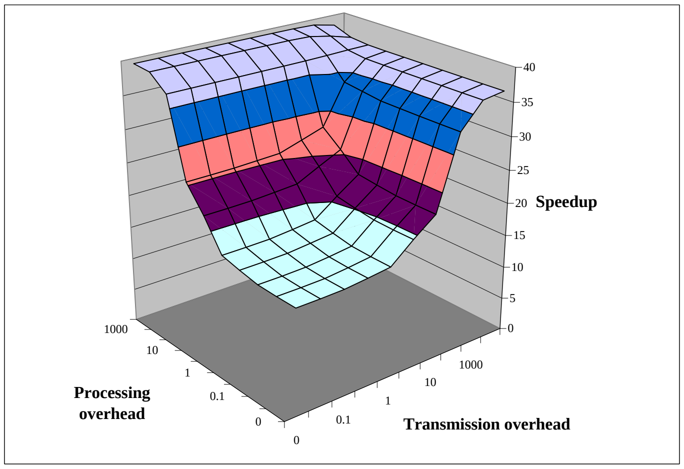


Figure B.49: Speedup variation: grid topology; external constraint compatibility 30%; local constraint compatibility 25%; domain size 10; Ratio of locally constrained agents 60%

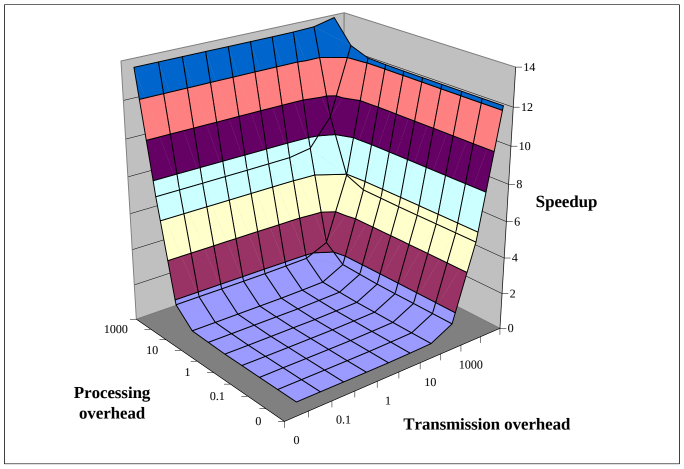


Figure B.50: Speedup variation: grid topology; external constraint compatibility 30%; local constraint compatibility 25%; domain size 80; Ratio of locally constrained agents 60%

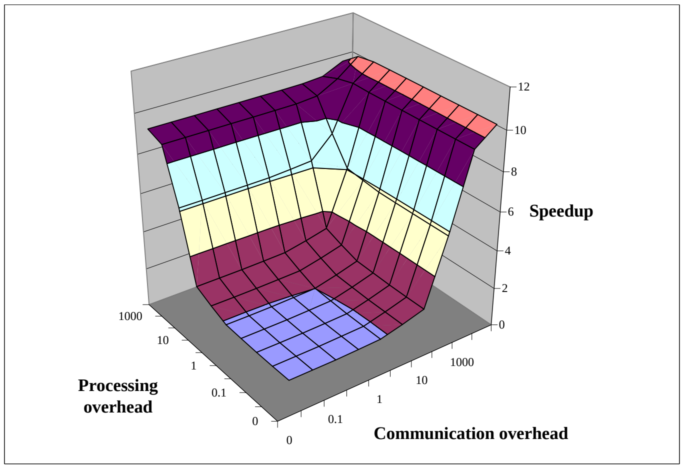


Figure B.51: Speedup variation: grid topology; external constraint compatibility 60%; local constraint compatibility 25%; domain size 40; Ratio of locally constrained agents 90%

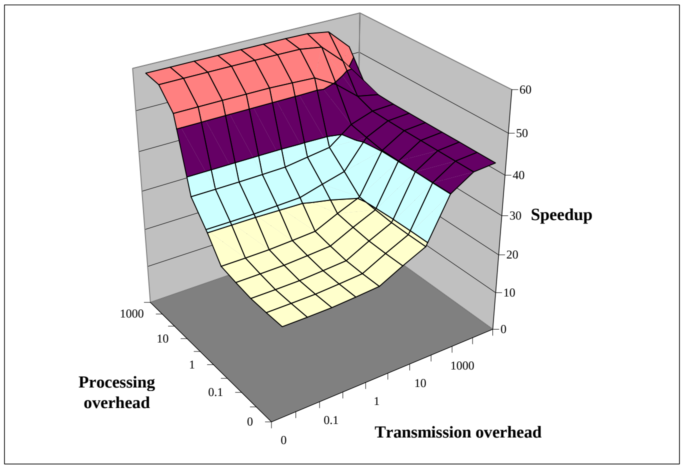


Figure B.52: Speedup variation: triangular topology; external constraint compatibility 30%; local constraint compatibility 25%; domain size 10; Ratio of locally constrained agents 30%

Appendix C

Computation Cost Saving by Building Blocks

In Chapter 4 (Section 4.2.1), it was shown that, when a given policy is based on agents' local states in a locally observable world, the evaluation of an MTDP policy can be reduced to a Markov chain analysis. While the space reduction significantly decreases computation cost, given a large state space, the Markov chain analysis still requires a high computation cost. In this appendix, we show that the building block based approach further reduces significant amount of computation cost and memory requirements, which makes the MTDP based model feasible in practice.

Number of agents in Markov chain	5	10	15	25
Runtime	101 *sec*	105 *min*	more than 10 hours	—

Table C.1: Comparison of runtime for Markov chain

To show the computation cost saved by building block based approach, we compare the runtime of Markov chain analysis with different number of agents (Table C.1). Table C.1 shows that the runtime increases exponentially as the number of agents increases in a Markov chain [1]. For the case with 15 agents, if we apply the building block based approach, the total time would be around 303 *sec* ($\approx$ 101 *sec* $\times$ 3) since the Markov

[1]For the case with 10 agents, we stopped the process after 10 hours, and the 25 agents case could not be run because of memory space (explained below)

chain with 15 agents can be divided into three building blocks (each building block with 5 agents takes 101 *sec*). Since the cut off time for 15 agents case is 10 *hours*, the speedup from using building blocks is more than 119 (= 10 *hours* / 303 *sec*).

In addition to the computation cost saving, the building block based approach gives a significant efficiency in terms of memories required for the computation. Table C.2 compares the memory space to store the table for state transition probabilities (assuming that the transition probability is represented in a single-precision floating point data type which requires 4 bytes). For simplification, a terminal state is not considered. While the Markov chain with 25 agents requires 4 tera bytes of memory for transition table, if we apply the building block based approach, the Markov chain can be divided into five building blocks reducing the space requirement into 20K bytes (= 5 $\times$ 4K bytes).

Number of agents in Markov chain	Number of world states	Memory requirements
5	2^5	4K bytes (= $2^5 \times 2^5 \times 4bytes$)
10	2^{10}	4M bytes (= $2^{10} \times 2^{10} \times 4bytes$)
15	2^{15}	4G bytes (= $2^{15} \times 2^{15} \times 4bytes$)
20	2^{20}	4T bytes (= $2^{20} \times 2^{20} \times 4bytes$)

Table C.2: Memory space for transition table in Markov chain analysis

Appendix D

Efficiency of Novel Conflict Resolution Strategies in Sensor Networks

While an example of DCSP mapping for distributed sensor networks was presented in Chapter 3 (Section 3.1.2), in this appendix, we provide another illustration in the distributed sensor domain which lays out our ideas of local constraint communication and novel conflict resolution strategies using a detailed example.

In the DCSP mapping (Section 3.1.2), each tracker agent is modeled as a variable whose values are radar sector combinations of sensors controlled by the tracker plus a special value NT (no target). Following local and external constraints are given to tracker agents (local and external constraints are numbered for reference purpose):

- External constraint (**C1**): two tracker agents cannot share any sensor in their values.
- Local constraint (**C2**): if a tracker agent is notified of target detection by any associated sensor, the tracker's value cannot be NT.

Given conflicts in selecting sensor combinations (with appropriate sectors), the tracker agents must resolve them as quickly as possible to reduce RMS (root mean square) error, a major criterion for tracking accuracy, that is measured by the distance between a target's real position and its detected position. The RMS error for a target increases if the target is not tracked at all or tracked with less than three sensors.

The following example (Figure D.1) shows how the extra communication of local constraints and novel conflict resolution strategies improve performance in convergence in the DCSP mapping of this sensor domain. The reduction of search space and inter-agent communication cycles will be presented. For this example, we add time constraints which requires targets to be tracked in a certain time period. Therefore, the value (sensor combinations) of trackers includes an additional component for the time period (τ_i): e.g., $< sector^1_0, sector^2_2, sector^4_0 : \tau_i >$ where $sector^i_j$ indicates sensor i's sector j.

In the Figure D.1, tracker agents (tarcker1, 2, 3, and 5) have targets (T1, T2, T3, and T4) in their sensing areas. In addition to the local and external constraints defined above, tracker agents have the following additional constraints regarding time and the availability of sensors:

- **C3**: Each tracker agent must track a target either at τ_1 or τ_2.
- **C4**: Tracker1 must track its target at τ_1.
- **C5**: Tracker3 must track its target at τ_2
- **C6**: Sensor8 (denoted as $S8$) is not available since its power is low: this is an additional temporal local constraint for tracker3.

Local constraints can be represented as explicit specification of available values from the tracker agents' domain. Thus, given local constraints, agents' domains are reduced. For instance, tracker3 has originally 9 values (NT plus 4 combinations of sensors at τ_1 or τ_2):

- $< sector^3_0, sector^4_2, sector^7_0 : \tau_1 >$
- $< sector^3_0, sector^4_2, sector^8_1 : \tau_1 >$
- $< sector^3_0, sector^7_0, sector^8_1 : \tau_1 >$

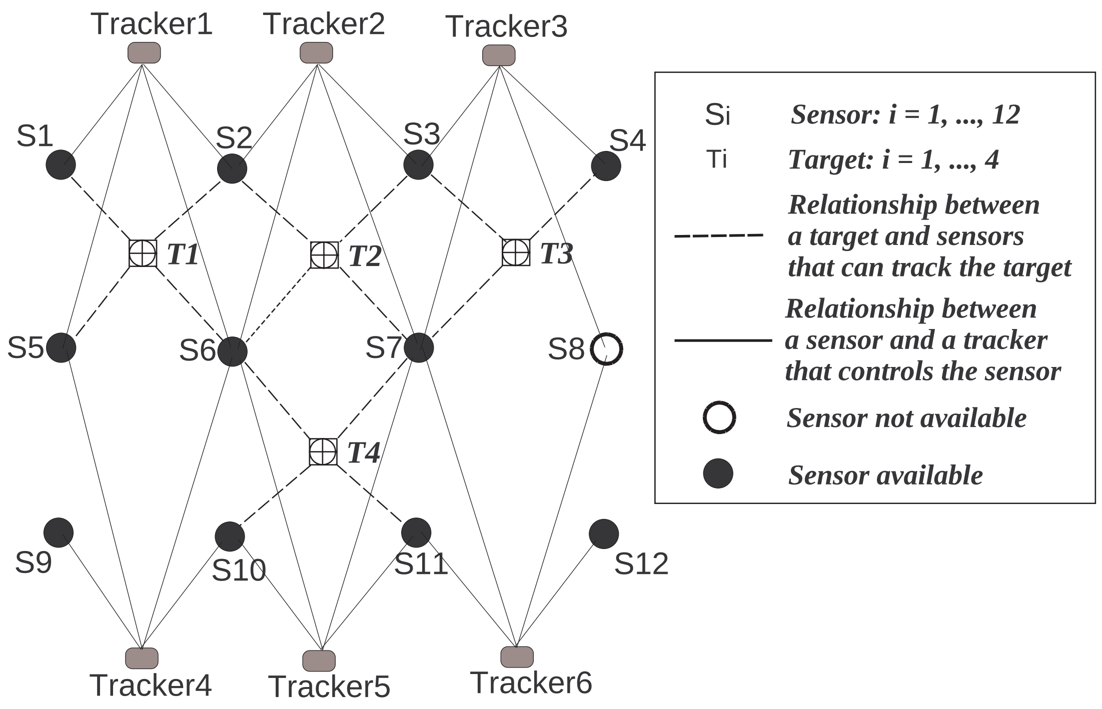

Tracker1
Tracker2
Tracker3
S1
S2
S3
S4
T1
T2
T3
S5
S6
S7
S8
T4
S9
S10
S11
S12
Tracker4
Tracker5
Tracker6
Si Sensor: i = 1, ..., 12
Ti Target: i = 1, ..., 4
Relationship between a target and sensors that can track the target
Relationship between a sensor and a tracker that controls the sensor
Sensor not available
Sensor available

Figure D.1: Distributed sensor networks with targets (for simplification, radar sectors for each sensor is not drawn.)

- $< sector_2^4, sector_0^7, sector_1^8 : \tau_1 >$
- $< sector_0^3, sector_2^4, sector_0^7 : \tau_2 >$
- $< sector_0^3, sector_2^4, sector_1^8 : \tau_2 >$
- $< sector_0^3, sector_0^7, sector_1^8 : \tau_2 >$
- $< sector_2^4, sector_0^7, sector_1^8 : \tau_2 >$
- NT

From tracker3's original domain, $C1$ eliminates NT, $C5$ eliminates 4 values (4 combinations of sensors at τ_1), and $C6$ eliminates 3 values that includes the sensor8 ($S8$) that is not available, ($< sector_0^3, sector_2^4, sector_1^8 : \tau_2 >$, $< sector_0^3, sector_0^7, sector_1^8 : \tau_2 >$, and $< sector_2^4, sector_0^7, sector_1^8 : \tau_2 >$). The three local constraints of tracker3 can be represented as a single (unary) local constraint:

- Available value set for tracker3 is $\{< sector_0^3, sector_2^4, sector_0^7 : \tau_2 >\}$

Henceforth, we assume that all the local constraints are represented as unary constraints that specify the available values for the tracker agent variables. Furthermore, for brevity, *constraint propagation* that was described in Chapter 3 (Section 3.2.1) refers to the *extra communication of local constraint and constraint propagation* since the constraint propagation in DCSP is not feasible without local constraint communication.

In the example above, the original search space before the constraint propagation has 256 value combinations: that is, 4 values for tracker1 (4 combinations of sensors at τ_1), 8 values for tracker2 (4 combinations of sensors at tau_1 or τ_2), 1 value for tracker3 (1 sensor combination at τ_2), and 8 values for tracker5 (4 combinations of sensors at tau_1 or τ_2). With constraint propagation, the original search space is reduced to 28 value combinations: 2 values for tracker1, 2 values for tracker2, 1 value for tracker3, and 7 values for tracker5. Refer to Section D.1 for detailed process of constraint propagation. This is about 9-fold decrease of search space.

Assume that, given the targets (T1, T2, T3, and T4), trackers select the following initial values: $< sector_2^2, sector_0^5, sector_1^6 : \tau_1 >$ for tracker1, $< sector_0^2, sector_2^3, sector_0^6 : \tau_1 >$ for tracker2, $< sector_0^3, sector_2^4, sector_0^7 : \tau_2 >$ for tracker3, NT for tracker4, $< sector_2^7, sector_0^{10}, sector_1^{11} : \tau_2 >$ for tracker5, and NT for tracker 6. With this initial value assignment, 10 inter-agent communication cycles are consumed until a solution is found without constraint propagation. In contrast, 3 cycles are consumed with constraint propagation. Section D.2 presents how the Asynchronous Weak Commitment algorithm solves the given problem (as shown in Figure

D.1) with the above initialization with and without extra local constraint communication. In addition to the reduction of search space and inter-agent communication cycles, the benefit of efficient value ordering heuristics can be seen in Figure D.1. Assume that only target T1 exists. When a target T1 is detected, tracker1 has 4 choices of sensor combinations: $< sector_0^1, sector_2^2, sector_0^5 : \tau_1 >$, $< sector_0^1, sector_2^2, sector_1^6 : \tau_1 >$, $< sector_0^1, sector_0^5, sector_1^6 : \tau_1 >$, and $< sector_2^2, sector_0^5, sector_1^6 : \tau_1 >$. If tracker2 also has to track its target at τ_1, while the former two values give one choice for tracker2, the latter two values give no choice to tracker2 (because, if both $S2$ and $S6$ are required for T1, when another target T2 appears in tracker2's region, tracker2 has only two sensors available). Here, it sounds more reasonable to select a value with more flexibility towards other agents. This example illustrates the benefit of such a heuristic that gives more flexibility to others. Indeed, the experimental results (described in the thesis proposal presentation) show the performance improvement from flexibility-based value ordering heuristics.

D.1 Search space reduction from local constraint communication and constraint propagation

In this section, we show how constraint propagation reduces the search space of a given problem. As constraints are propagated, the local constraints (specifying available values) are changed. For brevity, each sensor is represented only by its index without sector index. Since only one sector can be activated at one time, a sensor can be included in only one tracker's value so that constraint propagation can be achieved by referring to only sensor index. In the real value assignment, appropriate sector indices will be attached. For instance, tracker1's value $< 1, 5, 6 : \tau_1 >$ indicates $< sector_0^1, sector_0^5,$ $sector_2^6 : \tau_1 >$. The initial local constraints of tracker1, 2, 3, and 5 are as follows.

- Tracker1: Available value set is $\{< 1, 2, 5 : \tau_1 >, < 1, 2, 6 : \tau_1 >, < 1, 5, 6 : \tau_1 >,$ $< 2, 5, 6 : \tau_1 >\}$.
  - C1 (constraint) eliminates NT, and C6 eliminates 4 values (4 combinations of sensors at τ_2).
- Tracker2: Available value set is $\{< 2, 3, 6 : \tau_1 >, < 2, 3, 7 : \tau_1 >, < 2, 6, 7 :$ $\tau_1 >, < 3, 6, 7 : \tau_1 >, < 2, 3, 6 : \tau_2 >, < 2, 3, 7 : \tau_2 >, < 2, 6, 7 : \tau_2 >,$ $< 3, 6, 7 : \tau_2 >\}$.
  - Tracker2 has only C1 as a local constraint. Thus, it eliminates only NT from the original domain.
- Tracker3: Available value set is $\{< 3, 4, 7 : \tau_2 >\}$ (4 combinations of sensors at τ_1 or τ_2).
  - Tracker3's local constrains are listed above.
- Tracker5: Available value set is $\{< 6, 7, 10 : \tau_1 >, < 6, 7, 11 : \tau_1 >, < 6, 10, 11 :$ $\tau_1 >, < 7, 10, 11 : \tau_1 >, < 6, 7, 10 : \tau_2 >, < 6, 7, 11 : \tau_2 >, < 6, 10, 11 : \tau_2 >,$ $< 7, 10, 11 : \tau_2 >\}$ (4 combinations of sensors at τ_1 or τ_2).
  - Tracker5 has only C1 as a local constraint. Thus, it eliminates only NT from the original domain.
- Other tracker agents (tracker 4 and 6) retain their original domain values.

Constraint propagation is performed at several iterations until no more domains are changed. Constraint propagation in DCSP is done as follows. With communicated local constrains, each agent infers the current domain of neighboring agents and eliminates the values from its own domain that do not have any value in a neighbor's domain that can satisfy the external constraint between itself and the neighbor.

1. 1st iteration

   - Tracker1 propagates the local constraints from M2 and M5, but all of its values have a compatible value in the domains of M2 and M5.
   - Tracker2 propagates the local constraints from tracker1, 3, and 5. Two values at τ_1 are incompatible with tracker1 since they give no choice to tracker1: $< 2, 3, 6 : \tau_1 >$ and $< 2, 6, 7 : \tau_1 >$. Four values at τ_2 are deleted since any value at the time period τ_2 is not compatible with tracker3's unique choice $< 3, 4, 7 : \tau_2 >$. With tracker5, no propagation. Therefore, tracker2's domain is reduced to two values: $< 2, 3, 7 : \tau_1 >$ and $< 3, 6, 7 : \tau_1 >$.
   - Tracker3 has only one value $< 3, 4, 7 : \tau_2 >$ which has a compatible value with tracker2 and tracker5.
   - Tracker5's value $< 6, 7, 11 : \tau_2 >$ is not compatible with tracker3's unique choice. Other values are compatible with at least one value in the domains of tracker1 and 2. Therefore, M5's domain is reduced to seven values (original 8 values minus $< 6, 7, 11 : \tau_2 >$).

2. 2nd iteration

   - Two values of tracker1 are compatible with tracker2's deleted values at τ_2. Now, $< 1, 2, 6 : \tau_1 >$ and $< 2, 5, 6 : \tau_2 >$ give no choice to tracker2. Therefore, tracker1's domain is reduced to two values $< 1, 2, 5 : \tau_1 >$ and $< 1, 5, 6 : \tau_1 >$.

3. 3rd iteration

    - No more propagation

With constraint propagation, original search space is reduced to 28 (= $2 \times 2 \times 1 \times 7$) value combinations.

D.2 Comparison of numbers of cycles with and without constraint propagation

In measuring the number of cycles, we assume that agent communication is done in a synchronous way. After each cycle, agents simultaneously communicate their values and priorities. Note that, in AWC, agents need to satisfy the constraint with only higher agents. If the priorities are same, the order is defined by the alphanumerical order of indexes. Without constraint propagation, 10 inter-agent communication cycles are required as follows (priorities are in parentheses). As was done in Section D.1, for brevity, each sensor is represented only by its index without sector index.

1. Cycle 1
   Tracker1(0):$< 2, 5, 6 : \tau_1 >$, tracker2(0):$< 2, 3, 6 : \tau_1 >$, tracker3(0):$< 3, 4, 7 : \tau_2 >$, tracker5(0):$< 7, 10, 11 : \tau_1 >$

2. Cycle 2
   Tracker1(0):$< 2, 5, 6 : \tau_1 >$, tracker2(0):$< 2, 3, 6 : \tau_2 >$, tracker3(0):$< 3, 4, 7 : \tau_2 >$, tracker5(0):$< 7, 10, 11 : \tau_1 >$

    - Tracker2 cannot find a value at τ_1 to satisfy the constraint with tracker1. It selects a new value with τ_2.

3. Cycle 3

   Tracker1(0):$< 2,5,6 : \tau_1 >$, tracker2(0):$< 2,3,6 : \tau_2 >$, tracker3(0 $\rightarrow$ 1):$< 3,4,7 : \tau_2 >$, tracker5(0):$< 7,10,11 : \tau_1 >$

   - Tracker3 cannot satisfy the constraints with tracker2 since it is restricted to τ_2 and the current value is its unique choice. It increases its priority from 0 to 1.

4. Cycle 4

   Tracker1(0):$< 2,5,6 : \tau_1 >$, tracker2(0 $\rightarrow$ 2):$< 2,3,6 : \tau_1 >$, tracker3(1):$< 3,4,7 : \tau_2 >$, tracker5(0):$< 7,10,11 : \tau_1 >$

   - Tracker2 finds a conflict with tracker3 (now, tracker3 is higher than tracker2). It cannot find a value that can satisfy the constraints with tracker1 and tracker3. Let's assume that it selects a new value to minimize the number of conflicts: $< 2,3,6 : \tau_1 >$ has one conflict with tracker1, not with tracker3. Tracker2's priority also increases to 2 (which is "max priority of neighbors + 1").

5. Cycle 5

   Tracker1(0 $\rightarrow$ 3):$< 1,2,6 : \tau_1 >$, tracker2(2):$< 2,3,6 : \tau_1 >$, tracker3(1):$< 3,4,7 : \tau_2 >$, tracker5(0):$< 7,10,11 : \tau_1 >$

   - Trakcer1 finds a conflict with tracker2 but it cannot find a value without a conflict (tracker1 is restricted to τ_1). Let's assume it selects a new value $< 1,2,6 : \tau_1 >$. While $< 1,5,6 : \tau_1 >$ is available, we assume the worst case. Tracker1 increases its priority from 0 to 3.

6. Cycle 6

   Tracker1(3):$< 1, 2, 6 : \tau_1 >$, tracker2(2):$< 2, 6, 7 : \tau_2 >$, tracker3(1):$< 3, 4, 7 : \tau_2 >$, tracker5(0):$< 7, 10, 11 : \tau_1 >$

   - Tracker2 has a conflict with tracker1. Since tracker2 cannot find a value at τ_1, it selects a new value $< 2, 6, 7 : \tau_2 >$ that satisfy the constraint with tracker1. The value $< 2, 3, 6 : \tau_2 >$ (that was tried at cycle 2) can be avoided by saving nogoods: nogood is a value combination that cannot be part of any solution.

7. Cycle 7

   Tracker1(3):$< 1, 2, 6 : \tau_1 >$, tracker2(2):$< 2, 6, 7 : \tau_2 >$, tracker3(1 $\rightarrow$ 3):$< 3, 4, 7 : \tau_2 >$, tracker5(0):$< 7, 10, 11 : \tau_1 >$

   - As in cycle 3, tracker3 cannot satisfy the constraints with tracker2 since it is restricted to τ_2. It increases its priority from 1 to 3.

8. Cycle 8

   Tracker1(3):$< 1, 2, 6 : \tau_1 >$, tracker2(2 $\rightarrow$ 4):$< 2, 6, 7 : \tau_1 >$, tracker3(3):$< 3, 4, 7 : \tau_2 >$, tracker5(0):$< 7, 10, 11 : \tau_1 >$

   - As in cycle 4, tracker2 finds a conflict with tracker3. It cannot find a value that can satisfy the constraints with tracker1 and tracker3. It selects a new value $< 2, 6, 7 : \tau_1 >$ that has a conflict with tracker1, but not with tracker3. For the worse case, $< 2, 3, 7 : \tau_1 >$ is avoided here. tracker2's priority also increases to 4.

9. Cycle 9

   Tracker1(3 $\rightarrow$ 5):$< 1, 5, 6 : \tau_1 >$, tracker2(4):$< 2, 6, 7 : \tau_1 >$, tracker3(3 $\rightarrow$ 5):$< 3, 4, 7 : \tau_2 >$, tracker5(0):$< 6, 10, 11 : \tau_1 >$

   - Tracker1 finds a conflict with tracker2 but it cannot find a value that does not conflict with tracker2. Tracker1 selects a new value $< 1, 5, 6 : \tau_1 >$ and increase its priority from 3 to 5.
   - If nogood $\{$tracker1:$< 1, 2, 6 : \tau_1 >$, tracker3:$< 3, 4, 7 : \tau_2 >\}$ is saved at cycle 8, tracker3 finds a match between "its current value & known tracker1's value" and the nogood. Since tracker3 has no alternative choice, it increases its priority and creates a nogood $\{$tracker1:$< 1, 2, 6 : \tau_1 >\}$ for tracker1.
   - Tracker5 finds a conflict with tracker2 for S7 at τ_1. It selects a new value $< 6, 10, 11 : \tau_2 >$ that does not have any constraint violation.

10. Cycle 10

    Tracker1(5):$< 1, 5, 6 : \tau_1 >$, tracker2(4):$< 2, 3, 7 : \tau_1 >$, tracker3(3):$< 3, 4, 7 : \tau_2 >$, tracker5(0):$< 6, 10, 11 : \tau_1 >$

    - tracker2 has a conflict with tracker1 and finds a satisfying value $< 2, 3, 7 : \tau_1 >$.

In contrast, with propagations, 3 cycles are consumed until a solution is found as follows. Note that constraint propagation is interleaved with value selection. That is, it is not pre-processed before search.

1. Cycle 1

   Trackr1(0):$< 2, 5, 6 : \tau_1 >$, tracker2(0):$< 2, 3, 6 : \tau_1 >$, tracker3(0):$< 3, 4, 7 : \tau_2 >$, tracker5(0):$< 7, 10, 11 : \tau_1 >$

2. Cycle 2

   Tracker1(0):$< 2,5,6 : \tau_1 >$, tracker2($0 \rightarrow 1$):$< 2,3,7 : \tau_1 >$, tracker3(0):$< 3,4,7 : \tau_2 >$, tracker5(0):$< 7,10,11 : \tau_1 >$

   - Tracker2 finds a conflict with tracker1. Here, tracker2's domain is reduced to $\{< 2,3,7 : \tau_1 >, < 3,6,7 : \tau_1 >\}$. While both values cannot satisfy the constraint with tracker1, tracker2 selects one of two available values and increases its priority from 0 to 1. Assume that tracker2 selects $< 2,3,7 : \tau_1 >$ (selecting $< 3,6,7 : \tau_1 >$ leads to the same number of cycles).

3. Cycle 3

   Tracker1(0):$< 1,5,6 : \tau_1 >$, tracker2(1):$< 2,3,7 : \tau_1 >$, tracker3(0):$< 3,4,7 : \tau_2 >$, tracker5(0):$< 6,10,11 : \tau_1 >$

   - Tracker1 finds a conflict with tracker2 (now, tracker2 is higher than tracker1). At this cycle, tracker1's domain is reduced to $\{< 1,2,5 : \tau_1 >$, $< 1,5,6 : \tau_1 >\}$. tracker1 selects a new value $< 1,5,6 : \tau_1 >$ that can satisfy the constraint with tracker2.

   - Tracker5 also finds a conflict with tracker2. It selects a new value, $< 6,10,11 : \tau_2 >$, that can satisfy the constraints with its higher agents (tracker1, tracker2, tracker3). Note that tracker5 eliminated the value $< 6,7,11 : \tau_2 >$ at cycle 1.

This example shows that local constraint communication and constraint propagation enables agents to find a solution with less number of cycles.