


2015

# Application of Techniques for MAP Estimation to Distributed Constraint Optimization Problem

Yoonheui Kim

*University of Massachusetts - Amherst*

Follow this and additional works at: [https://scholarworks.umass.edu/dissertations\\_2](https://scholarworks.umass.edu/dissertations_2)

 Part of the [Operational Research Commons](#), and the [Other Computer Engineering Commons](#)

---

## Recommended Citation

Kim, Yoonheui, "Application of Techniques for MAP Estimation to Distributed Constraint Optimization Problem" (2015). *Doctoral Dissertations*. 546.

[https://scholarworks.umass.edu/dissertations\\_2/546](https://scholarworks.umass.edu/dissertations_2/546)

This Open Access Dissertation is brought to you for free and open access by the Dissertations and Theses at ScholarWorks@UMass Amherst. It has been accepted for inclusion in Doctoral Dissertations by an authorized administrator of ScholarWorks@UMass Amherst. For more information, please contact [scholarworks@library.umass.edu](mailto:scholarworks@library.umass.edu).

**APPLICATION OF TECHNIQUES FOR MAP  
ESTIMATION TO DISTRIBUTED CONSTRAINT  
OPTIMIZATION PROBLEM**

A Dissertation Presented

by

YOONHEUI KIM

Submitted to the Graduate School of the  
University of Massachusetts Amherst in partial fulfillment  
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

SEPTEMBER 2015

School of Computer Science

© Copyright by Yoonheui Kim 2015  
All Rights Reserved

**APPLICATION OF TECHNIQUES FOR MAP  
ESTIMATION TO DISTRIBUTED CONSTRAINT  
OPTIMIZATION PROBLEM**

A Dissertation Presented

by

YOONHEUI KIM

Approved as to style and content by:

---

Victor Lesser, Chair

---

Shlomo Zilberstein, Member

---

Andrew Mcgregor, Member

---

Jon Machta, Member

---

Lori Clarke, Chair  
School of Computer Science

## ABSTRACT

# APPLICATION OF TECHNIQUES FOR MAP ESTIMATION TO DISTRIBUTED CONSTRAINT OPTIMIZATION PROBLEM

SEPTEMBER 2015

YOONHEUI KIM

B.S., SEOUL NATIONAL

M.S., UNIVERSITY OF SOUTHERN CALIFORNIA

Ph.D, UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Victor Lesser

The problem of efficiently finding near-optimal decisions in multi-agent systems has become increasingly important because of the growing number of multi-agent applications with large numbers of agents operating in real-world environments. In these systems, agents are often subject to tight resource constraints and agents have only local views. When agents have non-global constraints, each of which is independent, the problem can be formalized as a distributed constraint optimization problem (DCOP). The DCOP is closely associated with the problem of inference on graphical models. Many approaches from inference literature have been adopted to solve DCOPs. We focus on the Max-Sum algorithm and the Action-GDL algorithm that are DCOP variants of the popular inference algorithm called the Max-Product algorithm and the Belief Propagation algorithm respectively. The Max-Sum algorithm

and the Action-GDL algorithm are well-suited for multi-agent systems because it is distributed by nature and requires less communication than most DCOP algorithms. However, the resource requirements of these algorithms are still high for some multi-agent domains and various aspects of the algorithms have not been well studied for use in general multi-agent settings.

This thesis is concerned with a variety of issues of applying the Max-Sum algorithms and the Action-GDL algorithm to general multi-agent settings. We develop a hybrid algorithm of ADOPT and Action-GDL in order to overcome the communication complexity of DCOPs. Secondly, we extend the Max-Sum algorithm to operate more efficiently in more general multi-agent settings in which computational complexity is high. We provide an algorithm that has a lower expected computational complexity for DCOPs even with  $n$ -ary constraints. Finally, In most DCOP literature, a one-to-one mapping between a variable and an agent is assumed. However, in real applications, many-to-one mappings are prevalent and can also be beneficial in terms of communication and hardware cost in situations where agents are acting as independent computing units. We consider how to exploit such mapping in order to increase efficiency.

# TABLE OF CONTENTS

	Page
<b>ABSTRACT</b> .....	<b>iv</b>
<b>LIST OF TABLES</b> .....	<b>ix</b>
<b>LIST OF FIGURES</b> .....	<b>x</b>
 <b>CHAPTER</b>	
<b>1. INTRODUCTION</b> .....	<b>1</b>
1.1 Introduction .....	1
1.2 Action-GDL and Max-Sum and MAP estimation .....	4
1.3 Motivating Examples .....	6
1.3.1 Meteorological Command and Control .....	6
1.3.2 Sensor Network .....	9
1.4 Optimizing Message Passing Algorithms for DCOPs .....	12
1.4.1 Applying Inference Technique to Reduce Communication Overhead of DCOP .....	13
1.4.2 Applying Inference Technique to Reduce Computational Burden in DCOPs .....	14
1.4.3 Exploiting the Mapping of Agents and Variables .....	15
1.5 Contributions .....	17
1.6 Thesis Organization .....	18
<b>2. LITERATURE REVIEW ON DCOP ALGORITHMS</b> .....	<b>19</b>
2.1 Exact Algorithms .....	20
2.2 Approximate Algorithms .....	24
2.3 Summary .....	32

<b>3. APPLYING INFERENCE TECHNIQUE TO REDUCE COMMUNICATION OVERHEAD OF DCOP</b> .....	<b>33</b>
3.1 Background .....	33
3.1.1 AND/OR Search Tree and Context-minimal AND/OR Search Graph .....	34
3.1.2 Distributed Constraint Optimization and Junction-Tree .....	37
3.1.3 DJAO(k) .....	38
3.1.3.1 First phase: heuristics generation .....	38
3.1.3.2 Second phase: search on AND/OR junction graph .....	39
3.1.3.3 Search in distributed settings .....	43
3.1.3.4 DJAO on AND/OR search junction graph .....	43
3.1.3.5 Bounds on partial solution .....	44
3.1.3.6 DJAO(k) .....	45
3.1.3.7 Approximate DJAO(k) .....	50
3.2 An Example of DJAO .....	50
3.2.1 A Simple Example of DJAO .....	50
3.2.2 A More Complicated Example of DJAO .....	54
3.3 Empirical Evaluation .....	57
3.4 Conclusions .....	61
<b>4. APPLYING INFERENCE TECHNIQUE TO REDUCE COMPUTATIONAL BURDEN IN DCOPS</b> .....	<b>63</b>
4.1 Background .....	63
4.2 Fast Belief Proagation for N-ary DCOP .....	65
4.2.1 G-FBP .....	67
4.2.2 G-FBP Algorithm with Partial Lists .....	69
4.2.3 Time Complexity and Selection of K .....	69
4.2.4 Independence Assumption and Correlation Measure .....	73
4.3 Experiments .....	75
4.3.1 Random Graphs .....	75
4.3.2 Multiagent Radar Coordination Domain .....	77
4.4 Conclusion .....	80
<b>5. EXPLOITING THE MAPPING OF AGENTS AND VARIABLES</b> .....	<b>81</b>



5.1	The Configuration of the Semi-centralized Structure and NetRad domain .....	81
5.2	Exploiting Semi-Centralization .....	82
5.2.1	Using Organization Structure .....	84
5.2.2	Starting with Known Policy .....	86
5.2.3	Using the Structure for Policy Generation .....	88
5.3	Experimental Results .....	90
5.3.1	Experimental Setting .....	90
5.3.2	Performance of Max-Sum on NetRad .....	91
5.3.3	Starting with Initial Policy .....	93
5.3.4	Performance of Max-Sum in a Two-Level Hierarchy .....	94
5.4	Conclusion .....	95
<b>6.</b>	<b>CONCLUSIONS AND FUTURE RESEARCH .....</b>	<b>101</b>
6.1	Contributions .....	101
6.2	Future Research .....	103
	<b>BIBLIOGRAPHY .....</b>	<b>106</b>

## LIST OF TABLES

Table	Page
3.1 Performance of Optimal DJAO(k) on Random Binary DCOP Instances .....	58
3.2 Sensor Network Instances .....	60

## LIST OF FIGURES

Figure	Page
1.1 Graph Coloring Example and Two constraint graph representation . . . . .	2
1.2 A result of radars' scan in the NetRad system . . . . .	7
1.3 Sensor Network example and two constraint graph representations for EAV and PEAV formulations . . . . .	11
3.1 an AND/OR search graph . . . . .	36
3.2 an AND/OR search graph based on a junction tree . . . . .	40
3.3 Example of junction tree and the constraint functions . . . . .	50
3.4 Merged constraint function in $A_2$ . . . . .	51
3.5 Messages in Action-GDL . . . . .	51
3.6 Filtered Messages in DJAO . . . . .	51
3.7 The resulting function at $A_1$ (root agent) . . . . .	53
3.8 Updated utility function at the root after the search . . . . .	53
3.9 Example of junction tree and the added constraint functions . . . . .	54
3.10 Messages from $A_4$ to $A_3$ . . . . .	55
3.11 Local function and messages combined for $A_3$ . . . . .	55
3.12 Message before and after filtering from $A_3$ to $A_1$ . . . . .	55
3.13 The resulting function at the root . . . . .	56
3.14 The resulting function at the root . . . . .	57

3.15	The resulting function at the root . . . . .	57
3.16	Performance of Approximate DJAO(k=10) . . . . .	59
4.1	Example of FBP technique. The largest item 15 of $\mathbf{v}_a$ that has index 6 is summed with 3 in $\mathbf{v}_b$ with the same index (which maps to specific value combination of variables). Therefore, items with value smaller than 3 in $\mathbf{v}_b$ can be ignored as any value smaller than 3 cannot yield a value larger than (15 + 3). We also limit the items smaller than 11 in $\mathbf{v}_a$ by applying the same idea. In this example only 2 computations are required to compute the maximum value using this technique. . . . .	64
4.2	Example of Message List Generation. Each value in boxes denotes individually sorted message values from the variable nodes to the function nodes. The domain size is 2 for $v_b$ and it is 3 for $v_a$ and $v_c$ thus the message size. In order to compute the message to $v_a$ , messages from $v_b$ and $v_c$ ( $q_{b \rightarrow 1}$ and $q_{c \rightarrow 1}$ respectively) are summed to generate the partial message list [15, 11] instead of the complete list [15, 11, 8, 8, 4, 1] with $ L_a  = 2$ . . . . .	68
4.3	Example of G-FBP technique as in Algorithm 1 . . . . .	71
4.4	The computation time of Max-Sum(MS), Max-Sum with G-FBP(MS+G-FBP) and Max-Sum with G-FBP with correlation measure(selective MS+G-FBP) . For Figure(a), the domain size of 10 was used. Datapoint (5, 3.6) is omitted to see the general trend. The performance of algorithms was (507178.5, 55070.6, 195056.9) for MS, MS+G-FBP and selective MS+G-FBP respectively. For Figure(b) the arity setting of (3, 2.8) was used. . . . .	76
4.5	(a) The computation time ratio of MS+G-FBP and MS+GSC-FBP to MS. K value is in brackets. (b) Time taken at each cycle. $K = 11$ . . . . .	79
4.6	Performance improvement using correlation measure. $K = 11$ . . . . .	79
5.1	An example structure of the NetRad domain where an agent owns multiple variables: An MC&C controls and manages multiple radars. . . . .	82
5.2	2-level Hierarchy Scheme . . . . .	89
5.3	Radar 1 (R1) can choose to scan Event 1 (Ev1), Event 2 (Ev2) or to scan both depending on the utility. Scanning all phenomena in a radar's range of sufficient quality may not be possible given the time limit to scan. . . . .	90

5.4	Gen:Genetic, <b>MS</b> :Max-Sum, <b>Neg</b> :Negotiation. The algorithm is run with the same number of tasks (weather phenomena) as the number of radars. The Genetic algorithm is run with a computation time limit of 10 minutes. We set the time limit to 10 minutes in order to get reasonable optimization. Given less than 10 minutes, the utility generated by the centralized optimization were significantly lower than the other approaches. ....	96
5.5	Experiment with different number of phenomena. The basis is 48 weather phenomena and this is increased to 120 phenomena (i.e. 2.5). ....	97
5.6	The results using initial policy (a) The decentralized computation time of Max-Sum including policy generation time (b) T-test result on hypothesis that each policy improves the computation time of regular Max-Sum with $\alpha = 0.05$ (c) Value convergence trend at each round (d) Number of Max-Sum rounds ....	98
5.7	Performance of MS-Init. <b>MS</b> :Max-Sum, <b>Neg</b> :Negotiaton, <b>MS-Init</b> : Max-Sum with IMS ....	99
5.8	Performance of MS2L ....	100

# CHAPTER 1

## INTRODUCTION

### 1.1 Introduction

In cooperative multi-agent systems, agents make local decisions that attempt to maximize the overall performance of the system subject to certain constraints and limitations. The decisions that agents make based solely on their local views are often suboptimal for many applications. Thus, coordination among agents is crucial to arrive at optimal decisions or near-optimal decisions. This coordination problem has been of primary concern to the multi-agent system field since its inception [1, 2]. There has been a vast body of literature that has focused on developing algorithms which generate strategies where multiple agents act together towards a common objective. Of late, many researchers are framing this coordination problem as a distributed constraint optimization problem (DCOP) in which there are multiple relations among agents and each relation is independent of each other and involves a number of agents [3].

Distributed constraint optimization has been a popular approach among a wide group of researchers due to its straightforward representation as a graphical model and the abundant literature on inference on graphical models that can be applied to it. A DCOP consists of variables, the constraint functions among the variables, and the agents who own a subset of variables. The goal of the DCOP is to find a set of variable values (decisions) that maximizes the sum of the constraint functions. DCOP have been successfully applied to distributed task allocation [4], sensor coordination [5, 6], and coalition formation [7]. However, distributed constraint optimization is NP-

hard [3], and the computational complexity and the communication overhead required by many complete algorithms make them unusable for real world systems [8, 9]. Thus, our emphasis in this thesis will be on complete and approximate distributed constraint optimization algorithms that use much less resources than traditional approaches.

Formally, a DCOP can be defined as a tuple  $\langle \mathbf{A}, \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$ .

- $\mathbf{A} = \{A_1, \dots, A_k\}$  is a set of agents,
- A set of variables  $\mathbf{X} = \{X_1, \dots, X_r\}$ , where each variable has a finite domain (maximum size  $N$ ) of possible values that it can be assigned.
- $\mathbf{D} = \{\mathbf{D}_1, \dots, \mathbf{D}_r\}$ , where a variable  $X_i$  can take a value  $x_i$  in the domain  $\mathbf{D}_i$ ,
- A set of constraint functions  $\mathbf{F} = (F_1, \dots, F_k)$ , where each constraint function,  $F_j : \mathbf{X}^j \rightarrow \mathfrak{R}$ , takes as input any setting of the variables  $\mathbf{X}^j \subseteq \mathbf{X}$  and provides a real valued utility.

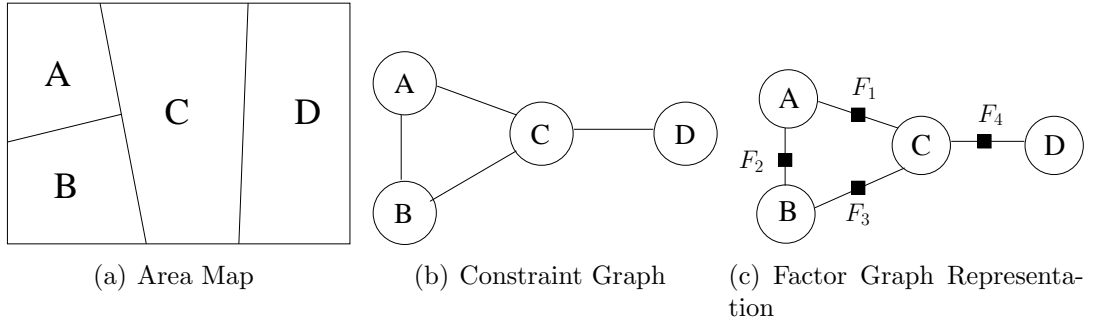


Figure 1.1: Graph Coloring Example and Two constraint graph representation

In a DCOP, we assume that each agent exclusively owns one or more variables and determines the value assignment on its owned variables. An agent only knows about the constraint functions associated with its variables. The DCOP problem can be represented using a *constraint network* as shown in Figure 1.1, where there is a node corresponding to each variable  $X_i$ . There is an edge (hyper-edge) for each constraint  $F_j$  that connects all the variables that are involved in the function  $F_j$ . In the graph

coloring example where adjacent areas are supposed to be in different colors, each area is represented as a node and the node can take a color as its value. Because two adjacent areas have a constraint to be colored differently, the nodes that represent these areas are connected by an edge. In the factor graph notation, constraints are represented explicitly as nodes called factor nodes and are shown as black squares in Figure 1.1(c).

The objective in the DCOP is to find the complete variable configuration  $\mathbf{x}$  that maximizes  $\sum_j F_j(\mathbf{x}^j)$ . This problem is equivalent to a special subclass of the well studied problem of MAP estimation in graphical models [10]. DCOP finds the agents' action choices given the current constraint relations on variables whereas MAP estimation problem finds the variable assignment that is most likely given the observation. In other words, DCOP is equivalent to a distributed MAP estimation where no observation is given. Due to these similarities, several message-passing algorithms for MAP estimation such as Max-Product [11] have been adapted to DCOPs. The examples include the Max-Sum algorithm [12] and the Action-GDL algorithm [13].

However, these algorithms are not practical for some DCOP domains due to their high computational and communication complexity. Moreover, DCOP is designed for a distributed environment where each variable is owned by an agent and these adapted algorithms do not exploit these domain characteristics, which leaves room for improvement. Since MAP estimation has been an active research fields for various type of problems, there have been many techniques developed for various environments and adopting these techniques to DCOP seems a natural step to improve DCOP algorithms. However, the MAP estimation algorithms using message passing are not directly applicable to these environment without modifications for two reasons. First, even though these MAP estimation algorithms using message passing can be easily transformed into a DCOP algorithm, this converted algorithm may require a large number of messages or large-sized messages making them not applicable for



real distributed applications with low communication bandwidth. Secondly, MAP estimation algorithms are designed for a repeated usage given a single fixed model and different observations, whereas in DCOP, the model is constructed to be solved once given the relations on variables. Therefore, preprocessing techniques that are applied offline cannot be directly applied to the DCOP algorithms.

In summary, this thesis studies the issues that arise when we adapt inference techniques to resource-constrained multi-agent domains. Our goal is to improve the efficiency of inference-based DCOP algorithms by exploiting the different characteristics of multi-agent domains and the different problem structures in the domains. Specifically this thesis investigates the following questions using the Max-Sum algorithm and the Action-GDL algorithm as prototypical examples:

- How can the most important information be selected to minimize communication for DCOPs in environments with scarce communication resources?
- How can the offline preprocessing step in inference algorithms be avoided in the algorithms for one-shot DCOP problems?
- How can the situation where agents manage multiple variables be exploited in terms of reducing communication and computation resources needed to solve DCOP?

## 1.2 Action-GDL and Max-Sum and MAP estimation

Max-Sum [12] and Action-GDL[13] are message-passing DCOP algorithms belonging to the class known as Generalized Distributive Law (GDL) [14]. Max-Sum and Action-GDL are simple variations of the Max-Product algorithm where the global utility function is maximized.

Action-GDL produces the optimal solution for any graph. Action-GDL performs message-passing on the tree structure [15] corresponding to the DCOP problem. In this graph, cliques which contain a set of variables are associated with constraint

functions. The algorithm consists of two phases, utility propagation and value propagation. The utility message  $\mu_{ij}$  from clique  $i$  to clique  $j$  :

$$\mu_{ij}(\mathbf{s}_{ij}) = \max_{\mathbf{s}_{ij}} \left[ F(\mathbf{x}_i) + \sum_{k \in \mathbf{C}_{ch}} \mu_{k \rightarrow i}(\mathbf{s}_{ik}) \right], \quad (1.1)$$

where  $\mathbf{s}_{ij}$  contains the variables that exists in both clique  $i$  and  $j$ , and  $x_i$  contains variables in clique  $i$ , and  $\mathbf{C}_{ch}$  contains the child cliques of clique  $i$ . This algorithm produces an optimal solution but requires messages with exponential size, thus limiting its applicability to real distributed domains.

Max-Sum produces the optimal solution in an acyclic graph or a good approximate solution in a cyclic graph [11]. The Max-Sum algorithm iteratively performs message-passing on the factor graph [16] corresponding to the DCOP problem. In this graph, there is a variable node for each variable and there is a factor node for each constraint function. A function node is connected to a variable node if the corresponding constraint function contains that variable in its domain. The messages exchanged in Max-Sum are of two types:

The message  $q_{i \rightarrow j}$  **from Variable  $i$  to Function  $j$**  :

$$q_{i \rightarrow j}(x_i) = \alpha_{ij} + \sum_{k \in \mathbf{M}_i \setminus j} r_{k \rightarrow i}(x_i) \quad (1.2)$$

$\alpha_{ij}$  is a scalar set such that  $\sum_{x_i} q_{i \rightarrow j}(x_i) = 0$ , and  $\mathbf{M}_i$  contains the indices of function nodes connected to variable node  $i$ .

The message  $r_{j \rightarrow i}$  **from Function  $j$  to Variable  $i$** :

$$r_{j \rightarrow i}(x_i) = \max_{\mathbf{x}_j \setminus i} \left[ F_j(\mathbf{x}_j) + \sum_{k \in \mathbf{N}_j \setminus i} q_{k \rightarrow j}(x_k) \right], \quad (1.3)$$

where  $\mathbf{N}_j$  contains the indices of variable nodes connected to the function node  $j$  in the factor graph.

The objective of Maximum a posteriori estimation is to find the values of variables that maximize the sum of the node and edge potentials. In junction tree algorithm

and loopy belief propagation, algorithms that solve MAP estimation, on a pairwise graphical model, computing a message  $m_{A \rightarrow B}$  between two neighboring cliques  $A = (\mathbf{i}, \mathbf{j})$  and  $B = (\mathbf{i}, \mathbf{k})$  is equivalent in complexity to solving

$$m_{A \rightarrow B}(y_i) = \Psi_i(y_i) + \max_{y_j} [\underbrace{\Psi_j(y_j)}_{\mathbf{v}_a} + \underbrace{\Phi_{i,j}(y_i, y_j)}_{\mathbf{v}_b}], \quad (1.4)$$

where  $\Phi_{i,j}(y_i, y_j)$  is the edge potential and  $\Psi_i(y_i)$  is the sum of node potential  $\Phi(y_i|x_i)$  and any first-order messages over  $y_i$ , that is, the sum of the values only related to  $y_i$  given the observation  $x_i$  (similarly for  $\Psi_j(y_j)$ ).

For a DCOP, the values of node potential and value potential in the equation 1.2, are fixed given the values of  $y_i$  and  $y_j$  because no observation is considered in DCOP and therefore, Max-Sum and Action-GDL are solving a distributed MAP estimation where no observation is given or assumed.

### 1.3 Motivating Examples

In order to motivate the thesis from a practical perspective, this section describes two examples of application domains where the techniques developed in this thesis will be evaluated.

#### 1.3.1 Meteorological Command and Control

NetRad detects and predicts hazardous weather phenomena using a network of radars to sense the atmosphere [17]. The goal of NetRad is to detect a tornado within 60 seconds of formation and to closely track its centroid. Netrad consists of a dense network of small adaptive radars, where radars' scanning ranges overlap. The radars collaborate with each other to sense the same volume of the atmosphere as some measurements on weather phenomena can only be performed using multiple

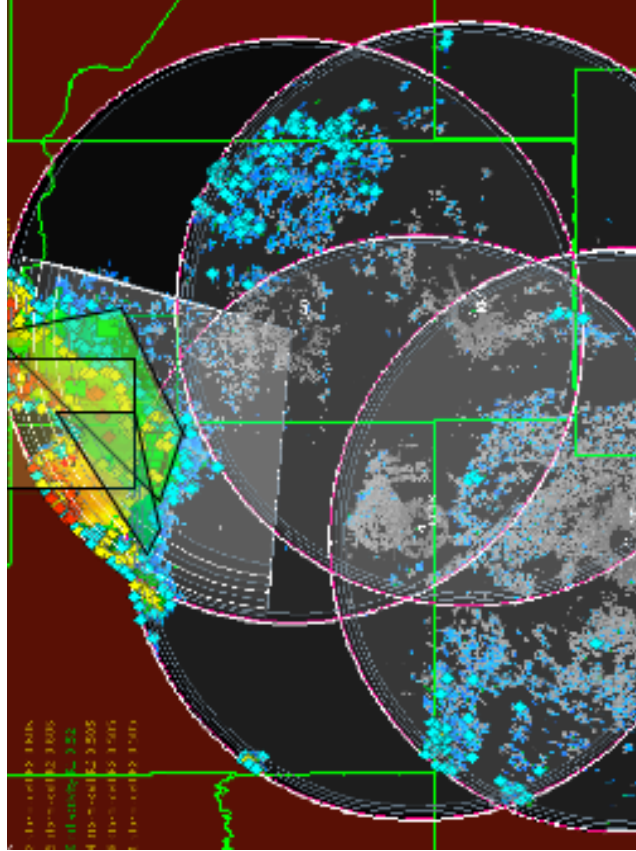


Figure 1.2: A result of radars' scan in the NetRad system

radars [17] or to avoid redundant scanning at the same volume by multiple radars where it is not needed.

A network of meteorological Command and Control (MC&C) [18] agents are the main controlling components of NetRad. Each MC&C is responsible for controlling one or more radars. The MC&C agent ingests data from its associated radars, identifies meteorological features in this data, reports important features to end-users, and determines each radar's future scan strategy based on detected features and end-user requirements. The MC&C agent dynamically schedules the radars in order to adjust to the weather phenomena at each moment and to meet changing end-user requirements. The system works on a cycle of 60 seconds and in each cycle, the system gathers and processes the sensed data and schedules the radars for the next cycle.

The system requires careful planning to get the maximum coverage on the weather phenomena for several reasons. First, each cycle is too short to cover the full range of each radar, and the radars are able to scan only a limited spatial volume at each cycle. Second, there are pinpointing tasks that require the coordinated scanning by multiple radars of the same region and they produce zero utilities if the requirements on the number of concurrently scanning radars are not met. Third, each phenomenon has a different degree of importance (for example, storms and tornadoes have higher priorities than other phenomena), thus each has a different maximum total utility.

This application domain can be formulated as a DCOP in the following way: the radars are mapped to variables, an MCC to an agent, and a phenomenon to a constraint. The domain of a variable is the corresponding radar’s possible scanning strategies; for a discrete domain, scanning strategies are limited to those that contain more than one phenomena. A constraint function associated with a phenomenon is defined over the variables associated with the radars in whose range is the phenomenon. The value of the constraint functions are determined by the quality of the scans of the associated radars. The objective of this problem is defined as finding the radar scan strategy that maximizes the sum of utility functions for all phenomena in the system [5].

More formally, the goal of the system is to find a radar scan strategy  $r_1, \dots, r_n$  which maximize the sum  $U$  of the utilities for all phenomena and represented as,

$$U = \sum_j u_j(\mathbf{r}^{p_j}) \times w_j \tag{1.5}$$

where  $u_j$  denotes the utility for phenomena  $p_j$ . For each phenomena  $p_j$ , the weight  $w_j$  is a constant determined by the requested user or the weather pattern and  $\mathbf{r}^{p_j}$  denotes the scanning policy of radars which have phenomena  $p_j$  in range. The local utility function  $u_j$  works as a constraint function with parameter  $\mathbf{r}^{p_j}$ . Since these variables have finite domains and constraints, the radar scheduling problem can be

modeled naturally as a distributed constraint optimization problem with local functions involving subsets of variables.

The constraint graph in the radar domain has a special structure common in multi-agent domains where variables and constraints represent physical entities. In the radar domain, the scanning range of each radar is finite, and only spatially closely located radars can scan a shared location. Therefore, a constraint created for each phenomena is connected to a finite number of variables and the combination of variables that a constraint can connect to is limited to sets of variables that share scannable locations. Also, this domain can be easily extended to more complex domain such as one involving n-ary constraints, which this thesis addresses. In the dense network of radars, a phenomenon that can be scanned by two or more radars forms an n-ary constraint.

In this domain, a weather phenomena may span across radar ranges of multiple MC&Cs. Coordination between multiple MC&C requires communication which is costly. However, communication for coordination among radars within a single MC&C has no cost.

### 1.3.2 Sensor Network

The second application domain we consider is a sensor network domain in [19], where sensors are coordinated in order to make observations. The sensors are located on each side of the corridors with a certain interval as shown in Figure 1.3(a) where green circles denote sensors along the corridor. Sensors on all four corners of one location must be focused on this location for them to sense objects in it. The sensors determine the direction to focus at a given time in order to coordinate observation with highest rewards. It is a specialized case of DiMES framework (Distributed Multi-Event Scheduling). DiMES is a framework to model the problems occurring in real-world domains involving joint activities.

In the DiMES framework, a set of resources  $R$  is assigned to an event set  $E$ . Time domain is represented as time interval as  $[T_{earliest} + (t - 1)\delta, T_{earliest} + t\delta]$  where  $\delta$  is the duration of each time slot. We assume equal-length time slots without a loss of generality. A  $k$ -th event  $E^k$  is characterized with three elements,  $(A^k, L^k, V^k)$ , the set of resources, the number of contiguous time slots for which the resources  $A^k$  are needed, and the value vector which describes the heterogeneous importance of an event to the resources respectively.  $V_n^k$  is an element of  $V^k$  which denotes the value per time slot to the  $n$ -th resource for scheduling event  $E^k$ . We assume that a resource cannot schedule two events simultaneously and the value of scheduling an event is independent of the time the event is assigned.

A schedule  $S$  is defined as a mapping from the event set to the time domain. We assume the event is not disjoint, i.e., event  $E^k$  must be scheduled in  $L^k$  contiguous slots. The event is *scheduled* when all resources in  $A^k$  agree to assign the time slots to event  $E^k$ . A schedule is *conflicted* if two events with at least one common resource are scheduled in a manner such that assigned time slots overlap. The utility of a resource is the sum of the values from scheduled events. The DiMES problem is NP-hard.

Three DCOP formulations with binary constraints have been suggested in [19] for DiMES problem: time slots as variables (TSAV), events as variables (EAV), and private events as variables (PEAV). We describe the two commonly used formulations EAV and PEAV among them. In these formulations, the variables in DCOP are defined for each event and each variable can take on a valid starting time slot so that the starting time is early so that the remaining time is enough for the event to finish. PEAV formulation is identical to EAV except that valuation information for each resource is only known to agents representing the resource's interests. This privacy requirement leads to additional variables that represents each resource. Conserving privacy is represented as internal links within variables pertaining to the specific resource and the valuation information is only on these internal links.

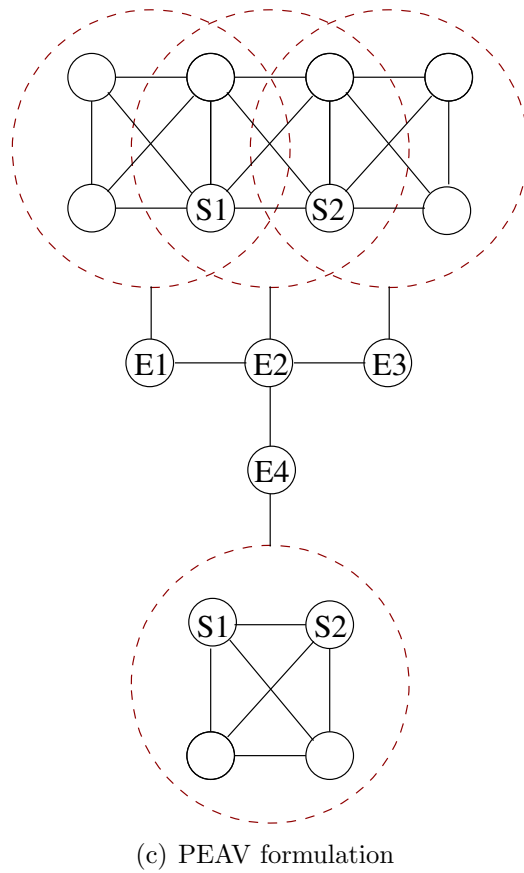
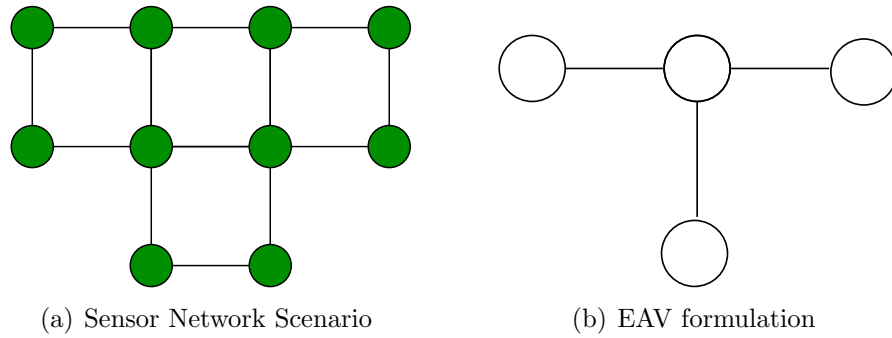


Figure 1.3: Sensor Network example and two constraint graph representations for EAV and PEAV formulations



In this thesis, the sensor network domain is formulated with PEAV formulation where variables are represented as an observation of each location. Each constraint is formulated between locations constraining that two adjacent locations cannot be observed at a time; thus, the domain of each variable is possible time slots plus a null value that denotes the event is not scheduled. Because each observation is assumed to be made in one time slot, the constraint of consecutive assignment does not exist.

The Figure 1.3(b) shows a constraint graph of EAV formulation resulted from the scenario in Figure 1.3(a). There exists 10 sensors at each vertex of squares along the corridors. There are 4 locations that needs to be observed. A constraint function is created for each pair of the locations that use the same resource and this constraint function describes a condition that the value of these two variables cannot be identical. For example, a constraint is created between the two adjacent locations as shown in Figure 1.3(b) because the sensors needed for observation on these locations partially overlap. The constraint function is defined such that a penalty is charged if the two variables have the same value, i.e. the observations at the adjacent locations happens at the same time. Taking this example to PEAV adds variables for each sensors in order to have the valuation information for each observation on the internal links only. Figure 1.3(c) shows this more complicated constraint graph with sensor variables. The connection between sensor variables to the event variables are represented as the connection between dotted circles and the event variable for simplicity. In addition, sensor variables  $S1$  and  $S2$  are drawn twice for simplicity as well. These examples based on DiMES framework create a moderate sized DCOPs that can be solved using exact DCOP solvers.

## 1.4 Optimizing Message Passing Algorithms for DCOPs

In the next three subsections, we briefly outline the research that we have completed as part of the thesis.

### 1.4.1 Applying Inference Technique to Reduce Communication Overhead of DCOP

We start with problems that arise when we apply inference techniques to DCOP environments with constrained communication resources. DCOP algorithms can be categorized into inference-based and search-based algorithms. Action-GDL [13] and ADOPT [3] are some of the most well-known exact algorithms for each category. Search-based algorithms are efficient in that the search can be terminated without searching the full space whereas, inference-based algorithms can reduce the number of messages by sending the aggregated utility. However, the message size of Action-GDL is exponential whereas the number of messages are exponential in ADOPT and a framework that have a reasonable size messages and number of messages is desired.

The search space for graphical models [20] is constructed using independence relations in the models to efficiently search the solution space. Therefore, we develop a similar approach that captures benefits from both Action-GDL and ADOPT. We extend the AND/OR search space by Dechter *et al*[20] that exploits independencies encoded by the graphical model. An AND/OR search tree is a search space with additive AND nodes whose subtrees denote disjoint search spaces under different variables in addition to OR nodes in traditional search trees whose subtrees denote disjoint search spaces under values of variables. By using these independencies, the technique reduces the size of search space from  $O(\exp(n))$  to  $O(n \cdot \exp(m))$ , where  $m$  is the depth of the pseudo-tree [21] and  $n$  is the number of variables. DCOP algorithms such as ADOPT [3] and BnB-ADOPT [22] can be viewed as distributed search algorithms on this AND/OR search space.

However, on the search space used with ADOPT and BnB-ADOPT, functions are evaluated only when their scope is fully assigned along the path. The search backtracks to evaluate different variable assignment which occurs among the domain of a single variable at each level. This complete decentralization in value selection

in the distributed setting results in an exponential number of messages in ADOPT. In [23], we introduce AND/OR search graph on a junction tree where each level is associated with each clique in the junction tree in a DCOP (See Fig. 2), consequently yielding a more compact search graph with a lower number of nodes. By using the fact that communication complexity is far greater concern in the domain, we provide a scheme to generate heuristic functions to perform efficient search on this search space. This work was published in AAAI 2014 [23] and will be described in Chapter 3.

#### 1.4.2 Applying Inference Technique to Reduce Computational Burden in DCOPs

We also address the computational complexity of DCOPs for n-ary DCOPs. Since the inception of work on DCOPs in the multi-agent community, most research has focused on developing the algorithms for the simplest form of DCOPs with binary constraints. Although this is a representative class of the problems, various environments which can straightforwardly adopt DCOP formalism cannot be easily represented in this class and extra modeling effort is required. Both example domains described in Chapter 1.3 need n-ary constraints. Although the DCOP with n-ary constraints can be translated into binary constraints, the translation introduces extra variables and an exponential number of constraints and makes the model unnecessarily complex. Also, this translation step, which differs for each domain, is a burden to the system designer who tries to efficiently model the problem structure. Additionally it simplifies the constraint graph structure which can save communication

However, computing on these n-ary constraints requires exponential time and lowering the computational burden on these constraints is essential in using these models. We applied recent work by McAuley et al [24] that uses order statistics of variable configuration and reduces the computational complexity of the task to find the maximum sum of variable values as given in Equation 1.2. Faster maximization computation

was achieved through the offline presorting of the values of local functions based on different variable configurations. By filtering variable configurations that cannot lead to the maximum value, a lower expected complexity is achieved given independence assumptions about the data. Because the objective of the DCOP is to find a set of agents' actions that maximizes the sum of constraint functions, any DCOP algorithm requires the same operation on each node that finds the variable configuration that maximizes the sum of multiple constraint functions.

However, the application of inference techniques to these extended DCOPs is not straightforward because the settings of inference techniques are different from those of multi-agent domains. The computation is done centrally and there is no notion of communication cost. Also, the inference techniques often assume repeated use of the constructed graphical models, which then enables the use of preprocessing steps. Thus, applying these approaches to DCOP environments requires significant extension and modification of inference algorithms. The technique by McAuley et al [24] requires computationally expensive preprocessing which is not feasible in DCOP domains where each model is used only once. Therefore, we provided a modified maximization operator that does not require significant preprocessing and still obtains lower expected complexity than that of the Max-Sum algorithm. We also provide a correlation measure which can be used to selectively apply the scheme relaxing the order statistics independence assumption given in the McAuley's scheme. We experimentally show the advantage of the approach over the standard Max-Sum algorithm. This work was published in AAMAS 2013 [25] and will be described in Chapter 4

### **1.4.3 Exploiting the Mapping of Agents and Variables**

Finally, We address issues regarding agents in multi-agent domains where agents are associated with real hardware. Agents in DCOPs in real distributed applications are individual physical entities with computing units. Consequently, communication

among these agents entails cost. In the system, these agents form a management layer that controls the value of nodes, computes the solutions, and exchanges messages produced during the management process. Thus, the computation within agents is serialized, whereas the computation of agents runs concurrently. Often, in the DCOP literature, a one-to-one mapping between a variable and an agent is assumed for simplicity. However, this one-to-one mapping introduces a communication cost for each message between the nodes and is not beneficial in environments with scarce communication resources. Also, having as many agents as the number of variables increases the hardware cost in the system.<sup>1</sup> Therefore, we consider domains where one agent manages more than one node. This one-to-many mapping brings up interesting questions. Among many, we question how to modify a DCOP algorithm to exploit such a one-to-many mapping to save resources.

In Chapter 5, we studied the problem of exploiting such a mapping. The agents and variables form a partially centralized structure as the agents manages multiple variables in the system. This research resulted in two modifications to the Max-Sum algorithm that exploit the partial centralization imposed by the hardware structure that significantly reduces communication overhead. We developed a variant of the Max-Sum algorithm which starts with a pre-computed policy computed using information on variables within the single agent. Also, we provide a message passing schedule for the Max-Sum algorithm where agents process the local information aggressively before sending out the resulting messages to other agents. We show experimentally these modifications significantly improve the communication and computa-

---

<sup>1</sup>We assume that a processor contains one agent and this is commonly assumed in multi-agent systems because we consider an agent an autonomous entity that can reason independent of other agents. Although there can be a different model that multiple agents map to a single processor, then our question would be to generate an efficient mapping between each processor to the nodes contained in the single processor. In that case, agents are processes and multiple processes are assigned to a single processor. Because the communication among processors are significantly costlier than the communication across processes in a single processor, the mapping from nodes to processors needs to be considered.

tional performance of the Max-Sum algorithm in NetRad radar domain. This work was published in IAT-2010 [5].

## 1.5 Contributions

The work described in this thesis makes important contributions to the state of the art in the algorithms for DCOP. We improve the efficiency and extend the applicability to more complicated settings of DCOP algorithms, the Max-Sum algorithm and Action-GDL algorithm. The existing contributions of this work can be summarized as follows:

- We built a search space that requires much less messages than ADOPT by using junction tree structure unlike the search structure in ADOPT. Also, we develop a two phase search algorithm, which we call DJAO, that constructs a heuristic function using filtering technique for an efficient search on the constructed search space. This algorithm significantly reduces communication overhead without losing any accuracy by avoiding sending all required information as in Action-GDL. This is the first work that combines features of ADOPT and Action-GDL for DCOPs. Because DJAO is a search algorithm, the search can be prematurely terminated within a fixed error bound if desired leading to further reduction in communication.
- We extend the technique that reduces the expected complexity of the operation to find the maximum sum to n-ary DCOP. Also, we provide a correlation measure to determine dynamically the appropriate cases to apply the technique since its efficiency is sensitive to characteristics of the data sets. There is no existing work on lowering the computational difficulty in solving n-ary DCOPs.
- We present a novel message-passing schedule for the Max-Sum algorithm in domains where multiple variables map to a single agent. This technique facilitates effective problem solving through the use of a locally generated pre-computed

policy and two phase propagation on the Max-Sum algorithm and saves communication and computational resources. To the best of our knowledge, this work is the first research that exploits many variables to one agent mapping given by the hardware structure to efficiently solve DCOPs.

## 1.6 Thesis Organization

The remainder of this thesis is structured in the following manner: In Chapter 2, we discuss related research on distributed constraint optimization, including both complete and approximate algorithms. Subsequently, in Chapter 3, we develop an algorithm that combines ADOPT and Action-GDL and provide a scheme to generate heuristics which reduces communication overhead for this algorithm. Next, in Chapter 4, we provide an algorithm that reduce computational burden for n-ary DCOPs. After that, in Chapter 5, we present various techniques that exploits partial centralization imposed by the hardware. We finally summarize the contributions in this thesis and outline future directions in Chapter 6.

## CHAPTER 2

### LITERATURE REVIEW ON DCOP ALGORITHMS

A DCOP is a non-linear optimization problem which involves multiple agents. It is NP-hard [3], thus computationally intensive. There is a communication overhead because the solution is computed in a distributed manner by multiple agents. Because the application domains of the DCOPs are often resource-constrained, there has been extensive research on developing more efficient exact and approximate algorithms. Our focus is on the Max-Sum algorithm, one of the most used approximate algorithms for DCOP and Action-GDL algorithm, one of the best exact algorithms for DCOP. Both approaches are belief-propagation-based algorithms among many. The Max-Sum algorithm is applicable to general DCOPs and is known to produce reasonable solutions for cyclic graphs with minimal communication. The Action-GDL algorithm is an exact algorithm and is known to require only  $(2N)$  messages where  $N$  is the number of agents.

We review exact algorithms and approximate DCOP algorithms in this Chapter. Typically, DCOP algorithms are classified into two categories, search-based algorithms and belief-propagation-based ones (or dynamic programming based). Many search-based algorithms originated from constraint satisfaction algorithms whereas belief-propagation based algorithms originated from inference algorithms for graphical models. Therefore, we first briefly overview these two categories of exact algorithms for DCOPs including Action-GDL and then present approximate algorithms in both classes including the Max-Sum algorithm in the later section. Next, we focus on the variants of the Max-Sum algorithm, one of which algorithms this thesis con-



centrates on. Finally, we overview a new emerging class of algorithms using linear and quadratic programming for DCOPs.

## 2.1 Exact Algorithms

The complete search algorithms distributedly search the solution space sufficiently so that they can guarantee the optimality of the found solution. Firstly, we review search-based algorithms and then belief-propagation based algorithms.

In the main loop of search-based algorithms, a search process is distributedly conducted via messages. The search ends when the agents perceive that the total utility cannot be improved by changing values of local variables.

In OptAPO [26], the search is conducted by multiple mediators which are dynamically selected among agents. Each mediator asynchronously conducts a centralized branch and bound search on the constructed search tree with variables within its scope and finds the optimal assignment of variables within the scope. A mediator controls the values of the variables in its scope. When variables outside the scope (external variables) are in a conflicting relation with the variables inside the scope (internal variables) and this conflict cannot be resolved by changing the values of internal variables, conflicting external variables are added to the scope of the mediator. The mediator then conducts another centralized branch and bound search within the new scope and propagates new values of variables in the optimal assignment to other nodes outside the scope. This local branch and bound search occurs until the termination increasing scope. The messages in OptAPO are exchanged to control this mediation process until each agent finds that it cannot improve the solution. The mediator receives the value assignment and the constraint functions of internal variables and, consequently, the maximum message size does not asymptotically exceed the total constraint function size. The only significant computation is the branch and bound search of the mediator. Because a mediator increases its scope until the

optimal solution is found, the mediator might include all variables in his scope. The mediator exploits the context of previous solution of smaller scope in this case so that the search may quickly terminate; however, this whole process of repetition of checking local solutions and increasing the scope may become equivalent to a centralized branch and bound search in the worst cases.

In ADOPT [3], the search occurs on trees constructed based on partial ordering of variables. Each node participates in a distributed search process by exchanging messages with its parent and children. Limited information is shared among the nodes. Each message contains either values of the neighboring variables or overall cost information of subtree such as an upper bound, a lower bound and a threshold. Each node limits the search space of its subtree by setting its value to a locally optimal value and the search backtracks whenever the bounds cannot be satisfied with the current local values. The size of each message does not exceed  $O(N + D)$  where  $N$  is the the variable domain size and  $D$  is the number of neighbors. The computation required in the search process is the summation over the bounds received from the neighbors and choosing a value assignment that maximizes the benefit over possible values in its variable domain. Thus, the computation in each node is  $O(N + D)$  as well. BnB-ADOPT [22] is a similar version with different search strategy. It uses depth-first search rather than the best-first search of ADOPT and it uses much less computational and communication resources than ADOPT for most DCOP domains. However, the bottleneck of both algorithms is in the number of sent messages. These algorithms work as if each step in the search process is conducted in a different node. As a consequence, the number of messages becomes exponential in the number of variable and the domain size of each variable.

There are also complete dynamic programming-based algorithms such as Action-GDL and DPOP, and they are analogous to the junction tree inference algorithm [27]. Action-GDL constructs a clique tree, DPOP a pseudo tree. They compute solutions

based on received messages on the trees, utilizing dynamic programming. Each message represents a marginalized utility distribution over the domain of shared variables. In both Action-GDL and DPOP, messages are delivered in two passes, one from leaf nodes to the root node and the other from the root node to leaf nodes. Therefore, the number of messages is linear in the number of variables. It is shown that a pseudo tree that DPOP constructs during the execution is a particular junction tree among many possible junction trees in Action-GDL [13]. The computational complexity and the communication overhead is analogous to that of junction tree belief propagation algorithm. The size of each message is exponential in the induced width of created clique trees for action-GDL and in the maximum number of parents of any node plus the children in the pseudo tree for DPOP. It has been shown that constructing a clique tree with the minimum induced width is NP-hard [27]. The computational complexity relates to the size of the clique node where each node goes through each possible combination of variable values within the clique. Although the number of messages of Action-GDL and DPOP is linear in the number of variables, the size of messages can be exponential in the number of variables common in two neighboring cliques. It is reported that the size of total messages can be as large as multi-gigabytes for network of fewer than 100 variables [9].

ADOPT, Action-GDL and DPOP all require a tremendous amount of communication in order to guarantee optimality of their solutions. OptAPO is reported to require less communication than the other approaches, however it still requires a significant number of messages and it is reported that it conducts multiple mediation process when the network gets large and suffers scalability issue [28, 9]. Overall, although the complete search algorithms produce optimal solutions, they in general require too much communication and lack scalability for real practical applications and therefore approaches are developed for reducing this overhead.

There are algorithms that are variants of DPOP and Action-GDL algorithms. These algorithms focus on reducing the communication overhead but not the computational effort. This is due to the fact that one of main challenges in exact belief-propagation based algorithms such as DPOP and Action-GDL is the exponential growth in message size with the linear increase of number of variables in cliques. Some algorithms are developed in order to reduce the message size using function filtering [29]. Function filtering produces a modified function which returns a predetermined value for the values that exceed a certain boundary value. Thus, the value which equals this boundary value in the modified function are not transmitted, and therefore reduces the size of messages in the next iteration. [29] performs multiple iterations of DPOP with increasing message size and terminates when the computed error bound is in an acceptable range. [30] decomposes the constraint function into lower-arity functions summarizing out a set of variables and the communication can be saved by transmitting these lower-arity constraint function values. These approaches perform repetitive operations on the constraint functions thus they are computationally more expensive than the original algorithms though saving communication resources. Additionally, PC-DPOP [28] reduces communication by limiting communication up to a constant by partially centralizing the computation for a group of nodes, which creates messages larger than the specified size in DPOP. It sends utility functions instead of messages limiting message sizes in these cases. However, the message size cannot be smaller than the local constraint functions size and sending constraint functions can be also costly in many domains.

Additionally, MB-DPOP [31] solves a DCOP for domains with a limited cache size. The algorithm identifies cycle-cutset nodes whose values are instantiated throughout the algorithm to limit the maximum cache size and perform a DPOP with fixed contexts on cycle-cutset nodes. Although the algorithm limits the cache size, it iterates over all instantiations. Therefore, the solution quality is the same with DPOP

and the computational and communication overhead in this algorithm is at least identical to the original DPOP.

## 2.2 Approximate Algorithms

Similarly to the optimal DCOP algorithms, approximate DCOP algorithms also can be divided into search-based or belief-propagation based algorithms. Search-based approximation algorithms conduct search on very limited solution subspaces. In many algorithms, agents utilize the value information of a small subset of variables, i.e. their direct neighbors to decide their variable values. In belief-propagation based approximate algorithms, agents perform the utility distribution calculation given the limited information on neighbors' utility distribution. This utility information can be inaccurate due to the cyclic graph structure in the algorithms such as the Max-Sum algorithm. On these cyclic graphs, there exist multiple paths that utility information can be delivered. Thus, some utility information is multiply counted, and leads to inexact utility calculation. On the other hand, message contents can be summarized for saving communication resource generating approximate solutions.

The search-based approximate algorithms find the best possible configuration of a local group of nodes. Two of the most well-known approximate search algorithms are Distributed Breakout algorithm (DBA) and Distributed Stochastic algorithm (DSA) [32]. Both DBA and DSA originate from approximate algorithms for constraint satisfaction problems and were modified into algorithms for constraint optimization problems. In these algorithms, agents constantly update their variable values given the configuration of direct neighbors until convergence without any quality guarantees.

In DBA, a node with the highest number of conflicts in the neighborhood receives the highest priority and gets a chance to improve its value. The node decreases its priority after changing its value or after finding out that it cannot improve the

current solution. Other nodes with the next highest priority get a chance to change at the next iteration. Agents will continue this operation until termination. At each iteration, each node shares the priority information and the value of each variable. Therefore, a fixed communication cost is incurred at each iteration. At each iteration, the nodes with the highest priorities in the neighborhood are selected to perform a computation to choose their variable values which yield the lowest number of conflicts given the fixed values of neighborhood variables. Because DBA may not converge to a solution and may run indefinitely, the algorithm runs either until the convergence or a fixed number of iterations.

In DSA, each node changes the value stochastically when the cost can be improved. DSA does not transmit priority information as in DBA. Thus, there is no communication in an iteration where a node does not exchange a value. As a consequence, DSA often involves less communication than DBA. Like DBA, a node computes and finds the best possible value assignment of its variables given the assignment of the neighborhood variables.

In addition to DBA and DSA originated from approximate algorithms for DisCSP, the approximate version of ADOPT with a quality bound has also been developed [3]. It is a slight variation of ADOPT in that it starts with an acceptable error bound. In this variant, nodes allow an error within the bound and do not try to improve the solution when the cost is in an acceptable range. Therefore, the search space where the cost is within the given threshold is not explored. Even with this allowance, the behavioral aspect of the algorithm remains similar and the algorithm still often requires a significant number of messages and backtracks to parent nodes in the tree many times when the cost exceeds the allowable range.

Another class of search-based approximate algorithms, which has attracted much attention recently, is the DALO algorithm [33] which provides a quality guarantee using conditions on optimality. The optimality analysis on the solutions of DALO

can be constructed. It guarantees the quality of the solution  $q$  in proportion to the quality of optimal solution  $q^*$ , that is  $q = \alpha q^*$ . This optimality is achieved when a fixed subset of variables cannot improve their solution by a single variable value change within the group.

There are many different optimality notions which define different optimality conditions. The first optimality that provides a quality guarantee is k-optimality [34]. A solution that satisfies k-optimality has a property that the utility of the solution cannot be improved by changing variables' values in any k-sized neighborhood. A k-sized neighborhood refers to any group of k variables that are connected to each other through constraints. A quality bound is derived by considering the assignments that 1) exactly k variable values are different from a k-optimal solution and 2) the values of deviated variables from the k-optimal solution equals the variable value in the globally optimal solution. Consequently, for these assignments, constraint functions yield the same values in k-optimal solution or optimal solution depending on whether the associated variables are deviated or not. A relation between k-optimal solution and optimal solution is derived using relations on constraint function values on these assignments. Furthermore, on graphs where we can limit the assignments to those that all deviated variables are connected through constraints, a tighter bound can be achieved. The t-optimality [33] is a condition that provides the guarantee on the solution quality for a locally optimal solution in variable groups which contain all nodes within the distance of  $t$ . For example, t-optimality of  $t = 1$  forms groups of variables which contain all directly connected neighbors of each variable. As a result, the size of each group can be arbitrarily large in t-optimality depending on the connectivity. A star-shaped graph contains one group that includes all variables. The last and the most recent optimality guarantee, C-optimality [35], combines these two criteria and can handle both distance and region size optimality, thus subsumes k- and t-optimality.

The DALO algorithm finds the variable assignment that satisfies one of these properties on variable values. The DALO algorithm is similar to OptAPO in that a leader of groups controls the local search process in each neighborhood. In DALO, the neighborhood does not change as in an earlier version of OptAPO [36] during the algorithm and only guarantees the optimality within the neighborhood. In the main loop of DALO, the algorithm first exchange all the constraint functions within the defined neighborhoods and elects the leader of each neighborhood. Then, the leader continues to centrally compute the optimal solution of its region repeatedly until a local optima is achieved in all regions. Conflicts in variable values are resolved using heuristics that gives priorities to certain leaders to ensure incremental improvement of the solution. Communication happens when a node transfers its variable value to the leaders of all neighborhoods that the node belongs to upon a value change and a leader node sends control messages that prevents simultaneous variable value change in the overlapped neighborhoods. Each leader centrally computes the best variable assignment in the neighborhood assuming the fixed variable values outside the neighborhood and change the variable values in the neighborhood to the values that yield the best result. Thus, the computational complexity can be as high as the total number of possible assignment of the variables in the neighborhood.

Although the solution quality bound is provided by these algorithms, in order to gain a tight bound, a bigger neighborhood is needed. This requirement causes more computation and communication because the computation for each neighborhood is centralized. Moreover, these bounds are often loose. In [35], it was shown that having almost entire graph-sized neighborhood (5 for a graph with 6 variables) only produces  $\frac{2}{3}$  and  $\frac{1}{2}$  for C-optimality and t-optimality respectively. This means the achieved solution is guaranteed to be greater than two third of the optimal solution for C-optimality and a half for t-optimality respectively.



Among belief-propagation based algorithms for DCOP, the Max-Sum algorithm [12] is the DCOP variation of loopy belief propagation algorithm [37]. Loopy belief propagation algorithm is known as a fast and efficient algorithm for many inference problems in practice and so does Max-Sum have these performance characteristics in many multi-agent domains [38, 6, 12, 7]. In the Max-Sum algorithm, a constraint function is represented as a node called a function node. Variable nodes and function nodes form bipartite graphs called factor graphs. The main routine of the algorithm is repetition of computing messages and exchanging the messages between variable and function nodes until values in the messages converge. Each message contains the possible maximum benefit for each value of the variable from the nodes' local view. The Max-Sum produces an optimal solution when the bipartite factor graph forms a tree. When a factor graph has cycles, the optimality of the solution is not guaranteed, however it is known to produce good solutions even with cycles [12, 5]. The size of these messages equals the single variable domain size and the communication overhead is significantly lower than other DCOP algorithms and thus suitable to many resource-constrained applications in multi-agent fields. The function nodes find variables configurations which maximize the constraint function values by iterating all possible combination of variable values connected to these nodes. This procedure determines the computational complexity of the algorithm. Therefore, the computational complexity is higher than the algorithms such as DBA and DSA where variables choose their value given the fixed values of other variables, but much lower than the exact algorithms such as Action-GDL and DPOP that reformulate the constraint graphs into a tree-like structure.

BnB Max-Sum [6] was developed as a computationally more efficient variants of Max-Sum. It performs a branch and bound search in function nodes to reduce the computational complexity of the maximization operator. In each function node, a search tree is constructed with variables associated with the constraint function. The

search tree branches out with values of a variable at each level and on the tree a branch and bound search is performed. Thus, each node may not need to go through all possible combinations of variables and possibly reduce the computational complexity. Because a branch and bound search process estimates the upper and lower bound of a constraint function without instantiating the variable values located in the subtree, constraint functions that can be evaluated only with a subset of variable values are required. This requirement on the constraint function limits the application to many realistic applications with constraint functions that cannot be evaluated with partial variables assignment.

Fast Max-Sum [7] minimizes the size of the solution space of the constraint function significantly in the task allocation domain by reducing the domain size of the variables. In the task allocation domain, the utility of a task is represented as a constraint function of agents' actions. The domain of this function is each agent's action choices. With the underlying assumption that execution of one task does not affect the utility for other tasks, from the perspective of task A, actions other than the actions for task A can be summarized as  $\neg A$ , that is, not executing the task A. Thus, all agents have only two action choices such as  $A$  and  $\neg A$  from task A's perspective. This property significantly reduces the possible number of action combination of multiple agents for each task. However, in some domains, the assumption that every task and action are independent may not be feasible. If an action of an agent affects multiple tasks or different combination of multiple agents' actions yield different results, this technique cannot be applied. For example, in the Radar domain, scanning one phenomenon may affect multiple variables associated with other phenomena when a single radar' scanning range includes those phenomena.

Bounded Max-Sum [8] provides a quality measure which measures the ratio of the solution to the optimal solution. The algorithm modifies a DCOP by removing arcs which produce cycles and then solves the modified problem with the resulting

tree. When it removes each arc, it selects a specific arc that produces the minimal distance from the original problem, thus minimizing the solution quality loss. By using these computed loss from removing arcs, the algorithm computes the distance from the original optimal solution. Although the problem is now solved on the tree, the complexity of solving on the tree remains the same as the original constraint functions are used. Also, there is overhead of selecting which arcs to remove. This step has the same complexity with solving the original problem. Arc removal is a deterministic process and there is no way to adjust the quality bound and the quality bound is bound to a constraint graph. This bound can be only obtained after constructing the tree structure from the resulting constraint graph of DCOPs.

Another variant of Max-Sum is Max-Sum-AD [39] with value propagation step which operates in a tree-like structure by communicating in a pre-defined order. Max-Sum-AD only sends messages to neighbors with a lower priority in one iteration, only to the ones with a higher priority in another iteration. After the message propagation step, the algorithm then propagates variable values in order for agents to select the actual best given the neighboring variables' actual values. Therefore, each agent does not select a value that is most likely to maximize the utility as in the Max-Sum, but a value that actually produces the computed utility.

Tractable higher order potentials [40, 41] are used to reduce the computational complexity of binary Max-Sum where variables only take on two values. By using these special types of constraint functions for the binary Max-Sum, the computation of Max-Sum can be linear in the number of variables in function nodes instead of exponential. For example, a constraint function that satisfies the one and only condition which takes on a value 1 only for one value assignment, the linear complexity can be obtained because the one assignment that takes on the value 1 needs to be identified and others automatically takes a value 0.

Finally, the divide-and-coordinate approach [38] iteratively divides the search space into segments (divide stage) and searches a solution within the subproblem and then coordinates the solution among agents (coordinate stage). Because each agent always solves problems in the context of each segment, the algorithms using this approach do not give an optimal solution. However, the quality of solution for subproblem is always an upper bound of the quality of equivalent subpart of global solution. Therefore, the quality bound of a solution can be computed with the sum of solutions of subproblems and the overall quality. DacSA algorithm [38] uses this divide-and-coordinate approach. In the divide step in DacSA, each agent formulates and solves a subproblem as a linear program concerning its own variables with associated constraints. Constraints on coordination across agents are specified as Lagrangian multipliers. In the coordinate step, the algorithm updates the coordination parameters using the subgradient method.

DeQed [42] is another algorithm that is based on the divide-and-coordinate approach, thus is very similar to DacSa. However, DeQed uses a different encoding of variables. In DeQed, a single variable in the linear program is a vector of  $D \times 1$  dimension. Each variable is a vector that denotes which value the corresponding variable takes in DCOP, i.e. the  $i$ -th value of this vector takes the value 1 if the corresponding variable in DCOP takes the  $i$ -th value and all the rest take 0. This encoding produces a corresponding quadratic program and solving the program is quite the same with DacSa.

Additionally, message passing algorithms using linear programming and quadratic programming for MAP estimation [43, 44] have been developed. Although these are not DCOP algorithms, these have potentials for DCOP algorithms as they are inherently distributed thus scalable and are able to provide bounds on the solution by computing exact solutions of relaxed problems.

## 2.3 Summary

There have been many algorithms developed to solve DCOPs in multi-agent domains. These algorithms can be categorized as search-based or belief-propagation based algorithms depending on the information the messages transfer. Also, they are categorized as exact methods or heuristic methods based on the exactness of the solution. Because of the resource constraints in multi-agent domains, exact methods require too much computation and communication resources and are generally not appropriate for realistic applications. Therefore, there has been much efforts to reduce these resource requirements for both approximate and exact algorithms in these domains.

Among the DCOP algorithms, the belief propagation based algorithms has shown promising performance and interesting characteristics. The Action-GDL algorithm is an exact algorithm which only requires a linear number of messages providing an optimal solution. The Max-Sum algorithm produces good solutions in practice. There has been an intensive research on belief propagation algorithms in machine learning community and many approaches can be readily applied to DCOPs. For these reasons, we have chosen the Max-Sum algorithm and the Action-GDL algorithm as the focus of our thesis for trying to improve and extend it to more complex settings of DCOP technology.

## CHAPTER 3

### APPLYING INFERENCE TECHNIQUE TO REDUCE COMMUNICATION OVERHEAD OF DCOP

This chapter addresses exponential message size in exact algorithms such as Action-GDL and DPOP. It is the first algorithm that combines Action-GDL and ADOPT. It requires much less communication than Action-GDL and DPOP in many domains. Additionally, the algorithm has the anytime property that can be terminated given the allowed error bound.

#### 3.1 Background

The challenges in exact algorithms such as Action-GDL and DPOP lie in exponential message sizes in the induced width of the dual constraint graph. That is, the size of a message between agents is exponential in the number of variables in the separator of two neighboring cliques in the dual graph. This exponential message size can be close to gigabytes for DCOP in realistic applications.

Many researchers have tried to overcome this communication complexity of Action-GDL and DPOP. Several variants of DPOP have been developed for reducing communication. [28] is a variation of DPOP where the computation is partially centralized to avoid sending large messages. However, this approach may centralize a large part of the problem solving. Because the potential function is known to only local agents, potential function needs to be transferred, leading to additional communication. [45] prunes the message size by removing violating configurations when there are hard constraints. This approach exploits hard constraint in order to save communication.

However, this approach does not apply to the domains without such constraints. [29] uses the function filtering approach to reduce the communication overhead. A function filtering estimates the possible lower bound and filters all variable configurations that lead to a smaller value. This approach sends unpruned Action-GDL messages based on the computed lower bound. The search is implicit in that it constantly constructs Action-GDL messages to update the bounds.

Our approach to this problem was to formulate a novel algorithm that uses search to efficiently find a solution with low communication overhead; this algorithm, called distributed junction tree AO search algorithm (DJAO), conducts a distributed search on the AND/OR space built based on distributed junction trees [15]. DJAO operates in two phases. In the first phase, heuristic upper and lower bounds for variable value configurations are created using a bottom-up propagation scheme similar in character to Action-GDL [13]. Except that instead of transmitting values for all configurations, we transmit only the filtered upper and lower bounds of configuration values. The next phase using these heuristics conducts an ADOPT-like [3, 22] search on AND/OR search graph based on the junction tree, which we call AND/OR search junction graph, to find a solution with desired precision. This two-phase strategy reduces overall communication significantly.

### 3.1.1 AND/OR Search Tree and Context-minimal AND/OR Search Graph

AND/OR search space [46] is introduced to exploit independencies encoded by the graphical model. AND/OR search tree is a search space with additive AND nodes whose subtrees denote disjoint search spaces under different variables in addition to OR nodes in traditional search trees whose subtrees denote disjoint search spaces under values of variables. AND nodes decompose the search space in their subtrees under Generalized Distributive Law framework [14]. It reduces the size of DCOP search space from  $O(\exp(n))$  to  $O(n \cdot \exp(m))$ , where  $m$  is the depth of the pseudo-

tree and  $n$  is the number of variables. In connection with DCOP, ADOPT [3] and BnB-ADOPT [22] can be viewed as a distributed search algorithms on this AND/OR search space.

**Definition 1 (AND/OR search tree)**

*Given a COP instance  $P$ , its primal graph  $G$  and a pseudo-tree  $T$  of  $G$ , the associated AND/OR search tree  $S_T(P)$  has alternating levels of OR nodes and AND nodes. The OR nodes are labeled  $X_i$  and correspond to variables. The AND nodes are labeled  $\langle X_i, a \rangle$  and correspond to value assignments in the domains of variables. The root of the AND/OR search tree is an OR node, labeled with the root of  $T$ . The children of an OR node  $X_i$  are AND nodes labeled with assignments  $\langle X_i, a \rangle$ , consistent along the path from the root. The children of an AND node  $\langle X_i, a \rangle$  are OR nodes labeled with the children of variable  $X_i$  in  $T$ . The path of a node  $n \in S_T$ , denoted  $Path_{S_T}(n)$ , is the path from the root of  $S_T$  to  $n$ , and corresponds to a partial value assignment to all variables along the path.*

An example of AND/OR search tree is given in Fig. 3.1(a). Because AND nodes decompose the problem into separate subproblems, variables in different subtrees of an AND node  $n$  are considered independently given the value assignment along the path to  $n$ . The arcs in  $S_T$  are annotated by appropriate labels of the cost functions.

**Definition 2 (label)** *The label  $l(X_i, \langle X_i, a \rangle)$  is defined as the sum of all the cost functions values for which variable  $X_i$  is contained in their scope and whose scope is fully assigned along the path from root to  $n$ .*

**Definition 3 (value)** *The value  $v(n)$  of a node  $n \in S_T$ , is defined recursively as follows: (i) if  $n = \langle X_i, a \rangle$  is a terminal AND node then  $v(n) = l(X_i, \langle X_i, a \rangle)$ ; (ii) if  $n = \langle X_i, a \rangle$  is an internal AND node then  $v(n) = l(X_i, \langle X_i, a \rangle) + \sum_{n' \in succ(n)} v(n')$ ; (iii) if  $n = X_i$  is an internal OR node then  $v(n) = \max_{n' \in succ(n)} v(n')$ , where  $succ(n)$  are the children of  $n$  in  $S_T$ .*



In [20], AND/OR search graph shown in Fig. 3.1(c) was introduced to reduce the size of the search tree in Fig 3.1(a) by merging two nodes that root identical subtrees. **Context-based merge** operation is defined as merging nodes when two AND nodes share same variable assignments on the ancestors of these nodes, that have connections in G to these nodes or their descendants, or two OR nodes share assignments on these nodes and ancestors of nodes, that have connections in G to nodes' descendants.

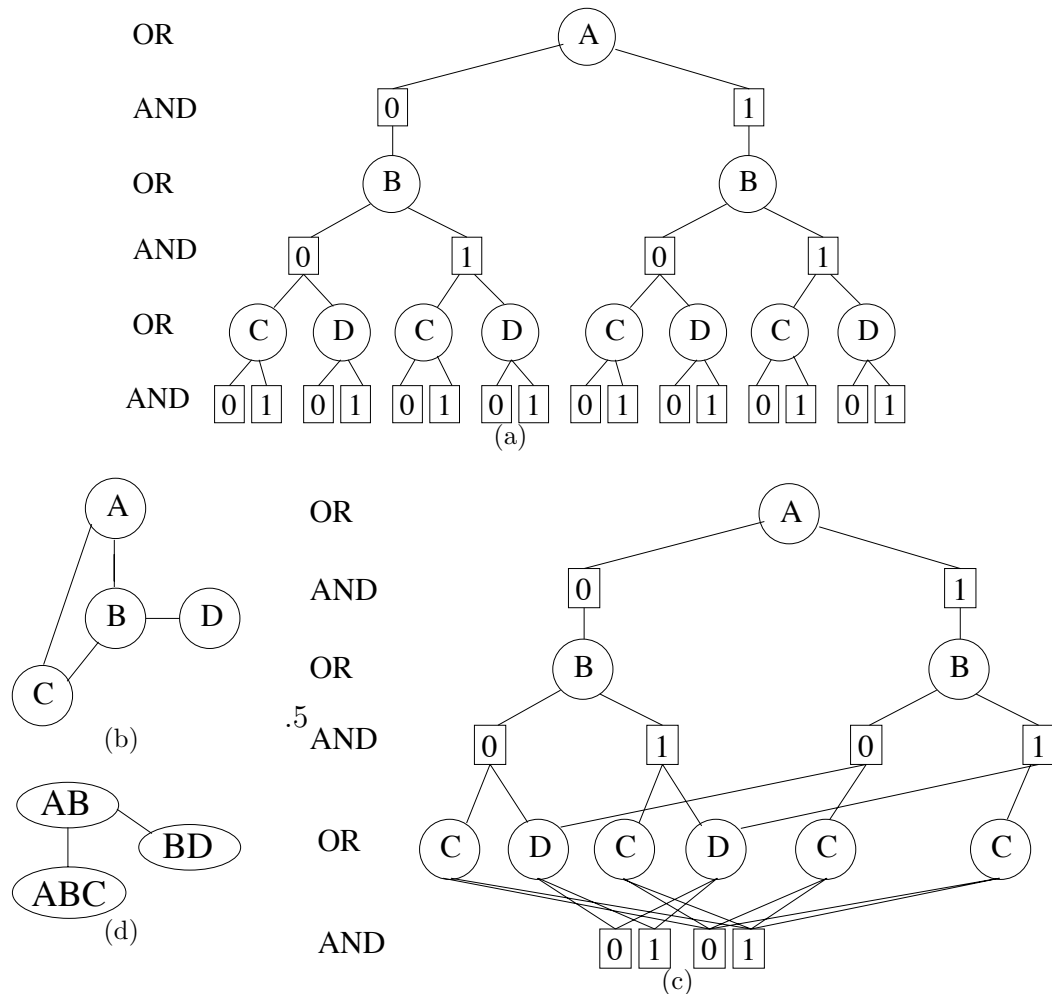


Figure 3.1: an AND/OR search graph

**Definition 4 (context minimal AND/OR graph)** *The AND/OR search graph of  $G$  that is closed under context-based merge operator is called context minimal AND/OR search graph.*

### 3.1.2 Distributed Constraint Optimization and Junction-Tree

A distributed constraint optimization problem (DCOP) instance  $P = \langle \mathbf{A}, \mathbf{X}, D, \mathbf{F} \rangle$  is formally defined by the following parameters:

- A set of variables  $\mathbf{X} = \{X_1, \dots, X_r\}$ , where each variable has a finite domain  $D$  (maximum size  $N$ ) of possible values that it can be assigned.
- A set of constraint functions  $\mathbf{F} = (F_1, \dots, F_k)$ , where each constraint function,  $F_j : \mathbf{X}_j \rightarrow \mathfrak{R}$ , takes as input any setting of the variables  $\mathbf{X}_j \subseteq \mathbf{X}$  and provides a real valued utility.

In DCOP, we assume that each variable  $x_i$  is owned by an agent  $a_i \in \mathbf{A}$  and that an agent only knows about the constraint functions in which it is involved. The DCOP can be represented using a *constraint network*, where there is a node corresponding to each variable  $x_i$  and where there is an edge (hyper-edge) for each constraint  $F_j$  that connects all variables that are involved in the function  $F_j$ .

The objective in the DCOP is to find the complete variable configuration  $\mathbf{x}$  that maximizes  $\sum_{F_j \in \mathbf{F}} F_j(\mathbf{x}_j)$ .

The *dual constraint graph* [21] is a transformation of a non-binary network into a special type of binary network. It contains cliques (or c-variables) domains of which ranges over all possible value combinations permitted by the corresponding constraint functions, and shared variables in any two adjacent cliques have same values.

A junction tree (or join tree) [27]  $T$  is a subgraph of the dual graph which is a tree and satisfies the condition that cliques associated with a variable  $x$  form a connected subset of  $T$ . A *Junction tree* is represented as a tuple  $\langle \mathbf{X}, \mathbf{C}, \mathbf{S}, \mathbf{F} \rangle$ . where  $\mathbf{X}$  is a set of variables,  $\mathbf{C}$  is a set of cliques, where each clique  $C_i$  is a subset of variables  $C_i \subseteq \mathbf{X}$ ;

$\mathbf{S}$  is a set of separators, where each separator is an arc between two adjacent cliques containing their intersection; and  $\mathbf{F}$  is a set of potentials, where each potential in  $\mathbf{F}$  is assigned to each clique in  $\mathbf{C}$ .

A distributed junction tree [15] decomposes a DCOP into a series of subproblems, some of which can be solved in parallel. A subproblem represented as a clique  $c_i \in \mathbf{C}$  can be solved independently given the local constraint functions  $f_i \in \mathbf{F}$  and the values from neighbors on separator  $s_i \in \mathbf{S}$ . Separators  $S$  specify which values will be used in the neighboring cliques in order to compute the solution for its local subproblem.

### 3.1.3 DJAO(k)

#### 3.1.3.1 First phase: heuristics generation

Preprocessing techniques to supply the search with heuristic values has successfully been used to enhance both centralized and decentralized search methods. [47, 48]. In this section we describe a scheme for generating initial heuristic estimates  $h_{UB}$  and  $h_{LB}$  used in  $DJAO(k)$ , based on a new function filtering technique, which we call Soft Filtering, described here. [30, 29] used the Function Filtering technique [29] on DCOPs to prune variable configurations of local nodes, that do not yield the optimal solution. We use the soft function filtering technique to generate heuristics that maintains the tuples that potentially yield the optimal solution while summarizing the rest with upper and lower bounds. Unlike the heuristics in [47, 48] which are generated by solving lower complexity problems than the original, DJAO solves the original problem and focuses on reducing communication by filtering tuples that are unlikely to be part of the optimal solution.

The Soft Filtering technique used in DJAO summarizes constraint functions to reduce communication required for transmitting such function. A simple difference from Function Filtering is that the Soft Filtering technique provides summarized lower and upper bounds on filtered configurations. Let the variable configuration

$S$  in message  $m$  be divided into two sets filtered configurations  $S_F$ , and non-filtered configurations  $S_{NF}$ . Let  $UB$  be the upper bounds of values on variable configurations and  $LB$  the lower bounds.

The values  $UB_m$  and  $LB_m$  in the messages are filtered as follows.

$$UB_m(v) = \begin{cases} UB(v) & \text{if } v \in S_{NF} \\ \max_{S_F} UB(v) & \text{if } v \in S_F \end{cases}$$

$LB_m$  is similarly defined with max replaced with min.

**Example** Consider a function  $F$  with 10 values, from 0 to 9 for the domain  $v_0, \dots, v_{10}$ . Thus,  $F$  has LB and UB such that  $LB(v_i) = UB(v_i)$ . The soft filtered functions  $UB_m$  and  $LB_m$  by 90% of  $F$  is  $UB_m(v_i) = 8, LB_m(v_i) = 0$  for  $0 \leq i \leq 8$ ,  $UB_m(v_9) = LB_m(v_9) = 9$ .

Filtered configurations are summarized as a *filtered* tuple with a single upper and lower bounds, therefore reducing the number of items in each message from  $\|2S\|$  to  $2\|S_{NF}\| + 2$ . The optimal strategy is guaranteed to remain in the search space as no solution is completely dropped. This summarization builds a basis for the next phase where an ADOPT-like search finds a solution within a desired accuracy. Among many ways to select which items to filter, we select items in the bottom  $(100 - d)\%$  of the function range.

### 3.1.3.2 Second phase: search on AND/OR junction graph

On the pseudo-tree based search graph, functions are evaluated only when their scope is fully assigned along the path. The search backtracks to evaluate different variable assignment which occurs among the domain of a single variable at each level. This complete decentralization in value selection in the distributed setting results in the exponential number of messages in ADOPT. Instead, we introduce AND/OR search graph on a junction tree where each level is associated with functions in a DCOP (See Fig. 3.2), consequently yielding a more compact search graph with a

lower number of nodes. This search graph is a context-minimal AND/OR search graph upon construction.

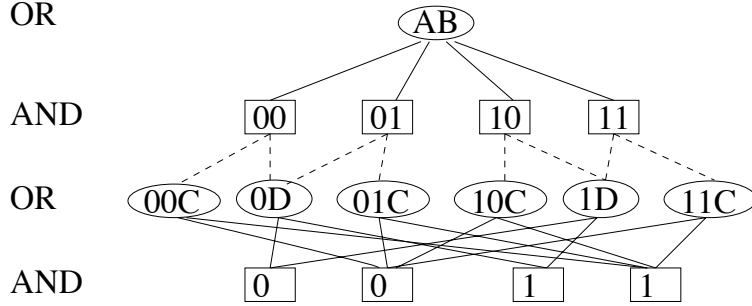


Figure 3.2: an AND/OR search graph based on a junction tree

**Definition 5 (AND/OR search junction graph)**

Given a DCOP instance  $P$  and its junction tree  $T$ , the associated AND/OR search graph  $S_T(P)$  has alternating levels of OR nodes and AND nodes. The OR nodes are labeled  $C_i: \langle \mathbf{S}_i, \mathbf{a}, \mathbf{N}_i \rangle$  where  $\mathbf{a}$  are variables assignments in the domains of variables in separators  $\mathbf{S}_i$  whose value are propagated from ancestors and newly appeared variables  $\mathbf{N}_i$  in clique  $C_i$ . These OR nodes correspond to the cliques with partial assignment. The AND nodes are labeled  $\langle \mathbf{S}_{ij}, \mathbf{b} \rangle$  and correspond to value assignments in the domains of the separator between clique  $C_i$  and its child  $C_j$ . The root of the AND/OR search graph is an OR node, labeled with the root of  $T$ . The children of an AND node  $\langle \mathbf{S}_{ij}, \mathbf{b} \rangle$  are OR nodes who is labeled with  $C_j: \langle \mathbf{S}_j, \mathbf{b}, \mathbf{N}_j \rangle$  with the same assignment on variables in separators.

**Example** Consider the graphical model in Fig. 3.1(b) describing a graph coloring problem over domains  $\{0,1\}$ . An AND/OR search graph based on a possible pseudo-tree is given in Fig 3.1(c) and an AND/OR search junction graph in Fig 3.1(d) is given in Fig. 3.2. Observe that the function evaluation on  $l(\{A, B\}, a)$  occurs at the expansion of nodes at level 3 in Fig. 3.1(c) instead of at level 1 in Fig. 3.2 generating

unguided search until the third expansion. It also elongates the backtrack path leading to an increase in the number of nodes needed to visit to evaluate a single function.

**Theorem 1** *Given a DCOP instance  $P$  and a junction tree  $T$ , its AND/OR search graph is sound and complete. It contains all and only solutions.*

[ Proof: By definition, all the arcs of  $S_T(P)$  are consistent. Also, by definition of junction trees,  $S_T(P)$  contains at least one AND node that assigns values of each variable. Therefore, the assignment that consists of labels of the AND nodes in the solution graph is a solution of  $P$ . Also, by definition of the AND/OR tree, every solution of  $P$  corresponds to a variables assignment in  $S_T(P)$ . Finally, the value  $v(n)$  of a variable assignment is derived by combination of arc values along the corresponding  $APT(n)$ . By construction, each function of  $F$  contribute to one and only one arc value on  $APT(n)$  which matches the variable assignment. Therefore, it yields the cost of a solution.  $\square$

Intuitively, the solution space in AND/OR search graph is identical to the junction tree, thus it contains all and only solutions. Consequently, any search algorithm that traverses the AND/OR search graph in a depth-first manner is guaranteed to have a time complexity equal to the time complexity of Action-GDL [13] on the same junction tree which is exponential in the tree width.

**Theorem 2** *The size of search tree has exactly same size as the total complexity of junction tree as no subtree is redundant. The depth of the graph does not exceed the number of agents.*

The search result for its subtree is stored at each node, therefore no identical subtree is explored twice and a value assignment on a cost function is never repeated. The arcs in  $S_T$  are annotated by appropriate *labels* of the cost functions. The nodes in  $S_T$  are associated with a *value*, accumulating the result of the computation resulted from the subtree below. We define *labels* similar with one defined in Def. 3.

**Definition 6** *label:* The label  $l(C_i \langle \mathbf{S}_i, \mathbf{a}, \mathbf{N}_i \rangle, \langle \mathbf{S}_{ij}, \mathbf{b} \rangle)$  of the arc from the OR node to the AND node  $\langle \mathbf{S}_{ij}, \mathbf{b} \rangle$  is defined as the cost function values contained in the clique  $C_i$  whose scope is fully assigned with values from the parent OR node and child AND node.

The **value** of  $v(n)$  of a node  $n \in S_T(P)$  is computed in the same way as in Def. 3. Likewise, the value of each node can be recursively computed from leaves to root. We can show that:

**Proposition 1** *Given an AND/OR search graph  $S_T(P)$  of a DCOP instance, the value function  $v(n)$  is the maximum cost solution to the subproblem rooted at  $n$ , subject to the current variable instantiation along the path from root to  $n$ . If  $n$  is the root of  $S_T$ , then  $v(n)$  is the maximum cost solution to  $P$ .*

[Proof: By construction, value of each AND node is deterministic given the child nodes and the path  $PSG$  to the node. The value of each OR node is the maximum value of its child nodes, thus unless value of its child nodes has suboptimal value, it chooses the best value. By construction, every possible variable assignment is examined in the subtree, thus the value function  $v(n)$  is the maximum cost function.

Intuitively, We verify the value of nodes are identical to the values produced during the execution of Action-GDL. The *value* of an AND node is identical to the value of corresponding assignments in the messages from the corresponding clique of Action-GDL given the context along the search path to these nodes. Valuation of OR nodes is combination of local utility functions and values of its child nodes and corresponds to the maximum achievable value of variable assignments given the variable assignments along the search path.

**Proposition 2** *AND/OR search junction graph  $S_T(P)$  is context-minimal upon construction*

The separators  $\mathbf{S}_i$  and  $\mathbf{S}_{ij}$  contain variables that build a context for each clique and a single node is created for each value assignment in the separator, thus it is context-minimal.

### 3.1.3.3 Search in distributed settings

Each agent in the system distributedly conducts its share of search for the nodes on  $S_T(P)$  it owns. Agents are responsible for valuation of owned nodes and path determination.

**Definition 7** *agent ownership: Each node in  $S_T(P)$  is owned by an agent. Agent  $A_i$  owns all nodes associated with its own clique  $C_i$ : OR nodes  $C_i : \langle \mathbf{S}_i, \mathbf{a}, \mathbf{N}_i \rangle$  and child AND nodes of these are assigned to  $A_i$ .*

For example, suppose clique  $AB$ ,  $ABC$  and  $BD$  in Fig 3.1(d) are owned by agent  $A_1$ ,  $A_2$ , and  $A_3$  respectively. OR nodes of clique BD are  $C_3 : \langle B, 0, D \rangle, C_3 : \langle B, 1, D \rangle$ . These nodes and child AND nodes of these are assigned to  $A_3$ .

Search procedure between nodes belonging to different agents incurs communication. When a child OR node  $C_i : \langle \mathbf{S}_i, \mathbf{a}, \mathbf{N}_i \rangle$  is chosen for expansion, agent transmits partial assignments  $\mathbf{a}$  to an agent who owns the child nodes. Updated function values are sent to the agent who owns the parent node on the search path when the search backs up. Search paths that incur communication are displayed as dotted lines in Fig. 3.2.

### 3.1.3.4 DJAO on AND/OR search junction graph

If each node  $n \in S_T(P)$  is assigned a *heuristic lower-bound estimate*  $LB(n)$  and *heuristic upper-bound estimate*  $UP(n)$ , then we can calculate the lower and upper bound estimates of assignments and dominated search space can be pruned.



### 3.1.3.5 Bounds on partial solution

Similarly to [46], a partially expanded search graph, denoted as  $PSG$ , contains the root node, will have a *frontier* containing all the nodes that were generated but not expanded. Each expansion of a leaf node of a  $PSG$  updates the lower and upper bound estimates on AND/OR search graph. An active partial subtree  $APT(n)$  rooted at a node  $n$  ending at a tip node  $t$  contains the path between  $n$  and  $t$ , and all Or children of AND nodes on the path. A dynamic heuristic function of a node  $n$  relative to the current  $PSG$  given the initial heuristic functions  $h_{UB}$  and  $h_{LB}$  can be computed.

**Definition 8 (Dynamic Lower and Upper Bound)** *Given an active partial tree  $APT(n)$ , the dynamic heuristic estimate of upper and lower bound function,  $UB(n)$  and  $LB(n)$ , is defined recursively as follows: (i) if there is a single node  $n$  in  $APT(n)$  and is evaluated, then  $UB(n) = v(n) = LB(n)$  else if  $n$  is a single node in  $APT$   $UB(n) = h_{UB}(n)$  and  $LB(n) = h_{LB}(n)$ ; (ii)  $n = \langle \mathbf{S}_{ij}, \mathbf{b} \rangle$  is an AND node, having OR children  $m_1, \dots, m_k$ , and label =  $l(C_i : \langle \mathbf{S}_i, \mathbf{a}, N_i \rangle, \langle \mathbf{S}_{ij}, \mathbf{b} \rangle)$ , then*

$$UB(n) = \min(h_{UB}(n), label + \sum_{i=1}^k UB(m_i))$$

$$LB(n) = \max(h_{LB}(n), label + \sum_{i=1}^k LB(m_i)) ;$$

(iii) if  $n = C_i : \langle P_i, \mathbf{a}, N_i \rangle$  where  $n$  is an OR node, having an AND child  $m$ , then  $UB(n) = \min(h(n), UB(m))$  and  $LB(n) = \max(h(n), LB(m_i))$ .

**Theorem 3**  $LB(n)$  is a lower bound on the optimal solution to the subproblem rooted at  $n$ , namely  $LB(n) \leq v(n)$ , and also by definition  $LB(n) \geq h_{LB}(n)$ . Also,  $UB(n) \geq v(n)$  and  $UB(n) \leq h_{UB}(n)$ .

**Proof:** We will prove by induction assuming the correctness of heuristics that  $v(n) \leq h_{UB}(n), v(n) \geq h_{LB}(n)$ .

**Basis:** At leaf nodes of AND/OR junction search graph, it is trivial that  $v(n) = UB(n) = LB(n)$  as  $v(n)$  is computed using local constraints and does not involve any heuristic function.

**Induction step:** At any AND node having OR children  $m_1, \dots, m_k$ ,

$$v(n) = label + \sum_i v(m_i) \leq label + \sum_i UB(m_i),$$

where  $v(m_i) \leq UB(m_i)$ .

$$v(n) \leq \min(h_{UB}(n), label + \sum_i UB(m_i)) = UB(n),$$

where  $v(n) \leq h_{UB}(n)$ .

At any OR node having AND child  $m = \operatorname{argmax}_i v(m_i)$ ,

$$v(n) = v(m) \leq UB(m)$$

$$v(n) \leq \min(h_{UB}(n), UB(m)) = UB(n).$$

$LB(n) \leq v(n)$  can be proved similarly. Therefore,

$$LB(n) \leq v(n) \leq UB(n). \quad \square$$

Also,  $UB(n)$  and  $LB(n)$  provides tighter bounds than the initial heuristic functions.

**Proposition 3 (Pruning rule)** *Given PSG, for any AND node  $n$  and its sibling  $m$ , if  $UB(n) < LB(m)$  or  $UB(n) = LB(n)$  then subtree below  $n$  can be pruned.*

### 3.1.3.6 DJAO(k)

We now set up a DJAO search on  $S_T(P)$  whose nodes are assigned to agents. Starting from the root agent given the initial heuristic upper and lower bound functions  $h_{UB}$  and  $h_{LB}$ , the objective is to search one of the solution that satisfies the termination condition while pruning dominated candidate solutions.

---

**Algorithm 1: DJAO(k)(1)**

---

```
procedure Init()
  wait  $\leftarrow$  0 ; // number of waited messages
   $k_i \leftarrow 0, k_c \leftarrow 0$  ; // own and child's k value
   $m_b \leftarrow nil$  ; // OR node in  $par(a_i)$  to backtrack to
   $m^*, m^{**}$  ; // OR node with max, second max UB
   $n_c$  ; // AND node context-compatible with  $m_b$ 
procedure RootRun()
  Init();
  UpdateM();
  if (CheckTermination()) then
    | Send(TERMINATE) to  $\forall c \in succ(a_i)$ ;
    | terminate;
  end
   $k_i \leftarrow UB(m^*) - \max(UB(m^{**}) - k, LB(m^*))$ ;
  wait  $\leftarrow \|succ(a_i)\|$ ;
  Send(VALUE,  $m^*, k_i$ );
  loop forever
  while (message queue is not empty) do
    | pop msg off message queue;
    | When Received(msg);
    | if (CheckTermination()) then
    | | Send(TERMINATE) to  $\forall c \in succ(a_i)$ ;
    | | terminate;
    | else if ( wait==0) then
    | | UpdateM();
    | |  $k_i = UB(m^*) - \max(UB(m^{**}) - k, LB(m^*))$ ;
    | | Send(VALUE,  $m^*, k_i$ ) to  $\forall c \in succ(a_i)$ ;
  end
  procedure Run()
  Init();
  loop forever
  while (message queue is not empty) do
    | pop msg off message queue;
    | When Received(msg);
    | if (Decide BackUp() && wait==0) then
    | | Send(COST,  $m_b, UB(m_b), LB(m_b)$ ) to  $par(a_i)$ 
    | else if (wait==0) then
    | | UpdateM();
    | | Send(VALUE,  $m^*, k_c$ ), to  $\forall c \in succ(a_i)$ 
  end
```

---

---

**Algorithm 2:** DJAO(k)(2)

---

```
procedure UpdateM()
   $m^* = \operatorname{argmax} UB(m)$ , for  $m \in \operatorname{succ}(n_r)$ ;
   $m^{**} = \operatorname{argmax} UB(m)$ , for  $m \in \operatorname{succ}(n_r) \setminus m^*$ ;
  procedure When Received(COST,  $m, v_{UB}, v_{LB}$ )
     $wait \leftarrow wait - 1, UB(m) \leftarrow v_{UB}, LB(m) \leftarrow v_{LB}$ ;
     $UB'(n) \leftarrow UB(n)$ , where  $n = \operatorname{par}(m)$  ;
     $UB(n) \leftarrow \max(UB(n), UB(m))$  ;
     $LB(n) \leftarrow \max(LB(n), LB(m))$ ;
  procedure When Received(VALUE,  $m, k$ )
     $k_i \leftarrow k, m_b \leftarrow m, wait \leftarrow \|\operatorname{succ}(a_i)\|$ ;
     $n_c \leftarrow \text{context} - \text{compatible}(m)$ ;
    procedure Decide BackUp()
      if  $(UB(n_c) - UB'(n_c) \geq k_i)$  then
        | return true;
      else
        |  $k_c \leftarrow (k_i - (UB(n) - UB'(n))) / \|\operatorname{succ}(a_i)\|$ ;
        | return false;
    end
  procedure When Received(TERMINATE)
    Send(TERMINATE) to  $\forall c \in \operatorname{succ}(a_i)$ ;
    terminate;
  procedure Check Termination()
    if  $(UB(n_r) == LB(n_r))$  then
      | return true;
    else if for  $\forall m \in \operatorname{succ}(n_r) \setminus m^*, UB(m) \leq LB(m^*)$  then
      | return true;
    return false;
```

---

DJAO agents use three types of messages: VALUE, COST, and TERMINATE. At the start, the root agent expands the OR nodes from its AND node and selects the best branch in the subtree and sends VALUE messages containing variable values on the chosen branch to its child nodes.

Upon receipt of a VALUE message, an agent evaluates whether the back-up condition is satisfied for the given value assignments  $\mathbf{b}$  in the message. If the back-up condition is satisfied, the agent backs up with updated values by sending a COST message to its parent. Otherwise, it expands the OR nodes compatible with  $\mathbf{b}$  and selects the best branch and sends VALUE message to its children.

Upon receipt of COST message containing the updated lower and upper bounds on the chosen expanded OR nodes from all child nodes, it recalculates the lower and upper bounds of its AND node. It then re-evaluates the back-up condition for the received VALUE message. Unless it satisfies the back-up condition, then the question of which branch to select is re-examined and the agents sends another VALUE message to its children. These steps are repeated until a termination condition in Prop 4 holds for the root agent. It then sends a TERMINATE message to each of its children and terminate. Upon receipt of a TERMINATE message, each agent does the same.

**Proposition 4** *Given an OR node  $n$  and AND nodes*

*$m_1, \dots, m_k$  at the root agent, DJAO( $k$ ) is terminated if UB and LB satisfies the condition  $UB(n) = LB(n)$  or*

*$\exists i, UB(m_j) \leq LB(m_i)$  for  $\forall j, i \neq j$ .*

Each agent stores the lower and upper bounds of expanded nodes and updates these values upon each COST message arrival. The memory requirement for each agent does not exceed  $O(n^d)$  where  $n$  is the size of variable domain and  $d$  is the induced width of the junction tree.

Among many different search strategies which determines the back-up condition for solving COP and DCOP, best-first search and depth-first branch-and-bound search

have been primarily studied [46, 49, 3, 22]. Best-first search always follows the best item found and in the distributed setting whenever there is an update, agents propagate it to all ancestors whose best items may change. On the other hand, depth-first search retains the current path until it is certainly dominated or the true value of node  $v(n)$  is found. In [50], *ADOPT(k)* that provides a trade-off between these two extremes, where the search keeps the current path until the distance between the best solution on the current path and the best solution found so far becomes greater than a given constant  $k$ .

Similarly, we developed *DJAO(k)*, which subsumes both depth-first and best-first search strategy on AND/OR search junction graph. It performs depth-first when  $k = \infty$ , best-first when  $k = \epsilon$ , and a hybrid when  $\epsilon < k < \infty$ , where  $k$  is the distance between the best found solution  $UB(m^*)$  and the next best solution  $UB(m^{**})$  found so far. Unlike *ADOPT(k)* which uses the best solutions based on the subproblems provided by the node’s subtree, the search uses a measurement that considers a more global perspective on the current best solution. The search backtracks when  $UB(m^*) \leq \max(UB(m^{**}) - k, LB(m^*))$ , which occurs as soon as the best solution is dominated by the second best with the best-first strategy with  $k = \epsilon$ , and when the true value for  $m^*$  is found (Thus,  $UB(m^*) = LB(m^*)$ .) with the depth-first strategy with  $k = \infty$ .

Algorithm 1 and 2 shows the pseudocode of *DJAO(k)*, where  $a_i$  is a generic agent,  $par(a_i)$  its parent agent,  $succ(a_i)$  its set of child agents,  $par(n)$  the parent node of the node  $n$  in the search graph,  $succ(n)$  the set of node  $n$ ’s child nodes, and  $n_r$  the AND node of the root agent. The root agent runs *RootRun()* which contains search initiation whereas all other agents runs *Run()*. The pseudo-code uses a predicate **context – compatible**( $m$ ) to select a node whose variable value assignment matches that of the node  $m$ .

### 3.1.3.7 Approximate DJAO(k)

An approximate version of the algorithm can be obtained by relaxing the constraint on upper and lower bound gap similar to search-based DCOP algorithms [3, 51]. Approximate DJAO with an error bound  $e$  terminates when the solution contains no more than error  $e$  such that that value of the found solution  $n$  is no worse than  $v(n^*) - e$ , where  $n^*$  is the optimal solution. The corresponding termination condition is  $LB(n) \geq UB(n) - e$  or  $\exists i, UB(m_j) \leq LB(m_i) + e$  for  $\forall j, i \neq j$ . Also, the search backtracks when  $UB(m^*) \leq \max(UB(m^{**}) - k, LB(m^*) + e)$ .

## 3.2 An Example of DJAO

### 3.2.1 A Simple Example of DJAO

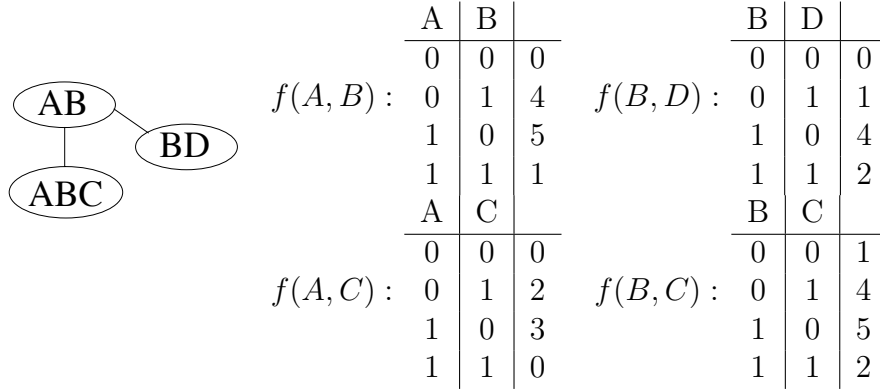


Figure 3.3: Example of junction tree and the constraint functions

On the junction tree in Figure 3.3, let there be three agents,  $A_1, A_2$  and  $A_3$  for the cliques  $AB, ABC$  and  $BD$  respectively. The constraint function  $f(A, B)$  are assigned to clique  $AB$ ,  $f(A, C)$  and  $f(B, C)$  to clique  $ABC$ ,  $f(B, D)$  to  $BD$ .

#### Phase 1:

The first phase starts by the agent  $A_2$  computing the local potential  $b$  by merging  $f(A, C)$  and  $f(B, C)$  in the clique  $ABC$  as well as  $A_3$  (Figure 3.4).

$f(A, B, C) :$	A	B	C	
	0	0	0	1
	0	0	1	6
	0	1	0	5
	0	1	1	4
	1	0	0	4
	1	0	1	4
	1	1	0	8
	1	1	1	2

Figure 3.4: Merged constraint function in  $A_2$

Each agent who does not own the root node generates a filtered message once they received from all the child agents (agents who own the child OR nodes). The message from  $A_2$  and  $A_3$  to  $A_1$  in Action-GDL would be as shown in Figure 3.5.

$M_{A_3 \rightarrow A_1} :$	B			
	0	1	$M_{A_2 \rightarrow A_1} :$	
	1	4		
			A	B
			0	0
			0	1
			1	0
			1	1
				6
				5
				4
				8

Figure 3.5: Messages in Action-GDL

Filtered messages are created and sent with the filtering rate  $l = 80$ . FS denotes the filtered set of variable configurations. For the message  $M_{A_2 \rightarrow A_1}$ , the function range is  $4 (= 8 - 4)$ , thus items with upper bound equal or less than  $(4 + 4 * 0.8)$  are filtered except  $(A=1, B=1)$  as shown in Figure 3.6

$M_{A_3 \rightarrow A_1} :$	B	$h_{LB}$	$h_{UB}$		
	1	4	4	$M_{A_2 \rightarrow A_1} :$	
	FS	1	1		
				A	B
				1	1
				FS	4
					$h_{LB}$
					$h_{UB}$
					8
					8
					6

Figure 3.6: Filtered Messages in DJAO



Once messages received,  $A_1$  calculates potential  $b$  as the total sum of received messages and local functions(Figure 3.7).

	A	B	LB	UB
	0	0	5	7
$f(AB) :$	0	1	12	14
	1	0	10	12
	1	1	13	13

Figure 3.7: The resulting function at  $A_1$  (root agent)

### The second phase

$A_1$  checks the termination condition on the possible solution with the highest upper bound ( $A=0, B=1$ ). Since  $LB(A=0, B=1)$  does not dominate  $UB(A=1, B=1)$ , the search starts.  $A_1$  computes the distance  $k_1$  between the maximum and the second maximum upper bounds. The distance  $k_1 = 14 - 13 = 1$ . The algorithm backtracks to the source of upper and lower bound gap from received messages with a target of reducing the upper bound by  $k_1$ . The search backtracks when the upper bound decreases by equal or more than  $\min(k, k_1)$ . The upper and lower bound gap originates only from  $M_{A_2 \rightarrow A_1}$ . Therefore,  $A_1$  sends a VALUE message with a variables configuration ( $A=0, B=1$ ) and  $\min(k, k_1)$  to  $A_2$ .  $A_2$  receives this VALUE message and prepares a COST message as it is a leaf node. It then sends a COST message  $LB(0, 1) = 5, UB(0, 1) = 5$ . Upon receipt of the COST message, the root updates its table.

	A	B	LB	UB
	0	0	6	7
$f(A, B) :$	0	1	12	12
	1	0	11	12
	1	1	13	13

Figure 3.8: Updated utility function at the root after the search

Since the lower bound of  $(A, B) = (1, 1)$  dominates upper bounds of all other configurations, the termination condition is satisfied and the search terminates.

### 3.2.2 A More Complicated Example of DJAO

We assume another agent  $A_4$  added to the problem which handles clique  $BDE$ . Let there be additional potentials  $f(B, E)$  and  $f(D, E)$  as given in Fig. 3.9.

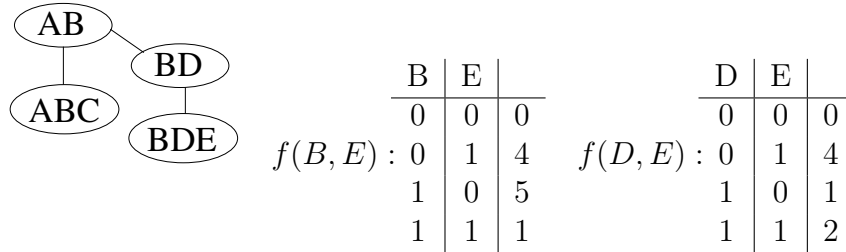


Figure 3.9: Example of junction tree and the added constraint functions

#### The first phase:

Firstly, in addition to  $A_2$ , the agents  $A_4$  computes the local potential merging  $f(B, E)$  and  $f(D, E)$ .

B	D	E	
0	0	0	0
0	0	1	8
0	1	0	1
0	1	1	6
1	0	0	5
1	0	1	4
1	1	0	6
1	1	1	3

 $f(B, D, E) :$ 

The function values are summarized in the OR node on the variables in the separators. A normal message and a filtered message with 80% filtering from  $A_4$  to  $A_3$  would be as follows.

	B	D	8
$M_{A_4 \rightarrow A_3} :$	0	0	6
	1	0	5
	1	1	6

	B	D	$h_{LB}$	$h_{UB}$
$M_{A_4 \rightarrow A_3} :$	0	0	8	8
	FS		5	6

Figure 3.10: Messages from  $A_4$  to  $A_3$

This filtered message then merged to compute the local potentials for the agent  $A_3$  as given in. 3.11.

	B	D	$h_{LB}$	$h_{UB}$
$f'(B, D) :$	0	0	8	8
	0	1	6	7
	1	0	9	10
	1	1	7	8

Figure 3.11: Local function and messages combined for  $A_3$

A message from  $A_3$  to  $A_1$  before filtering  $M$  and after filtering  $MF$  changes to Fig. 3.12.  $A_1$  receives all messages and combines with local functions as shown in Fig. 3.13.

	B	$h_{LB}$	$h_{UB}$		B	$h_{LB}$	$h_{UB}$
$M_{A_3 \rightarrow A_1} :$	0	8	8	$MF_{A_3 \rightarrow A_1} :$	1	9	10
	1	9	10		FS	8	8

Figure 3.12: Message before and after filtering from  $A_3$  to  $A_1$

	A	B	$LB$	$UB$
$f(AB) :$	0	0	12	14
	0	1	17	20
	1	0	17	19
	1	1	18	19

Figure 3.13: The resulting function at the root

### The second phase

The second phase in which the search is conducted starts in the root checking the termination condition for the item with the highest upper bound. The distance  $D_1$  between the maximum and the second maximum is  $20 - 19 = 1$ . The source of upper and lower bound gap originates from both messages, it backtracks to recover to both  $A_3$  and  $A_2$ . The targeted change limit on the bound  $D_1/W = 1/2$ , where the  $W$  is the number of child agents it backtracks and is 2 in the example. It sends a VALUE message of  $(A=0, B=1)$  and  $(B=1)$  to  $A_2$  and  $A_3$  respectively.

When  $A_3$  receives this VALUE message, it checks the source of the bound gap. It finds the gap originated from the received message ( and not from local filtering in the process of message  $M_{A_3 \rightarrow A_1}$  production ) and the upper bound cannot be changed locally. It backtracks to recover  $1/2$  and sends a VALUE message to  $A_4$ . The maximum value for the variable configuration  $(B=1)$  is enabled by  $(D=0)$ , and thus it sends a VALUE message of  $(B=1, D=0)$ .  $A_4$  replies with a COST value  $LB(1,0) = 5, UB(1,0) = 5$ .  $A_3$  updates the cost table based on this COST message and returns a COST message  $LB(1) = 9, UB(1) = 9$ .

Upon receiving a VALUE message,  $A_2$  does the same for the previous example and replies with a COST message  $LB(0,1) = 5, UB(0,1) = 5$ . The updated table at the agent  $A_1$  is given in Fig. 3.14

Because there is no single variable configuration that dominates, the search continues and  $A_1$  sends  $A_2$  responsible for the gap a VALUE message of  $(A=1, B=0)$  and

	A	B	<i>LB</i>	<i>UB</i>
$f(AB) :$	0	0	12	14
	0	1	18	18
	1	0	17	19
	1	1	18	18

Figure 3.14: The resulting function at the root

receives a COST message  $LB(1,0) = 4, UB(1,0) = 4$  in return. The updated cost table of  $A_1$  is shown in Fig. 3.15.

	A	B	<i>LB</i>	<i>UB</i>
$f(AB) :$	0	0	12	14
	0	1	18	18
	1	0	17	17
	1	1	18	18

Figure 3.15: The resulting function at the root

As a result, both  $(A=0, B=1)$  and  $(A=1, B=1)$  dominates and the search terminates with existing upper and lower bound gap on the variable configuration  $(A=0, B=0)$ .

### 3.3 Empirical Evaluation

In this section we evaluate the performance of DJAO search. For each experiment, we report the communication costs, NCCCs(non-concurrent constraint check), and solution quality for approximate solutions with respect to optimal solutions. We used a DJAO that sends VALUE messages to at most 25 nodes when there are ties. We evaluate and compare our approach with Action-GDL and ADOPT(k) with  $k=4000$  which was reasonable among 400, 4000, and 40000. Communication costs are

$c$	Algorithm	Total Bytes	NCCCs	Msgs
20	DJAO( $k = \epsilon$ )	<b>227744</b>	25627367	667
	DJAO( $k = 10$ )	260708	25991283	615
	DJAO( $k = 100$ )	229503	20898049	664
	DJAO( $k = 500$ )	271962	34144648	1485
	Action-GDL	394690	2741859	18
	ADOPT(K=4000)	1217757	4341532	206082
25	DJAO( $k = \epsilon$ )	1540523	794393531	2888
	DJAO( $k = 10$ )	1509601	835729595	2804
	DJAO( $k = 100$ )	<b>1508705</b>	570976502	2478
	DJAO( $k = 500$ )	2516606	1792901180	9263
	Action-GDL	3679104	25942425	18
	ADOPT(K=4000)	54556373	417172726	8992463
30	DJAO( $k = \epsilon$ )	<b>1513317</b>	734471121	3203
	DJAO( $k = 10$ )	2204580	1117448830	3505
	DJAO( $k = 100$ )	1513698	553233251	2910
	DJAO( $k = 500$ )	4084228	2759254975	17741
	Action-GDL	4568592	33038068	18
	ADOPT(K=4000)	37285466	219714985	6334245

Table 3.1: Performance of Optimal DJAO(k)  
on Random Binary DCOP Instances

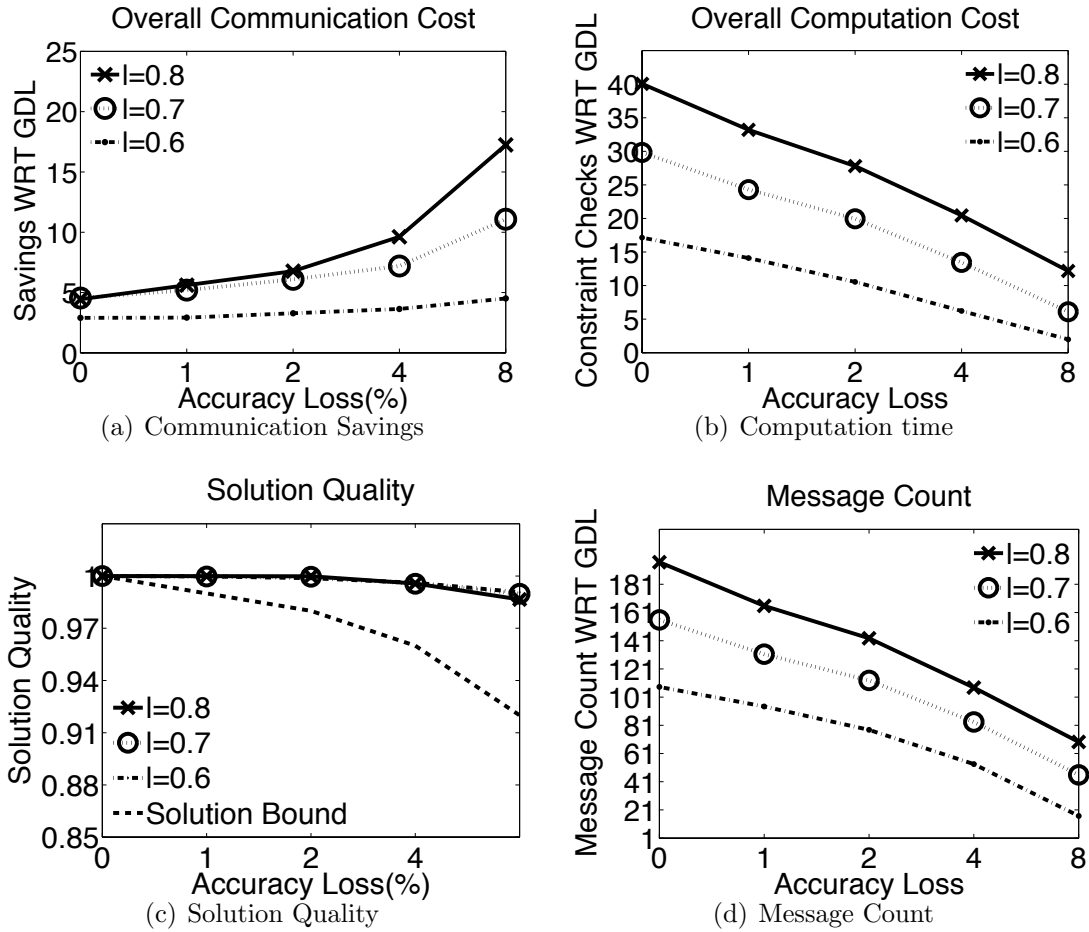


Figure 3.16: Performance of Approximate DJAO(k=10)



	Algorithm	Total Bytes	NCCCs	Msgs
A	DJAO( $k = \epsilon$ )	163,360	22,217,301	315
	DJAO( $k = 100,000,000$ )	<b>155,021</b>	25,413,935	326
	Action-GDL	3,624,186	19,905,921	126
	ADOPT(K=30,000,000)	11,121,364	24,068,428	2,005,732
B	DJAO( $DJAOk = \epsilon$ )	278,168	30,441,957	325
	DJAO( $k = 100,000,000$ )	<b>238,696</b>	21,998,565	342
	Action-GDL	4,274,606	24553995	126
	ADOPT(K=30,000,000)	54,735,040	166,542,715	9,869,280
C	DJAO( $k = \epsilon$ )	207,801	9,379,233	194
	DJAO( $k = 100,000,000$ )	<b>120,516</b>	7,509,783	191
	Action-GDL	1,294,382	6,419,231	78
	ADOPT(K=30,000,000)	1,009,124	2,625,136	178,301
D	DJAO( $k = \epsilon$ )	718,528	31,980,359	405
	DJAO( $k = 100,000,000$ )	<b>482,654</b>	43,316,277	474
	Action-GDL	10,321,229	58,754,948	126
	ADOPT(K=30,000,000)	21,573,856	66,347,439	3,812,541

Table 3.2: Sensor Network Instances

measured as the number of bytes sent during execution<sup>1</sup> and the message count of both UTIL and VALUE messages. For approximate results, we show the true utility of found solution which is often much higher than the estimated lower bound.

We experimented on random binary DCOP instances with 10 variables of domain size 10. The function cost are randomly generated over the range  $\langle 0, \dots, 200 \rangle$ . We varied the number of constraint functions  $c$  20, 25 and 30 and averaged our results over 20 instances for each value of  $c$ . Total communication amount is largely determined by the structure of junction tree. Thus, five junction trees were constructed for each problem instance. Initial heuristics were generated using soft filtering where the bottom 70% is filtered. The table shows DJAO requires less communication than both ADOPT and Action-GDL, especially when the network connectivity is high. The best-first search-like DJAO when  $k = \epsilon$  consistently performed well in these experiments.

---

<sup>1</sup>A single variable value is 4bytes, and cost 8bytes.

Table. 3.2 shows the results on sensor network instances from a public repository [52] with a heuristic which filters 90% ( $l = 0.9$ ). Table. 3.1 and 3.2 show DJAO requires up to an order of magnitude less communication than both ADOPT<sup>2</sup> and Action-GDL without significant increase in computational costs. Compared to the random DCOP instances, function ranges in sensor network instances are wider and a large set of clearly dominated variable configurations results in significant communication savings with DJAO. DJAO with high  $k$  values performed better in terms of communication on these lower connectivity graphs compared to the random DCOP instances.

Lastly, in an experiment on the same problem instances, we measured the methods' trend as the solution quality guarantee changes. We evaluated 20 instances for each quality loss ranging from  $loss = 0\%$  to  $loss = 8\%$  using a heuristics which filters 80% ( $l=0.8$ ) and 70% ( $l=0.7$ ). Results in Fig. 3.16 show that DJAO gains significant savings in communication as the error bound increases. With 80% heuristics, it transmits about 18 times less information than that of Action-GDL when  $loss = 8\%$  while it uses 13 times more computation (NCCCs) and about 130 messages per agent.

### 3.4 Conclusions

We addressed the problem of solving DCOP exactly and with precise approximation bounds by developing a new distributed algorithm called DJAO. There are three novel ideas in DJAO. Firstly, it uses an AND/OR junction graph representation, which builds a basis for efficient search in the distributed settings. The second is a two phase search strategy that combines characteristics of ADOPT and Action-GDL. The third is a soft filtering technique to significantly reduce communication

---

<sup>2</sup>Results from [50]

without losing any accuracy. We also showed experimentally significant reduction in communication required by DJAO in comparison with ADOPT and Action-GDL.

## CHAPTER 4

### APPLYING INFERENCE TECHNIQUE TO REDUCE COMPUTATIONAL BURDEN IN DCOPS

The objective of this chapter is to address problems when we apply techniques for MAP estimation to DCOPs. Due to different settings of inference problem on graphical models from those of multi-agent domains, we need to modify techniques for MAP estimation to apply to DCOPs. We are particularly interested in recent work by McAuley et al. [24] that uses order statistics of variable configuration to reduce the computational complexity of the maximization operation. We apply this technique to Max-Sum in order to save computational resources in various DCOP environments.

#### 4.1 Background

This section presents the work by McAuley et al. [24] on reducing the expected complexity of maximization operation. Their technique, called Fast Belief Propagation (FBP), finds a particular variable configurations that yields the maximum sum of two sorted lists of length  $N$  with an expected complexity of  $O(\sqrt{N})$ . Assuming the order statistics of variables on the lists are independent, the technique achieves an expected time of  $O(N\sqrt{N})$  to compute a single Max-Sum message for a binary constraint, which is smaller than  $O(N^2)$  required by the naive approach.

The Fast Belief Propagation (FBP) [24] optimizes the maximization operator of Equation (1.3) by using presorted constraint functions. Given a binary constraint, it uses two lists—a list of presorted constraint function values and a list of incoming

message values that are sorted online. This operation amounts to maximizing the sum of two lists  $\mathbf{v}_a$  and  $\mathbf{v}_b$ :

$$\max_i \{ \mathbf{v}_a[i] + \mathbf{v}_b[i] \} \quad (4.1)$$

The FBP algorithm performs the above operation with an expected  $O(\sqrt{N})$  time complexity, as opposed to  $O(N)$  of the naive algorithm, where  $N$  is the length of the list.

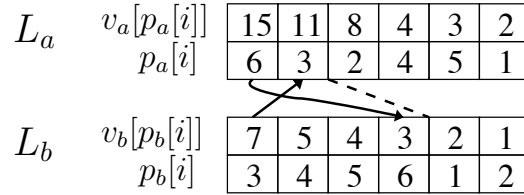


Figure 4.1: Example of FBP technique. The largest item 15 of  $\mathbf{v}_a$  that has index 6 is summed with 3 in  $\mathbf{v}_b$  with the same index (which maps to specific value combination of variables). Therefore, items with value smaller than 3 in  $\mathbf{v}_b$  can be ignored as any value smaller than 3 cannot yield a value larger than  $(15 + 3)$ . We also limit the items smaller than 11 in  $\mathbf{v}_a$  by applying the same idea. In this example only 2 computations are required to compute the maximum value using this technique.

Figure 4.1 describes the main idea of the FBP algorithm. The list  $\mathbf{p}_a$  and  $\mathbf{p}_b$  denote the permutation arrays of  $\mathbf{v}_a$  and  $\mathbf{v}_b$ . For further details, please refer to [24].

As the expected computational complexity to find the maximum of two such lists is  $O(\sqrt{N})$ , the FBP algorithm achieves the total expected complexity of  $O(N^{1.5})$  for the Max-Sum maximization operation, which is better than  $O(N^2)$  time required by the naive algorithm. The main drawback of the FBP approach is that it requires the complete problem to be specified ahead of time as constraint functions need to be presorted. Further, this approach is only applicable to pairwise graphs and runtime guarantees do not extend to arbitrary arity graphs.

## 4.2 Fast Belief Proagation for N-ary DCOP

Despite the importance of n-ary constraints in real applications, there has been little work devoted to developing algorithms that handle n-ary constraints [53, 32, 12, 13] Further, these studies do not directly tackle the computational difficulty in handling n-ary constraints in DCOPs.

Max-Sum performs repetitive maximization operations for each constraint to select the locally best configuration of its associated variables given its local function and a set of incoming messages. The complexity of this step grows exponentially as the number of associated variables (constraint arity) grows. We address this bottleneck and also provide formal guarantees regarding the expected runtime improvement, which could be very significant, up to an exponential improvement over the naive scheme. Reducing such computational overhead is particularly crucial in the multiagent setting, where agents are often assumed to be resource constrained such as mobile robots [6, 54]. There has been research [29, 30] in DCOP that tries to reduce the amount of communication, however there has not been any improvement in the overall computational complexity of DCOP algorithms. An exception is the work using tractable higher order potentials for binary DCOP [41], which applies only to a special class of DCOP.

Although the FBP technique offers substantial benefit on binary constraints, it cannot be directly used on graphs with n-ary constraints. Moreover, based on their theoretical analysis on n-ary constraint functions, the benefit decreases with higher-arity functions. Additionally, the order statistics of variables on lists summed in the scheme should be positively correlated or independent for the theoretical analysis to hold. However, this property easily gets violated in domains where a variable's value can affect multiple constraint function values in the opposite ways. When the order statistics of these lists are negatively correlated, the technique may perform worse than a simple technique using dynamic programming. Lastly, the constraint graphs

are required to be given offline for computational savings. Often, a DCOP is a one-shot problem in which an expensive preprocessing sorting step that dominates the actual problem complexity is not realistic. Additionally, in many DCOP applications constraint function values and domains of variables often change dynamically.

To remedy these limitations, we have developed a variant of McAuley’s technique, which we call Generalized Fast Belief Propagation (G-FBP). Our approach is fundamentally different from that of FBP in that it does not require the offline and complete sorting of data as in FBP; rather it uses *partially sorted* lists. The key idea behind our approach is that often, only a small representative sample of values from different message/value lists is needed to efficiently perform the maximization procedure. Further, our approach works for arbitrary arity graphs as opposed to pairwise graphs required by the FBP algorithm [24]. We also provide a theoretical analysis of the expected runtime complexity for this general case and show that we can indeed achieve the  $O(N\sqrt{N})$  complexity for pairwise graphs with only partially sorted lists. For  $m$ -ary graphs, this translates into an expected complexity of  $O(mN^{\frac{m+1}{2}})$  as opposed to the exhaustive approach’s complexity of  $O(mN^m)$ , which is a significant reduction. Further, we also note that an advanced version of FBP is presented in [55], which has a theoretical expected complexity of  $O(mN^{\frac{(m-1)^2}{m}+1})$  for general  $m$ -ary graphs. Our approach has strictly better expected complexity.

Additionally, we devise a correlation measure which decides whether the order statistics of lists are negatively correlated. We then use this measure to conditionally apply G-FBP scheme to a particular maximization operation. Given the definition of correlation on order statistics, we show that this measure correctly computes the correlation.

Finally, we add another feature to our approach, which leads to an extended version of G-FBP called GSC-FBP. This approach reuses computation from the previous

iterations results. Its effectiveness lies in the fact that messages become less likely to change in later stages of the algorithm.

#### 4.2.1 G-FBP

G-FBP that uses *two partially* sorted lists, to find the maximum sum as in Equation (1.3), instead of completely sorted lists used in FBP. We construct two lists called the *value list* and the *message list* by selecting and sorting only the top  $KN^{\frac{m-1}{2}}$  items of both lists in FBP where  $N$  is the domain size,  $m$  is the number of associated variables and  $K$  is a constant. The main idea behind such select-then-sort operation is that for the maximization operation, only the top  $KN^{\frac{m-1}{2}}$  will be required most of the time; the unsorted entries are not accessed in most cases. *Partial sorting* and using a single *message list* are keys to generalizing the approach to n-ary constraints while keeping the same complexity.

**Value list:** Intuitively, the value list corresponds to a partially sorted version of the constraint function  $F_j$  given the specific value of a single variable in Equation (1.3). There is one value list defined for every constraint function  $F_j$  and every value of variable  $X_i$  that is in the scope of  $F_j$ . It contains  $\langle \text{index-value} \rangle$  pairs as:

$$L_b(j, x_i) = \{ \langle \mathbf{x}^j, F_j(\mathbf{x}^j) \rangle | \mathbf{x}^j(i) = x_i \} \quad (4.2)$$

where  $\mathbf{x}^j$  is a complete assignment to all the variables in the scope of function  $F_j$ ; the condition  $\mathbf{x}^j(i) = x_i$  implies that the  $i$ -th variable is fixed to a particular value  $x_i$  in every  $\mathbf{x}^j$ . If the constraint function is  $m$ -ary (or involve  $m$  variables), then the length of each value list is  $N^{m-1}$ . Instead of completely sorting this list, which is expensive, we select the top  $KN^{\frac{m-1}{2}}$  values of  $L_b(j, x_i)$ , which are then sorted in decreasing order and inserted back to the front of this list.

**Message list:** Intuitively, the message list represents a partially sorted list corresponding to the sum of incoming messages  $q$  to a function node, as shown in the second term of Equation (1.3). There is one message list defined for every constraint



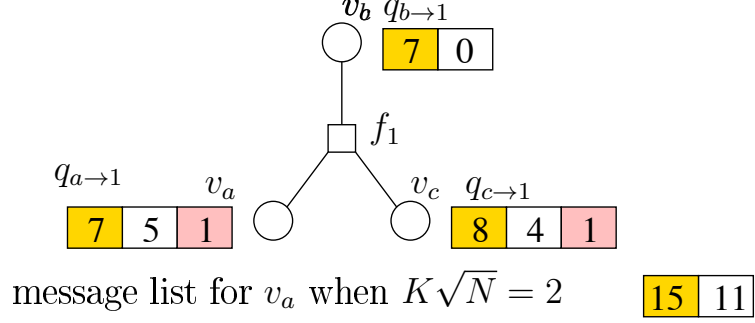


Figure 4.2: Example of Message List Generation. Each value in boxes denotes individually sorted message values from the variable nodes to the function nodes. The domain size is 2 for  $v_b$  and it is 3 for  $v_a$  and  $v_c$  thus the message size. In order to compute the message to  $v_a$ , messages from  $v_b$  and  $v_c$  ( $q_{b \rightarrow 1}$  and  $q_{c \rightarrow 1}$  respectively) are summed to generate the partial message list [15, 11] instead of the complete list [15, 11, 8, 8, 4, 1] with  $|L_a| = 2$ .

function  $F_j$  and every variable  $X_i$  (Not every value of variable) that is in the scope of  $F_j$ . It contains  $\langle \text{index-value} \rangle$  pairs as:

$$L_a(j, X_i) = \left\{ \left\langle \mathbf{x}^j \setminus X_i, \sum_{k \in N_j \setminus i} q_{k \rightarrow j}(\mathbf{x}^j(k)) \right\rangle \right\} \quad (4.3)$$

where  $\mathbf{x}^j(k)$  denotes the assignment to the variable  $X_k$  under  $\mathbf{x}^j$ . The length of every complete list  $L_a$  is  $N^{m-1}$  and selecting the top most items requires iterating over all values. Fortunately, each message contains values on a single neighboring variable and are independent of each other in Max-Sum. Using the independency among messages, we do not iterate the items in the list  $L_a$  completely in order to select the top elements. In our implementation, we incrementally construct each message list partially that only contains the top  $KN^{\frac{m-1}{2}}$  items sorted in descending order, without ever generating the complete  $N^{m-1}$  sized lists. Although these message lists are constructed per iteration unlike value lists, there are only  $m$  such lists for each  $m$ -ary constraint in contrast to  $(m \times N)$  value lists. The overhead of constructing message lists for a single message is  $O(\log mK \times N^{\frac{m-1}{2}})$  and this does not dominate the expected complexity  $O(N \times N^{\frac{m-1}{2}})$  of computing a message for an  $m$ -ary constraint for a reasonable  $K$ . Figure 4.2 shows an example of a partial message list.

### 4.2.2 G-FBP Algorithm with Partial Lists

We now describe the complete steps of the G-FBP maximization procedure that operates using such partial value and message lists where the ranks of items in the unsorted part are not known. For ease of exposition, we describe the Algorithm 1 in terms of the maximization problem in Equation (4.1). The main difference of G-FBP from the FBP algorithm is the lists  $\mathbf{v}_a$  and  $\mathbf{v}_b$  are partially sorted. Thus, we need to process items whose matching items are not found in the other list. In Algorithm 1 lines 13–15 saves the missing items for the later processing in lines 27–30. Also, we need to detect the case when the maximization cannot be performed using the sorted component of lists (lines 32–24). In these cases, we compute sums for all variable value combinations to find the maximum. The example of applying this procedure is shown in Figure 4.3. Algorithm 2 describes the steps of computing Equation (1.3) using G-FBP in Alg. 1 in Max-Sum.

### 4.2.3 Time Complexity and Selection of $K$

The main intuition behind G-FBP is that the probability of finding the maximum value within the sorted section is very high with an appropriate  $K$  given the independence assumption of two lists. [24] uses enumerative combinatorics to construct the analysis on the probability of items with the same index not existing in topmost  $M$  items in the lists of size  $N$ . Under the assumption that the order statistics of two sorted lists are independent, this probability is computed as the probability of getting  $M$  red-colored balls where we randomly select  $M$  balls out of the box in which there are  $(N - M)$  red-colored balls and  $M$  blue-colored balls.

Using the same notion, the probability of **not** finding the matching items within  $K\sqrt{N}$  items in the lists of size  $N$  is

---

**Algorithm 1** G-FBP( $v_a, v_b$ ) : Find  $\max(v_a[i] + v_b[i])$ 

---

**Require:** permutation array  $p_a$  and  $p_b$  that partially sort  $v_a$  and  $v_b$  in decreasing order (i.e.  $p_a[i]$  is the index of  $i$ th largest value of  $v_a$ ).

- 1: {Initialization}
- 2:  $S_b^{miss} \leftarrow \phi, S_a^{miss} \leftarrow \phi, val_{max} \leftarrow -\infty$
- 3:  $end_a \leftarrow len(p_a), end_b \leftarrow len(p_b)$
- 4:  $itr_a \leftarrow 0, boundfound \leftarrow false$
- 5: **if**  $(p_a[1] \in p_b) \wedge (p_b[1] \in p_a)$  **then**
- 6:    $index_{max} \leftarrow argmax_{i \in \{p_a[1], p_b[1]\}} \{v_a[i] + v_b[i]\}$
- 7:    $val_{max} \leftarrow v_a[index_{max}] + v_b[index_{max}]$
- 8:    $end_a \leftarrow p_a^{-1}[p_b[1]], end_b \leftarrow p_b^{-1}[p_a[1]]$
- 9:    $boundfound \leftarrow true$
- 10: **end if**
- 11: **while**  $itr_a < end_a$  {Until bounding items are found} **do**
- 12:    $itr_a \leftarrow itr_a + 1$
- 13:   **if**  $p_a[itr_a] \notin p_b$  **then**
- 14:      $S_b^{miss} \leftarrow S_b^{miss} \cup \{p_a[itr_a]\}$
- 15:   **else**
- 16:      $boundfound \leftarrow true$
- 17:     **if**  $v_a[p_a[itr_a]] + v_b[p_a[itr_a]] > val_{max}$  **then**
- 18:        $index_{max} \leftarrow p_a[itr_a]$
- 19:        $val_{max} \leftarrow v_a[index_{max}] + v_b[index_{max}]$
- 20:     **end if**
- 21:     **if**  $p_b^{-1}[p_a[itr_a]] < end_b$  **then**
- 22:        $end_b \leftarrow p_b^{-1}[p_a[itr_a]]$
- 23:     **end if**
- 24:   **end if**
- 25: **end while**
- 26: repeat 10-24 while interchanging a and b
- 27: {Process unmatched items by directly calculating from the constraint function and messages}
- 28: **for all**  $i \in S_a^{miss} \cup S_b^{miss}$  **do**
- 29:   compute the value  $v_a[i] + v_b[i]$  by going through the value table and messages and update  $val_{max}$ .
- 30: **end for**
- 31: {failure case}
- 32: **if**  $boundfound == false$  **then**
- 33:   process unsorted part of the list and update  $val_{max}$
- 34: **end if**
- 35: return  $val_{max}$

---

Step 0														
$L_a$	$v_a[p_a[i]]$ <table border="1"> <tr><td>15</td><td>11</td><td>?</td><td>?</td><td>?</td><td>?</td></tr> <tr><td>6</td><td>3</td><td>?</td><td>?</td><td>?</td><td>?</td></tr> </table>	15	11	?	?	?	?	6	3	?	?	?	?	<p>We are given a message list <math>L_a</math> of length 2 and a value list <math>L_b</math> of only 2 items sorted where the length of sorted parts is 2.</p> <p>Note that the part of the list <math>L_a</math> beyond 2 items is not computed and the shaded part of <math>L_b</math> is not sorted.</p>
15	11	?	?	?	?									
6	3	?	?	?	?									
	$p_a[i]$ <table border="1"> <tr><td>6</td><td>3</td><td>?</td><td>?</td><td>?</td><td>?</td></tr> </table>	6	3	?	?	?	?							
6	3	?	?	?	?									
$L_b$	$v_b[p_b[i]]$ <table border="1"> <tr><td>7</td><td>5</td><td>1</td><td>2</td><td>4</td><td>3</td></tr> <tr><td>3</td><td>4</td><td>2</td><td>1</td><td>5</td><td>6</td></tr> </table>	7	5	1	2	4	3	3	4	2	1	5	6	
7	5	1	2	4	3									
3	4	2	1	5	6									
Step 1														
$L_a$	$v_a[p_a[i]]$ <table border="1"> <tr><td>15</td><td>11</td><td>?</td><td>?</td><td>?</td><td>?</td></tr> <tr><td>6</td><td>3</td><td>?</td><td>?</td><td>?</td><td>?</td></tr> </table>	15	11	?	?	?	?	6	3	?	?	?	?	<p>In step 1, we process the first item in <math>L_a</math> and cannot locate the matching item in <math>L_b</math> with index 6 (We do not keep the location of unsorted items). We add this item with index 6 to <math>S_b^{miss}</math> and continue.</p> <p><math>S_b^{miss} = \{6\}</math></p>
15	11	?	?	?	?									
6	3	?	?	?	?									
	$p_a[i]$ <table border="1"> <tr><td>6</td><td>3</td><td>?</td><td>?</td><td>?</td><td>?</td></tr> </table>	6	3	?	?	?	?							
6	3	?	?	?	?									
$L_b$	$v_b[p_b[i]]$ <table border="1"> <tr><td>7</td><td>5</td><td>1</td><td>2</td><td>4</td><td>3</td></tr> <tr><td>3</td><td>4</td><td>2</td><td>1</td><td>5</td><td>6</td></tr> </table>	7	5	1	2	4	3	3	4	2	1	5	6	
7	5	1	2	4	3									
3	4	2	1	5	6									
Step 2														
$L_a$	$v_a[p_a[i]]$ <table border="1"> <tr><td>15</td><td>11</td><td>?</td><td>?</td><td>?</td><td>?</td></tr> <tr><td>6</td><td>3</td><td>?</td><td>?</td><td>?</td><td>?</td></tr> </table>	15	11	?	?	?	?	6	3	?	?	?	?	<p>In step 2, we try to process an item in <math>L_b</math> and we find a matching item in <math>L_a</math> with index 3 and thus we can find the temporary maximum of 18.</p> <p><math>val_{max} \leftarrow 18</math></p> <p>We set <math>a_{end}</math> to the location of the matching item as we found one.</p> <p><math>a_{end} \leftarrow 2</math></p>
15	11	?	?	?	?									
6	3	?	?	?	?									
	$p_a[i]$ <table border="1"> <tr><td>6</td><td>3</td><td>?</td><td>?</td><td>?</td><td>?</td></tr> </table>	6	3	?	?	?	?							
6	3	?	?	?	?									
$L_b$	$v_b[p_b[i]]$ <table border="1"> <tr><td>7</td><td>5</td><td>1</td><td>2</td><td>4</td><td>3</td></tr> <tr><td>3</td><td>4</td><td>2</td><td>1</td><td>5</td><td>6</td></tr> </table>	7	5	1	2	4	3	3	4	2	1	5	6	
7	5	1	2	4	3									
3	4	2	1	5	6									
Step 3														
		<p>In step 3, we proceed in <math>L_a</math> and reached the second entry on <math>L_a</math>. We have already reached <math>a_{end}</math> and are done with lists.</p> <p>We process <math>S_b^{miss}</math> and compute the item with the index 6 using the constraint function and received messages and find the value 18 and do not update <math>val_{max}</math> as it is not larger than the current maximum. At this point, since there are no more items in <math>S_b^{miss}</math> and <math>S_a^{miss}</math>, the algorithm terminates.</p>												

Figure 4.3: Example of G-FBP technique as in Algorithm 1

---

**Algorithm 2** compute  $r_{j \rightarrow i}$  in Equation (1.3) with G-FBP

---

```

1: messagechanged = false
2: if cycle==0 then
3:   for all  $x_i \in X_i$  do
4:     construct  $L_b(j, x_i)$ 
5:   end for
6: end if
7: for all  $k \in N_j \setminus i$  do
8:   if  $q_{k \rightarrow j}$  has changed then
9:     messagechanged = true
10:  end if
11: end for
12: if messagechanged == true then
13:   construct  $L_a(j, X_i)$ 
14: end if
15: for all  $x_i \in X_i$  do
16:    $r_{j \rightarrow i}(x_i) = G - FBP(L_b(j, x_i), L_a(j, X_i))$ 
17: end for
18: return  $r_{j \rightarrow i}$ 

```

---

$$P(X > K\sqrt{N}; N) = \frac{(N - K\sqrt{N})!(N - K\sqrt{N})!}{(N - 2K\sqrt{N})!N!} \quad (4.4)$$

$$\leq \left( \frac{(N - K\sqrt{N})}{N} \right)^{K\sqrt{N}}, \quad (4.5)$$

where Equation 4.5 can be derived by simply expanding Equation 4.4 and using the relation  $\frac{N-K\sqrt{N}-i}{N-i} < \frac{N-K\sqrt{N}}{N}$  to replace the intermediate terms. For the case of list size  $N = 10,000$ ,  $K\sqrt{N} = 200$ , the probability bound is as small as 0.0176. In other words, when there are two lists of length 10000 and 200 items are selected and sorted, the probability of finding matching items in 200 items on two lists is as large as 0.9824.

In Alg. 1, the algorithm iterates over all items if any set of items with the same index (that is, same variable configuration) is not found in the sorted part of the lists (see line 31–34). Therefore, finding such items is critical to the performance of the algorithm. In order to increase the probability of finding the matching items in sorted part of the lists, we increase the size of sorted parts. More specifically, with

G-FBP that uses partially sorted lists, a specific condition is required to hold for the expected complexity for finding the maximum to be  $O(\sqrt{N})$  given the lists of size  $N$ .

**Theorem 4** *The expected time complexity of  $O(\sqrt{N})$  holds with partial lists when  $(1 - \frac{K}{\sqrt{N}})^{K\sqrt{N}} < \frac{1}{\sqrt{N}}$ .*

**Proof:** The expected running time is estimated based on the number of summed items evaluated in order to find the maximum. The expected number of summations  $E(\Sigma)$  is given as  $\sum_{i=0}^{N-1} P(X > i; N)$ . The probability  $P(X > i; N)$  is the probability that the rank  $X$  of an item is not smaller than  $i$ . In our setting with partial lists, the probability of certain items to be in the unsorted part equals the probability of not finding the maximum within  $K\sqrt{N}$ . Thus, the probability becomes:

$$P(X > i; N) = P(X > K\sqrt{N}) \text{ if } i > K\sqrt{N} \quad (4.6)$$

We re-write the expected number of summations as

$$E(\Sigma) = \sum_{i=0}^{K\sqrt{N}} P(X > i) + \sum_{i=K\sqrt{N}+1}^{N-1} P(X > K\sqrt{N}) \quad (4.7)$$

$$= \sum_{i=0}^{K\sqrt{N}} P(X > i) + \sum_{i=K\sqrt{N}+1}^{N-1} \frac{(N - K\sqrt{N})!(N - K\sqrt{N})!}{(N - 2K\sqrt{N})!N!} \quad (4.8)$$

$$\leq \sum_{i=0}^{N-1} P(X > i) + \sum_{i=K\sqrt{N}+1}^{N-1} \left(1 - \frac{K\sqrt{N}}{N}\right)^{K\sqrt{N}} \quad (4.9)$$

In Equation 4.9, we already know from [55] that the first term is  $O(\sqrt{N})$ . The second summation equates to  $(N - K\sqrt{N})/\sqrt{N}$  by the condition of the theorem and is dominated by  $\sqrt{N}$ . Therefore, the expected time complexity is  $O(\sqrt{N})\square$

#### 4.2.4 Independence Assumption and Correlation Measure

The guarantee of the expected complexity of the FBP technique is constructed based on the assumption that the ranks of the items on the two lists are independent. However, the independence assumption of the FBP technique does not hold generally. If the two lists are negatively correlated, the expected complexity does not hold. It

is likely that the G-FBP scheme fails to find the maximum item using partial lists, thereby increasing the time complexity of the algorithm. Therefore, if we can detect negative correlation, then we avoid applying the G-FBP approach.

We modify the Spearman's rank correlation measure [56] to measure the correlation among two partially sorted lists.

Let  $x$  and  $y$  be two lists of length  $N$  where  $r_{x_i}$  and  $r_{y_i}$  are the ranks of the respective items with index  $i$ . Let  $r_m = K\sqrt{N} + \frac{1}{2}$  be the imaginary median rank. Our redefined correlation measure is :

$$\rho' = \frac{\sum_i (k_{x_i})(k_{y_i})}{\sqrt{\sum_i k_{x_i}^2 \sum_i k_{y_i}^2}} \quad (4.10)$$

where the rank (for each list) is calculated as:

$$k_i = \begin{cases} \frac{(N-K\sqrt{N})}{K\sqrt{N}}(r_i - r_m), & \text{if } r_i < r_m \\ \left(\frac{(K\sqrt{N}+1+2K\sqrt{N})}{2} - r_m\right), & \text{if } r_i > r_m. \end{cases}$$

**Definition 9** Given the item  $x_i$  of list  $x$ , and rank of two positions  $r^1$  and  $r^2$  on list  $y$  such that  $|r^1 - r_{x_i}| < |r^2 - r_{x_i}|$ , the ranks of two lists  $x$  and  $y$  of equal length are positively correlated when  $P(r_{y_i} = r^1) > P(r_{y_i} = r^2)$ . They are negatively correlated when  $P(r_{y_i} = r^1) < P(r_{y_i} = r^2)$ . They are independent when  $P(r_{y_i} = r^1) = P(r_{y_i} = r^2)$ .

That is, if the lists are positively correlated, the items with same index are likely to appear at nearby locations in two lists. Using the above definition, we can state the following result about our modified correlation measure:

**Theorem 5** For any sample set  $s$  of the items with ranks in the range  $0 \leq r \leq \frac{3}{4}r_m$ , the following holds. When the ranks of the two lists are independent, then the expected value of the correlation measure for a set  $s$  is  $E(\rho'_s) = 0$ . When the two lists are positively correlated, then  $E(\rho'_s) \geq 0$  and when the lists are negatively correlated, then  $E(\rho'_s) \leq 0$ .

With the Definition 9, we can construct a relation of the probabilities of an item being at specific ranks. We use this relation to compute the sign of the expected

value of  $k_i$  of an item in set  $s$  and also the sign of  $E(k_{x_i}k_{y_i})$  in the numerator in Equation 4.10.  $E(\sum X) = \sum E(X)$ , so we can determine the sign of the expected value of the correlation measure of set  $s$ .

### 4.3 Experiments

We evaluated the effectiveness of our approach against the Max-Sum algorithm on two sets of problems with n-ary constraints. We call the Max-Sum with G-FBP technique as MS+G-FBP. Also, the approach that selectively applies G-FBP based on data correlation is experimented as selective Max-Sum+G-FBP. For fairness of comparison, we used an implementation of Max-Sum that uses dynamic programming with the worst case complexity of  $O(N^m)$  for a single constraint instead of the standard Max-Sum with  $O(mN^m)$  where the arity is  $m$  and the domain size is  $N$ . The two sets of DCOP instances that are used in our experiments are:

- 50 instances of random graphs with 25 variables with domain sizes from 10 to 30, and 15 constraints with the maximum arity of 2,3,4 or 5 with different average constraint arities.
- 25 instances of graphs in the radar coordination domain with 48 variables with domain size up to 15 and 96 constraints with the maximum arity 4.

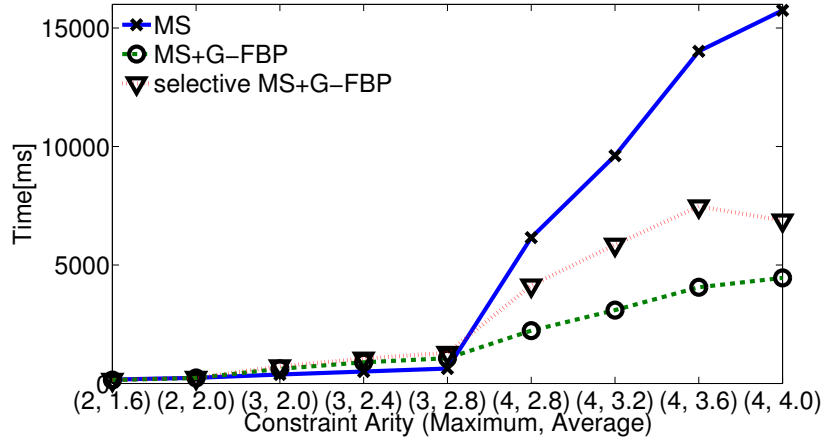
We focus on the computational aspect of the algorithms because our approach does not affect the solution quality. Because the computational complexity of DCOPs is determined by the constraint arity among the parameters related to graph topology as well as by the domain size, we experiment on varying these two parameters .

#### 4.3.1 Random Graphs

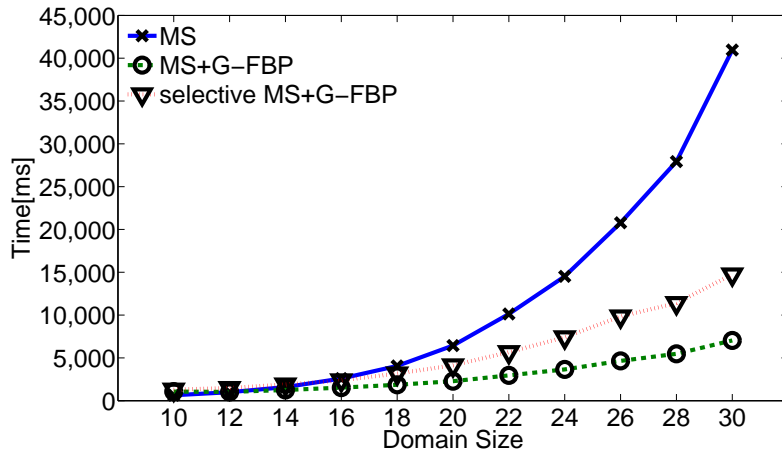
Our initial tests perform a comparison over randomly generated DCOPs with n-ary constraints. We characterize each scenario by the maximum constraint arity ( $m_{max}$ ), an average constraint arity ( $m_{avg}$ ) and the variable domain size ( $N$ ). We have



explored scenarios with  $m_{max}$  from 2 to 5,  $m_{avg}$  from 1.6 to 4.4 with an increment of 0.4 and  $N$  from 10 to 30. From our knowledge, this problem set is one of the most computationally expensive problems for a DCOP. For the first problem set, we fixed the value  $K$  to 2 in regard to the length of the sorted parts of lists. This value is chosen based on the probability analysis from Section 4.2.3.



(a) Computation time as the constraint arity increases



(b) Computation time as the domain size increases

Figure 4.4: The computation time of Max-Sum(MS), Max-Sum with G-FBP(MS+G-FBP) and Max-Sum with G-FBP with correlation measure(selective MS+G-FBP) . For Figure(a), the domain size of 10 was used. Datapoint (5, 3.6) is omitted to see the general trend. The performance of algorithms was (507178.5, 55070.6, 195056.9) for MS, MS+G-FBP and selective MS+G-FBP respectively. For Figure(b) the arity setting of (3, 2.8) was used.

Figure 4.4 shows the results for various arity settings and domain sizes. We observe that Max-Sum with G-FBP technique (MS+G-FBP) clearly outperforms Max-Sum (MS) for higher arities and larger domain sizes. Concretely, with G-FBP technique, the performance improved by 89% for an arity setting of (5,3.6) and the domain size of 10 and the performance improved by 82% for the domain size of 30 and arity setting of (3, 2.8). As the constraint arity and domain size increases, the number of entries that MS+G-FBP examines does not increase as much as the number of total entries. This increases the gain of MS+G-FBP. For lower arities and smaller domain size, MS performs better than MS+G-FBP because the overhead of sorting partial lists dominates the gain from finding the maximum value for shorter lists.

However, the use of the correlation measure in selective MS+G-FBP is not beneficial in this problem sets. Because the randomly generated constraint values leads the independence explained in Section 4.2.4 to hold, and thus there is no benefit in selectively applying G-FBP here and causes an additional overhead of computing the correlation measure as shown in Figure 4.4.

### 4.3.2 Multiagent Radar Coordination Domain

Our next problem set is created from the abstracted radar coordination and scheduling application based on the real-time adaptive NetRad system [57]. Radars collect real-time data on the location and importance of phenomena and the system schedules the radars to focus their sensing on scheduled weather phenomena. This scheduling step can be thought of as a DCOP. See [5], for more details on the formulation.

We developed a simulator in the Farm simulator framework [58]. Although it is a simulation environment, the utility functions are constructed based on the real scenario and the same utility function is used in the deployed system [57]. Our scenario involves 48 radars with a scenario of 96 phenomena with random locations,

size, and type. The radars are placed in a grid with overlapping regions with other radars. This scenario creates problem instances with 48 variables, 96 constraints with the maximum arity of 4. In this data set, we do not directly control the constraint arity nor the domain size. These numbers vary in the experiments, so we categorized the computational difficulty of each problem instance by the maximum factor size. The maximum *factor size* is computed as the number of recorded entries in the constraint functions totaling  $O(mN^m)$  for an  $m$ -ary constraint, where  $N$  is the maximal domain size of associated variables. We report the average runtime of 25 runs.

As shown in Figure 4.5(a), both MS+G-FBP and MS+GSC-FBP outperform MS with an appropriate  $K$  as discussed in Section 4.2.3. However, there is not a significant difference between them when a reasonable  $K$  is chosen for at least this domain. As shown in Figure 4.5(b), the time savings in later iterations of MS+G-FBP dominate the sorting overhead in the initial iteration, and leads to superior performance to MS. MS+G-FBP takes 36% less computation time than MS for the factor sizes in the range  $(10000, 40000]$  and  $K=11$ . The performance improvement by MS+G-FBP is not so significant as on the first dataset, because most constraints have lower arity and some variables have smaller domain size than the one related to the maximum factor size and also because of the data dependencies. Unlike randomly generated data sets, here variables have more structured dependencies through constraint functions and the independence assumption in Section 4.2.4 does not hold in this domain and MS+G-FBP performs poorly on instances of strongly negatively correlated lists. Therefore, we examine the use of correlation measure on this domain further.

On Experiments in Figure 4.6, we selectively applied G-FBP scheme (selective MS+G-FBP) when the correlation measure with the sample set of  $\frac{K\sqrt{N}}{2}$  largest items in a value list computed as in Equation 4.10 is positive. Selective MS+G-FBP takes 55% less computation time than MS in contrast to 36% for MS+G-FBP. As in Figure 4.6(b), selective MS+G-FBP almost always finds the maximum item in sorted

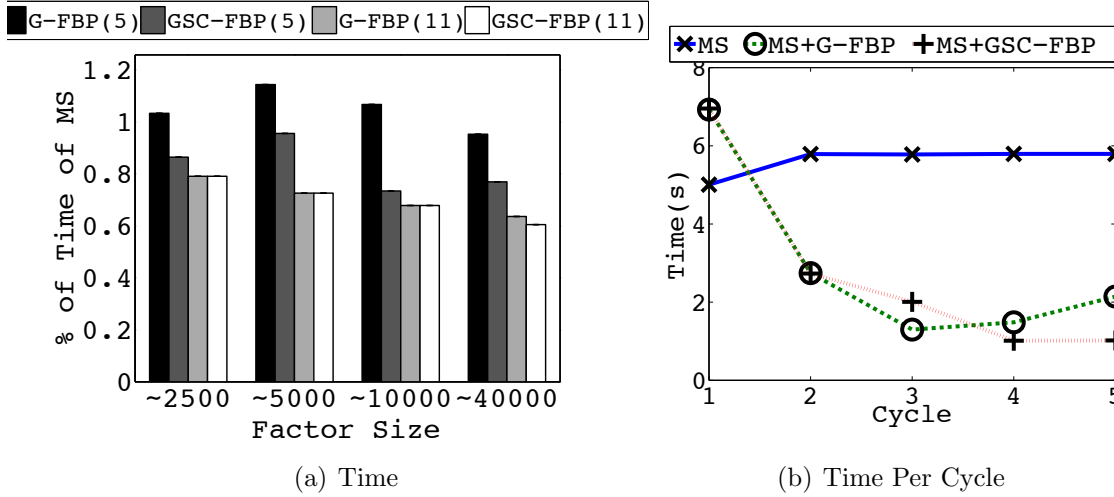


Figure 4.5: (a) The computation time ratio of MS+G-FBP and MS+GSC-FBP to MS.  $K$  value is in brackets. (b) Time taken at each cycle.  $K = 11$ .

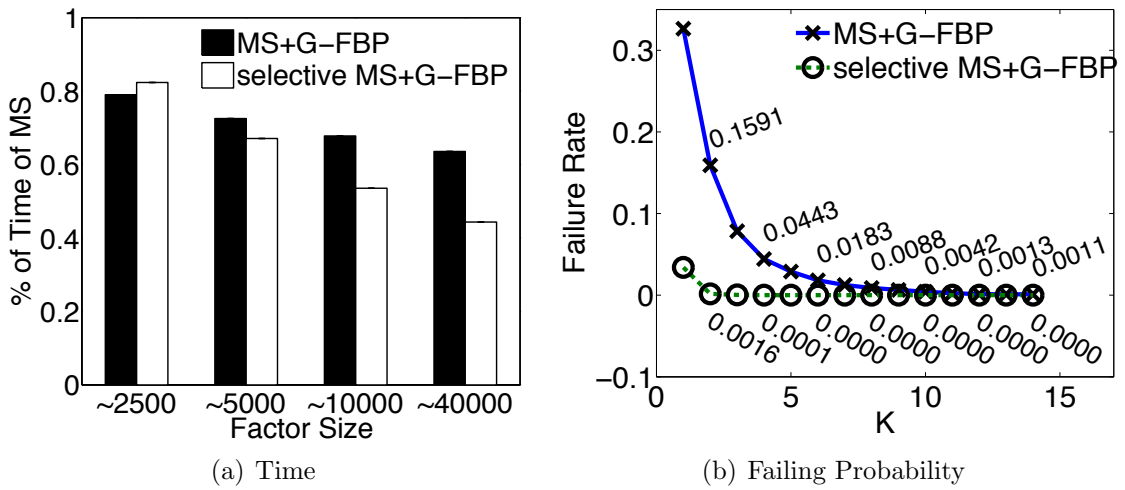


Figure 4.6: Performance improvement using correlation measure.  $K = 11$ .

parts of lists and the failure rate becomes zero when  $K > 8$ . Note that MS+G-FBP sometimes fails even with larger  $K$  values.

#### 4.4 Conclusion

We presented a new approach, called generalized fast belief propagation (G-FBP), which optimizes the key computational bottleneck of the maximization operator in the popular Max-Sum algorithm. Our approach is applicable to a general setting in the context of arbitrary arity graphs as opposed to some previous approaches which operate only on pairwise graphs. We provide a significant reduction in the time complexity of computing a single message in the Max-Sum algorithm from  $O(N^m)$  to  $O(mN^{\frac{m+1}{2}})$  for general  $m$ -ary graphs. The key idea of our approach that distinguishes it from previous approaches is that only a small number of values are accessed from partially sorted lists to efficiently perform the maximization operation in Max-Sum, rather than performing the complete sorting. We also provide theoretical results regarding the number of samples required and a proof of expected complexity.

We also devised a correlation measure to determine whether to apply G-FBP based on correlation of order statistics of data. We showed that in the radar domain where the data correlation occurs, using this correlation measure brings down the probability of using G-FBP on negative correlated data close to zero.

Finally, we added another extension that reuses computation from the previous iteration. It shows effectiveness when the messages are near convergence and reduce the computation time further from Max-Sum with G-FBP.

## CHAPTER 5

### EXPLOITING THE MAPPING OF AGENTS AND VARIABLES

In this chapter, we discuss how to utilize agents as a component of the DCOP model. In order to efficiently solve DCOPs, we consider environments where an agent may control more than one variable. When agents control multiple variables, the variables in each agent form a partially centralized structure. We tackle the issue of how to modify distributed algorithms to exploit this partial centralization. We are interested in how to reduce the communication and computational burdens by exploiting low overhead communication among variables within an agent. This problem has not been studied extensively by the DCOP community where agents and nodes commonly have a one-to-one mapping in which every message leads to communication among agents.

#### 5.1 The Configuration of the Semi-centralized Structure and NetRad domain

In addition to the basic model of DCOP  $\langle A, X, D, F \rangle$  in Section 1.1, We define an additional parameter  $R$  to the DCOP model that maps agents to variables and constraint functions. This mapping function  $R$  assigns the variables  $\mathbf{v}$  in  $\mathbf{X}$  to an agent in  $\mathbf{A}$ . We call the variables that are mapped into agent  $a$  as members of  $a$ . Each agent controls the value of its members.

The agents handles outgoing and incoming messages of the members as well as the computation regarding the members. The communication among members of an

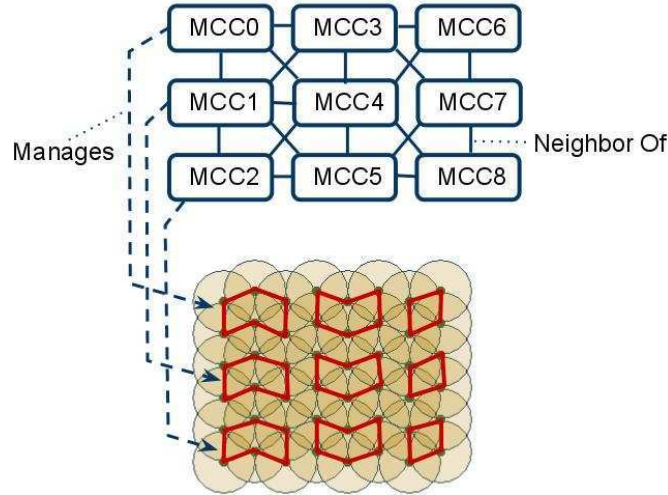


Figure 5.1: An example structure of the NetRad domain where an agent owns multiple variables: An MC&C controls and manages multiple radars

agent is free whereas across members of two different agents are commonly not free. In NetRad domain already introduced in Section 1.3.1, each MC&C agent contains a subgraph that represents the radars (variable nodes) that are controlled by that MC&C. However, these subgraphs are not totally disjoint since weather phenomena (function nodes) may span multiple MC&Cs where radars in these MC&Cs can scan the phenomena. The phenomena that are shared among MC&Cs incur communication cost because the radars in the scannable range of these phenomena reside in multiple MC&C and they need to be coordinated through non-free communication messages.

## 5.2 Exploiting Semi-Centralization

Decentralized coordination in large-scale systems of cooperative agents requires large computational and communication resources and often is not feasible in realistic problems requiring real-time deadlines: The decentralized computation of an optimal coordination policy often requires control overhead involving the construction of the structure for solving the problem, e.g. establishing a pseudo-tree for DPOP. In a

realistic decentralized setting of multiple agents, the establishment of a constraint optimization problem often occurs online and needs to be solved as quickly as possible and thus control overhead should be minimized. Also, the requirement for quick delivery of the solution limits the amount of communication or computation. Even without these problems, as networks get very large, communication and computation burden still can be extensive given the time and bandwidth constraints and thus minimizing these resource requirements in these settings is valuable.

There have been studies [59, 60] on exploiting semi-centralization on distributed constraint satisfaction problems (DisCSP), however both works, which focus on pruning non-satisfying local solutions before checking satisfiability across agents, do not apply to DCOPs where local pruning may eliminate globally optimal solutions. There are works [26] on DCOP that try to partially centralize computation in order to efficiently find the optimal solution in local neighborhoods. The partial centralization reduces communication and achieve the optimality of the solution. However, this centralization is not concerned with the mapping from variables to agents. The scope of local search is determined by the constraints among variables and intermediate state of the search process of each agent. Thus, instead of naively solving multiple nodes' tasks in each processor, as we would in the totally centralized environment, we try to use the partial centralization given by the hardware structure to reduce the communication and computational resources needed.

Given the fixed mapping of nodes and agents, we try to exploit partial centralization within agents. In order to improve the algorithmic performance using partial centralization, we exploit locality of nodes' dependency. We hypothesize that because the nodes' dependency is restricted to a subset of nodes that are adjacent or close in proximity in the constraint graph, readily transferring information of such proximal nodes helps nodes' reach solutions more quickly. Since nodes tied to one agent has



such proximity and access to the information of other nodes contained in one agent have much lower cost, aggressive information processing within agents is reasonable.

### 5.2.1 Using Organization Structure

In so doing, we modify the Max-Sum algorithm to cater to this partial centralization in each processor by adding a local processing stage and creating denser messages reducing the communication overhead and computational burden. We modify Max-Sum to work on two levels for general graphs with cycles in order to increase the algorithm efficiency in the context of clustered hardware resources. We modified the message passing schedule of Max-Sum to propagate sometimes within a subgraph of the factor graph associated with clustered hardware resources. There is no modification to the algorithm except skipping the computation of outgoing messages to the nodes outside the partition, thus saving inter-processor communication. This modified Max-Sum, which we call MS2L, alternates between a global propagation cycle and a local propagation cycle so as to ensure that the utility values can also travel to other parts of the graph.

In the algorithm, information is shared among nodes through messages and the algorithm converges to a single point when there is no new information flowing in any direction [17]. We conjecture that the communication can be more efficient when this information sharing is delayed until a subset of nodes become closer to consensus. By delaying sending messages outside the local processor until more developed values are constructed within each MCC, we expect to reduce inter-processor communication without affecting overall performance. Also, in order to avoid getting into local optima, we ensure that the algorithm periodically communicates globally. Therefore we modify Max-Sum to have following message passing schedule.

This scheme is different from simplifying the problem by breaking the network into several subgraphs. It only delays the message delivery to make communication more

efficient and the computational complexity remains same even in local flooding. This modification explores how Max-Sum can adapt to the system’s organizational structure and its associated communication topology as well as the utility structure. In the NetRad domain, as the connection between function nodes and variables nodes are determined based on the spatial location of the phenomena and corresponding radars, dependency structure between nodes are simpler and factor graphs constructed following the domain are more easily decomposable. We exploit this property of the graph structure in the domain and modify Max-Sum to reduce the required resource for global level propagation.

Also, in NetRad, the system has an organizational structure where an MCC manages several radars. In this system, MCCs communicate to coordinate with neighboring radars. Because there is a cost to communicate across MCCs, it is beneficial to simplify computation and communication occurring across MCCs. Therefore, we adapted Max-Sum to exploit this structure effectively by skipping some outgoing messages from MCCs in alternating cycles.

1. (Initialization) At any vertices, carry out the global flooding step.
  
2. (Local flooding) Both variable and function nodes send messages only to the neighbors within the same MCC. For each local neighbor, given the newest message on each edge, compute the message values for each local neighbor and send. Let the neighbors of variable node  $i$  be  $N_i$  and the nodes in MCC  $k$   $m_k$ . In function nodes, it sends the same message to a subset of neighbors  $N_i \cap m_k$ . In variable nodes, it computes the message using the previous messages from neighbors outside the MCC. At cycle  $t$ , the message from the variable to function node is,

$$q_{i \rightarrow j}^t(x_i) = \alpha_{ij} + \sum_{k \in N_i \cap m_k \setminus j} r_{k \rightarrow i}^t(x_i) + \sum_{k \in N_i \setminus m_k} r_{k \rightarrow i}^{t-1}(x_i)$$

3. (Global flooding) For all neighbors, do a regular message calculation using the newest message on each edge. Function nodes compute the messages at cycle  $t$  for all neighbors using messages at  $t-1$  for neighbors  $N_i \setminus m_k$ . The function node does not have updated messages for all neighbors due to local propagation in the previous cycle thus it combines previous messages from neighbors outside MCC.

$$r_{j \rightarrow i}^t(x_i) = \max_{\mathbf{x}_j \setminus i} [F_j(\mathbf{x}_j) + \sum_{k \in (N_j \cap m_k \setminus i)} q_{k \rightarrow j}^t(x_k) + \sum_{k \in (N_j \setminus (i \cup m_k))} q_{k \rightarrow j}^{t-1}(x_k)]$$

4. Repeat step 2 and 3.

### 5.2.2 Starting with Known Policy

We also generate an initial policy that incorporates the information of constraint functions local to each processor for the Max-Sum algorithm which normally starts with no information on neighboring constraints. This information is computed using partial constraint graphs. These partial constraint graphs are constructed based on local variables and functions within the agent, and thus can be quickly generated and computed. As a result of this local processing that has low communication cost, the algorithm starts closer to the final solution.

We speculate that a good known policy can be used to create such starting messages as it incorporates non-local information than just the information in a local function. In the Max-Sum algorithm, a node's outgoing messages are dependent on the incoming messages it received in the prior cycles. Also, the first initial messages will depend only on the local functions. The variable after the first message would take on the value

$$\tilde{x}_i = \arg \max_{x_i} \sum_{j \in N_i} \max_{\mathbf{x}_j \setminus i} F_j(\mathbf{x}_j) \quad (5.1)$$

This message would be the value assuming the best-case setting of other variables and only incorporates the local preferences. Given a known policy  $\hat{x}$ , we modify the algorithm for function nodes to send the following messages which do not involve maximization to the connected variable nodes. Function node  $j$  to variable node  $i$ :

$$F_j((\hat{\mathbf{x}}_j \setminus i) \cup x_i) \quad (5.2)$$

After receiving these messages, if a variable node were to take on a value, it would be:

$$\tilde{x}_i = \arg \max_{x_i} \sum_{j \in N_i} F_j((\hat{\mathbf{x}}_j \setminus i) \cup x_i) \quad (5.3)$$

**Proposition 1** *If the assignment  $\hat{\mathbf{x}}$  is such that no individual variable can by itself change its value to increase the global utility, then  $\hat{\mathbf{x}}$  is a solution to the assignment constraints imposed by Equation 5.3. If changing any individual variable's value will strictly decrease the global utility, then  $\hat{\mathbf{x}}$  is the unique solution for Equation 5.3.*

From the perspective of an arbitrary variable node  $i$ , all other nodes are fixed to the configuration specified by  $\hat{\mathbf{x}}$ . Maximizing  $\sum_{j \in N_i} F_j((\hat{\mathbf{x}}_j \setminus i) \cup x_i)$  leads to maximization of the global utility given the values of other variables. This is because only the functions for nodes  $j \in N_i$  are affected by  $x_i$ .

If  $\hat{x}_i$  were not a solution to this, then the algorithm which selected  $\hat{x}_i$  to be part of  $\hat{\mathbf{x}}$  could have instead selected  $\tilde{x}_i$  to receive a higher utility. Since by supposition, no individual variable can change its value to increase the global utility,  $\hat{x}_i$  is a solution to Equation 5.3. If changing any variable's value in  $\hat{\mathbf{x}}$  will decrease the global utility, then there can be only one solution to Equation 5.3. Since  $\hat{x}_i$  is a solution, it must be the unique solution.  $\square$

Thus, in the sense of the above property, we can insert a variable assignment into a factor graph as a starting solution. The property requires that no single variable can change its value to increase the utility. This is a desirable property for an optimization algorithm to have, and a fairly lax one. Any algorithm which does

not satisfy this constraint can be followed by a hill-climbing procedure in order to meet the requirement of Property 1.

In addition to what Property 1 can tell us, Equation 5.3 by itself looks quite a bit better than Equation 5.1. While the assignment still only considers directly neighboring function nodes, it does so using better assumptions. For nodes other than itself, it assumes a configuration that is known to exist rather than a separate maximization for each function node. The assumed variable assignments are also known to be consistent with a good global utility, and  $x_i$  will fit itself into this assignment.

After the messages from function nodes to variable nodes, we allow the variable nodes to send one set of messages before proceeding with the regular algorithm. This is so the next set of messages from function nodes will have a starting point other than assuming uniform functions in variable node messages.

### 5.2.3 Using the Structure for Policy Generation

We provide a scheme which computes a policy which can be used as in Section 5.2.2. Instead of generating a policy for the whole problem, we tried to compute the locally optimal policy for subproblems associated with each MCC (See Figure 5.2). We break the full factor graph into factor subgraphs for each MCC that contains only the radars and phenomena in each MCC and each is smaller than the original factor graph. In this way, we first solve a smaller problem within MCCs and then solve a bigger problem using the information from the smaller problem. This is the key difference between local propagation in Section 5.2.1 as we break down the factor graph into subgraphs each of which has decreased complexity.

In order to accomplish this, we assign each phenomenon to one MCC to avoid redundant utilities for shared phenomena in computing the initial policy. Consequently, the domain of variable nodes and parameter values in the cost function at

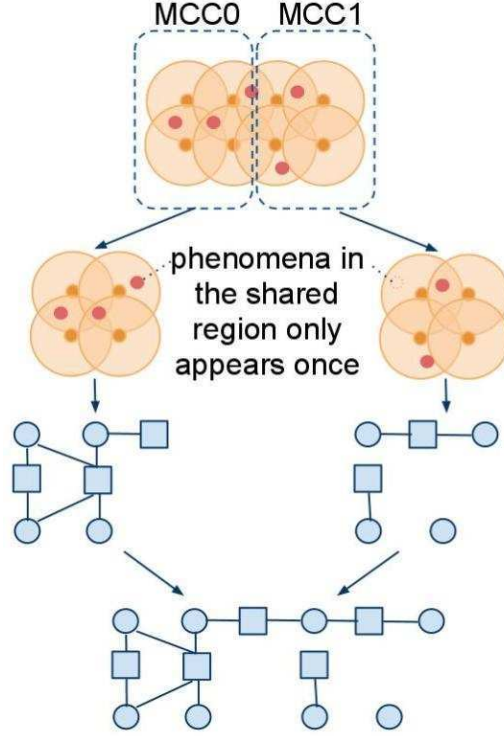


Figure 5.2: 2-level Hierarchy Scheme

the function nodes are smaller than the original problem. Thus the computation at each function node  $f_j$  that belongs to the set of nodes  $m_k$ , which belong to  $MCC_k$ , is done only for each neighbor  $v_i \in m_k$ .

$$r_{j \rightarrow i}(x_i) = \max_{\mathbf{x}_j \setminus i} [F'_j(\mathbf{x}_j \wedge m_k) + \sum_{k \in N_j \wedge m_k \setminus i} q_{k \rightarrow j}(x_k)] \quad (5.4)$$

The message from variable node  $v_i$  to function node  $f_j$  for  $f_j \in m_k$  is,

$$q_{i \rightarrow j}(x_i) = \alpha_{ij} + \sum_{k \in M_i \wedge m_k \setminus j} r_{k \rightarrow i}(x_i). \quad (5.5)$$

We assume that function  $F'_j$  with a subset of arguments that excludes the variable nodes outside the MCC can be deduced from the original function  $F_j$ . Additionally, the domain of variable  $v_i$  is a subset of its domain in the original problem only relevant to  $f_j \in m_k$ . The subproblem is used to create the policy used as prior information on local functions.

## 5.3 Experimental Results

### 5.3.1 Experimental Setting

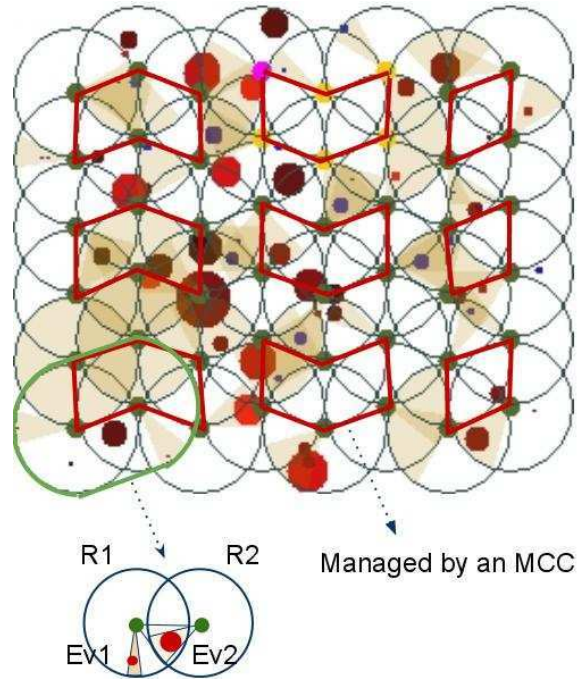


Figure 5.3: Radar 1 (R1) can choose to scan Event 1 (Ev1), Event 2 (Ev2) or to scan both depending on the utility. Scanning all phenomena in a radar’s range of sufficient quality may not be possible given the time limit to scan.

We experimented with the Max-Sum algorithm on an abstract simulation environment of the NetRad radar system developed in the Farm simulator framework [58]. In this simulator, weather tasks are abstracted as circular areas as shown in Figure 5.3. Aspects such as the utility function, the effective range of radars, and the separation between radars, however, are the same as in the operational testbed. For further information on the simulation environment, see [61].

For a statistically meaningful result, we repeated each instance for 100 runs by randomly generating the weather phenomena varying in their size, location and importance. To make the results more easily interpretable, each trial is run for only one radar scan cycle rather than for a sequence of radar scans.

The computation time is measured in CPU Time. The experiments were run on a single machine although we assumed that there are several computation units working in parallel in a simulated time step. The time complexity in the decentralized setting results from the sum of the longest time taken in the local computation at each MCC for each round.

We do not account for any communication delay in measuring the completion time. For measurement on communication amount, the control messages to construct the network as well including the connectivity establishment between nodes and information sharing on possible values that each variable can take were counted as communication. The total amount of communication is measured in bytes considering one double number as 8 bytes and one integer as 4 bytes.

### 5.3.2 Performance of Max-Sum on NetRad

In order to evaluate the performance of several alternative optimization algorithms, we varied the number of radars and the number of phenomena. We compare the performance of the Max-Sum algorithm, a decentralized negotiation algorithm [61], an exact distributed constraint optimization algorithm [62] and a centralized optimization algorithm based on a genetic algorithm that is currently used for local optimization in the negotiation algorithm in each MCC. The negotiation algorithm, specifically developed for the NetRad problem domain, is an iterative two step process performed concurrently at each MCC. In the first step, each MCC performs a local optimization based on its local tasks and knowledge of its neighboring MCCs's proposed scan schedules. In the second step, MCCs negotiate with their neighbors so as to make adjustments to their scheduling based on the strategy of other MCCs. This two step process for performing the distributed optimization tries to maximize the parallelism at the MCC level and to minimize communication among MCCs. In contrast, the standard Max-Sum algorithm does not consider such an organizational



structure and is completely decentralized. The Max-Sum algorithm does not explicitly take into account that certain communication links are within an MCC cluster and others are between MCC clusters. The genetic algorithm uses a centralized approach; no communication is required and it utilizes only one processor.

### **Different Network Sizes and Number of Phenomena**

In order to evaluate the general performance and the scalability of the alternative optimization algorithms, we evaluated the performance on different sized networks with different number of phenomena. In scenarios with different sized networks, there are the same number of phenomena as the number of radars in the network as shown in Figure 5.4. We also compared the algorithms with DPOP [62] which gives globally optimal solution on 48 and 96 radar cases; however, we were not able to use DPOP on bigger networks due to memory constraints. We have varied the parameters of the genetic algorithm to improve its performance but it still remains inferior to other methods. This conclusion applies similarly to an optimization algorithm based on Simulated Annealing not shown in the graph due to its general inferior performance on the problem.

In the next set of experiments, the results of which are shown in Figure 5.5, we increase the number of phenomena in a 48-radar network, thereby requiring more coordination among radars and studied how the algorithms perform. The solution quality of Max-Sum remains the highest while the time complexity remains lower than the negotiation algorithm. In terms of communication between MCCs, Max-Sum requires more communication than negotiation, but remains significantly lower in comparison to DPOP as seen in Table 5.4(f). The communication amount of the Max-Sum algorithm increases as shown in Figure 5.5(d) because the number of the variables connected to function nodes in Max-Sum increases as more weather phenomena are added. Further, communication in the negotiation algorithm decreases in denser

networks since this algorithm finds it difficult to find a good strategy to negotiate, resulting in an early termination within only a few negotiation cycles ; however, this is not a good thing since it achieves only 90% of the solution quality of Max-Sum in 120 phenomena case.

### 5.3.3 Starting with Initial Policy

As described in Section 5.2.2 and Section 5.2.3, we provided the algorithm with 3 different approaches for computing the initial policy of a 48 radar network with the limit of 10 rounds, where we took the results at the last cycle. The initial policies includes : 1) the solution from the genetic algorithm within MCCs (IGen), 2) the solution of the Max-Sum algorithm within MCCs (IMS) following the scheme described in Section 5.2.3, and 3) a randomly generated initial policy (IRand). The initial policy for IRand was generated in each function node, which leads to potential inconsistencies among multiple function nodes; however these inconsistencies are often quickly resolved.

The use of an initial policy in all cases helps the algorithm save computation time as shown in Figure 5.6(a) and also improves its anytime performance behavior as in Figure 5.6(c). The quality of the final solution rarely changes by more than a fraction of utility and the computation time including policy generation decreases. Given the initial policy, Max-Sum can almost instantly produce the initial messages avoiding the maximization step and the policy also reduces the number of rounds taken to converge.

Using initial policies, Max-Sum seems to produce a solution with high utility quickly because the messages at the initial round are not biased towards the local functions reaching the final solution quickly as shown in Figure 5.6(c).

IMS provides computation time saving as well as quick convergence speed and stability, although it requires specific domain structure to be effective. Addition-

ally IRand shows less anytime characteristics and some unstable performance, e.g. showing oscillation between multiple values ending up a higher number of rounds to converge as in Figure 5.6(d), but it is a quick and easy way to provide an initial policy without the need to synchronize policies over multiple nodes.

### Policy Generation Using Structure

We experimented with the algorithm using IMS further to show how much we can improve the algorithm using the policy. We ran Max-Sum for 5 cycles where, for Max-Sum with IMS (MS-Init), we replace the first two cycles with initial policy computing cycles.

As shown in Figure 5.7, not only does the performance quality remain similar to Max-Sum, the time complexity decreases by half as well as the communication amount. This computational saving is due to the fact that the computation on the factor graph using only local nodes is much simpler than the computation on the global-level factor graph and also the result of this computation leads to a quicker convergence on the global level. As messages are exchanged only within MCCs to compute the initial policy, the number and size of messages also decreases.

#### 5.3.4 Performance of Max-Sum in a Two-Level Hierarchy

We next tried MS2L, the alternating Max-Sum algorithm that uses a repetitive cycles of local and global propagation of messages as in Section 5.2.1. We experimented with increasing number of phenomena and also with the IMS policy replacing the first 2 cycles for generating the policy. We ran experiments with various number of phenomena, as we would like to see the result of MS2L varying the network connectivity which will increase the need for global propagation.

As shown in Figure 5.8, the utility of MS2L with IMS (MS2L-Init) remains similar. Moreover, the communication is reduced by half. It also shows the result of MS-50Comm where we randomly skip the communication 50% of the time and note that

the algorithm has not yet converged in the given number of cycles unlike MS2L. Even in the local propagation cycle, the utility is being propagated as effectively as the global propagation cycle and half of the global propagation cycles are enough to reach similar performance. Also by starting with IMS, the computation time can be also decreased as shown in Figure 5.8(b).

## 5.4 Conclusion

We applied the Max-Sum approximate constraint optimization algorithm in the NetRad system for coordinating and scheduling weather-sensing radars. In this system, Max-Sum generated policies with high utility but requires more communication and computation than the negotiation algorithm in some settings. We thus modified Max-Sum to start with initial policy to guide and expedite the search process. Using initial policy improves Max-Sum's anytime performance and saves computation and communication by 50% for networks with the same number of phenomena as the number of radars. As part of using a policy, we generated a scheme to create a starting policy which works in the two-level hierarchy solving a partial problem within the local processor. Additionally, we developed MS2L, an adapted message passing scheme which alternates between different scopes of the message propagation. This scheme proved the benefit of exploiting the organizational structure by requiring less computation and communication than all other tested algorithms with about 50% communication savings and 5-30% computational savings for dense network where the number of constraints exceed the number of radars.

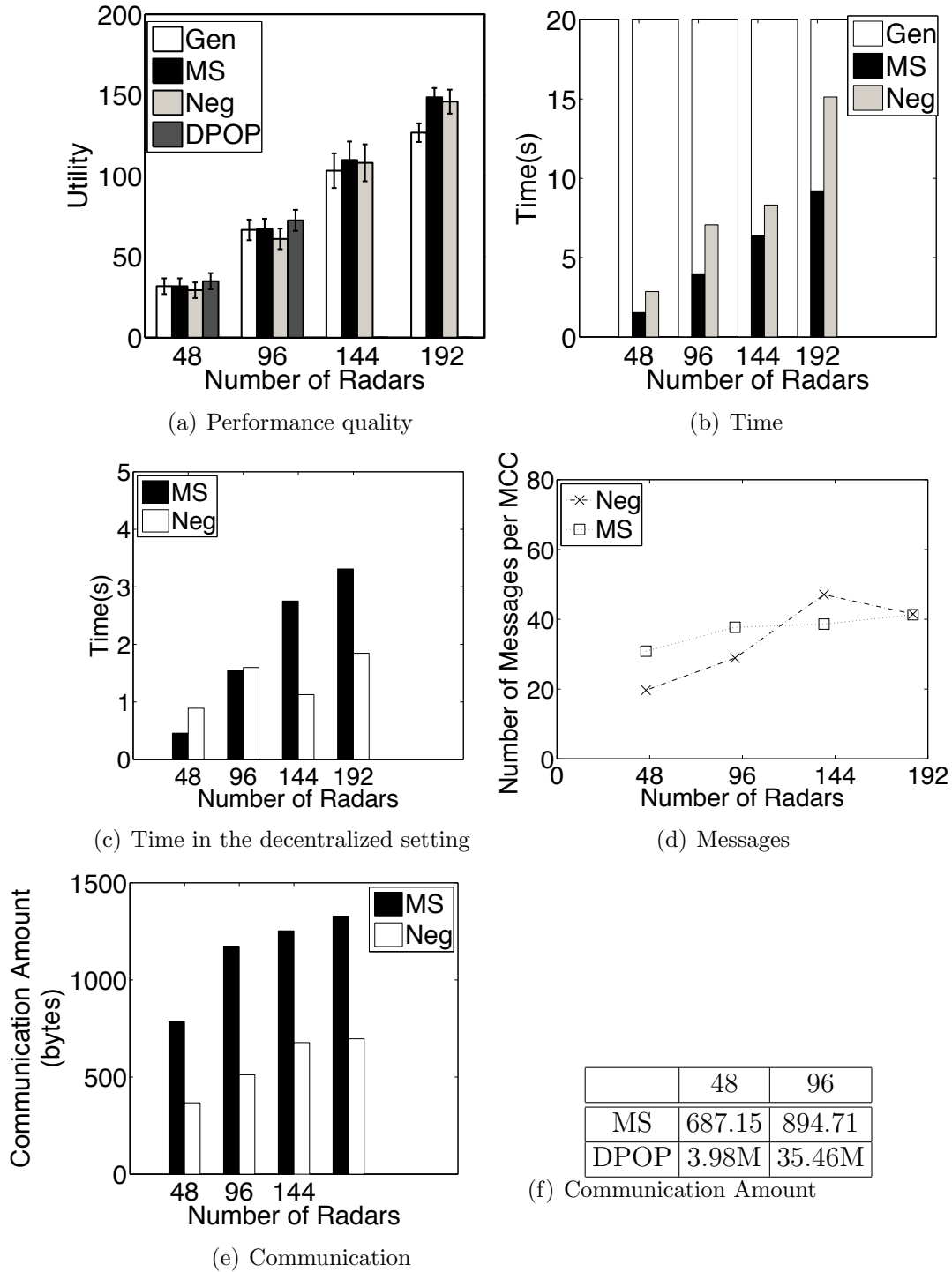
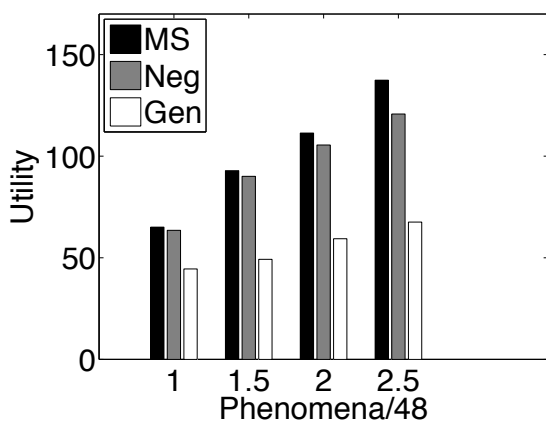
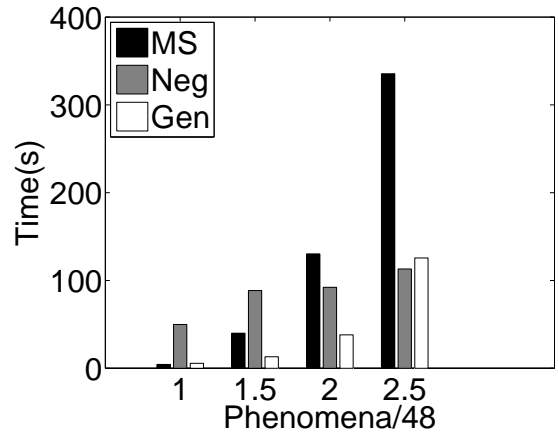


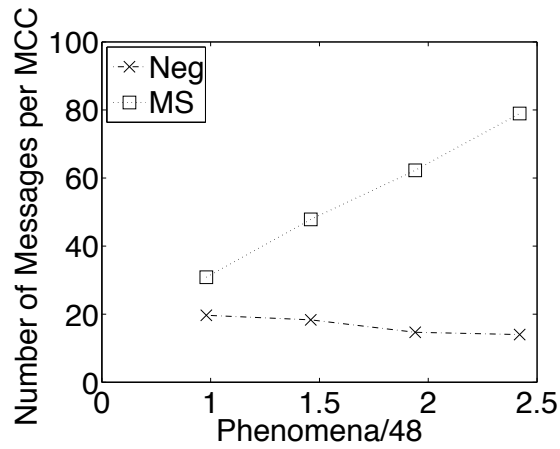
Figure 5.4: Gen:Genetic, MS:Max-Sum, Neg:Negotiation. The algorithm is run with the same number of tasks (weather phenomena) as the number of radars. The Genetic algorithm is run with a computation time limit of 10 minutes. We set the time limit to 10 minutes in order to get reasonable optimization. Given less than 10 minutes, the utility generated by the centralized optimization were significantly lower than the other approaches.



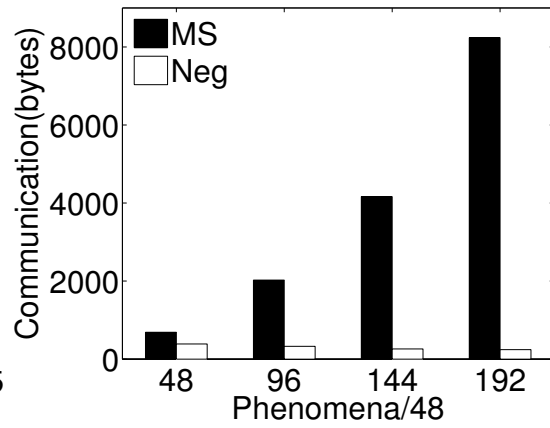
(a) Performance Quality



(b) Time Complexity

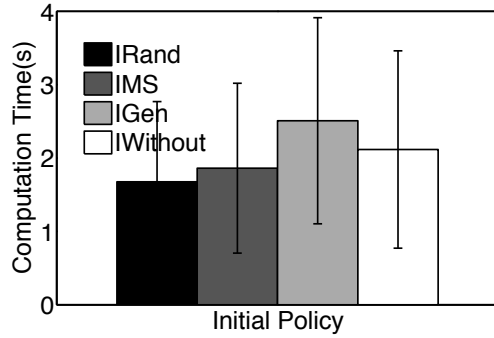


(c) Messages



(d) Communication

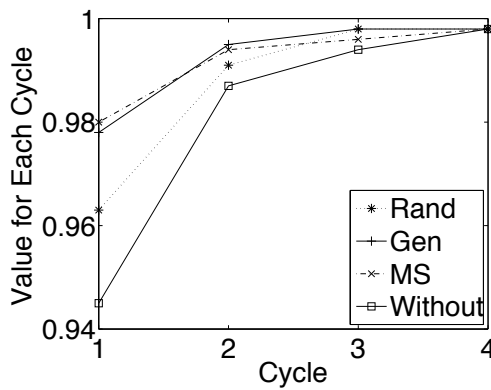
Figure 5.5: Experiment with different number of phenomena. The basis is 48 weather phenomena and this is increased to 120 phenomena (i.e. 2.5).



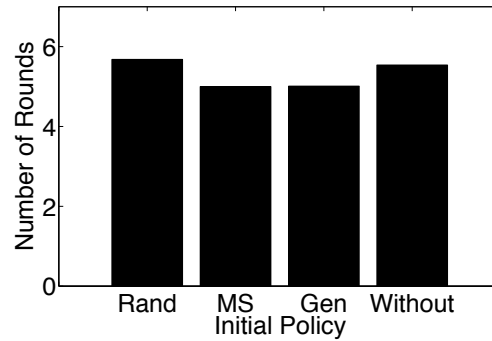
IRand	IMS	IGen
pass	pass	fail

(a)

(b)



(c)



(d)

Figure 5.6: The results using initial policy (a) The decentralized computation time of Max-Sum including policy generation time (b) T-test result on hypothesis that each policy improves the computation time of regular Max-Sum with  $\alpha = 0.05$  (c) Value convergence trend at each round (d) Number of Max-Sum rounds

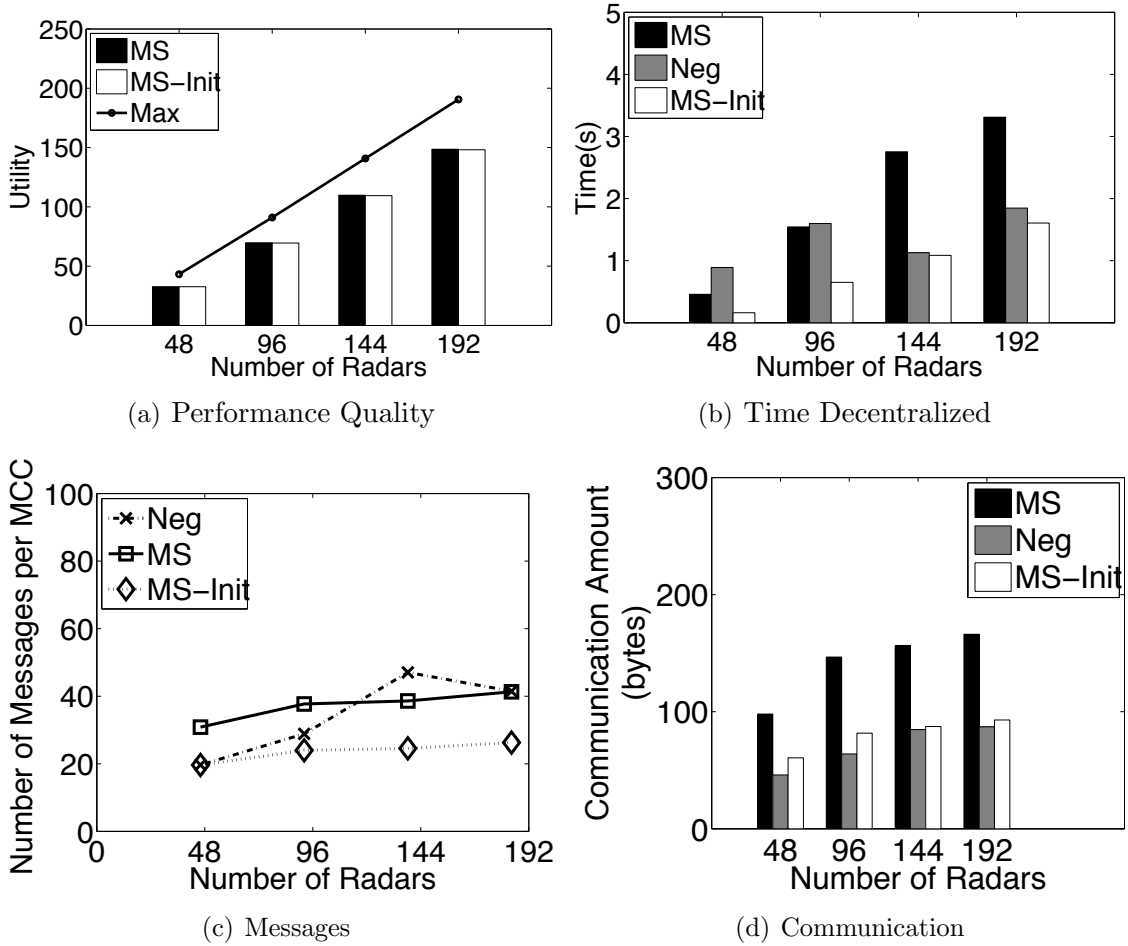


Figure 5.7: Performance of MS-Init. **MS**:Max-Sum, **Neg**:Negotiaton, **MS-Init**: Max-Sum with IMS



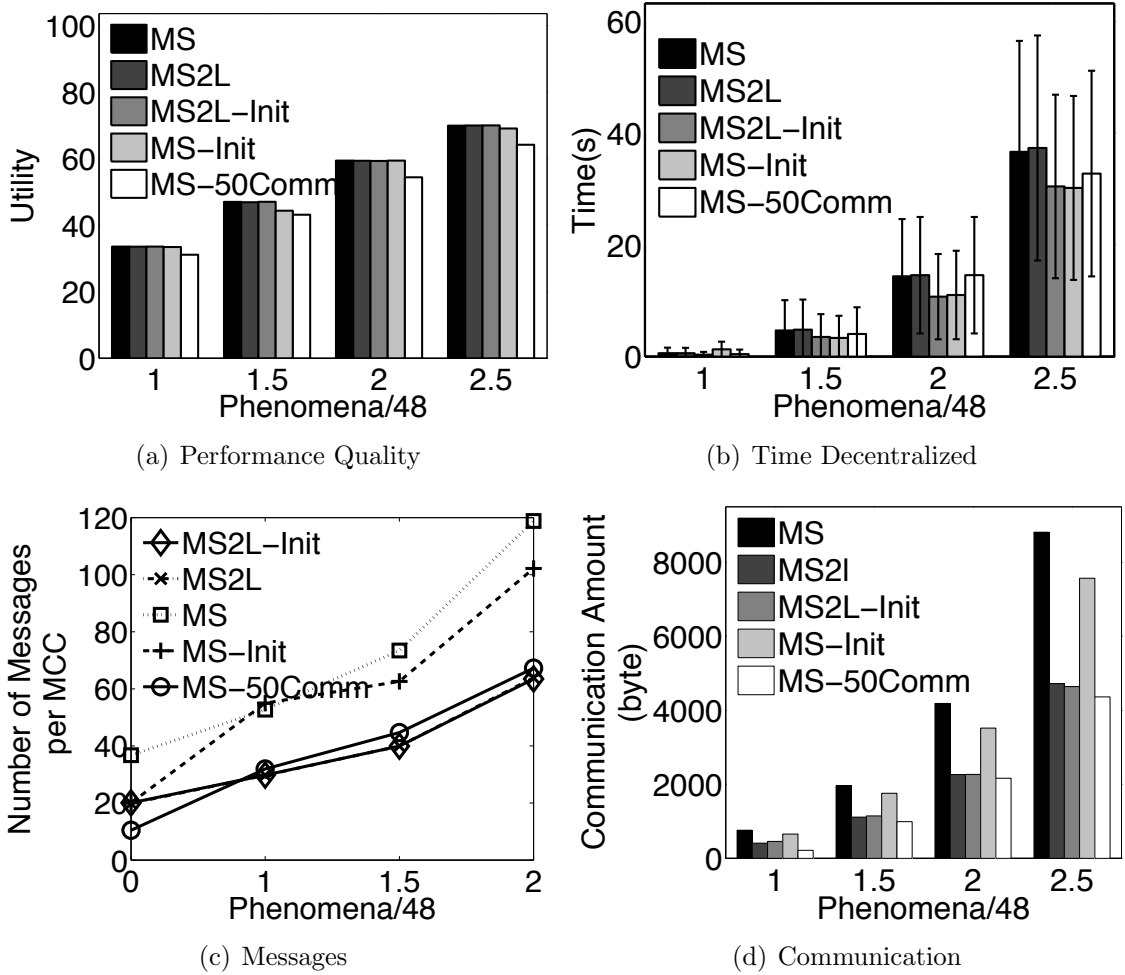


Figure 5.8: Performance of MS2L

## CHAPTER 6

### CONCLUSIONS AND FUTURE RESEARCH

This thesis sets out to develop new and extensions of existing distributed constraint optimization algorithms that require much less computational and communication resources than traditional approaches. We particularly utilize various techniques from graphical models researches as a starting point to construct these new approaches. We applied these techniques to applications including Sensor Network in a PEAV framework and, the CASA radar domain where there are fewer hardware processors than the number of agents and random instances with n-ary constraints. In this final chapter, we will summarize the research contributions of this work, as well as discuss directions for future research.

#### 6.1 Contributions

The work described in this thesis makes a number of important contributions to the state of the art in the distributed constraint optimization by providing approaches to handle high computational and communication complexity of DCOP algorithms. The contributions of this work can be summarized as follows:

- We addressed the problem of solving DCOP exactly and with precise approximation bounds by developing a new distributed algorithm called DJAO. There are three novel ideas in DJAO. Firstly, it uses an AND/OR junction graph representation, which builds a basis for efficient search in the distributed settings. The second is a two phase search strategy that combines characteristics of ADOPT and Action-GDL. Because it is a search strategy, the algorithm is also capable of

a bounded approximate optimization which requires significantly less communication. The third is a soft filtering technique to significantly reduce communication without losing any accuracy.

- We develop a technique, which we call G-FBP, that optimizes the key computational bottleneck of the maximization operator in the Max-Sum algorithm. Our approach uses order statistics and data correlation to reduce computational burden by partially sorting constraint functions. This is the first work on reducing the computational burden in n-ary constraints for DCOP. We provide a significant reduction in the time complexity of computing a single message in the Max-Sum algorithm from  $O(N^m)$  to  $O(mN^{\frac{m+1}{2}})$  for general m-ary graphs given the independence assumption on data. We also provide a correlation measure that selectively apply the technique given the data correlation. To our knowledge, this is the first work that a DCOP algorithm includes an adaptive scheme to emerging intermediate computation.
- We present techniques to reduce the communication overhead of the Max-Sum algorithm when each agent owns a set of variables. Our techniques exploits processor hardware organizational structure such that partial messages in the Max-Sum algorithm do not incur communication cost if message transfer occurs within a processor. Our work is the first work that exploits such setting for a DCOP. Our study shows that using the techniques communication can be decreased up to 50% by using the two-hierarchy message passing scheme and generated initial policy. Also, we showed that using initial policies, the algorithm shows a better anytime property leading us to save some computational resources as well. We empirically verified this result on NetRad radar domain.

In summary, this research provides new approaches to make DCOP applicable to a wider class of application domains.

## 6.2 Future Research

Future research in distributed constraint optimization includes improving existing algorithms for scalability, parallelism, and privacy. Existing algorithms for distributed constraint optimization domains can be applied to selective domains, However, many algorithms are not scalable to complex domains with a large size of problems due to computational and communication complexities. There are many approaches for MAP estimation from graphical models community to help the scalability issues in DCOP. We have applied some of the techniques from graphical models community to DCOPs to combat the complexity. In Chapter 3, we applied AND-OR search graph to provide a mechanism that combines ADOPT and Action-GDL. Additionally, in Chapter 4, we applied the technique for belief propagation that lowers the computational complexity for DCOPs with n-ary constraints. There are many more recent approaches for improving inference on graphical models. There have been many researches that try to perform more efficient search on AND/OR search graphs [63, 64] which can be applied to improve DJAO further. Additionally, there has been active development on algorithms for optimizing MAP estimation using convex optimization on linear programs and quadratic programs [44, 43]. Although there have been approaches that use linear programs and quadratic programs for DCOPs [38, 42], these are not corresponding translation of message passing algorithms constructed built using convex optimizations. There is a potential for a new class of DCOP algorithm based on the message-passing version of these algorithms in the same way that the Max-Sum and the Action-GDL have been developed .

The issue of scalability is also connected to parallelism as parallelism significantly enhances scalability of the algorithms. However, DCOP algorithms, especially inference-based algorithms such as Action-GDL, DPOP, Max-Sum as well as DJAO in Chapter 3 have limited parallelism as they have a structured message passing scheme. Maintaining communication efficiency while keeping parallelism is a delicate

issue. Parallel operation generally requires significantly more messaging than its non-parallel counterpart and balancing these two constraints is an interesting direction to pursue. Along the same line of research for increasing scalability and parallelism, there has been recent developments with linear programs on optimization problems [44] and there has been few works that applies the idea to DCOP. Applying those algorithms to DCOP seems very promising as linear programs can be transformed into distributed parallel algorithms.

Also, extending existing models and algorithms for new domains is another future research direction. In Chapter 5, we dealt with an environment where an agent manages more than one variable. We developed a scheme on how to modify a DCOP algorithm to exploit such one-to-many mapping to save resources. However, we have not studied how to come up with an efficient mapping which saves communication and time for this environment. Because the computation within an agent is serialized, evenly distributing the computational complexity across agents is crucial to reduce the end-to-end computation time. Also, when the variables which interact with each other are included in one agent, the amount of communication across agents can be reduced. Additionally, decomposition of the factor graph based on weak dependencies among nodes can be helpful as information needed to transfer can be minimal for links with weak dependencies.

Finally, an additional future research direction is to extend DCOP model for dynamic environments where new problems that are similar to past problems that have been already solved. In many practical application domains, the system constructs and solves a new DCOP whenever a new agent strategy is needed due to the changing environments. However, these new instances are quite similar to each other as well as their solutions if the environment gradually changes. Although there are algorithms that can react to the changes and partially reuse the solution [65, 66], the portion of the reused solution is minimal. A large part of the solution often needs to be

recomputed because a small change in the problem propagates to a large area due to the contingencies between nodes and such affected nodes need to compute their solution from scratch. However, solving the problem can be more efficiently done using information from previously seen problems. New variables appear and existing variables disappear and the constraint function values can change as well as the association between variables and constraint functions. Studying the impact of each change on various algorithms and studying the ways to reuse the previous decisions in order to expedite problem solving in these environments would be interesting. For example, in Chapter 4, we process the constraint function by selecting and sorting partial constraint function and such processed information can be reused effectively when only a small part of the function changes.

## BIBLIOGRAPHY

- [1] Randall Davis and Reid G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20(1):63 – 109, 1983.
- [2] Victor Lesser and Daniel. Corkill. Functionally Accurate, Cooperative Distributed Systems. *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-11(1):81–96, January 1981.
- [3] Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. Adopt: asynchronous distributed constraint optimization with quality guarantees. *Artif. Intell.*, 161(1-2):149–180, January 2005.
- [4] Kathryn Sarah Macarthur, Ruben Stranders, Sarvapali D. Ramchurn, and Nicholas R. Jennings. A distributed anytime algorithm for dynamic task allocation in multi-agent systems. In Wolfram Burgard and Dan Roth, editors, *AAAI*. AAAI Press, 2011.
- [5] Yoonheui Kim, Michael Krainin, and Victor Lesser. Effective Variants of the Max-Sum Algorithm for Radar Coordination and Scheduling. In *Proceedings of 2011 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 357–364, Lyon, France, October 2011.
- [6] Ruben Stranders, Alessandro Farinelli, Alex Rogers, and Nicholas R. Jennings. Decentralised coordination of mobile sensors using the max-sum algorithm. In *Proceedings of the 21st international joint conference on Artificial intelligence, IJCAI’09*, pages 299–304, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

- [7] Sarvapali D. Ramchurn, Alessandro Farinelli, Kathryn S. Macarthur, and Nicholas R. Jennings. Decentralized coordination in robocup rescue. *Comput. J.*, 53(9):1447–1461, 2010.
- [8] A. Rogers, A. Farinelli, R. Stranders, and N. R. Jennings. Bounded approximate decentralised coordination via the max-sum algorithm. *Artif. Intell.*, 175(2):730–759, February 2011.
- [9] Robert Junges and Ana L. C. Bazzan. Evaluating the performance of dcop algorithms in a real world, dynamic problem. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 2*, AAMAS '08, pages 599–606, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.
- [10] Akshat Kumar, William Yeoh, and Shlomo Zilberstein. On message-passing, map estimation in graphical models and dcops. In *Proceedings of the International Workshop on Distributed Constraint Reasoning (DCR)*, pages 57–70, 2011.
- [11] Yair Weiss and William T. Freeman. On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs. volume 47, pages 736–744, 2001.
- [12] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, pages 639–646, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.



- [13] Meritxell Vinyals, Juan A. Rodriguez-Aguilar, and Jesús Cerquides. Constructing a unifying theory of dynamic programming dcop algorithms via the generalized distributive law. *Autonomous Agents and Multi-Agent Systems*, 22(3):439–464, May 2011.
- [14] S.M. Aji and R.J. McEliece. The generalized distributive law. *Information Theory, IEEE Transactions on*, 46(2):325–343, Mar 2000.
- [15] Mark Paskin, Carlos Guestrin, and Jim McFadden. A robust architecture for distributed inference in sensor networks. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks, IPSN '05*, Piscataway, NJ, USA, 2005. IEEE Press.
- [16] F.R. Kschischang, B.J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *Information Theory, IEEE Transactions on*, 47(2):498–519, feb 2001.
- [17] David J. McLaughlin, V. Ch, Kelvin Droegemeier, Stephen Frasier, Jim Kurose, Francesc Junyent, Brenda Philips, Ra Cruz-pol, and Jose Colom. Distributed collaborative adaptive sensing (dcas) for improved detection. In *in Proc. American Meteorological Society Annual Meeting*, 2005.
- [18] M. Zink, D. Westbrook, S. Abdallah, B. Horling, E. Lyons, V. Lakamraju, V. Manfredi, J. Kurose, and K Hondl. Meteorological Command and Control: An End-to-end Architecture for a Hazardous Weather Detection Sensor Network. In *Proceedings of the ACM Workshop on End-to-End, Sense-and-Respond Systems, Applications, and Services (EESR 05)*, pages 37–42, Seattle, WA, 2005.

- [19] Rajiv T. Maheswaran, Milind Tambe, Emma Bowring, Jonathan P. Pearce, and Pradeep Varakantham. Taking dcop to the real world: Efficient complete solutions for distributed multi-event scheduling. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '04, pages 310–317, Washington, DC, USA, 2004. IEEE Computer Society.
- [20] Rina Dechter and Robert Mateescu. AND/OR search spaces for graphical models. *Artif. Intell.*, 171(2-3):73–106, February 2007.
- [21] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009.
- [22] William Yeoh, Ariel Felner, and Sven Koenig. Bnb-adopt: an asynchronous branch-and-bound dcop algorithm. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 2*, AAMAS '08, pages 591–598, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.
- [23] Yoonheui Kim and Victor Lesser. Djao: A communication-constrained dcop algorithm that combines features of adopt and action-gdl. 2014.
- [24] Julian John McAuley and Tibério S. Caetano. Exploiting data-independence for fast belief-propagation. In Johannes Fürnkranz and Thorsten Joachims, editors, *ICML*, pages 767–774. Omnipress, 2010.
- [25] Yoonheui Kim and Victor R. Lesser. Improved max-sum algorithm for dcop with n-ary constraints. In *AAMAS*, 2013.

- [26] Roger Mailler and Victor Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1, AAMAS '04*, pages 438–445, Washington, DC, USA, 2004. IEEE Computer Society.
- [27] Robert G. Cowell, A. Philip Dawid, Steffen L. Lauritzen, and David J. Spiegelhalter. *Probabilistic Networks and Expert Systems: Exact Computational Methods for Bayesian Networks*. Springer Publishing Company, Incorporated, 1st edition, 2007.
- [28] Adrian Petcu, Boi Faltings, and Roger Mailler. Pc-dpop: a new partial centralization algorithm for distributed optimization. In *Proceedings of the 20th international joint conference on Artificial intelligence, IJCAI'07*, pages 167–172, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.
- [29] Ismel Brito and Pedro Meseguer. Improving dpop with function filtering. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1, AAMAS '10*, pages 141–148, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.
- [30] Marc Pujol-Gonzalez, Jesus Cerquides, Pedro Meseguer, and J. A. Rodriguez-Aguilar. Communication-constrained dcops: message approximation in gdl with function filtering. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 1, AAMAS '11*, pages 379–386, Richland, SC, 2011. International Foundation for Autonomous Agents and Multiagent Systems.

- [31] Adrian Petcu and Boi Faltings. Mb-dpop: A new memory-bounded algorithm for distributed optimization. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI'07*, pages 1452–1457, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.
- [32] Weixiong Zhang, Guandong Wang, Zhao Xing, and Lars Wittenburg. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artif. Intell.*, 161(1-2):55–87, January 2005.
- [33] Christopher Kiekintveld, Zhengyu Yin, Atul Kumar, and Milind Tambe. Asynchronous algorithms for approximate distributed constraint optimization with quality bounds. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1, AAMAS '10*, pages 133–140, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.
- [34] Jonathan P. Pearce and Milind Tambe. Quality guarantees on k-optimal solutions for distributed constraint optimization problems. In *Proceedings of the 20th international joint conference on Artificial intelligence, IJCAI'07*, pages 1446–1451, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.
- [35] Meritxell Vinyals, Eric Shieh, Jesus Cerquides, Juan Antonio Rodriguez-Aguilar, Zhengyu Yin, Milind Tambe, and Emma Bowring. Quality guarantees for region optimal dcop algorithms. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 1, AAMAS '11*, pages 133–140, Richland, SC, 2011. International Foundation for Autonomous Agents and Multiagent Systems.

- [36] R. Mailler and V. Lesser. A cooperative mediation-based protocol for dynamic distributed resource allocation. *Trans. Sys. Man Cyber Part C*, 36(1):80–91, January 2006.
- [37] Kevin P. Murphy, Yair Weiss, and Michael I. Jordan. Loopy belief propagation for approximate inference: an empirical study. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, UAI'99, pages 467–475, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.
- [38] Meritxell Vinyals, Marc Pujol, J. A. Rodriguez-Aguilar, and Jesus Cerquides. Divide-and-coordinate: Dcops by agreement. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*, AAMAS '10, pages 149–156, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.
- [39] Roie Zivan and Hilla Peled. Max/min-sum distributed constraint optimization through value propagation on an alternating dag. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '12, pages 265–272, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems.
- [40] Daniel Tarlow, Inmar E. Givoni, and Richard S. Zemel. HOP-MAP: efficient message passing with high order potentials. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, pages 812–819, 2010.
- [41] Marc Pujol-Gonzalez, Jesús Cerquides, Gonzalo Escalada-Imaz, Pedro Meseguer, and Juan A. Rodríguez-Aguilar. On binary max-sum and tractable hops. volume 1113, Toulouse, France, 12/12/2013 2013.

- [42] Daisuke Hatano and Katsutoshi Hirayama. Deqed: An efficient divide-and-coordinate algorithm for dcop. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems, AAMAS '13*, pages 1325–1326, Richland, SC, 2013. International Foundation for Autonomous Agents and Multiagent Systems.
- [43] Amir Globerson and Tommi S. Jaakkola. Fixing max-product: Convergent message passing algorithms for map lp-relaxations. In J.C. Platt, D. Koller, Y. Singer, and S.T. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 553–560. Curran Associates, Inc., 2008.
- [44] Akshat Kumar and Shlomo Zilberstein. Message-passing algorithms for quadratic programming formulations of MAP estimation. In Fabio Gagliardi Cozman and Avi Pfeffer, editors, *UAI 2011, Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, Barcelona, Spain, July 14-17, 2011*, pages 428–435. AUAI Press, 2011.
- [45] Akshat Kumar, Adrian Petcu, and Boi Faltings. H-dpop: using hard constraints for search space pruning in dcop. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 1, AAAI'08*, pages 325–330. AAAI Press, 2008.
- [46] Radu Marinescu and Rina Dechter. AND/OR branch-and-bound for graphical models. In *Proceedings of the 19th international joint conference on Artificial intelligence, IJCAI'05*, pages 224–229, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.
- [47] Syed Ali, Sven Koenig, and Milind Tambe. Preprocessing techniques for accelerating the DCOP algorithm ADOPT. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS '05*, pages 1041–1048, New York, NY, USA, 2005. ACM.

- [48] Richard J. Wallace. Enhancements of branch and bound methods for the maximal constraint satisfaction problem. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 1*, AAAI'96, pages 188–195. AAAI Press, 1996.
- [49] Radu Marinescu and Rina Dechter. Best-first AND/OR search for graphical models. In *Proceedings of the 22Nd National Conference on Artificial Intelligence - Volume 2*, AAAI'07, pages 1171–1176. AAAI Press, 2007.
- [50] Patricia Gutierrez, Pedro Meseguer, and William Yeoh. Generalizing ADOPT and BnB-ADOPT. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume One*, IJCAI'11, pages 554–559. AAAI Press, 2011.
- [51] William Yeoh, Xiaoxun Sun, and Sven Koenig. Trading off solution quality for faster computation in dcop search algorithms. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, IJCAI'09, pages 354–360, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.
- [52] Zhengyu Yin. USC DCOP repository. 2008.
- [53] Federico Pecora, P. Jay Modi, and Paul Scerri. Reasoning about and dynamically posting n-ary constraints in adopt. 2006.
- [54] Manish Jain, Matthew Taylor, Milind Tambe, and Makoto Yokoo. Dcops meet the realworld: exploring unknown reward matrices with applications to mobile sensor networks. In *Proceedings of the 21st international joint conference on Artificial intelligence*, IJCAI'09, pages 181–186, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.
- [55] T. S. Caetano and J. J. McAuley. Faster algorithms for max-product message-passing. *Journal of Machine Learning Research*, 12(4):1349–1388, 2011.

- [56] C. Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 100(3/4):441–471, 1987.
- [57] James F. Kurose et al. An End-User-Responsive Sensor Network Architecture for Hazardous Weather Detection, Prediction and Response. In *Proceedings of the Second Asian Internet Engineering Conference, AINTEC*, pages 1–15, 2006.
- [58] Bryan Horling, Roger Mailler, and Victor Lesser. Farm: A Scalable Environment for Multi-Agent Development and Evaluation. In Alessandro Garcia Carlos Lucena, Jaelson Castro Alexander Romanovsky, and Paulo Alencar, editors, *Advances in Software Engineering for Multi-Agent Systems*, pages 220–237. Springer-Verlag, Berlin, February 2004.
- [59] M. Yokoo and K. Hirayama. Distributed constraint satisfaction algorithm for complex local problems. In *Proceedings of the 3rd International Conference on Multi Agent Systems, ICMAS '98*, pages 372–, Washington, DC, USA, 1998. IEEE Computer Society.
- [60] Aaron A. Armstrong and Edmund H. Durfee. Dynamic prioritization of complex agents in distributed constraint satisfaction problems. In *Proceedings of the fourteenth national conference on artificial intelligence and ninth conference on Innovative applications of artificial intelligence, AAAI'97/IAAI'97*, pages 822–822. AAAI Press, 1997.
- [61] Michael Krainin, Bo An, and Victor Lesser. An Application of Automated Negotiation to Distributed Task Allocation. In *2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2007)*, pages 138–145, Fremont, California, November 2007. IEEE Computer Society Press.



- [62] Adrian Petcu and Boi Faltings. A scalable method for multiagent constraint optimization. In *Proceedings of the 19th international joint conference on Artificial intelligence, IJCAI'05*, pages 266–271, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.
- [63] Rina Dechter Radu Marinescu and Alexander Ihler. And/or search for marginal map. In *Proceedings of Uncertainty in Artificial Intelligence (UAI 2014)*, 2014.
- [64] Radu Marinescu Akihiro Kishimoto. Recursive best-first and/or search for graphical models. In *Proceedings of Uncertainty in Artificial Intelligence (UAI 2014)*, 2014.
- [65] K. Macarthur, A. Farinelli, S. Ramchurn, and N. Jennings. Efficient, superstabilizing decentralised optimisation for dynamic task allocation environments. In *Proceedings of the 3rd International Workshop on Optimization in Multi-Agent Systems Agents*, 2010.
- [66] Adrian Petcu and Boi Faltings. Superstabilizing, fault-containing distributed combinatorial optimization. In *Proceedings of the 20th national conference on Artificial intelligence - Volume 1, AAAI'05*, pages 449–454. AAAI Press, 2005.