Privacy Sensitive Environment Decomposition for Hypertree Agent Organization

Construction

by

Abdulrahman Alshememry

A Thesis

Presented to

The University of Guelph

In partial fulfilment of requirements

for the degree of

Master of Science

in

Computer Science

Guelph, Ontario, Canada

© Abdulrahman Alshememry, August, 2018
Cooperative multiagent systems form an active area of research and practice in AI and software engineering. Decentralized probabilistic reasoning, constraint reasoning, and decision-theoretic reasoning are essential tasks of cooperative multiagent systems. Several frameworks for these tasks organize agents into a junction tree (JT). The JT agent organization has several computational advantages, including agent privacy during inference. During construction of the JT, however, existing frameworks utilize construction algorithms that leak the agent privacy. One exception is the HTBS algorithm, which constructs a JT if one exists without disclosing such private information. A limitation of the HTBS algorithm is that if no JT exists in the given agent environment decomposition, it only recognizes the non-existence. A novel algorithm suite DAER (Distributed Agent Environment Re-decomposition) is proposed to overcome this limitation by re-decomposing the environment to construct a JT. DAER has been evaluated in comparison with existing algorithms demonstrating significantly lower privacy loss.
Acknowledgments

I would like to thank my supervisor Dr. Yang Xiang for his great insights and instructions for completing this thesis. I would also like to thank my advisor committee member Dr. Chalrie Obimbo for his insights in my public seminar. My sincerest thanks to my employer King Saud University for giving me this opportunity to pursue my education by granting me a governmental scholarship provided from the government of Saudi Arabia and operated by the Saudi Arabian Cultural Bureau in Canada. I would like to thank my family for their constant support during my time in this program. Finally, my sincerest thanks to my wife Amerah Aldoghaithr, whom I got married to during the writing of this thesis, for her constant support and encouragement.
# Table of Contents

Abstract .............................................. ii

Acknowledgements ..................................... iii

List of Tables ........................................ vi

List of Figures ........................................ vii

1 Introduction ...................................... 1
  1.1 Multi-Agent Systems .............................. 1
     1.1.1 Agent Organization ...................... 2
     1.1.2 Example of MAS ......................... 3
  1.2 Agent Privacy ................................... 3
  1.3 Thesis Statement ................................ 4
  1.4 Overview of Thesis ............................. 5

2 Background ........................................ 6
  2.1 Agent Organization ............................. 6
  2.2 Junction Trees .................................. 7
  2.3 Quantification of Privacy Loss ............... 9
     2.3.1 Agents Privacy .......................... 10
     2.3.2 Privacy on Private Variables .......... 14
     2.3.3 Privacy on Shared Variables .......... 18
  2.4 Existing Algorithms ............................ 22
     2.4.1 Action-GDL Algorithm .................. 22
     2.4.2 DCTE Algorithm ......................... 24
     2.4.3 COORD-Plus Algorithm .................. 27
     2.4.4 DPMST Algorithm ....................... 28
  2.5 HTBS Algorithm ................................ 31
     2.5.1 Boundary Graph Based Condition ....... 32
     2.5.2 Boundary Set Based Condition .......... 34
2.5.3 Recognizing JT Existence Distributively ..................................... 35
2.5.4 HTBS Algorithm Suite ....................................................... 37
2.5.5 Examples of HTBS ........................................................... 40
2.5.6 Construction of JT Organization ............................................ 43

3 Methodology .............................................................................. 45
  3.1 DAER Algorithm .................................................................. 45
    3.1.1 The Idea Behind DAER ................................................... 46
    3.1.2 Algorithm Overview ....................................................... 48
    3.1.3 Leader Agent ................................................................. 52
    3.1.4 Elimination-Expansion Stage ......................................... 54
  3.2 New Privacy Loss Quantification Scheme ............................... 62
    3.2.1 Assigning Privacy Weight ............................................... 64
    3.2.2 Privacy Loss on Agent Identity ....................................... 69
    3.2.3 Privacy Loss on Bordering Relations ............................... 71
    3.2.4 Privacy Loss on Variables Identities and Domains ............ 71
  3.3 Demonstration of Soundness ................................................... 73
  3.4 Complexity ........................................................................... 80

4 Experimental Evaluation .......................................................... 82
  4.1 Experimental Setup ............................................................. 82
  4.2 Experimental Results .......................................................... 84

5 Conclusion .................................................................................. 99
  5.1 Summary of Contributions ................................................... 99
  5.2 Future Work ......................................................................... 100

Bibliography ................................................................................... 102

A Appendix .................................................................................... 105
List of Tables

3.1 Weights of agent identities of agents in Fig. 3.4 . . . . . . . . . . . . 65
3.2 Weights of the borders between agents in Fig. 3.4 . . . . . . . . . . . 66
3.3 Weights of private variables identities and domains in Fig. 3.4 . . . . 67
3.4 Weights of shared variables identities and domains in Fig. 3.4 . . . . 69
3.5 Weights of shared variables identities and domains in Fig. 3.5 . . . . 74
3.6 Weights of agent identities of agents in Fig. 3.5 . . . . . . . . . . . . 74
3.7 Weights of the borders between agents in Fig. 3.5 . . . . . . . . . . . 74

4.1 Boundary graphs type track through the execution of DAER, where 83
each row is a boundary set and each column is a round of execution .
4.2 Indexes of the experimental environment decompositions . . . . . . . 84
List of Figures

1.1 A MAS organized as a junction tree ............................................. 2
2.1 A junction graph with clusters as ovals and separators as squares . . 7
2.2 An example of a junction tree ..................................................... 8
2.3 A MAS with five agents ............................................................ 10
2.4 a) A dependency graph. b) DFS tree. c) A pseudotree ................. 23
2.5 Action-GDL constructing a JT as shown in [22] ............................ 23
2.6 a) Environment decomposition, b) possible spanning tree, c) resultant JT ............................................................................. 25
2.7 Execution steps when running DMPST ........................................ 29
2.8 Examples of boundary graphs ..................................................... 32
2.9 Recognizing the existence of JT distributively .............................. 37
2.10 (a) Communication graph with its boundary graph of type 1 in (b) . 40
2.11 (a) Communication graph with its boundary graph of type 2 in (b) . 41
2.12 (a) Communication graph with its boundary graph of type 3 in (b) . 42
2.13 HTBS rounds on Fig. 2.12(a) ..................................................... 43
3.1 The communication graph on Fig.2.12 (a) ................................... 47
3.2 Communication graph in (a) with its boundary set in (b) ............... 50
3.3 DAER iterations from (a) to (c), and the revised environment decom-
position (d) ..................................................................................... 52
3.4 A MAS with five agents ............................................................. 65
3.5 Environment decomposition with type 2 boundary graph ............. 73
3.6 The active boundaries of Fig. 3.5 after the first expansion .......... 78
3.7 The active boundaries of Fig. 3.5 after the second expansion in (a),
and after the third expansion in (b) .............................................. 79
3.8 The revised environment decomposition of Fig. 3.5 in (a), and the
resultant JT is show in (b) ............................................................ 80
4.1 PL of constructing a JT, MBS is between 1-5, and the ratio is 1:1 (µ,
σ): DAER(2.64, 0.46), Action-GDL(3.17, 0.52) DCTE(4.19, 0.60) . . 86
4.2 PL of constructing a JT, MBS is between 1-5, and the ratio is 2:1 ($\mu$, $\sigma$): DAER(2.65, 0.46), Action-GDL(3.17, 0.51) DCTE(4.44, 0.61) ... 87
4.3 PL of constructing a JT, MBS is between 1-5, and the ratio is 3:1 ($\mu$, $\sigma$): DAER(2.66, 0.46), Action-GDL(3.17, 0.51) DCTE(4.60, 0.62) ... 87
4.4 PL of constructing a JT, MBS is between 6-10, and the ratio is 1:1 ($\mu$, $\sigma$): DAER(2.35, 0.57), Action-GDL(3.29, 0.4) DCTE(4.41, 0.45) ... 87
4.5 PL of constructing a JT, MBS is between 6-10, and the ratio is 2:1 ($\mu$, $\sigma$): DAER(2.34, 0.56), Action-GDL(3.29, 0.39) DCTE(4.48, 0.45) ... 88
4.6 PL of constructing a JT, MBS is between 6-10, and the ratio is 3:1 ($\mu$, $\sigma$): DAER(2.34, 0.56), Action-GDL(3.29, 0.39) DCTE(4.84, 0.45) ... 88
4.7 PL of constructing a JT, MBS is between 11-15, and the ratio is 1:1 ($\mu$, $\sigma$): DAER(1.88, 0.59), Action-GDL(3.25, 0.77) DCTE(4.55, 0.87) ... 88
4.8 PL of constructing a JT, MBS is between 11-15, and the ratio is 2:1 ($\mu$, $\sigma$): DAER(1.88, 0.58), Action-GDL(3.25, 0.77) DCTE(4.82, 0.88) ... 89
4.9 PL of constructing a JT, MBS is between 11-15, and the ratio is 3:1 ($\mu$, $\sigma$): DAER(1.86, 0.6), Action-GDL(3.24, 0.77) DCTE(4.99, 0.88) ... 89
4.10 RN of constructing a JT, MBS is between 1-5, and the ratio is 1:1 ($\mu$, $\sigma$): DAER(3.36, 0.82), Action-GDL(1.94, 0.19) DCTE(3.17, 0.70) ... 91
4.11 RN of constructing a JT, MBS is between 1-5, and the ratio is 2:1 ($\mu$, $\sigma$): DAER(3.32, 0.86), Action-GDL(2.24, 0.33) DCTE(3.17, 0.72) ... 91
4.12 RN of constructing a JT, MBS is between 1-5, and the ratio is 3:1 ($\mu$, $\sigma$): DAER(3.32, 0.86), Action-GDL(2.51, 0.43) DCTE(3.18, 0.71) ... 92
4.13 RN of constructing a JT, MBS is between 6-10, and the ratio is 1:1 ($\mu$, $\sigma$): DAER(3.57, 0.75), Action-GDL(2.21 0.27) DCTE(3.44, 0.67) ... 92
4.14 RN of constructing a JT, MBS is between 6-10, and the ratio is 2:1 ($\mu$, $\sigma$): DAER(3.53, 0.78), Action-GDL(2.64, 0.35) DCTE(3.53, 0.70) ... 92
4.15 RN of constructing a JT, MBS is between 6-10, and the ratio is 3:1 ($\mu$, $\sigma$): DAER(3.53, 0.79), Action-GDL(2.99, 0.41) DCTE(3.51, 0.57) ... 93
4.16 RN of constructing a JT, MBS is between 11-15, and the ratio is 1:1 ($\mu$, $\sigma$): DAER(3.78, 1.23), Action-GDL(2.72, 0.50) DCTE(3.66, 1.15) ... 93
4.17 RN of constructing a JT, MBS is between 11-15, and the ratio is 2:1 ($\mu$, $\sigma$): DAER(3.77, 1.24), Action-GDL(3.30, 0.68) DCTE(3.75, 1.19) ... 93
4.18 RN of constructing a JT, MBS is between 11-15, and the ratio is 3:1 ($\mu$, $\sigma$): DAER(3.76, 1.25), Action-GDL(3.70, 0.76) DCTE(3.76, 1.14) ... 94
4.19 MP in constructing a JT, MBS is between 1-5, and the ratio is 1:1 ($\mu$, $\sigma$): DAER(5.28, 0.82), Action-GDL(3.65, 0.36) DCTE(3.82, 0.55) ... 95
4.20 MP in constructing a JT, MBS is between 1-5, and the ratio is 2:1 \((\mu, \sigma)\): DAER(5.28, 0.82), Action-GDL(4.04, 0.37) DCTE(3.82, 0.55) . . 96

4.21 MP in constructing a JT, MBS is between 1-5, and the ratio is 3:1 \((\mu, \sigma)\): DAER(5.28, 0.82), Action-GDL(4.31, 0.37) DCTE(3.82, 0.55) . . 96

4.22 MP in constructing a JT, MBS is between 6-10, and the ratio is 1:1 
\((\mu, \sigma)\): DAER(5.51, 0.63), Action-GDL(4.06, 0.25) DCTE(4.02, 0.43) 96

4.23 MP in constructing a JT, MBS is between 6-10, and the ratio is 2:1 
\((\mu, \sigma)\): DAER(5.50, 0.63), Action-GDL(4.46, 0.25) DCTE(4.02, 0.43) 97

4.24 MP in constructing a JT, MBS is between 6-10, and the ratio is 3:1 
\((\mu, \sigma)\): DAER(5.51, 0.63), Action-GDL(4.73, 0.25) DCTE(4.02, 0.43) 97

4.25 MP of constructing a JT, MBS is between 11-15, and the ratio is 1:1 
\((\mu, \sigma)\): DAER(5.46, 1.24), Action-GDL(4.54, 0.44) DCTE(3.99, 0.88) 97

4.26 MP of constructing a JT, MBS is between 11-15, and the ratio is 2:1 
\((\mu, \sigma)\): DAER(5.46, 1.24), Action-GDL(4.95, 0.44) DCTE(3.99, 0.88) 98

4.27 MP of constructing a JT, MBS is between 11-15, and the ratio is 3:1 
\((\mu, \sigma)\): DAER(5.46, 1.24), Action-GDL(5.22, 0.44) DCTE(3.99, 0.88) 98
Chapter 1

Introduction

Research in Computer Science has progressed very rapidly in the last several decades. This rapid advancement, has changed our daily life. Artificial Intelligence (AI) is one of the major fields in Computer Science. Intelligent systems are built in AI, also known as intelligent agents [14], to perform various tasks. When more than one agent is involved in such an intelligent system, the agents form a multi-agent system (MAS) that can solve more complex problems.

1.1 Multi-Agent Systems

A single agent system might not be sufficient in some cases. For example, troubleshooting a chemical plant might not be possible for a single agent. Another limitation of single agent systems is that when the agent fails, the whole system fails. On the other hand, multi-agent systems have the flexibility to simplify complex tasks by dividing it to the agents into easier smaller tasks that contribute to solving the problem [11]. Moreover, if a single agent fails in a MAS, the rest of the agents can still perform their tasks if their tasks are not dependent on the failing agent. The agents in a MAS can play cooperative or competitive roles depending on the nature of the environment and the tasks performed [14]. Cooperative multi-agent systems perform many tasks such as probabilistic reasoning, constraint reasoning, and decision
making.

1.1.1 Agent Organization

In MAS, each framework adapts a certain agent organization. MAS organizations control the communications between agents. The agent organization decides what agents each agent should communicate with, and what type of information should be exchanged. There are many organizations that frameworks can adapt to MAS. For example, some frameworks allow each agent to exchange all information with every other agent. Other frameworks organize the agents into a tree where each agent can communicate with its parent and children in a child-parent kind of communication. A special case of tree agent organization is to use junction trees (JT$s$). A junction tree is a cluster tree where each node in the tree is a set of variables instead of a single variable. An example of a JT can be seen on Fig. 1.1 (see Section 2.2). One advantage of the JT organization is that it protects the agents’ privacy once the JT is constructed.

Figure 1.1: A MAS organized as a junction tree
1.1.2 Example of MAS

Trouble-shooting a complex piece of equipment such as a chemical plant is an example of a MAS performing a probabilistic reasoning task [22]. Each component of the chemical plant might come from a different independent vendor. Each vendor supplies monitoring and troubleshooting agent. All the troubleshooting agents cooperate as a MAS to troubleshoot the chemical plant. Troubleshooting all the components of the chemical plant might be very complex for a single agent especially if the components come from different vendors. Another issue of such a MAS being operable by a single agent is the privacy. The privacy of all the monitoring agents and the components will be exposed to the single controlling agent. With this leak of privacy, vendors will hesitate to participate in such a MAS to protect their intellectual property [22]. A simpler example of MAS can be seen in Fig. 1.1. This system consist of four agents $A_1, A_2, A_3,$ and $A_4$. This MAS consist of nine variables $a, b, c, d, e, f, g, h,$ and $i$. Each agent controls a subset of the system’s variables. For example, the agent $A_1$ controls the variables $a, b, c,$ and $d$. The agents $A_1$ and $A_2$ are neighboring agents sharing the variable $d$.

1.2 Agent Privacy

In real-world applications, each agent might be designed by a different manufacturer. In those cases, agent’s privacy is very important for each manufacturer. There are three main categories of agent’s privacy which are privacy of private variables, privacy of shared variables, and privacy of agent identity and bordering relations.

**Private variables** are the variables that are contained in a single agent only. The identities of those variables, as well as their domain, should only be accessible to
their agent. For example, the variables $b$ and $c$ in Fig. 1.1 should only be known by the agent $A_1$.

**Shared variables** are the variables shared between two or more agents. The identities of those variables, as well as their domains, should only be accessible to the agents that contain them. For example, the variable $d$ in Fig. 1.1 should only be known by the agents $A_1$ and $A_2$.

**Agent identity and bordering relations** should only be known by the agent and its neighboring agents. For example, the identity of $A_2$ in Fig. 1.1 should only be known by $A_1$, and $A_3$, $A_4$ should not know about the identity of $A_2$. Moreover, the relation between $A_1$ and $A_2$ in Fig. 1.1 should only be known to $A_1$ and $A_2$, and should not be leaked to $A_3$ and $A_4$.

### 1.3 Thesis Statement

When a MAS is organized as a JT, there is no leak on agents’ privacy. However, while constructing the JT, most existing works take approaches that leak agents’ privacy. To calculate the privacy loss, a quantification scheme is presented in [22], but this quantification scheme considers all types of privacy loss as equal. In real-world applications, some private information holds significant importance, and not all types of private information have the same value. Therefore, a new quantification scheme is built on the one presented in [22] is proposed in the thesis to enable differentiating the importance of private information. Moreover, HTBS algorithm suite is the only known algorithm to construct the JT agent organization without any privacy leaks. However, when no JT exists in a given environment decomposition, HTBS can only recognize the non-existence of a JT, but it cannot construct one. A novel algorithm
suite DAER (Distributed Agent Environment Re-decomposition) is presented in the thesis that re-decompose the environment with minimal privacy loss to enable the construction of a JT.

1.4 Overview of Thesis

The rest of the thesis is organized as follows. First the background showing existing algorithms and how they build the JT agent organization, and how most of them leak privacy on the construction stage is presented in Chapter 2. The DAER algorithm suite with its algorithmic details and examples are shown in Chapter 3 along with a new privacy loss quantification scheme. Chapter 4 shows the experimental results of the DAER algorithm. Chapter 5 presents the conclusion and future work. A table of the abbreviations and HTBS algorithm suite’s procedures can be found in the appendix.
Chapter 2

Background

This chapter introduces detailed background of the thesis. Section 2.1 shows how existing frameworks deal with agent organization. The junction tree agent organization is discussed on in Section 2.2. Section 2.3 explains an existing privacy loss quantification scheme. The exiting algorithms that construct the JT agent organization is demonstrated in Section 2.4. Finally, how the HTBS algorithm suite works is explained in Section 2.5.

2.1 Agent Organization

Cooperative multi-agent systems perform various tasks, and some of those tasks are probabilistic reasoning, constraint reasoning, and decision making. Some frameworks of MAS for probabilistic reasoning can be found in [15, 18, 8]. MAS frameworks for constraint reasoning can be found in [1, 4, 16, 21]. Decision making has some frameworks as MAS that can be found in [20, 6, 13]. When organizing the agents in MAS, there are many approaches to organize such systems. The organization structure is not specified in some frameworks such as [3, 15]. The agents can be organized using a weighted directed graphs such as the work in [2]. Other frameworks organize the agents as a pseudotree like [10, 4], or a directed spanning tree such as [7]. Some frameworks adapt a junction tree (JT) organization (will be defined in Section
2.2 Junction Trees

A regular graph can be represented as $G = (V, E)$, where $V$ is the set of variables in the graph and $E$ is the set of edges. Cluster graphs are the graph where instead of having nodes as single variables linked with edges, each node is a clusters that is a set of variables instead of a single variable. Cluster graph can represent the agents where the variables controlled by each agent are grouped into a single cluster. A subset of cluster graphs is the junction graphs. Junction graphs are the cluster graphs, where whenever there exist two clusters sharing at least one variable, they
must be connected by a separator. A separator between any two clusters is the set of variables shared between those two clusters. Junction graphs are represented as a triplet $G = (V, \Omega, E)$, where $V$ is a nonempty set of variables, $\Omega$ is a set of clusters $(Q_1, Q_2, ..., Q_i)$ where each cluster is a subset of $V$ such that $\bigcup_{Q \in \Omega} Q = V$, and $E$ is a set of separators where $E = \{(Q_i, Q_j) | Q_i, Q_j \in \Omega, Q_i \neq Q_j, Q_i \cap Q_j \neq \phi\}$ [18]. An example junction graph can be seen in Fig. 2.1.

Junction trees are a subset of junction graphs. To create a junction tree instead of a junction graph two conditions must be met. First, it has to have the property of a tree. The main property of trees is to have exactly one path between any two nodes (clusters in cluster graphs). Second, if any nonadjacent clusters $Q_i$ and $Q_j$ share a variable, where $Q_i \cap Q_j \neq \phi$, their intersection must be contained in every clusters in path between $Q_i$ and $Q_j$ [22]. An example of a JT is shown in Fig. 2.2. The variables of the JT in Fig. 2.2 are $V = \{a, b, c, d, e, f, g, h, i\}$, the clusters are $\Omega = \{Q_1 = \{a, b, c, d\}, Q_2 = \{d, f\}, Q_3 = \{a, g, h, i\}, Q_4 = \{a, e, i\}\}$, and the separators are $E = \{(Q_1, Q_2), (Q_1, Q_3), (Q_3, Q_4)\}$. In this example, there is
exactly one path between any two clusters, and the intersection of any nonadjacent clusters is contained in the path between them. For example, there is exactly one path between $Q_1$ and $Q_4$, which is $(Q_1, Q_3, Q_4)$, and their intersection is the variable $a$ that is contained in every cluster in the path between them.

The main purpose of using the JT agent organization is that the computation complexity of the JT message passing is linear on the number of agents [18]. The JT agent organization also allows exact computations in probabilistic reasoning as shown in [18, 5].

2.3 Quantification of Privacy Loss

To quantify the loss of privacy, a quantification scheme is proposed in [22]. This scheme allows precise evaluation and comparison of privacy loss of the three types of privacy among different frameworks [22]. It presents six main equations to calculate the privacy loss on agent identities, bordering relations, private variables identities, private variables domains, shared variables identities, and shared variables domains. For each category, the system privacy loss is calculated, where it returns a value of the number of pieces of information leaked. To make this value clearer for comparison, the maximum system privacy loss is calculated, and then the normalized system privacy loss is calculated by dividing the system privacy loss over the maximum privacy loss. These six equations can be grouped under three main categories: 1) agents privacy, 2) private variables privacy, and 3) shared variables privacy.
2.3.1 Agents Privacy

**Agent identities** To quantify the privacy loss of agent identities, the functions $\text{border}(A_i, A_j)$ and $\text{know}(A_i, A_j)$ are used. The function $\text{border}(A_i, A_j)$ returns 1 if $A_i$ and $A_j$ are neighbors and 0 otherwise. For example, since $A_1$ and $A_2$ are bordering agents in Fig. 2.3, the function $\text{border}(A_1, A_2)$ will return 1, but $\text{border}(A_1, A_4)$ will return 0. The function $\text{know}(A_i, A_j)$ returns 1 if the agent $A_i$ knows the identity of the agent $A_j$ and 0 otherwise. For instance, $\text{know}(A_1, A_2)$ should return 1 when applied to Fig. 2.3 since $A_1$ and $A_2$ are neighbors. On the other hand, $\text{know}(A_1, A_4)$ should return 0 since they are not bordering agents. Therefore, whenever a non bordering agent $A_j$ knows the identity of $A_i$, it will be counted as one unit of privacy loss.

![Figure 2.3: A MAS with five agents](image-url)

The equation $\text{know}(A_i, A_j) - \text{border}(A_i, A_j)$ will return 1 only if the identity of $A_j$ is leaked to the agent $A_i$ while they are not neighboring agents. For example, $\text{know}(A_1, A_2) - \text{border}(A_1, A_2)$ in Fig. 2.3 will return $1 - 1 = 0$, where $\text{know}(A_1, A_4) - \text{border}(A_1, A_4)$ should return $0 - 0 = 0$. However, if the identity of the agent $A_1$ is
leaked to $A_4$, the equation $\text{know}(A_1, A_4) - \text{border}(A_1, A_4)$ will return $1 - 0 = 1$, which will count a privacy loss. To calculate the privacy loss on agent identities, this equation is applied to all the agents, and the summation is calculated. This equation calculates the system privacy loss of agent identities ($SPL_{aid}$):

$$SPL_{aid} = \sum_i \sum_{j \neq i} (\text{know}(A_i, A_j) - \text{border}(A_i, A_j))$$

The maximum system privacy loss ($MSPL_{aid}$) is then calculated, where it is assumed that all the identities have been leaked. As a result of this assumption, the function $\text{know}(A_i, A_j)$ will always return 1. The following equations calculate the $MSPL_{aid}$:

$$MSPL_{aid} = \sum_i \sum_{j \neq i} (1 - \text{border}(A_i, A_j))$$

The maximum privacy loss on agent identity in Fig. 2.3 will be broken down as follows:

$$((1 - \text{border}(A_1, A_2)) + (1 - \text{border}(A_1, A_3))$$

$$+ (1 - \text{border}(A_1, A_4)) + (1 - \text{border}(A_1, A_5)))$$

$$+ ((1 - \text{border}(A_2, A_1)) + (1 - \text{border}(A_2, A_3))$$

$$+ (1 - \text{border}(A_2, A_4)) + (1 - \text{border}(A_2, A_5)))$$

$$+ ((1 - \text{border}(A_3, A_1)) + (1 - \text{border}(A_3, A_2))$$

$$+ (1 - \text{border}(A_3, A_4)) + (1 - \text{border}(A_3, A_5)))$$

$$+ ((1 - \text{border}(A_4, A_1)) + (1 - \text{border}(A_4, A_2))$$

$$+ (1 - \text{border}(A_4, A_3)) + (1 - \text{border}(A_4, A_5)))$$

$$+ ((1 - \text{border}(A_5, A_1)) + (1 - \text{border}(A_5, A_2))$$

$$+ (1 - \text{border}(A_5, A_3)) + (1 - \text{border}(A_5, A_4)))$$

$$= ((1 - 1) + (1 - 1) + (1 - 0) + (1 - 0)) + ((1 - 1) + (1 - 0) + (1 - 0) + (1 - 1))$$

11
The normalized system privacy loss ($NSPL_{aid}$) is as follows:

$$NSPL_{aid} = SPL_{aid}/MSPL_{aid} = \frac{\sum_i \sum_{j \neq i} (\text{know}(A_i, A_j) - \text{border}(A_i, A_j))}{\sum_i \sum_{j \neq i} (1 - \text{border}(A_i, A_j))} \in [0, 1].$$

$NSPL_{aid}$ returns the normalized system privacy loss as a value between 0 and 1, where 0 means no privacy loss, and 1 means the maximum privacy loss.

**Bordering relations** The privacy of the bordering relations is another type of privacy loss that can occur during the construction of a JT. To calculate the privacy loss on bordering relations, the function $\text{knowBdr}(A_i, A_j, A_k)$ is used where it returns 1 if $A_j$ and $A_k$ are neighboring agents and the agent $A_i$ knows about their bordering relation and 0 otherwise. For instance, $A_1$ and $A_2$ are neighboring agents in Fig. 2.3, and agent $A_3$ should not know about this bordering relations, so the function $\text{knowBdr}(A_3, A_1, A_2)$ should return 0. However, if this border is leaked to $A_3$, the function $\text{knowBdr}(A_3, A_1, A_2)$ will return 1. To calculate if the border is leaked to another agent the following equation can be applied $\text{border}(A_j, A_k) \ast \text{knowBdr}(A_i, A_j, A_k)$. To calculate how many borders has been leaked to a certain agent $A_i$, the following summation can be applied:

$$\sum_{j \neq i} \sum_{k \neq j} \text{border}(A_j, A_k) \ast \text{knowBdr}(A_i, A_j, A_k)$$

To calculate the system privacy loss on bordering relations ($SPL_{bdr}$), the summation is applied to all agents as follows:

$$SPL_{bdr} = \sum_i \sum_{j \neq i} \sum_{k \neq i, k \neq j} \text{border}(A_j, A_k) \ast \text{knowBdr}(A_i, A_j, A_k)$$
The maximum system privacy loss \( MSPL_{bdr} \) assumes that all borders are leaked, so the function \( knowBdr(A_i, A_j, A_k) \) always returns 1. Therefore, the \( MSPL_{bdr} \) can be calculated as follows:

\[
MSPL_{bdr} = \sum_i \sum_{j \neq i} \sum_{k \neq i, k \neq j} border(A_j, A_k).
\]

However, the proposed equations to calculate the privacy leak in border relations in [22] are not accurate because of the double counting where \( A_j \) and \( A_k \) could exchange roles for the same border. For example, in Fig. 2.3, let \( A_4 \) be \( A_i \) in the equation, \( A_1 \) be \( A_j \), and \( A_2 \) be \( A_k \), the function \( border(A_1, A_2) \ast knowBdr(A_4, A_1, A_2) \) will return 1 if the border between \( A_1 \) and \( A_2 \) is leaked to \( A_4 \), and let \( A_4 \) be \( A_i \), \( A_2 \) be \( A_j \), and \( A_1 \) be \( A_k \), the function \( border(A_2, A_1) \ast knowBdr(A_4, A_2, A_1) \) will also return 1 if the border between \( A_2 \) and \( A_1 \) is leaked to \( A_4 \), which cause double counting on the same border. To solve this issue, the equations could be revised, so that \( k \) be \( k > j \) instead of \( k \neq j \) as follows:

\[
SPL_{bdr} = \sum_i \sum_{j \neq i} \sum_{k \neq i, k > j} border(A_j, A_k) \ast knowBdr(A_i, A_j, A_k).
\]

\[
MSPL_{bdr} = \sum_i \sum_{j \neq i} \sum_{k \neq i, k > j} border(A_j, A_k).
\]

The \( MSPL_{bdr} \) in Fig. 2.3 will be as follows:

\[
(border(A_2, A_3) + border(A_2, A_4) + border(A_2, A_5)
+ border(A_3, A_4) + border(A_3, A_5) + border(A_4, A_5))
+ (border(A_1, A_3) + border(A_1, A_4) + border(A_1, A_5)
+ border(A_3, A_4) + border(A_3, A_5) + border(A_4, A_5))
+ (border(A_1, A_2) + border(A_1, A_4) + border(A_1, A_5)
+ border(A_2, A_4) + border(A_2, A_5) + border(A_4, A_5)).
\]
+ (border(A_1, A_2) + border(A_1, A_3) + border(A_1, A_5))
+ border(A_2, A_3) + border(A_2, A_5) + border(A_3, A_5))
+ (border(A_1, A_2) + border(A_1, A_3) + border(A_1, A_4))
+ border(A_2, A_3) + border(A_2, A_4) + border(A_3, A_4))
= (0 + 0 + 1 + 1 + 0 + 0) + (1 + 0 + 0 + 1 + 0 + 0) + (1 + 0 + 0 + 0 + 1 + 0)
+ (1 + 1 + 0 + 0 + 1 + 0) + (1 + 1 + 0 + 0 + 0 + 1)
= 2 + 2 + 2 + 3 + 3 = 12

The normalized system privacy loss (NSPL_{bdr}) is as follows:

\[
NSPL_{bdr} = \frac{\sum_i \sum_{j \neq i} \sum_{k \neq i, k > j} \text{border}(A_j, A_k) \cdot \text{knowBdr}(A_i, A_j, A_k)}{\sum_i \sum_{j \neq i} \sum_{k \neq i, k > j} \text{border}(A_j, A_k)} \in [0, 1].
\]

### 2.3.2 Privacy on Private Variables

**Private variables identities** When calculating the privacy loss on the identity of private variables the function \( \text{private}(x, A_i) \) is used where it returns 1 when the variable \( x \) is private in the variable set \( V_i \) of the agent \( A_i \) (\( x \in V_i \)), and returns 0 otherwise. For example, in Fig. 2.3, \( \text{private}(b, A_1) \) will return 1 since the variable \( b \) is a private variable for the agent \( A_1 \), but \( \text{private}(a, A_1) \) will return 0 since the variable \( a \) is shared with agents \( A_3 \) and \( A_4 \). The function \( \text{knowID}(A_i, x) \) is also used when calculating the privacy loss on the identity of the private variables where it returns 1 if the agent \( A_i \) knows the identity of the variable \( x \) and 0 otherwise. For instance, \( \text{knowID}(A_1, b) \) will return 1 since the agent \( A_1 \) knows the identity of the variable \( b \), but \( \text{knowID}(A_3, b) \) should return 0 since the variable \( b \) is private to agent \( A_1 \). However, if \( \text{knowID}(A_3, b) \) returns 1, the identity of the variable \( b \) is leaked to \( A_3 \), which will be counted as a privacy loss.
To calculate the privacy loss on the identity of a private variable, the equation $\text{private}(x, A_j) \ast \text{knowID}(A_i, x)$ is used. This equation returns 1 if the focused variable is private to a certain agent, but there exist another agent that knows the identity of this private variable. For example, $\text{private}(b, A_1) \ast \text{knowID}(A_3, b)$ should return $1 \ast 0 = 0$, which means the identity of the variable $b$ is not leaked to the agent $A_3$. However, if the identity of the private variable $b$ is leaked to $A_3$, the equation $\text{private}(b, A_1) \ast \text{knowID}(A_3, b)$ will return $1 \ast 1 = 1$, which means there is a privacy loss on private variables identities. To calculate how much privacy on the identity of the private variables of $A_j$ is leaked to the agent $A_i$ we calculate following equation:

$$\sum_{x \in A_j} \text{private}(x, A_j) \ast \text{knowID}(A_i, x)$$

This equation will return a value that indicates the number of private variables identities in $A_j$ that has been leaked to agent $A_i$. For example, consider $A_2$ and $A_5$ in Fig. 2.3. The agent $A_2$ has one private variable, and one shared variable. To calculate how many private variable identities in $A_2$ has been leaked to $A_5$, the following equation is used:

$$\sum_{x \in A_j} \text{private}(x, A_2) \ast \text{knowID}(A_5, x)$$

Assuming there is no privacy loss, the equation will be broken down as follows:

$$(\text{private}(d, A_2) \ast \text{knowID}(d_5, x)) + (\text{private}(f, A_2) \ast \text{knowID}(A_5, f))$$

$$= 0 \ast 1 + 1 \ast 0 = 0$$

However, if the identity of the variable $f$ is leaked to $A_5$, the equation will be as follows:
\[
(\text{private}(d, A_2) \ast \text{knowID}(d_5, x)) + (\text{private}(f, A_2) \ast \text{knowID}(A_5, f)) = 0 \ast 1 + 1 \ast 1 = 1
\]

To calculate how much privacy on private identities is leaked to a certain agent from the whole system, the previous equation will be applied on all other agents, and the summation will be calculated as follows:

\[
\sum_{j \neq i} \sum_{x \in V_j} \text{private}(x, A_j) \ast \text{knowID}(A_i, x)
\]

This equation will return the number of private variables identities that has been leaked to the agent \(A_i\) from all other agents in the system. To calculate the system private loss on the identity of private variables (\(SPL_{pvid}\)), the equation is applied on all agents in the system, and the summation is calculated as follows:

\[
SPL_{pvid} = \sum_{i} \sum_{j \neq i} \sum_{x \in V_j} \text{private}(x, A_j) \ast \text{knowID}(A_i, x)
\]

The maximum system privacy loss on the identity of the private variables (\(MSPL_{pvid}\)) will assume that all the identities of private variables have been leaked. Therefore, the function \(\text{knowID}(A_i, x)\) will always return 1. The equation used to calculate the \(MSPL_{pvid}\) is

\[
MSPL_{pvid} = \sum_{i} \sum_{j \neq i} \sum_{x \in V_j} \text{private}(x, A_j).
\]

For simplicity, \(\text{private}(x, A_j)\) will be written as \(Pr(x, A_j)\) in the next example. The maximum privacy loss on identity of the private variables in Fig. 2.3 will be:

\[
\begin{align*}
((Pr(d, A_2) + Pr(f, A_2)) + (Pr(a, A_3) + Pr(g, A_3) + Pr(h, A_3) + Pr(i, A_3)) \\
+ (Pr(a, A_4) + Pr(e, A_4) + Pr(i, A_4)) + (Pr(d, A_5) + Pr(j, A_5) + Pr(k, A_5)) \\
+ ((Pr(a, A_1) + Pr(b, A_1) + Pr(c, A_1) + Pr(d, A_1)))
\end{align*}
\]
The normalized system privacy loss on the identity of private variables ($NSPL_{pvid}$) is as follows:

$$NSPL_{pvid} = \sum_i \sum_{j \neq i} \sum_{x \in V_j} \text{private}(x, A_j) \times \text{knowID}(A_i, x) \in [0, 1].$$

**Private variables domain** The domain of private variables is another type of private information under the private variables. The privacy leak on the domain
of private variables can be calculated using the function $knowDom(A_i, x)$ where it returns 1 if the agent $A_i$ knows the domain of the variable $x$. This function is similar to the previous function $knowID(A_i, x)$, but instead of focusing on the identity of the private variables, it focuses on the domain. The following equation calculates the system privacy loss on the domain of private variables ($SPL_{dom}$):

$$SPL_{pvdom} = \sum_i \sum_{j \neq i} \sum_{x \in V_j} \text{private}(x, A_j) \times knowDom(A_i, x).$$

The maximum system privacy loss on the domain of private variables ($MSPL_{pvdom}$) will assume that all the domains of private variables are leaked. Therefore, the value of the function $knowDom(A_i, x)$ will always return 1. As a result, the $MSPL_{pvdom}$ function will be the same as $MSPL_{pvid}$:

$$MSPL_{pvdom} = \sum_i \sum_{j \neq i} \sum_{x \in V_j} \text{private}(x, A_j).$$

The maximum system privacy loss on private variables domains ($MSPL_{pvdom}$) in Fig. 2.3 is the same as ($MSPL_{pvid}$), which is $6 + 7 + 6 + 7 + 6 = 32$.

The normalize system privacy loss ($NSPL_{pvdom}$) is:

$$NSPL_{pvdom} = \frac{\sum_i \sum_{j \neq i} \sum_{x \in V_j} \text{private}(x, A_j) \times knowDom(A_i, x)}{\sum_i \sum_{j \neq i} \sum_{x \in V_j} \text{private}(x, A_j)} \in [0, 1].$$

### 2.3.3 Privacy on Shared Variables

**Shared variables identities** The identities of the shared variables are the first concern in this category. To calculate the privacy loss on the identity of the shared variables, the function $shared(x, A_i, A_j)$ is used where it returns 1 if the variable $x$ is shared by the agents $A_i$ and $A_j$ and returns 0 otherwise. For example, the variable $i$ in Fig. 2.3 is shared between the agents $A_3$ and $A_4$, so the function $shared(i, A_3, A_4)$ will return 1. However, the variable $g$ is private to $A_3$, so $shared(g, A_3, A_4)$ will return 0.
0. To calculate if the identity of a shared variable is leaked to a non sharing agent, the equation 
\[ \text{shared}(x, A_j, A_k) \ast \text{knowID}(A_i, x) \] is used. This equation returns 1 if there exist a shared variable \( x \) that is shared between the agents \( A_j \) and \( A_k \), and the identity of this variable is leaked to another agent \( A_i \). For instance, the variable \( i \) is shared between the agents \( A_3 \) and \( A_4 \) in Fig. 2.3, and it is not shared with the agent \( A_i \). Therefore, the equation \( \text{shared}(i, A_3, A_4) \ast \text{knowID}(A_1, i) \) should return \( 1 \ast 0 = 0 \). However, if the identity of the shared variable \( i \) is leaked to \( A_1 \), the equation will return \( 1 \ast 1 = 1 \). The system privacy loss on the shared variables identities (\( SPL_{svid} \)) is as follows:

\[
SPL_{svid} = \sum_i \sum_{j \neq i} \sum_{k \neq i, k \neq j} \sum_{x \in V_j} \text{shared}(x, A_j, A_k) \ast \text{knowID}(A_i, x).
\]

The maximum system privacy loss on shared variables identities (\( MSPL_{svid} \)) assumes that all the identities of shared variables are leaked. Therefore, the function \( \text{knowID}(A_i, x) \) will always return 1 as follow:

\[
MSPL_{svid} = \sum_i \sum_{j \neq i} \sum_{k \neq i, k \neq j} \sum_{x \in V_j} \text{shared}(x, A_j, A_k).
\]

However, this equation has a problem where it counts privacy loss in some cases where it should not. For example, if the equation is applied to the MAS in Fig. 2.3, when it reaches \( i = 1, j = 2, k = 5 \), and \( x = d \), the equation becomes as follows.

\[
\text{shared}(d, A_2, A_5) \ast \text{knowID}(A_1, d)
\]

The function \( \text{shared}(d, A_2, A_5) \) will return 1 since the variable \( d \) is shared between \( A_2 \) and \( A_5 \), and \( \text{knowID}(A_1, d) \) will return 1 since \( d \in A_1 \), so the equation will count it as privacy loss where no privacy has been leaked. To fix this problem, the equation can be revised as follows. Let \( \text{notIN}(x, A_i) \) be a function that returns 1
if the variable $x$ is not contained in the variable set of the agent $A_i$ ($x \notin A_i$), and
returns 0 otherwise. For example, $notIN(i, A_1)$ should return 0 in Fig. 2.3. Using
this function, the equation can be revised as follow.

$$SPL_{svid} = \sum_i \sum_{j \neq i} \sum_{k \neq i, k \neq j} \sum_{x \in V_j} shared(x, A_j, A_k) \cdot knowID(A_i, x) \cdot notIN(x, A_i).$$

After this modification, the corresponding maximum system privacy loss on shared
variables identities is as follows:

$$MSPL_{svid} = \sum_i \sum_{j \neq i} \sum_{k \neq i, k \neq j} \sum_{x \in V_j} shared(x, A_j, A_k) \cdot notIN(x, A_i).$$

For simplicity, $shared(x, A_k, A_k)$ and $notIN(x, A_i)$ will be written as $Sh(x, A_j, A_k)$
and $N(x, A_i)$ respectively in the next example. The maximum system privacy loss on
shared variables identities in Fig. 2.3 is as follows:

$$((Sh(d, A_2, A_3) \cdot N(d, A_1) + Sh(f, A_2, A_3) \cdot N(f, A_1) + Sh(d, A_2, A_4) \cdot N(d, A_1)
+ Sh(f, A_2, A_4) \cdot N(f, A_1) + Sh(d, A_2, A_5) \cdot N(d, A_1) + Sh(f, A_2, A_5) \cdot N(f, A_1))
+ (\sum_{x \in A_3} Sh(x, A_3, A_2) \cdot N(x, A_1) + \sum_{x \in A_3} Sh(x, A_3, A_4) \cdot N(x, A_1)
+ \sum_{x \in A_3} Sh(x, A_3, A_5) \cdot N(x, A_1))
+ (\sum_{x \in A_4} Sh(x, A_4, A_2) \cdot N(x, A_1) + \sum_{x \in A_4} Sh(x, A_4, A_3) \cdot N(x, A_1)
+ \sum_{x \in A_4} Sh(x, A_4, A_5) \cdot N(x, A_1))
+ (\sum_{x \in A_5} Sh(x, A_5, A_2) \cdot N(x, A_1) + \sum_{x \in A_5} Sh(x, A_5, A_3) \cdot N(x, A_1)
+ \sum_{x \in A_5} Sh(x, A_5, A_4) \cdot N(x, A_1))))$$

20
\[ \sum_{k\neq 2, k\neq 1} \sum_{x \in A_1} Sh(x, A_1, A_k) \times N(x, A_2) + \sum_{k\neq 2, k\neq 3} \sum_{x \in A_3} Sh(x, A_3, A_k) \times N(x, A_2) \]

\[ + \sum_{k\neq 2, k\neq 4} \sum_{x \in A_4} Sh(x, A_4, A_k) \times N(x, A_2) + \sum_{k\neq 2, k\neq 5} \sum_{x \in A_5} Sh(x, A_5, A_k) \times N(x, A_2) \]

\[ + \sum_{j\neq 3} \sum_{k\neq 3, k\neq j} \sum_{x \in A_j} Sh(x, A_j, A_k) \times N(x, A_3) \]

\[ + \sum_{j\neq 4} \sum_{k\neq 4, k\neq j} \sum_{x \in A_j} Sh(x, A_j, A_k) \times N(x, A_4) \]

\[ + \sum_{j\neq 5} \sum_{k\neq 5, k\neq j} \sum_{x \in A_j} Sh(x, A_j, A_k) \times N(x, A_5) \]

\[ = 2 \quad \text{(when } i = 1) + 6 \quad (i = 2) + 4 \quad (i = 3) + 4 \quad (i = 4) + 6 \quad (i = 5) = 22 \]

The normalized system privacy loss on shared variables identities \( NSPL_{svid} \) is as follows:

\[ NSPL_{svid} = \frac{\sum_i \sum_{j\neq i} \sum_{k\neq i, k\neq j} \sum_{x \in V_j} shared(x, A_j, A_k) \times knowID(A_i, x) \times notIN(x, A_i)}{\sum_i \sum_{j\neq i} \sum_{k\neq i, k\neq j} \sum_{x \in V_j} shared(x, A_j, A_k) \times notIN(x, A_i)} \in [0, 1]. \]

**Shared variables domains**

The privacy loss on shared variables domains can be calculated using the same functions used before. The system privacy loss on shared variables domains \( SPL_{sdom} \) is as follows:

\[ SPL_{sdom} = \sum_i \sum_{j\neq i} \sum_{k\neq i, k\neq j} \sum_{x \in V_j} shared(x, A_j, A_k) \times knowDom(A_i, x) \times notIN(x, A_i) \]

The maximum system privacy loss on shared variables domain is same equation used for \( MSPL_{svid} \):

\[ MSPL_{sdom} = \sum_i \sum_{j\neq i} \sum_{k\neq i, k\neq j} \sum_{x \in V_j} shared(x, A_j, A_k) \times notIN(x, A_i) \]
The maximum system privacy loss on shared variables domains \((MSPL_{sdom})\) in Fig. 2.3 is the same as \((MSPL_{svi})\), which is \(2 + 6 + 4 + 4 + 6 = 22\). The normalized system privacy loss on shared variables domains \((NSPL_{sdom})\) is as follows:

\[
NSPL_{sdom} = \frac{\sum_i \sum_{j \neq i} \sum_{k \neq i, k \neq j} \sum_{x \in V_j} \text{shared}(x, A_j, A_k) \ast \text{knowDom}(A_i, x) \ast \text{notIN}(x, A_i)}{\sum_i \sum_{j \neq i} \sum_{k \neq i, k \neq j} \sum_{x \in V_j} \text{shared}(x, A_j, A_k) \ast \text{notIN}(x, A_i)}
\in [0, 1]
\]

2.4 Existing Algorithms

There are many frameworks that adapt the JT organization in MAS to perform various tasks. This section will focus on the algorithms Action-GDL [16], DCTE [1], COOD-Plus [21], and DPMST [22]. How each of these algorithms constructs the JT will be considered.

2.4.1 Action-GDL Algorithm

Action-GDL is a message passing algorithm based on the generalized distributive law (GDL) developed to effectively solve distributed constraint optimization problems (DCOPs) [16]. Action-GDL is designed so each physical agent controls a single variable. However, a virtual agent is assumed to control multiple physical agents, and physical agents will be considered as variables, as considered in [22].

First a pseudotree is constructed in the process to create a JT. Let \(G\) be a dependency graph as in Fig. 2.4(a). A DFS tree is constructed in order to create a pseudotree breaking tie alphabetically as in Fig. 2.4(b). Now a pseudotree is the resultant DFS tree with the undirected links that are not part of the DFS tree considered
as back links. Back links are represented in Fig. 2.4(c) as dotted links. The variables connected with back links are considered as a pseudoparent and a pseudochild.

Consider the MAS in Fig. 2.5(a). Action-GDL first constructs a pseudotree as shown in Fig. 2.5(b). Then the JT is constructed where each node in the pseudotree forms a cluster containing three main parts: 1) the node’s own variable, 2) the node’s parents and its pseudoparents, and 3) the variables of each child’s cluster except the child’s variables. The resultant JT is shown in Fig. 2.5(c). Since each agent has access to its own variables, each agent is assumed to gain access to every cluster containing at least one of its variables. For example, agent $A_3$ in Fig. 2.5(a) has the variables $b$ and $d$; therefore, $A_3$ has access to the clusters $\{a, b, c\}$ and $\{b, d\}$ because the cluster $\{a, b, c\}$ contains the variable $b$, and the cluster $\{b, d\}$ has the variables $b$ and $d$. 

23
Considering the privacy loss on private variables’ identities and domains in Fig. 2.5, the agent $A_2$ knows the identity and the domain of the variable $c$, which is a private variable in $A_1$, through the cluster $\{a, c\}$. The agent $A_2$ has access to the cluster $\{a, c\}$ through the variable $a$. Also, the private variable $c$ is leaked to the agent $A_3$ as well through the cluster $\{a, b, c\}$. Therefore, the privacy of private variables’ identities and domains is leaked when using Action-GDL to construct a JT. Shared variables’ identities and domains are also leaked in Action-GDL. For example, the shared variable $a$ between $A_1$ and $A_2$ is leaked to $A_3$ through the cluster $\{a, b, c\}$. The agent $A_3$ has access to the cluster $\{a, b, c\}$ through the variable $b$; therefore, $A_3$ has access to the identity and the domain of the shared variable $a$.

However, Action-GDL doesn’t incur any privacy loss on agent identities and bordering relations. To summarize, Action-GDL leaks the privacy of private and shared variables’ identities and domains, but it preserve the privacy of agent identities and bordering relations.

### 2.4.2 DCTE Algorithm

DCTE, as Action-GDL, is also an algorithm designed for constraint optimization problems (DCOP) [1]. However, since the thesis focus on the construction of the JT organization, only the construction part will be considered. DCTE constructs the JT tree using a method from [9] where instead of each physical agent controlling single variable (as in Action-GDL), each agent controls multiple variables. However, the method in [9] allows any two agents to communicate even if they are not neighbors, which may cause more privacy leaks where it can be protected. Therefore, the work in [22] considered a revised version where agents only communicate with their neighbors. The revised version in [22] will be considered here as well.
DCTE starts by creating a spanning tree. Each agent elects itself as the root and announce its choice to its neighboring agents. When an agent receives the choice of its neighboring agents that they elected themselves as the root, it compares the choices with its own choice to select the agent that has the minimum ID value to be the root. The ID value can be any value that differentiates the agents. For example, the MAC address is used as the ID value in [9]. After that each agent updates its choice of the root, and chooses its own parent and announces it to its neighboring agents. Then the choice of the root, and the choice of each agent’s parent is propagated through the system until all the agents agree on the same choices.

For example, when constructing a spanning tree from the environment decomposition in Fig. 2.6(a), the agent $A_1$ will be elected as the root, assuming the ID value is the agent number (e.g. the ID value of $A_1$ is 1), and the resultant spanning tree

Figure 2.6: a) Environment decomposition, b) possible spanning tree, c) resultant JT

For example, when constructing a spanning tree from the environment decomposition in Fig. 2.6(a), the agent $A_1$ will be elected as the root, assuming the ID value is the agent number (e.g. the ID value of $A_1$ is 1), and the resultant spanning tree
is shown in Fig. 2.6(b). This process guarantees a tree organization, and since each agent controls multiple variables, it guarantees a cluster tree organization. However, it’s not guaranteed to be a JT yet because the main property of JT, which is the running intersection, is not guaranteed. For example, the resultant spanning tree in Fig. 2.6(b) is not a JT because the agents $A_4$ and $A_3$ share the variable $i$, yet it’s not included in the path between them.

To ensure that the cluster tree is guaranteed to have running intersection and represent a JT, each agent announces all of its variables to its neighboring agents. Also, each agent announces all the variables reached to it to its neighboring agents. When an agent receives the same variable from two different agents, it adds this variable to its cluster. To elaborate, this step ensures that whenever there are two agents sharing a variable, that variable will be included in every agent in the path between them.

For instance, to ensure the running intersection in the spanning tree in Fig. 2.6(b), the agent $A_4$ will announce its variables $a, g, i$ to $A_1$. The agent $A_1$ will announce its own variables as well as the variables received from $A_4$, $a, b, f, g, i$, to $A_2$. After that, $A_2$ will announce $a, b, d, g, f, i$ to $A_3$. The agent $A_3$ will also announce its own variables $d, i$ to $A_2$. The agent $A_2$ will announce $d, f, i$ to $A_1$. After that, $A_1$ will announce $a, b, d, f, i$ to $A_4$. The agent $A_1$ has received the variable $i$ from $A_2$ and from $A_4$, so it adds the variable $i$ to its cluster. The agent $A_2$ has received the variable $i$ from $A_1$ and from $A_3$, so it adds the variable $i$ to its cluster as shown in Fig. 2.6(c). This method ensures that the cluster tree in Fig. 2.6(c) is a JT because it ensures the running intersection.

When considering the privacy leak using DCTE, this approach leaks all types of privacy. First, it leak the agent identity when electing the root, and announce it
through the system. Second, it leaks the agents’ bordering relations when announcing
the choice of parent. For example, when the agent $A_4$ in Fig. 2.6(a) announces that it
chose the agent $A_1$ as its parent to the agent $A_3$, the border relation between $A_4$ and
$A_1$ will be leaked to $A_3$. Third, when agents announce their variables to its neighbors
to ensure the running intersection, the identity and domain of all the private and the
shared variables will be leaked. For example, when agent $A_4$ announce the variables
$a, g, i$ to agent $A_1$, the identity and the domain of the private variable $g$ and the shared
variable $i$ will be leaked to $A_1$. To summarize, DCTE incurs all types of privacy loss
when constructing the JT.

2.4.3 COORD-Plus Algorithm

Some frameworks in probabilistic reasoning construct the JT organization through
a coordinator agent as in [18, 21]. The coordinator agent has access to agents’ iden-
tities, bordering relations, and shared variables identities. However, when it comes
to private variables, since the private variables are not crucial in the construction
process, the coordinator agent has access to only a single private variable from each
agent if the agent have private variables [18]. Since the coordinator agent has this
much access to agents’ information, the coordinator agent can recognize the existence,
and constructs a JT if it exists.

Only the coordinator agent has access to agents’ information, so there is no
privacy leaks between the other agents. However, considering the coordinator agent
as one of the agents, the privacy of all the other agents is leaked to this agent. This
method is refereed to as COORD-Plus in [22]. To sum up, COORD-Plus leaks the
privacy of agent identities, bordering relations, and the identities of both private and
shared variables to the coordinator agent.
2.4.4 DPMST Algorithm

DPMST is an algorithm designed for the purpose of constructing the JT organization with minimal privacy loss [22]. DPMST is a maximum spanning tree based algorithm that “extends Prim’s algorithm [12] distributively and grow an MST through a rooted control” [22]. Agents use four type of messages to build a JT:

- **Notify** A leaf on the MST tree notifies a neighbor, which was not in the MST, that it has been added to the MST.
- **Announce** The sending agent announces to all its neighbors (except the tree-parent) that it has been added to the MST.
- **Expand** The sending agent orders a tree-child to expand the MST by adding a new node.
- **Report** The sending agent reports back to its tree-parent the best outgoing weight (bow), or it reports termination.

DPMST construct a JT from the *boundary set* of the given MAS. The boundary set of an MAS is found first to preserve all the private variables information. Consider finding the boundary set $W$ for the MAS on Fig. 2.7 (1). To find the boundary set $W$, first the relative boundary $W_i$ for each agent $A_i$ is found. The boundary is the union of all the borders that the agent have. For example the boundary $W_2$ of the agent $A_2$ in Fig. 2.7(1) is as created as follows. $A_2$ has a border with the agent $A_1$ over the variables $b, v$ ($I_{12} = \{b, d\}$), so $W_2 = I_{12} = \{b, v\}$. The relative boundary set of the MAS in Fig. 2.7(1) is $W = \{\{a, b, d, v\}, \{b, v\}, \{a, i\}, \{a, d, i\}\}$, and can be represented graphically as the *communication graph* where each boundary is a cluster, and for every two boundaries $W_i$ and $W_j$ there is a link between them if their intersection is not empty ($W_i \cap W_j \neq \emptyset$). The communication graph of the MAS in Fig. 2.7(1) can be seen in Fig. 2.7(2), where the private variables $u$ and $n$, in agents $A_2$ and $A_3$ respectively, are not part of the communication graph.
Figure 2.7: Execution steps when running DMPST
It’s been proven in [22] that if a JT exists such that each cluster in the JT is a boundary in W, the JT can be applied to the MAS where each cluster is replaced with the relative agent of the boundary [22]. Therefore, creating a JT from the boundary set, represented in a communication graph, preserve the privacy of private variables.

DPMST constructs the JT from the communication graph. First, each boundary in the communication graph is considered as a node, and each link between any two boundaries is weighted by the number of shared variables between those two boundaries, Fig. 2.7(3). After that, a root node is chosen arbitrarily. For example, the node $x_1$ was chosen to be the root in Fig. 2.7(4). After the root get chosen, the root announces that it is in the MST by sending Announce message to its neighbors, and expand the MST by choosing the maximum link weight among its neighbors by sending Notify message to that neighbor, Fig. 2.7(4).

After that, whenever a new node is added to the MST, it will weight its links and report back the maximum weight to its tree-parent, and the tree parent will update its weighted links and report the updated weight to its tree-parent until it reaches the root. The root then will update its weighted links and choose the maximum weighted link by sending the Expand message to the neighbor that reported that weighted link. The Expand message will pass down until it reaches a tree leaf, and the tree leaf will send Notify message to its maximum weighted link, and add a new node to the MST.

Then these steps are repeated until all the tree leaves don’t have any neighboring nodes that are not part of the MST. In that case, the tree leaves will report to terminate, and the terminate report will pass up the tree until it reaches the root. When the root receive the terminate report, and the root has no more neighbors that are not part of the MST, the MST will be completed, and resultant JT will be the MST when the nodes are considered as their original clusters in the CG. These steps
are elaborated in the example in Fig. 2.7 where the CG is shown in (2), and the resultant JT is shown in (10).

However, the resultant MST forms a JT if and only if a JT exists in the given CG [22]. Therefore, DPMST can only be applied to create a JT if it is known that a JT exists in the CG. DPMST does not recognize the existence of the JT, so the existence must be found by other algorithms first. If the JT existence is known, DPMST constructs a junction tree without leaking any privacy on shared or private variables in the process. DPMST is reported to be the first algorithm to construct a MST without leaking any privacy on agent identities [22]. However, the paper reported the concern that some bordering relations might be inferred by agents while constructing the JT [22]. To sum up, DPMST preserve the privacy of shared and private variables, and agent identities, but some border relations might be inferred and leaked.

2.5 HTBS Algorithm

HTBS algorithm suite was developed to recognize the existence of a JT in a given environment decomposition without leaking any privacy [22]. Since DPMST can only build a JT only if one exists, HTBS was developed to recognize the existence to apply DPMST to build the JT if it exists. HTBS considered how the existence of a JT can be determined distributively without leaking any private information as measured in the privacy loss quantification scheme presented in Section 2.3.

However, a by-product of HTBS could construct the JT without leaking any private information as well. That way, HTBS surpassed DPMST by recognizing the existence of a JT, and constructing it if it exists without leaking border relations.
that could be leaked in DPMST. Moreover, HTBS is faster and more efficient than DPMST [22]. HTBS is the only known algorithm to construct a JT with no privacy loss on all types of agent’s privacy. HTBS recognize the existence of a JT by two “necessary and sufficient conditions for hypertree existence” [22]. Both conditions are derived from the boundary set of the given MAS. Hence, no privacy loss on private variables. The first condition is based on the boundary graph (BG), and the second is based on the boundary set.

### 2.5.1 Boundary Graph Based Condition

Boundary graph is an undirected graph derived from the boundary set of the given MAS. The nodes of the undirected graph are all the variables contained in the boundary set. However, the boundary graph must meet a specific condition
to viable. Every set of variables $V_i$ that are contained in the same boundary $W_i$ must be pairwise connected. For example, the boundary set $W$ of the MAS in Fig. 2.7(1) contains $\{W_1, W_2, W_3, W_4\}$, and each boundary has the following variables $\{\{a, b, d, v\}, \{b, v\}, \{a, i, n\}, \{a, d, i\}\}$ respectively. Since the variables $\{a, b, d, v\}$ are contained in the same boundary $W_1$, they have to be pairwise connected in the boundary graph. The boundary graph of the MAS in Fig. 2.7(1) can be seen on Fig. 2.8(a) where the variables in each boundary $\{\{a, b, d, v\}, \{b, v\}, \{a, i, n\}, \{a, d, i\}\}$ are pairwise connected.

The boundary graph is considered *chordal* when the BG has no cycle of size four or more without a chord. A *chord* is a link between two nodes in a cycle of size four or more that is not part of the cycle links. For example, the boundary graph in Fig. 2.8(a) has only one cycle of size four, which is $(a, b, v, d, a)$, and since this cycle has two chords, which is $(a, v)$ and $(b, d)$, the boundary graph is chordal. A *clique* is an undirected graph such that the boundary graph is a set of variables that are pairwise connected. A *maximum clique* is the maximum set of variables that are pairwise connected. For example, the variables $a, b, d$ are pairwise connected, but they don’t form a maximum clique because the variable $v$ is also pairwise connected with every one of them. However, the variables $a, b, d, v$ forms a maximum clique. Only maximum cliques are considered in this area, so it will be referred to as a *clique*.

JT exists in the given MAS if and only if the boundary graph is chordal, and for every clique in the boundary graph, there exist a boundary $W_i$ such that all the variables of the clique are contained in $W_i$ [22]. To elaborate, the work in [22] categorize the boundary graph into “three mutually exclusive and exhaustive types” [22].
**Type 1** The boundary graph is chordal, and all the cliques are boundary contained.

**Type 2** The boundary graph is not chordal.

**Type 3** The boundary graph is chordal, but there exist at least one clique that is not boundary contained.

A JT exists if and only if the boundary graph is type 1 [22]. For example, the MAS in Fig. 2.7(1) has the boundary graph in Fig. 2.8(a), which is a type 1 boundary graph; therefore, a JT agent organization exists for the given decomposition as shown in Fig. 2.7(10). However, the environment decomposition in Fig. 2.8(b) has the boundary graph in Fig. 2.8(c). The boundary graph in Fig. 2.8(b) has the clique \(a, b, d, i\) that is not contained in any boundary of Fig. 2.8(b), so it is a type 3 boundary graph. Therefore, the environment decomposition in Fig. 2.8 does not have a JT agent organization. Moreover, the environment decomposition in Fig. 2.8(d) has the boundary graph in Fig. 2.8(e). The boundary graph in Fig. 2.8(e) has a cycle of size four, which is \((c, a, e, d, c)\), that does not have a chord, so it’s not chordal (type 2 boundary graph). Therefore, the environment decomposition in Fig. 2.8(d) has no JT agent organization.

### 2.5.2 Boundary Set Based Condition

Another necessary and sufficient condition of a JT existence is derived from the boundary set of the given MAS. This condition realize the existence of a JT by using boundary elimination. A boundary \(W_i\) in the boundary set \(W\) can self eliminate if there exits another boundary \(W_j\) such that \(W_i\) is a subset or equal to \(W_j\), \(W_i \subseteq W_j\). When this condition is met, the boundary \(W_i\) can self eliminate relative to the boundary \(W_j\). If \(W_i\) self eliminate from the boundary set \(W\) relative to the
boundary $W_j$, the boundary $W_j$ needs to update its variable set since the boundaries are created from the union of their borders, and the bordering boundary $W_i$ has been eliminated ($I_{ij}$).

For example, when considering the boundary set of the MAS in Fig. 2.7(1), $W = \{\{a, b, d, v\}, \{b, v\}, \{a, i\}, \{a, d, i\}\}$, the boundary $\{b, v\}$ can self eliminate relative to the boundary $\{a, b, d, v\}$. After eliminating, the boundary $\{a, b, d, v\}$ will be updated to be $\{a, d\}$, which is the union of its remaining borders ($\{a, d\}$ with the boundary $\{a, d, i\}$, and $\{a\}$ with the boundary $\{a, i\}$). A JT exists if and only if the boundary set can perform boundary elimination until it becomes a single empty boundary [22]. For example, the boundary set of the MAS in Fig. 2.7(1), $W = \{\{a, b, d, v\}, \{b, v\}, \{a, i\}, \{a, d, i\}\}$, can perform self elimination as follows.

$\{b, v\}$ self eliminates relative to $\{a, b, d, v\}$  $W = \{\{a, d\}, \{a, i\}, \{a, d, i\}\}$
$\{a, d\}$ self eliminates relative to $\{a, d, i\}$  $W = \{\{a, i\}\}$
$\{a, i\}$ self eliminates relative to $\{a, i\}$  $W = \{\{\}\}$

Since the boundary set was eliminated to a single empty boundary, a JT exists on the given MAS as shown in Fig. 2.7(10). However, the boundary set of the environment decomposition in Fig. 2.8(b) has the boundary set $W = \{\{a, b, d\}, \{a, b, i\}, \{a, i\}, \{a, d, i\}\}$ that cannot perform boundary elimination to a single boundary. The boundary $\{a, i\}$ will self eliminate relative to the boundary $\{a, b, i\}$, and the updated boundary set is $W = \{\{a, b, d\}, \{a, b, i\}, \{a, d, i\}\}$. No more boundaries can be eliminated. Therefore, no JT exists in the MAS in Fig. 2.8(b).

2.5.3 Recognizing JT Existence Distributively

HTBS recognizes the existence of the JT agent organization by performing the boundary elimination distributively. Since each agent can determine whether or not it
can perform self elimination locally from its borders, all types of privacy are preserved. HTBS perform self elimination distributively in rounds, where in each round one boundary should be eliminated. A token is passed between the boundaries that allows them to perform the self elimination when possible, as well as to keep track of the rounds.

HTBS starts from the communication graph by selecting an arbitrary boundary to be the leader, and gives it the token of round 1. The selected boundary then determine if it can self eliminate or not relative to one of the bordering boundaries. If it can self eliminate relative to one of the neighboring boundaries, it will perform the self elimination and the boundary that allowed the selected leader to self eliminate will be the new leader of the next round, and receive a new token for the new round. However, if the selected leader cannot perform the self elimination, it will pass the token to one of the neighboring boundaries, and the token will passed between agents by depth-first-traversal (DFT). Whenever a boundary receive a token, it will check if it has received this token before or not (whether the boundary has been visited in this round or not). If the receiving boundary did receive the token before, it will report back that it has been visited in this round. Then the token will be passed until a boundary that has not been visited, and this boundary is able to self eliminate, then it will perform the self elimination and send a new token to the boundary that it to perform self elimination relative to it to start a new round of DFT. These steps are repeated until the boundary set is eliminated into a single boundary (a JT exist), or all the boundaries are visited in the current round, and no boundary can perform self elimination (no JT exists).

For example, the existence of a JT agent organization in the environment decomposition in Fig. 2.9(1) can be recognized distributively as follow. An arbitrary
boundary is selected as the leader of round 1, suppose that $W_1$ is selected to be the leader of round 1. $W_1$ will start with the first round of DFT with $tok^1$, and it will check if it can self eliminate, but it cannot, so it will pass the token to $W_2$. When $W_2$ receive $tok^1$, it will check if it can self eliminate, and it will realize that it can self eliminate relative to $W_1$, so it will self eliminate relative to $W_1$, and will notify $W_1$. The boundary $W_1$ will update its variables to be $\{a, d\}$, and start a new round of DFT with the $tok^2$ as shown in Fig. 2.9(2). After that, $W_1$ will realize that it can self eliminate relative to $W_4$, so it will self eliminate and notify $W_4$. When $W_4$ receive the message from $W_1$, it will update its variables to $\{a, i\}$, and start a new round of DFT with $tok^3$ as shown in Fig. 2.9(3). Finally, $W_4$ will self eliminate relative to $W_3$, and since the boundary set is eliminated to a single boundary, the existence of the JT agent organization is recognized.

### 2.5.4 HTBS Algorithm Suite

HTBS consist of four main procedures 1) $Response to Start New DFT(tok)$ 2) $Do DFT$ 3) $Response to DFT(tok)$ 4) $Response to Report$, and these four procedures are triggered by the following messages [22]:

![Figure 2.9: Recognizing the existence of JT distributively](image)
An agent notify a neighboring agent that it has been self-eliminated.

A request to start a new round of DFT with the given token.

A request to continue the ongoing round of DFT with the given token.

An agent report back to the previous agent in this DFT round that it has been visited in this round, or that the DFT round has been completed.

\( A_i \) is the agent running the procedure. \( A_c \) is the sender agent. Each agent has the following parameters: 1) \( state \in \{IN, OUT\} \) that indicates whether this agent has been eliminated or not, 2) \( nbsta(A_k) \in \{IN, OUT\} \) that indicates whether the bordering agent \( A_k \) has been eliminated or not, 3) \( curtok \) that keeps track of the round number, 4) \( visited(A_k) \) that indicates whether the neighboring agent \( A_k \) has been visited in this DFT round or not, 5) \( parent \) that keeps track of the sender \( A_c \) because the agent might send a report message back to \( A_c \), and 6) \( Y_i \), initiated as \( Y_i = W_i \), that keeps track of the active boundary of the agent since whenever a neighboring agent perform self-elimination relative to this agent, the active boundary will be updated.

In the first round, an external \( StartNewDFT(tok^1) \) is sent to an arbitrary agent. That agent will become the leader of the first DFT round. The procedure \( ResponsetoStartNewDFT(tok) \) will be executed first.

The procedure \( ResponsetoStartNewDFT(tok) \) will be executed as follows. When \( A_i \) receives the \( StartNewDFT(tok) \) message, \( A_i \) will check if the sender of this message is a neighboring agent or not. Receiving the \( StartNewDFT(tok) \) message from a neighboring agent means that the sender agent \( A_c \) has been self-eliminated. Therefore, \( A_i \) will update the state of \( A_c \) to \( OUT \). After that, \( A_i \) will check if there exist
other agents that has not been eliminated yet. If there are no remaining agents, the boundary set has been eliminated to a single boundary, which means a JT exists. However, when $A_i$ has remaining non-eliminated agents, $A_i$ will first update its active boundary set $Y_i$ from its borders with the remaining active agents. After that, $A_i$ will start a new round of DFT with itself as the leader of this round ($parent = nil$) and execute the procedure $DoDFT$.

When $A_i$ starts on executing $DoDFT$, $A_i$ will first check if it can perform self-elimination relative to one of the remaining active agents. If $A_i$ can perform self-elimination, it will notify its neighbors that it has been eliminated, so they update its $nbsta(A_i)$ to $OUT$. $A_i$ will also send the request message $StartNewDFT(\text{curtok} + 1)$ to the agent that allowed $A_i$ to self-eliminate relative to it. As a result, the agent receiving the $StartNewDFT(\text{curtok} + 1)$ message will start a new DFT round with a new token. However, if $A_i$ cannot perform self-elimination relative to a neighboring agent, it will continue the current round of DFT, and send the request $DFT(\text{tok})$ to the next agent. The procedure that response to this request is $ResponsetoDFT(\text{tok})$.

The procedure $ResponsetoDFT(\text{tok})$ has a simple task. The agent $A_i$ will check in this procedure whether the received token is fresh or not. If the token is not fresh, that means this agent has been visited in this round of DFT through another agent, so it reports back to the sender that it has been visited. On the other hand, if the token is fresh, the agent $A_i$ will run $DoDFT$, which either allows it to self-eliminate or pass the token to the next agent in this DFT round.

Finally, when an agent $A_i$ receives a report, that means the DFT round has not been completed yet. Therefore, $A_i$ will send the request to continue this round of DFT by passing the token to a bordering active agent. However, if there exist no bordering active agents, $A_i$ will report back to its parent $A_c$ that the DFT round has
not been completed. In the case of \( A_i \) being the leader of this round, its parent will be \( \text{nil} \). Therefore, if \( A_i \) has no bordering active agents, and \( A_i \) is the leader of this round, the boundary set has not been able to eliminate its boundaries into a single remaining boundary. Therefore, HTBS will recognize that no JT exists in the given boundary set.

### 2.5.5 Examples of HTBS

As shown in Section 2.5.1, the boundary graph can be of type 1, type 2, or type 3. HTBS will recognize the existence of the JT in type 1, and constructs. When HTBS is executed in type 2 or type 3, HTBS will recognize the non-existence of the JT organization. Three examples of HTBS executions on type 1, type 2, and type 3 is presented in this section.

#### Example of type 1 boundary graph

The communication graph in Fig. 2.10(a) has a type 1 boundary graph as shown in Fig. 2.10(b). Therefore, a JT exists on the given environment. HTBS will recognize the existence of the JT as follows:

\[ \text{StartNewDFT}(1) \] will be sent to \( W_1 \). \( W_1 \) will set \( \text{parent} = \text{nil} \), and start
Figure 2.11: (a) Communication graph with its boundary graph of type 2 in (b)

*DoDFT*. $W_1$ cannot self eliminate, so it will send $DFT(1)$ to $W_2$. Since $tok = 1$ is fresh to $W_2$, $W_2$ will set $parent = W_1$, and execute $DoDFT$. $W_2$ can self eliminate relative to $W_1$, so $W_2$ will set $state = OUT$. $W_2$ will send $StartNewDFT(2)$ to $W_1$. $W_1$ will set $nbsta(W_2) = OUT$, and updates $Y_1$ to $\{a, d\}$. $W_1$ will set $parent = nil$, and starts $DoDFT$. $W_1$ can self eliminate relative to $W_4$, so $W_1$ will set $state = OUT$. $W_1$ will send $StartNewDFT(3)$ to $W_4$. $W_4$ will set $nbsta(W_1) = OUT$, and update $Y_4$ to $\{a, i\}$. $W_4$ will set $parent = nil$, and start $DoDFT$. $W_4$ can self eliminate relative to $W_3$, so $W_4$ will set $state = OUT$. $W_4$ will send $StartNewDFT(4)$ to $W_3$. $W_3$ will set $nbsta(W_1) = OUT$. Since $W_3$ has no neighboring agent $A_k$ where $nbsta(A_k) = IN$, $W_3$ will announce that a JT exists. The resultant JT is shown in Fig. 2.7(10).

**Example of type 2 boundary graph** The communication graph in Fig. 2.11(a) has a type 2 boundary graph as shown in Fig. 2.11(b). Therefore, no JT exists on the given environment. HTBS will recognize the non-existence of the JT as follows:

$StartNewDFT(1)$ will be sent to $W_1$. $W_1$ will set $parent = nil$, and start $DoDFT$. $W_1$ cannot self eliminate, so it will send $DFT(1)$ to $W_2$. Since $tok = 1$ is
Figure 2.12: (a) Communication graph with its boundary graph of type 3 in (b)

fresh to $W_2$, $W_2$ will set $parent = W_1$, and execute $DoDFT$. $W_2$ cannot self eliminate, so it will send $DFT(1)$ to $W_3$. Since $tok = 1$ is fresh to $W_3$, $W_3$ will set $parent = W_2$, and execute $DoDFT$. $W_3$ cannot self eliminate, so it will send $DFT(1)$ to $W_4$. Since $tok = 1$ is fresh to $W_4$, $W_4$ will set $parent = W_3$, and execute $DoDFT$. $W_4$ cannot self eliminate, so it will send $DFT(1)$ to $W_1$. Since $tok = 1$ is not fresh to $W_1$, $W_1$ will send report back to $W_4$. Since $W_4$ has no remaining agents, $W_4$ will report back to $W_3$. Since $W_3$ has no remaining agents, $W_3$ will report back to $W_2$. Since $W_2$ has no remaining agents, $W_2$ will report back to $W_1$. $W_1$ has no remaining agents, and $W_1$ has $parent = nil$. Therefore, $W_1$ will announce “no JT exists”.

**Example of type 3 boundary graph** The communication graph in Fig. 2.12(a) has a type 2 boundary graph as shown in Fig. 2.12(b). Therefore, no JT exists on the given environment as shown in Fig. 2.13. HTBS will execute as follows:

StartNew$DFT(1)$ will be sent to $W_1$. $W_1$ will set $parent = nil$, and start $DoDFT$. $W_1$ cannot self eliminate, so it will send $DFT(1)$ to $W_2$. Since $tok = 1$ is fresh to $W_2$, $W_2$ will set $parent = W_1$, and execute $DoDFT$. $W_2$ cannot self eliminate, so it will send $DFT(1)$ to $W_3$. Since $tok = 1$ is fresh to $W_3$, $W_3$ will
set parent = W₂, and execute DoDFT. W₃ can self eliminate relative to W₂, so W₃ will set state = OUT. W₃ will send StartNewDFT(2) to W₂. W₂ will set nbsta(W₂) = OUT, and update Y₂ to a, b, i. W₂ will set parent = nil, and start DoDFT. W₂ cannot self eliminate, so it will send DFT(2) to W₁. Since tok = 2 is fresh to W₁, W₁ will set parent = W₂, and execute DoDFT. W₁ cannot self eliminate, so it will send DFT(2) to W₄. Since tok = 2 is fresh to W₄, W₄ will set parent = W₁, and execute DoDFT. W₄ cannot self eliminate, so it will send DFT(2) to W₂. Since tok = 2 is not fresh to W₂, W₂ will send report back to W₄. W₄ has no remaining agents, so W₄ will report back to W₁. W₁ has no remaining agents, so W₁ will report back to W₂. W₂ has no remaining agents, and W₂ has parent = nil. Therefore, W₂ will announce “no JT exists”.

2.5.6 Construction of JT Organization

HTBS was built to recognize the existence of the JT organization. After that, the work on [22] stated that the path of the StartNewDFT(tok) messages forms a JT. Therefore, if HTBS keeps track of every StartNewDFT(tok) message during executing, and connects the sender with the receiver of each of those messages, HTBS will construct a JT with no privacy loss. For instance, in the example presented in Section 2.5.5 for type 1 boundary graph, HTBS was able to recognize the exis-
tence of the JT agent organization. To construct the JT, HTBS will trace all the $StartNewDFT(tok)$ messages as follows:

The first message $StartNewDFT(1)$ was to initiate the execution of HTBS. The second message $StartNewDFT(2)$ is sent from $W_2$ to $W_1$. Therefore, $W_1$ and $W_2$ are connected in the JT. After that, $W_1$ will send the third message $StartNewDFT(3)$ to $W_4$. Hence, $W_1$ and $W_4$ are also connected in the JT. The fourth message $StartNewDFT(4)$ is sent from $W_4$ to $W_3$. As a result, $W_4$ and $W_3$ are connected in the JT. Finally, $W_3$ will announce the existence of the JT. Therefore, the agent $A_1$ will be connected with $A_2$ and $A_4$. The agent $A_2$ is connected with $A_1$. The agent $A_3$ is connected with $A_4$. The agent $A_4$ is connected with $A_1$ and $A_3$.

As a result, HTBS has surpassed DPMST by recognizing the existence of a JT organization and constructing the JT if it exists without leaking any privacy [22].
Chapter 3

Methodology

This chapter will introduce the new algorithm suite DAER (Distributed Agent Environment Re-decomposition), and a new quantification scheme to support it. The DAER algorithm suite is introduced in Section 3.1. The new quantification scheme is introduced in Section 3.2. An example of DAER execution using the new quantification scheme that shows the soundness of DAER is presented in Section 3.3. Finally, Section 3.4 will show the complexity of DAER.

3.1 DAER Algorithm

When the given environment decomposition has a boundary graph of type 2 or 3, HTBS only recognizes the non-existence of the JT organization. However, in practical applications, the user still needs to construct the JT. The user may use existing algorithms such as Action-GDL and DCTE. Since Action-GDL and DCTE will leak agents privacy even when JT exists while HTBS does not as demonstrated in Section 2.4.1 and Section 2.4.2, a new method is developed based on an extension of HTBS. DAER (Distributed Agent Environment Re-decomposition) is introduced to revise the environment decomposition while minimizing privacy loss. Using DAER, users will be able to construct the JT agent organization for environment decomposition with type 2 and type 3 boundary graph with the agents privacy on mind.
3.1.1 The Idea Behind DAER

HTBS is the only known algorithm to construct a JT without leaking any private information. Therefore, DAER follows the same approach as HTBS. The HTBS algorithm suite recognize the (non-)existence of the JT organization by performing self-elimination on the boundaries of the multi-agent system. If HTBS was able to perform self-elimination on the MAS boundaries to a single remaining boundary, a JT organization exists and can be constructed. However, when HTBS fails to perform self-elimination to a single boundary, the system will consist of at least three active boundaries (after elimination), and none of those boundaries can self-eliminates.

At this stage, DAER will revise one of the remaining boundaries to allow another boundary to perform self-elimination. When this revision is done, at least one boundary will be able to perform self elimination. Therefore, if the number of the remaining active boundaries before the revision was \( n \), the number of remaining active boundaries after the revision is at most \( n - 1 \). Then if DAER revised the another boundary to allow one more self-elimination, the number of the remaining active boundary will be at most \( n - 2 \) after the second revision. After that, DAER will keep revising and reducing the number of remaining active boundaries at least one boundary at each stage. At some point, the boundaries will eliminated until there only remain a single empty boundary, which means the JT agent organization now exist.

For example, communication graph in Fig. 3.1 does not have a JT organization as demonstrated in Section 2.5.5. However, if the boundary \( W_1 \) is expanded to \( \{a, b, d, i\} \), both \( W_2 \) and \( W_4 \) can perform self-elimination, and a JT organization will exist. Moreover, if the boundary \( W_2 \) is expanded to \( \{a, b, i, d\} \), both \( W_1 \) and \( W_4 \) can perform self-elimination, and a JT organization will exist. Furthermore, if the boundary \( W_4 \) is expanded to \( \{a, d, i, b\} \), both \( W_1 \) and \( W_2 \) can perform self-elimination, and
a JT organization will exist.

![Communication Graph](image)

Figure 3.1: The communication graph on Fig.2.12 (a)

If a boundary is expanded, what kind of privacy loss is occurred? The first leak of private information is the identities and domains of the shared variables that have been newly added to this boundary. Since the boundary did not have the new shared variables before the expansion, the agent did not have access to those variables. The second leak of private information is the identities of the agents that contains the newly shared variables. Since the expanded agent will be connected with every other agent (if not already connected) that contains the newly shared variables, the identities of all those agents will be leaked to the expanded agent. The third leak of private information is bordering relations of the agents that contains the newly shared variables. The expanded agent, when connected with new agents via the newly shared variables, can infer the bordering relations of the newly connected agents.

For example, if the boundary $W_1$ in Fig. 3.1 is expanded to \{a, b, d, i\}, the identity and the domain of the shared variable $i$ will be leaked to the agent $A_1$. However, since $W_1$ is already connected with $W_2$ and $W_4$, there is no privacy loss on agent identities. Moreover, since $W_1$, $W_2$, and $W_4$ already share the variable $a$, the agent $A_1$ already knows about the border between $A_2$ and $A_4$. Hence, there is no privacy loss in bordering relations.

Therefore, in some cases, the privacy of agent identities and bordering relations
is preserved. If a boundary is expanded, there is privacy loss on shared variables identities and domains, and a potential privacy loss on agent identities and bordering relations.

Since DAER’s purpose is to revise the environment decomposition while keeping the privacy to minimum, a new privacy loss quantification scheme is introduced in Section 3.2. This scheme is better than the one introduced in [22] because its distributed nature, and allowing DAER to evaluate the importance of each piece of private information.

3.1.2 Algorithm Overview

When HTBS fails to perform self-elimination to a single boundary, the decomposition have to be revised. Since allowing HTBS to continue performing self-elimination to a single boundary is enough to ensure the existence of a JT organization, the purpose of the revision should be allowing one of the remaining non-eliminated agents to perform self-elimination. An agent $A_i$ can perform self elimination if there exists an adjacent agent $A_j$, where the boundary $W_i$ of the agent $A_i$ is a subset or equal to the border $I_{ij}$ between $A_i$ and $A_j$ ($W_i \subseteq I_{ij}$). DAER allows an agent $A_i$ to perform self-elimination by expanding a neighboring agent $A_j$ to the point that the boundary $W_i$ of $A_i$ becomes a subset or equal to the border between $A_i$ and $A_j$.

The main question becomes: Which boundary should be expanded? The simple answer is the expansion that incurs the minimum privacy loss. When an agent $A_i$ cannot perform the self-elimination relative to any neighboring agent, how can $A_i$ revise the decomposition to be able to self-eliminate? The agent $A_i$ can share its whole boundary with a neighboring agent $A_j$, which will guarantees $A_i$ to be able to self-eliminate relative to $A_j$. However, since $A_i$ and $A_j$ are already bordering agents,
$A_i$ and $A_j$ already shares some variables in their boundaries. Therefore, $A_i$ can share the variables in its boundaries that are not contained in the border $I_{ij} (W_i \setminus I_{ij})$, which will still guarantees $A_i$ to self eliminate relative to $A_j$. Using the new quantification scheme, each agent $A_i$ can estimate the privacy loss if it shares $W_i \setminus I_{ij}$ with every neighboring agent $A_j$ and finds the one incurring the minimum privacy loss.

DAER revise the decomposition by running HTBS, and when if HTBS announces no JT exists, DAER expands one of the boundaries with a function called Elimination-Expansion (EE), which will be explained later. The main stages of DAER are 1) an HTBS stage, and 2) Elimination-Expansion stage. Running these two stages sequentially guarantees the existence of the JT organization. Given an agent environment $(A, \Omega, W)$, where $A$ is the list of agents $(A_1, A_2, ..., A_n)$, $\Omega$ is the lists of variables controlled by each agent $(\Omega_1, \Omega_2, ..., \Omega_n)$, and $W$ is the list of boundaries $(W_1, W_2, ..., W_n)$. DAER, as in HTBS, takes the $(A, W)$ as the input and it executes as follows:

**Algorithm 1** $DAER(A, W)$

1. do
2. $hasJT = HTBS(A, W);$  
3. if $hasJT = true$, halt;
4. each active agent estimates the best expansion plan with one of its neighbors;
5. select one active agent $A_i$ with its boundary $W_i$,
   and its best expansion plan neighbor $A_j$ with its boundary $W_j$;
6. $A_i$ shares $W_i \setminus W_j$ with $A_j$,
   and it performs self-elimination; \textbackslash\textbackslash A_i$ becomes inactive;
7. $A_j$ expands its boundary $W_j = W_i \cup W_j$;
When the given environment decomposition already admits a JT organization, HTBS will recognize the existence and construct the JT in lines 2 and 3 of the algorithm. However, if the given decomposition needs to be revised to admit the JT organization, the elimination-expansion stage is executed in lines 4-7. After every stage of elimination-expansion, HTBS stage will check whether the revised environment admits a JT organization or not. Since each run of the elimination-expansion stage guarantees that at least one active agent will be able to perform self-elimination, HTBS is guaranteed to return true after a finite number of executions, and the algorithm will halt in line 3.

For example, the environment decomposition in Fig. 3.2(a) has a type 2 boundary graph as shown in Fig. 3.2(b), where the cycle \((a, b, c, d, a)\) is a cycle of size four without a chord. Therefore the environment decomposition does not admit a JT organization. DAER will revise the decomposition as follows:

First, the first HTBS stage will be executed, and no agent can perform self-
elimination, so the elimination-expansion stage is necessary. After that, each agent will plan its best expansion plan. For example, \( W_4 \) has two possible plans, where it can share the variable \( b \) with \( W_3 \) or share \( c \) with \( W_1 \). The estimated privacy loss of those two plans is calculated using the new quantification scheme in Section 3.2.

For now, it suffices to assume that the privacy loss depends on the number of shared variables breaking ties arbitrarily. Assume the best expansion plan is \( W_5 \) sharing \( e \) with \( W_2 \). The boundary \( W_2 \) will be expanded into \( a, d, f, e \) and \( W_5 \) will be eliminated.

After that, HTBS will be executed, but no agent can self-eliminates. In the second elimination-expansion stage the active agents will be \( \{A_1, A_2, A_3, A_4\} \) and their active boundaries will be \( \{\{a, b, f\}, \{a, d, f, e\}, \{c, d, e\}, \{b, c\}\} \) as shown in Fig. 3.3(a). Each agent will find its best expansion plan, and suppose the best plan is \( W_3 \) sharing the variable \( c \) with \( W_2 \). The boundary \( W_2 \) will be expanded to \( a, d, f, e, c \) and will be connected with \( W_4 \) through the variable \( c \).

HTBS will then be executed, but no agent can self-eliminates. In the third elimination-expansion stage the active agents will be \( \{A_1, A_2, A_4\} \) and their active boundaries will be \( \{\{a, b, f\}, \{a, f, c\}, \{b, c\}\} \) as shown in Fig. 3.3(b). Each agent will finds its best expansion plan, and suppose the best plan is \( W_1 \) sharing the variable \( b \) with \( W_2 \). The boundary \( W_2 \) will be expanded to \( a, d, f, e, c, b \) and \( W_4 \) will self-eliminates.

After that, the active agents will be \( \{A_2, A_4\} \) and their active boundaries will be \( \{\{b, c\}, \{b, c\}\} \) as shown in Fig. 3.3(c). HTBS will be executed and will be able to perform self-elimination to a single boundary. The environment at this stage has been revised to allow a JT agent organization. The final environment decomposition after the revision is as follows: \( \{A_1, A_2, A_3, A_4, A_5\} \) and their relative boundaries will be \( \{\{a, b, f\}, \{a, d, f, e, c, b\}, \{c, d, e\}, \{b, c\}, \{e, d, f\}\} \) as shown in Fig. 3.3(d).
3.1.3 Leader Agent

When the environment decomposition does not admit a JT organization, DAER will revise the decomposition by expanding one boundary of an agent in the given environment. However, when each agent estimate their best expansion plan, which agent decides what expansion to perform? When done in a centralized approach, recognizing the best plan is not a challenge. Moreover, having a coordinator agent as a leader agent will find the best plan right away. However, multi-agent systems are distributed naturally, so a centralized approach is not applicable. Also, having a coordinator agent will leak the information of all the agents to this coordinator.
agent as shown in Section 2.4.3. Therefore, using a coordinator agent is avoided. The question becomes which agent will be the leader of each stage?

The HTBS algorithm suite assumes that the algorithm is initiated externally, and the leading agent is also specified externally. DAER will also assume that the leader of the first HTBS stage is specified externally. The role of this agent is to lead the first stage of HTBS only. After the first stage, the leader of each new round is specified within the execution of DAER. Since any agent can become a leader for any stage, the lead is following algorithm demonstrate how the leader is selected for each stage. The leader agent is initiated by receiving a message \( m \) or by an event \( e \). When the agent \( A_i \) receive the message \( m \), or when the event \( e \) occurs, the agent \( A_i \) will respond as follows:

\begin{algorithm}
\caption{Lead\((m, e)\)}
\begin{algorithmic}[1]
\State if \( e \) is an external request to start the first round of HTBS, \begin{algorithmic}
\State initiate HTBS;
\State else if \( e \) is a declaration of “no JT exists” by \( A_i \), \begin{algorithmic}
\State initiate elimination-expansion stage;
\State else if \( m \) is a boundary expansion request from a neighboring agent, \begin{algorithmic}
\State expand the boundary \( W_i \) of the agent \( A_i \);
\State initiate HTBS;
\State else if \( e \) is a declaration of “JT exists” by \( A_i \), \begin{algorithmic}
\State initiate halting;
\end{algorithmic}
\end{algorithmic}
\end{algorithmic}
\end{algorithmic}
\end{algorithm}

The leader agent of the first round is specified externally. This agent will execute lines 1-2 of the algorithm. After that, if HTBS recognize the existence of the JT, the agent who declares the existence of the JT organization will be the leader, and
initiate the halting process in lines 8-9. After that, the JT is found by tracing back the messages of \textit{StartNewDFT\textsubscript{tok}} as shown in Section 2.5.6. However, if HTBS recognize the non-existence of the JT organization, the agent that declares the non-existence becomes the leader of the next elimination-expansion stage as shown in lines 3-4. When the elimination-expansion concludes with the best expansion plan, the agent whose boundary will expand will receive a request to expand its boundary. When this agent receive the request to expand, it will expand its boundary then becomes the leader of the next HTBS stage as shown in lines 5-7. This algorithm allows the leader agent to be specified internally for every stage except the first HTBS stage, which is specified externally.

\subsection{Elimination-Expansion Stage}

The elimination-expansion stage is demonstrated in Fig. 3.2 and Fig. 3.3. However the example explained in Section 3.1.2 does not show how the elimination-expansion stage is executed distributively. The expansion-elimination stage executes lines 5-7 from Algorithm 1. At the start the elimination-expansion stage, each active agent knows its active boundary and its active neighbors. Only the active agents participate in the elimination-expansion stage since the purpose of DAER is to allow all agents to perform the self-elimination. Hence, the agents eliminated in the HTBS stage (inactive agents) will not affect or be affected by the expansion. However, the eliminated agents will be considered when calculating the privacy loss of each possible expansion because their identities and border relations might be leaked.

The elimination-expansion stage can be broken down into two sub stages. The first sub stage is the bidding stage, where the best expansion plan is found. The second sub stage is the expansion stage, where the expansion happens.
Bidding During Elimination-Expansion Stage

Each agent, using the new quantification scheme, can estimate the best expansion plan allowing its self-elimination. The new quantification scheme will be introduced in Section 3.2. Each agent estimates its best expansion plan in line 4 of Algorithm 1. Using the procedure GetBestExpPlan, which can be executed internally within each agent, each agent will know the expansion plan that allows it to self-eliminate while incurring the lowest privacy loss. More details on the procedure GetBestExpPlan will be presented in Section 3.2.

In the elimination-expansion stage, the expansion is decided based on multiple algorithms presented here. These algorithms are initiated by messages passed between active agents. A distributed depth-first-search (DFS) algorithm is included among these algorithms. The DFS is a well-known algorithm, but it’s presented here because it’s modified to fit the elimination-expansion purpose. The leader agent selected in Algorithm 2 in lines 3-4 will be the root of the DFS algorithm. DFS is executed as follows.

The agent executing the algorithm is referred to as $A_i$. The agent $A_c$ represent the sender of the message. The flag visited returns true if the agent $A_i$ has been visited in the current DFS, and false otherwise (initialized to false). The visited($A_k$) returns true if the neighboring agent $A_k$ has been visited, and false otherwise (initialized to false). The variable parent refers to the agent’s parent in the DFS tree. The variable myScore refer to the best expansion plan score, which can be calculated using GetBestExpPlan. The variable bestScore refers to the score of the best expansion plan reached to this agent. The variable bestScore($A_k$) refers to the best expansion score reported from the DFS child $A_k$.  

55
Algorithm 3 *DFS*

1. $A_i$ gets its best expansion plan: $myScore = GetBestExpPlan$;
2. if $A_i$ is the leader,  \select in Algorithm 2
3.  

\[\text{parent} = \text{nil}; \quad \text{visited} = \text{true};\]
4. run $ForwardDFS(myScore)$;
5. return;
6. get inScore from message;
7. $visited(A_c) = \text{true}$;
8. if $visited = \text{true}$,  \select $A_i$ is visited
9. label $A_c$ as non-adjacent in DFS;
10. send $NonChildReport$ to $A_c$;
11. else
12. $parent = A_c$;
13. $bestScore = \min(myScore, inScore)$;
14. run $ForwardDFS(bestScore)$;

When the agent $A_i$ execute the DFS algorithm, $A_i$ will first acquire the best expansion plan that allows it to self-eliminate. After that, if $A_i$ is the leader agent in this DFS round, specified by Algorithm 2 lines 3-4, it will appoint itself as the root of the DFS tree, and will try to expand the DFS by calling $ForwardDFS(bestScore)$ with it's own score as the best score. Hence, $A_i$ will execute lines 1-5.

However, if $A_i$ is not the leading agent, it will first mark the sender agent as visited, which means the sender agent is already in the DFS tree. After that, if $A_i$ is already a part of the DFS tree, it will notify the sender agent by sending a
NonChildReport. If $A_i$ is not a in the DFS, it will add itself in the DFS with the sender agent $A_c$ as the DFS tree-parent. $A_i$ will then calculate its best expansion plan and save it as myScore. $A_i$ will then try to expand the DFS tree by calling ForwardDFS(bestScore), where best score is the minimum between its own score (myScore) and the best score passed through the DFS tree (bestScore). When an agent tries to expand the DFS by running ForwardDFS(bestScore), it will be executed as follows:

**Algorithm 4 ForwardDFS(bestScore)**

1. if exists a neighboring agent $A_k$ such that visited($A_k$) = false, 
2. send DFS(bestScore) to $A_k$; 
3. visited($A_k$) = true; 
4. else 
5. if parent ≠ nil, 
6. send ChildReport(bestScore) to parent; 
7. else run Notify(bestScore);

When the agent $A_i$ executes ForwardDFS, $A_i$ will first look for an unvisited agent $A_k$ to add to the DFS tree. However, if there exist no neighboring agent that is unvisited, $A_i$ will report back to its DFS tree-parent the best score reached to $A_i$ by using ChildReport(bestScore). If $A_i$ is the root of the DFS tree, it will notify the agent with then best expansion plan to perform the expansion by using the Notify(bestScore). On the other hand, if $A_i$ finds an unvisited agent, it will try to add it to the DFS and pass the best score to it by sending DFS(bestScore) to that agent.
When the agent $A_i$ tries to add a new neighboring agent to the DFS tree by sending $DFS(bestScore)$, sometimes the neighboring agent is already in the DFS, and will report back with a *NonChildReport* (Algorithm 3, lines 8-10). In this case, $A_i$ will respond to the *NonChildReport* as follows:

**Algorithm 5** *RespondNonChildReport*

1. label $A_c$ as non-adjacent in the DFS tree;
2. run $ForwardDFS(bestScore)$;

The agent $A_i$ will recognize that the sender agent is non-adjacent in the DFS tree. Therefore, $A_i$ will run $ForwardDFS(bestScore)$ again to look for another unvisited agent to add to the DFS tree. However, if the sender reported with a *ChildReport* instead, $A_i$ will update the best score associated with the sender to the received score ($bestScore(A_c) = inScore$). After that, $A_i$ will look if it has any other unvisited neighbors to add to the DFS tree by running $ForwardDFS$ again. The response to *ChildReport* is as follows:

**Algorithm 6** *RespondChildReport*

1. label $A_c$ as a DFS tree child;
2. retrieve $inScore$ from message and assign: $bestScore(A_c) = inScore$;
3. $bestScore = min(bestScore, inScore)$;
4. run $ForwardDFS(bestScore)$;

By running the Algorithm 3 to Algorithm 6, all the agents will be added to the DFS tree, and the best score will be reported back to the DFS tree root. The only type of information exchanged between the agents is the score of the best expansion.
plan. Hence, there is no privacy loss on agent identities, bordering relations, shared variables, and private variables during the bidding stage.

Expansion During Elimination-Expansion Stage

When the root agent executes line 7 of Algorithm 4, the root notifies the agent with the best expansion score as follows:

**Algorithm 7 Notify(winningScore)**

1. if $myScore = winningScore$,
2. run $ShareVariable$;
3. else,
4. select the DFS tree-child ($A_k$) that has
   the winning score ($bestScore(A_c) = winningScore$);
5. send Notify(winningScore) to $A_k$;

When the agent $A_i$ executes Algorithm 7, it first checks if its own score ($myScore$) is the winning score of the DFS tree search. If $A_i$ has the best expansion plan ($myScore = winningScore$), the agent $A_i$ will perform the expansion associated with its best score by executing $ShareVariable$, which is introduced in Algorithm 8. However, if $A_i$ does not have the best expansion plan ($myScore \neq winningScore$), the agent $A_i$ will pass the winning score to its DFS tree-child that is associated with the winning score. It is guaranteed that $A_i$ will find a DFS tree-child associated with the winning score because of the following:

If the winning agent $A_i$ is the root of the DFS tree, $A_i$ will perform its best expansion without passing the notify message to a DFS tree-child. Therefore, $A_i$
does not need to have a DFS tree-child with the best score associated to it in order for this algorithm to execute correctly. However, if winning agent $A_i$ is not the DFS tree root, the best score of the winning is agent $A_i$ has been reported to the root through $ChildReport$. Therefore, the winning score is associated to $A_i$ by its DFS tree-parent. Moreover, $A_i$’s DFS tree-parent will report the best score of $A_i$ to its own parent using $ChildReport$, and that parent will report this score to its own parent until it reaches the root. Hence, the path between the root and the winning agent $A_i$ is defined by tracing the $ChildReport$ messages that contains the winning score.

When the agent $A_i$ that has the winning score of the best expansion plan receive the notify message, $A_i$ will perform its best expansion plan. The best expansion plan is found in Algorithm 3 line 1 using the new quantification scheme, which will be introduced in Section 3.2. The best expansion plan is the expansion that allows $A_i$ to self-eliminate relative to a neighboring agent $A_k$ by sharing some shared variables.

**Algorithm 8 ShareVariable**

1. find $Q = W_i \setminus I_{ik}$;
2. Find $S_Q$ which is the set of agents sharing variables in $Q$ with $A_i$;
3. send $Expand(Q, S_Q)$ to $A_k$;

To perform the expansion, $A_i$ calculate the set of variables ($Q$) needed to be shared with the neighboring agent $A_k$ that allows self-elimination. After that, $A_i$ finds the list of neighboring agents that share some variables in $Q$ and assign it to $S_Q$. The list of agents sharing variables in $Q$ is needed because all the agents in $S_Q$ will be connected to $A_k$ after the expansion. When $A_k$ receives the expand message from $A_i$, it will expand its boundary and connect with all the agents in $S_Q$. This step
incurs privacy loss on agents’ privacy.

The privacy of the identities and domains of the variables in $Q$ will be leaked to $A_k$. Moreover, the identity of every agent in $S_Q$ that was not connected to $A_k$ will be leaked to $A_k$, and the identity of $A_k$ will be leaked to every agent in $S_Q$. The bordering relations between $A_i$ and every agent in $S_Q$ will be leaked to $A_k$. The border relation between $A_i$ and $A_k$ will also be leaked to all the agents in $S_Q$.

Since this step is where the privacy loss occur, finding the expansion plan that incur the least privacy loss is important in DAER. The best expansion plan for each agent is calculated in Algorithm 3, line 1, and will be elaborated in Section 3.2.

When the agent $A_i$ receive an expand message from the agent $A_c$, the agent $A_i$ will perform the expansion as follows:

**Algorithm 9** *RespondToExpand*

1. Let $S$ be the list of adjacent agents of $A_i$;
2. Let $T$ be the list of variables shared between $A_i$ and $A_c$;
3. for each agent $A_j$ where $A_j \neq A_c$,
   
   $T = T \setminus I_{ij}$;
4. retrieve $Q$ and $S_Q$ from the expand message;
5. $W_i = (W_i \cup Q) \setminus T$;
6. connect with every agent in $S_Q$ \(\setminus\) if not already connected;
7. $S = S \cup S_Q$;

When the agent $A_i$ receive an expand message, it will update its boundary $W_i$ to include the newly shared variables and exclude the variables that are uniquely shared with $A_c$. The agent $A_i$ will exclude the variables that are uniquely shared
with $A_c$ because the agent $A_c$ is the agent that initiated the expansion. The agent $A_c$ will initiate the expansion if and only if $A_c$ can perform self-elimination after the expansion. Therefore, $A_c$ will not be active in the next round. Hence, the agent $A_i$ will update its active boundary to exclude the variables shared with $A_c$. The uniquely shared variables between $A_i$ and $A_c$ is found in lines 2-4 in Algorithm 9 as the variable set $T$.

After finding $T$, the agent $A_i$ will then include the newly shared variables $Q$ to its active boundary and exclude $T$ as in line 6 of the algorithm. Moreover, $A_i$ will be connected with all the agents containing some or all the variables in $Q$ in line 7. Hence the list of adjacent agents of $A_i$ denoted to as $S$ will be expanded as in line 8.

When this expansion happens, $A_c$ (the sender of the expand message) will be able to self-eliminate, and the agent $A_i$ will have its active boundary updated. This is the last step of the elimination-expansion stage. After this step, the agent $A_i$ will start a new round of HTBS as the leader agent as the leader agent as shown in lines 5-7 in Algorithm 2. The HTBS will either be able to find a JT organization by performing self-elimination a single boundary, or will get to the point where no agent can self-eliminate. If no agent can perform the self-elimination, HTBS will initiate another round of elimination-expansion stage. The HTBS stage and the elimination-expansion stage will keep executing until HTBS announces that a JT exists.

### 3.2 New Privacy Loss Quantification Scheme

The DAER algorithm suite revise the environment decomposition to allow the construction of the JT agent organization. However, during this revision, some private information has to be leaked. In Section 3.1, agents calculate the estimated
privacy loss using the function $GetBestExpPlan$. This function is based on the new quantification scheme presented in this section.

The privacy loss quantification scheme presented in [22] has the issue that all types of privacy loss are treated equally. For instance, if a private variable identity is leaked from one agent to another, it counts the same as if an agent identity, a bordering relation, or a shared variable identity is leaked. However, in real-world applications that is not the case. Private information varies in their importance. Having a privacy loss in the most critical piece of information is not the same as losing an insignificant piece of information.

Another issue of the presented scheme in Section 2.3 is that the privacy loss is counted in a centralized manner. The scheme was presented to allow comparison between different algorithms in constructing the JT organization, so the centralized approach was sufficient. However, when privacy loss is needed to be counted using this scheme during execution, as in DAER, one of the agents must have access to everything to be able to calculate the privacy loss, which contradicts the purpose of preserving privacy. Furthermore, during the execution of DAER, agents should be able to calculate the privacy loss of sharing a variable with another agent without leaking its privacy to other agents. The scheme presented in [22] does not support internal privacy loss evaluation. To overcome this issue, a new privacy loss quantification scheme is presented in this section. The main purpose of this scheme is to allow differentiating between the types of privacy loss and enabling agents to calculate privacy loss internally.
3.2.1 Assigning Privacy Weight

Agent identities The identities of the agents are not always of the same importance. Some agents hold more significance than others, and the knowledge of their existence should not be revealed beyond its neighboring agents. On the other hand, some agents have their identities known to almost every other agent in the system, and the knowledge of their existence does not hold vital importance. To differentiate between the importance of different agents, a weight is assigned to each agent. This weight should only be known by the agent itself and its neighboring agents. However, a problem rises in who assign this weight, and based on what? In this thesis, it is assumed that the weight of agents is assigned by the user.

Let $W(A_i) \in [0, 1]$ be the weight of the agent $A_i$, where 1 is the highest value of importance, and 0 being not important at all. This weight can be accessed by the agent itself, and every neighboring agent of the agent $A_i$. Since agent identities are known by neighboring agents, the weight of the identities is also known to the neighboring agents. Hence, no privacy loss occurs on agent identities by accessing this weight.

For example, consider assigning the privacy of the agent identities on the MAS in Fig. 3.4. The Table 3.1, shows possible agent identities weights assigned randomly. Using the weights in this table, agents will consider the potential privacy loss on agent identities when calculating the best expansion plan. When this example was applied to the previous quantification scheme in Section 2.3, all agent identities were treated equally. However, in this case, the identity of the agent $A_1$ holds the most significance. Therefore, agents will try to avoid leaking the privacy of the agent $A_1$. The identity of the agent $A_2$ on the other hand, has the least value of significance. As a result, when an agent consider leaking either the identity of $A_1$ or $A_2$, leaking the identity
of the agent $A_2$ will be considered as a better option.

![Figure 3.4: A MAS with five agents](image)

Table 3.1: Weights of agent identities of agents in Fig. 3.4

<table>
<thead>
<tr>
<th>Agent</th>
<th>Weight</th>
<th>Agent</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$</td>
<td>0.730</td>
<td>$A_2$</td>
<td>0.100</td>
</tr>
<tr>
<td>$A_3$</td>
<td>0.410</td>
<td>$A_4$</td>
<td>0.407</td>
</tr>
<tr>
<td>$A_5$</td>
<td>0.208</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bordering relations** The privacy of the bordering relations also does not hold the same level importance. For example, the knowledge on the relation between two significant agents in the system holds more value than the knowledge on the relation between two of the less significant agents. Therefore, a weight is assigned to value each border. This weight should only be known by the two agents having this border. This thesis, same as in agent identities, also assumes that the weight of each border is assigned by the user of the multi-agent system.

Let $W(A_i, A_j) \in [0, 1]$ be the weight of the border between $A_i$ and $A_j$, where 1 is the highest value of importance, and 0 being not important at all. This weight is only known to the agents that are part of this relation. For example, the weight of
the relation between $A_1$ and $A_2$ in Fig. 3.4 is only known by the agents $A_1$ and $A_2$. Hence, there is no privacy loss on bordering relations when assigning and accessing the weight of the bordering relations.

For example, consider assigning the privacy of the bordering relations on the MAS in Fig. 3.4. The Table 3.2, shows possible bordering relations weights assigned randomly. Using the weights in this table, agents will consider the potential privacy loss on bordering relations when calculating the best expansion plan. When this example was applied to the previous quantification scheme in Section 2.3, all bordering relations were treated equally. However, in this case, the bordering relation between the agents $A_3$ and $A_4$ holds the most significance. Therefore, the agents $A_3$ and $A_4$ will try to avoid leaking the privacy of this bordering relation. The bordering relation between $A_1$ and $A_2$ on the other hand, has the least value of significance. As a result, DEAR will value the privacy of the bordering relation between $A_3$ and $A_4$ more than the bordering relation between $A_1$ and $A_2$.

**Private variables** The privacy of the private variables identities and domains should also be valued based on importance. Some private variables are crucial in the system, and their privacy hold significant importance. On the other hand, some private variables do not hold the same level of importance. Therefore, a weight is also assigned to the private variables identities and domains. This weight is assigned to the private variables by the agent that contains the private variables.

Let $W(x) \in [0,1]$ be the weight of the identity of private variable $x$, where 1 is
### Table 3.3: Weights of private variables identities and domains in Fig. 3.4

<table>
<thead>
<tr>
<th>Variable</th>
<th>ID weight</th>
<th>Domain weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>0.911</td>
<td>0.937</td>
</tr>
<tr>
<td>c</td>
<td>0.487</td>
<td>0.397</td>
</tr>
<tr>
<td>e</td>
<td>0.583</td>
<td>0.660</td>
</tr>
<tr>
<td>f</td>
<td>0.159</td>
<td>0.294</td>
</tr>
<tr>
<td>g</td>
<td>0.369</td>
<td>0.506</td>
</tr>
<tr>
<td>h</td>
<td>0.864</td>
<td>0.116</td>
</tr>
<tr>
<td>j</td>
<td>0.0423</td>
<td>0.157</td>
</tr>
<tr>
<td>k</td>
<td>0.762</td>
<td>0.378</td>
</tr>
</tbody>
</table>

the highest value of importance, and 0 being not important at all. Let \( W_d(x) \) be the weight of the domain of private variable \( x \), where 1 is the highest value of importance, and 0 being not important at all. These two weights are only known to the agent \( A_i \) that contains the variable \( x \) (\( x \in A_i \)). For example, the weight of the identity and the domain of private variable \( b \) in Fig. 3.4 is only known to the agent \( A_1 \). Hence, there is no privacy loss on private variables when assigning and accessing the weights of private variables.

For example, consider assigning the privacy of the private variables identities and domains on the MAS in Fig. 3.4. The Table 3.3, shows possible private variables identities and domains weights assigned randomly. When this example was applied to the previous quantification scheme in Section 2.3, all agent identities were treated equally. Since DAER does not leak any information in private variables, these weights are insignificant when using the function \( GetBestExpPlan \). However, these weights are used later in Chapter 4 when calculating the privacy loss in Action-GDL and DCTE.

**Shared Variables** The privacy of the shared variables identities and domains should also be valued based on importance. Some shared variables are crucial in
the system, and are only shared with very few agents. On the other hand, some shared variables do not hold the same level of importance. Therefore, a weight is also assigned to the shared variables identities and domains. This weight is assigned to the shared variables by the agents that contains the shared variables. Let $x$ be a shared variable that is shared by 3 different agents $A_i$, $A_j$, and $A_k$. If $A_i$, $A_j$ and $A_k$ assign different weights to same shared variable $x$, the estimate privacy loss when sharing the variable $x$ will be different when valued by $A_i$, $A_j$, and $A_k$. Moreover, allowing the agents $A_i$, $A_j$, and $A_k$ to communicate to agree on the weight of the shared variable $x$ will cause potential privacy loss and won’t be efficient. Therefore, the weights of the shared variables identities and domains are assumed to be agreed upon between all the agents sharing those variable.

Let $W(x) \in [0, 1]$ be the weight of the identity of the shared variable $x$, where 1 is the highest value of importance, and 0 being not important at all. Let $W_d(x)$ be the weight of the domain of the shared variable $x$, where 1 is the highest value of importance, and 0 being not important at all. These two weights are only known to the agent $A_i$ that contains the variable $x$ ($x \in A_i$). For example, the weight of the identity and the domain of the shared variable $a$ in Fig. 3.4 is only known to the agents $A_1$, $A_3$, and $A_4$. Hence, there is no privacy loss on the shared variables when assigning and accessing the weights of shared variables.

For example, consider assigning the privacy of the private variables identities and domains on the MAS in Fig. 3.4. The Table 3.4, shows possible shared variables identities and domains weights assigned randomly. When this example was applied to the previous quantification scheme in Section 2.3, all agent identities were treated equally. However, using this table, the identity of the shared variable $d$ will be valued most since it has the highest weight. On the other hand, the domain of
<table>
<thead>
<tr>
<th>Variable</th>
<th>ID weight</th>
<th>Domain weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0.554</td>
<td>0.947</td>
</tr>
<tr>
<td>d</td>
<td>0.914</td>
<td>0.348</td>
</tr>
<tr>
<td>i</td>
<td>0.539</td>
<td>0.771</td>
</tr>
</tbody>
</table>

Table 3.4: Weights of shared variables identities and domains in Fig. 3.4

the agent $a$ is more important than the domain of the variable $d$. Therefore, both weights are considered to value the estimated privacy loss when using the function $\text{GetBestExpPlan}$.

3.2.2 Privacy Loss on Agent Identity

When an agent shares a shared variables to a new agent, the identities of all the agents sharing this variable previously will be known to the new agent as shown in Section 3.1. Therefore, the privacy of the agent identities is consider when calculating the estimate privacy loss of each expansion plan. For example, if the agents $A_i$ and $A_j$ shares the variable $x$, and the $A_i$ is estimating the privacy loss of sharing $x$ with another agent $A_k$. If $A_i$ shares the variable $x$ with $A_k$, the agents $A_j$ and $A_k$ will be connected. Hence, the identity of the agent $A_j$ will be leaked to $A_k$, and the identity of $A_k$ will be leaked to $A_j$. However, if the agents $A_j$ and $A_k$ are connected prior this expansion, no privacy loss occurs regarding their identities.

Let $ncv(A_j, A_k)$ be a function that returns 1 if there are no common variables between $A_j$ and $A_k$, and it returns 0 otherwise. Using the same example, when $A_i$ shares the variable $x$, which is shared between $A_i$ and $A_j$, to $A_k$, the privacy leak on the identities of $A_j$ and $A_k$ will only occur when $ncv(A_j, A_k)$ returns 1 ($A_j$ and $A_k$ are not connected prior the expansion). However, the agent $A_i$ cannot tell whether or not $A_j$ and $A_k$ are connected or not unless the three agents are sharing some variables.
together. Let \( ncv(A_i, A_j, A_k) \) be a function that returns 1 if there are no common variables across the three agents \( A_i, A_j, \) and \( A_k, \) and returns 0 otherwise. If the function \( ncv(A_i, A_j, A_k) \) returns 0, the agent \( A_i \) already knows that the agents \( A_j \) and \( A_k \) are already connected since they all share some variables. Hence, whenever \( ncv(A_i, A_j, A_k) \) returns 0, \( ncv(A_j, A_k) \) will return 0 as well. However, if the function \( ncv(A_i, A_j, A_k) \) returns 1, the agent \( A_i \) cannot determine whether \( A_j \) and \( A_k \) are connected or not.

When agent \( A_i \) shares a set of shared variables \( S \) with a neighboring agent \( A_j, \) the privacy loss is calculated as follows. Let \( A_S \) be an agent that contains at least one variable in \( S, \) and \( ncv(A_i, A_S, A_j) = 1. \) The following equation calculate the actual privacy loss on agent identities:

\[
Loss_{id} = \sum_{A_S} ncv(A_S, A_j) \ast (W(A_S) + W(A_j))
\]

Hence, the maximum privacy loss occur if for every agent \( A_S, \) the function \( ncv(A_S, A_j) \) returns 1. The maximum privacy loss is calculated as follows:

\[
MaxLoss_{id} = \sum_{A_S} (W(A_S) + W(A_j))
\]

The minimum privacy loss occurs when for every agent \( A_S, \) the function \( ncv(A_S, A_j) \) returns 0. Hence, the actual privacy loss is between 0 and the maximum privacy loss.

\[
Loss_{id} \in [0, MaxLoss_{id}]
\]

Since the agent \( A_i \) cannot determine the value of \( ncv(A_S, A_j), \) the agent \( A_i \) will always assume the maximum privacy loss will occur.
3.2.3 Privacy Loss on Bordering Relations

The privacy of the bordering relations is not directly leaked during the expansion stage, but it can be inferred. For example, let the variable $x$ be a shared variable between the agents $A_i$ and $A_j$. Suppose the agent $A_i$ decide to share the variable $x$ with another agent $A_k$. The identity and domain of the the variable $x$ will directly be leaked to $A_k$. The identities of the agents $A_j$ and $A_k$ might also be leaked as shown in Section 3.2.2. When the agent $A_i$ share the variable $x$ with $A_k$, $A_i$ will self-eliminate and the agents $A_j$ and $A_k$ will be connected. Hence, the agent $A_j$ can infer that the agents $A_i$ and $A_k$ have a bordering relation, and the agent $A_k$ can infer that $A_i$ and $A_j$ have a bordering relation as well. However, if the three agents are connected prior the expansion ($ncv(A_i, A_j, A_k) = 0$), no privacy loss on agent bordering relations occur.

To calculate the privacy loss when the agent $A_i$ shares a set of shared variables $S$ with a neighboring agent $A_j$, the following function is used:

$$\text{Loss}_{br} = \sum_{A_S} (W(A_i, A_S) + W(A_i, A_j))$$

$A_S$ is an agent that contains at least one variable in $S$.

3.2.4 Privacy Loss on Variables Identities and Domains

The privacy loss on private variables can be calculated directly by the agent containing those variables. Since each agent has access to the weights its own private variables, the privacy loss when a private variable identity or domain is leaked, is the weight of the private variable identity or domain. However, since DAER does not leak any private information regarding the private variables, the weights of private variables identities and domains are insignificant to DAER.

However, the privacy loss on shared variables is considered in DAER. The privacy
loss on the shared variables identities and domains can be calculated within every agent containing those private variables. Since the weight of the shared variables identities and domain are agreed upon as mentioned in Section 3.2.1, the estimated privacy loss on the shared variables identities and domains will have the same value among all the agents.

Since every shared variable are contained at least in 2 different boundaries, taking only the weight of the shared variable is not enough to evaluate the privacy loss. The number of agents containing the shared variables should be considered. The privacy loss could be calculated by multiplying the number of agents $N$ sharing the private variable $s$ with the weight of $s$ ($N \times W(s)$).

However, this simplification leads to an unwanted behavior. For example, if the variable $s$ is shared between 4 different agents ($A_1, A_2, A_3, A_4$), the privacy loss would be $4 \times W(s)$. After that, say the variable $s$ is shared with another agent $A_5$. It will incur $4 \times W(s)$ privacy loss units. After that, when another expansion is needed and one of the agents wants to estimate the privacy loss on sharing the variable $s$ to yet another agent $A_6$, The privacy loss will be calculated as $5 \times W(s)$, which does not make sense. If the privacy loss of sharing the variable to the agent $A_5$ in the first round was $4 \times W(s)$, why would it be counted as $5 \times W(s)$ when sharing with $A_6$? Both agent $A_5$ and $A_6$ did not have the variable initially, and should assume equal value of privacy loss when getting a new shared variable.

To avoid this issue, we assume that $W(s)$ and $W_d(s)$ has the number of agents sharing $s$ initially already calculated. Therefore, to calculate the privacy loss on shared variables identities when sharing a set of shared variables $SH$, the following
equation is used:

\[ \text{Loss}_s = \sum_{s \in SH} W(s) \]

Respectively, the privacy loss on shared variables domains will be:

\[ \text{Loss}_{svd} = \sum_{s \in SH} W_d(s) \]

The following section contains a detailed example of how DAER works. The calculation of the estimated privacy loss while calculating \textit{GetBestExpPlan} is also shown in the following section.

### 3.3 Demonstration of Soundness

![Environment decomposition with type 2 boundary graph](image)

Figure 3.5: Environment decomposition with type 2 boundary graph

To demonstrate the soundness of DAER, the following example is presented. This example consists of five agents, where no agent can perform self-elimination. The environment decomposition in Fig. 3.5 has a boundary graph of type 2. DAER will revise the environment decomposition to enable the JT agent organization. The
weights are shown in the following tables, where the shared variables table have the initial weight of the shared variable identity and domain, and W(s), which take into consideration both values and the number of agents sharing the variable as mentioned earlier:

<table>
<thead>
<tr>
<th>Variable</th>
<th>ID weight</th>
<th>Domain weight</th>
<th>W(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0.462</td>
<td>0.920</td>
<td>1.383</td>
</tr>
<tr>
<td>b</td>
<td>0.665</td>
<td>0.248</td>
<td>0.913</td>
</tr>
<tr>
<td>c</td>
<td>0.325</td>
<td>0.884</td>
<td>1.209</td>
</tr>
<tr>
<td>d</td>
<td>0.024</td>
<td>0.439</td>
<td>0.464</td>
</tr>
<tr>
<td>x</td>
<td>0.732</td>
<td>0.886</td>
<td>1.618</td>
</tr>
<tr>
<td>y</td>
<td>0.665</td>
<td>0.882</td>
<td>1.546</td>
</tr>
<tr>
<td>z</td>
<td>0.839</td>
<td>0.219</td>
<td>1.058</td>
</tr>
</tbody>
</table>

Table 3.5: Weights of shared variables identities and domains in Fig. 3.5

<table>
<thead>
<tr>
<th>Agent</th>
<th>Weight</th>
<th>Agent</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>A₁</td>
<td>0.177</td>
<td>A₂</td>
<td>0.088</td>
</tr>
<tr>
<td>A₃</td>
<td>0.612</td>
<td>A₄</td>
<td>0.489</td>
</tr>
<tr>
<td>A₅</td>
<td>0.731</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.6: Weights of agent identities of agents in Fig. 3.5

<table>
<thead>
<tr>
<th>Border</th>
<th>Weight</th>
<th>Border</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>A₁ − A₂</td>
<td>0.428</td>
<td>A₁ − A₃</td>
<td>0.698</td>
</tr>
<tr>
<td>A₁ − A₄</td>
<td>0.753</td>
<td>A₁ − A₅</td>
<td>0.945</td>
</tr>
<tr>
<td>A₂ − A₄</td>
<td>0.346</td>
<td>A₂ − A₅</td>
<td>0.462</td>
</tr>
<tr>
<td>A₃ − A₄</td>
<td>0.760</td>
<td>A₃ − A₅</td>
<td>0.826</td>
</tr>
</tbody>
</table>

Table 3.7: Weights of the borders between agents in Fig. 3.5

The first stage of DAER will be to run the HTBS stage for the first time. The leader of this stage is selected externally as shown in Section 3.1.3. Suppose the
request to start the first stage $\text{StartNewDFT}(1)$ was sent to $A_5$, which makes it the leader of the first stage.

$A_5$ will start the HTBS stage with $\text{tok} = 1$ and tries to perform self-elimination, but it will not be able to. The HTBS stage will continue executing by passing $DFT(\text{tok})$ between the agents trying to find an agent that can perform self-elimination. No agent can perform self-elimination at this stage, so the agent $A_5$ will recognize the non-existence of the JT organization and will start the Elimination-Expansion ($EE$) stage as the leader.

When the $EE$ stage starts, the bidding sub stage will begin to find the best expansion plan. $A_5$ will estimates its possible expansion plans, and will find the best expansion plan with the lowest estimated privacy loss as follows:

The first possible expansion to enable the agent $A_5$ to perform self-elimination is to share the variable $a$ with the agent $A_1$. The weigh of the variable $a$ is $W(a) = 1.383$. Since $A_5$ only shares the variable $a$ with the agent $A_2$ and $ncv(A_5, A_2, A_2) = 0$, there is no privacy loss on agent identities and bordering relations. Hence, the evaluated privacy loss of sharing $a$ with $A_1$ is $Loss = 1.383$.

The second possible expansion is to share the variables $a, z$ with the agent $A_3$. The weight of the variables $a, z$ are $W(a) = 1.383, W(z) = 1.058$. The agent $A_5$ shares $z$ with the agent $A_1$ and $A_2$, and shares the variable $a$ with $A_2$. Therefore, the privacy loss on agent identities and bordering relations with those two agents must be considered. Since $ncv(A_5, A_1, A_3) = 0$, there is no privacy loss in the identity and the bordering relation with $A_1$. However, since $ncv(A_5, A_2, A_3) = 1$, the agent $A_5$ will assume that $A_2$ and $A_3$ are not connected, which is accurate in this instance. The identity weight of $A_2$ is 0.088, which will be leaked to $A_3$. The identity weight of $A_3$ is 0.612, which will be leaked to $A_2$. The bordering relation weight between $A_5$ and $A_2$
is 0.462, which will be leaked to $A_3$. The bordering relation weight between $A_5$ and $A_3$ is 0.826, which will be leaked to $A_2$. The estimated privacy loss of this expansion plan is $Loss = 1.383 + 1.058 + 0.088 + 0.612 + 0.462 + 0.826 = 4.429$.

The agent $A_5$ has no more possible expansions. The best expansion plan for $A_5$ is the first expansion, which incurs $Loss = 1.383$. Therefore, $A_5$ will set $myScore = bestScore = 1.383$ and forwards $DFS(1.383)$ to the agent $A_1$. The agent $A_1$ will receive the DFS message, and assign $A_5$ as its DFS tree parent. The agent $A_1$ then calculate the best expansion plan of its own as follows:

The first possible expansion of $A_1$ is to share the variables $c, d$ with the agent $A_5$. This expansion will incur privacy loss on the shared variables $c, d$, agent identities of the agents $A_4$ and $A_5$, and bordering relations of $A_1 - A_4$ and $A_1 - A_5$. The estimated privacy loss of this expansion is $Loss = 1.209 + 0.464 + 0.489 + 0.731 + 0.753 + 0.945 = 4.591$.

The second possible expansion is to share the variables $c, d, x$ with the agent $A_2$. This expansion will incur privacy loss on the shared variables $c, d, x$, agent identities of the agents $A_3$ and $A_2$, and bordering relations of $A_1 - A_3$ and $A_1 - A_2$. However, since $A_1$ shares the variable $c$ with the agent $A_4$, and $ncv(A_1, A_4, A_2) = 1$, the agent $A_1$ will assume that $A_4$ and $A_2$ are not connected, which is not accurate in this instance. Therefore, $A_1$ will also estimate the loss of $A_4$ and $A_1 - A_4$ to $A_2$, and $A_2$ and $A_1 - A_2$ to $A_4$. Therefore, the estimated privacy loss is $Loss = 1.209 + 0.464 + 1.618 + 0.612 + 0.088 + 0.698 + 0.428 + 0.489 + 0.088 + 0.753 + 0.428 = 6.875$.

The last next possible expansion for $A_1$ is to share $z$ with $A_3$. This expansion will incur $Loss = 2.884$. The last possible expansion for $A_1$ is to share the variables $d, x, z$ with the agent $A_4$, which incurs $Loss = 7.816$. The best possible expansion for $A_1$ is sharing $z$ with $A_3$, which incurs $Loss = 2.884$. The agent $A_1$ will assign $myScore =$
2.884. However, since $bestScore < myScore$, $A_1$ will not update $bestScore$. The agent $A_1$ will forward $DFS(1.383)$ to the agent $A_2$.

The agent $A_2$ will receive the DFS message, and assign $A_1$ as its DFS tree parent, and will estimate its best expansion plan. The possible expansion plans for $A_2$ are as follows: 1) sharing $y$ with $A_5$, which will incur $Loss = 3.573$ 2) sharing $a, y$ with $A_1$, which will incur $Loss = 4.368$ 3) sharing $a, z$ with $A_4$, which will incur $Loss = 5.907$. The best expansion plan for $A_2$ is sharing $y$ with $A_5$, so $A_2$ will assign $myScore = 3.573$. After that, $A_2$ will forward forward $DFS(1.383)$ to the agent $A_5$. However, $A_5$ will report back with a $NonChildReport$, so it will forward $DFS(1.383)$ to the agent $A_4$.

The agent $A_4$ will receive the DFS message, and assign $A_2$ as its DFS tree parent, and will estimate its best expansion plan. The possible expansion plans for $A_4$ is to share the variable $y$ with $A_3$, which incurs $Loss = 3.352$. The agent $A_4$ will and assign $myScore = 3.352$, and tries to forward $DFS(1.383)$ to $A_1$, but $A_1$ will reply with a $NonChildReport$, so it will forward $DFS(1.383)$ to $A_3$.

The agent $A_3$ will receive the DFS message, and assign $A_4$ as its DFS tree parent, and will estimate its best expansion plan. The possible expansion plans for $A_3$ is to share the variable $b$ with $A_1$, which incurs $Loss = 0.913$. The agent $A_3$ will and assign $myScore = bestScore = 0.913$, and tries to forward $DFS(0.913)$ to $A_1$ and $A_5$, but both of them will reply with a $NonChildReport$.

Since $A_3$ has no more neighbors outside the DFS tree, agent $A_3$ will send the message $ChildReport(0.913)$ to $A_4$. The agent $A_4$ will assign $bestScore(A_3) = 0.913$, and send a $ChildReport(0.913)$ to $A_2$. The agent $A_2$ will assign $bestScore(A_4) = 0.913$, and send a $ChildReport(0.913)$ to $A_1$. The agent $A_1$ will assign $bestScore(A_2) = 0.913$, and send a $ChildReport(0.913)$ to $A_5$. The agent $A_5$ will assign $bestScore(A_1) =$
0.913, and update \textit{bestScore} = 0.913. Since, \( A_5 \) is the root, it will check if \textit{myScore} = \textit{bestScore}, but it’s not, so it will start the expansion stage to notify the winner.

The agent \( A_5 \) will send \textit{Notify}(0.913) to \( A_1 \). Since \( A_1 \)’s \textit{myScore} is not the winning score, it will send \textit{Notify}(0.913) to \( A_2 \). The agent \( A_2 \) will send \textit{Notify}(0.913) to \( A_4 \), and the agent \( A_4 \) will send \textit{Notify}(0.913) to \( A_3 \). Agent \( A_3 \) recognize that it has the winning score. Therefore, \( A_3 \) will share the variable \( b \) with the agent \( A_1 \), perform self-elimination, and send \textit{Expand}(\( b \)) to \( A_1 \) as shown in Fig. 3.6.

![Figure 3.6: The active boundaries of Fig. 3.5 after the first expansion](image)

The agent \( A_1 \) will expand it’s boundary then update it with the remaining active boundaries to be \( W_1 = b, c, x, z \) as shown in Fig. 3.6. Agent \( A_1 \) is now the leader of the next round starting with the HTBS stage with \( tok = 2 \). The HTBS stage will recognize the non-existence of the JT, and will start the elimination-expansion stage with \( A_1 \) as the leader.

The best expansion plan in this elimination-expansion stage will be the agent \( A_5 \) sharing the variable \( a \) with \( A_1 \), which will incur \textit{Loss} = 1.383. The agent \( A_5 \) will perform self elimination, and \( A_1 \) will update it’s active boundary as shown in
Fig. 3.7(a). The agent $A_1$ will be the leader of the next round with $tok = 3$. The best expansion plan of the third round will be the agent $A_4$ sharing the variable $y$ with $A_1$, which will incur $Loss = 2.910$. The agent $A_4$ will perform self elimination, and $A_1$ will update its boundary as shown in Fig. 3.7(b). The agent $A_1$ will be the leader of the next round with $tok = 4$.

![Diagram of active boundaries](image)

Figure 3.7: The active boundaries of Fig. 3.5 after the second expansion in (a), and after the third expansion in (b)

On the fourth round, the HTBS stage will recognize the existence of the JT. Hence, the environment decomposition now admits to the JT agent organization, and has been revised to have a type 1 boundary graph. The total privacy loss is the summation of privacy loss throughout the rounds: $Loss_{total} = 0.913 + 1.383 + 2.910 = 5.206$.

The boundary sets after the expansions is as follows. The boundaries $W_2, W_3, W_4, W_5$ remained the same, but the boundary $W_1$ has been expanded to $W_1 = a, b, c, d, x, y, z$. The revised environment decomposition is shown in Fig. 3.8(a), and the resultant JT is shown in Fig. 3.8(b).
3.4 Complexity

Let $\eta$ be the initial number of boundaries, and $e$ be the initial number of borders. The newly generated borders during the elimination-expansion stage are not considered since they are not significant. DAER algorithm suite consists of two main stages, which is the HTBS and the elimination-expansion stages. The HTBS stage consists of a depth-first-search (DFS) over the boundaries of the communication graph. The DFS works over the remaining active boundaries of the communication graph, which
has $e$ borders at most. Hence, the number of messages passed during the HTBS stage is $O(e)$. After that, the elimination-expansion stage runs DFS with the number of messages being $O(e)$ as well. In each elimination-expansion stage, at least one boundary is eliminated. Hence, the maximum number of DAER iterations is $\eta$. The complexity of the DAER algorithm suite is $O(e)$ for each stage, and these stages incur $\eta$ times at most. Therefore, the overall complexity of DAER is $O(e \eta)$. 
Chapter 4

Experimental Evaluation

This chapter presents the experimental results of DAER execution. The experimental setup used in experiments are explained in Section 4.1. In Section 4.2 the results of DAER execution compared to Action-GDL and DCTE is shown. The results in Section 4.2 present the comparison of DAER, Action-GDL, and DCTE in terms of three aspects: privacy loss, runtime, and the number of messages.

4.1 Experimental Setup

The proposed algorithm is evaluated using randomly generated boundary sets. The experiments of this study is supported by the WEBWEAVR-IV toolkit [17]. The boundary graph of the generated boundary sets can fall into one of the three types explained in Section 2.5.1. Type 1 boundary graph admits the JT agent organization, and can be constructed without any privacy loss using the HTBS algorithm. Hence, DAER will construct the JT agent organization without any privacy loss on the HTBS stage, and will not execute the elimination-expansion stage. Therefore, type 1 boundary graph are excluded from the experiments data. The execution of the HTBS algorithm on type 1 boundary graphs have been experimented throughly in [22].

Type 2 and type 3 boundary graph do not admit the JT agent organization, so the elimination-expansion stage is necessary when DAER constructs the JT. DAER
algorithm was run on 12 different randomly generated boundary sets containing 6 sets for each type, and the type has been tracked throughout the execution rounds of DAER. Each round of execution contains the HTBS stage and the elimination-expansion stage. It turns out that type 2 boundary sets can turn into type 3 boundary set during the execution, and type 3 can turn into type 2 as well. The results of the 12 examples can be seen on Table 4.1.

Table 4.1: Boundary graphs type track through the execution of DAER, where each row is a boundary set and each column is a round of execution

<table>
<thead>
<tr>
<th></th>
<th>BS-1:</th>
<th>BS-2:</th>
<th>BS-3:</th>
<th>BS-4:</th>
<th>BS-5:</th>
<th>BS-6:</th>
<th>BS-7:</th>
<th>BS-8:</th>
<th>BS-9:</th>
<th>BS-10:</th>
<th>BS-11:</th>
<th>BS-12:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

From Table 4.1 it can be seen that type 2 boundary graphs are more likely to turn into type 3 boundary graphs than the other way round. Therefore, the experimental setup used in the evaluation are all of type 2 boundary graph. The batches generated for testing are based on three main parameters.

The first parameter is the number of agents in the generated environment decomposition. The maximum border size is the second parameter. The third parameter is the ratio between the number of private variables to the number of the shared variables. The number of agents has been divided into 4 possible values, where the
generated environment can have the number of agents between 10-50, 51-100, 101-150, or 151-200 agents in the given environment decomposition. The maximum border size can be between 1-5, 6-10, or 11-15 variables in the maximum border. The ratio between private to shared variables can be 1:1, 2:1, or 3:1.

Using the three mentioned parameters, 36 batches were generated, where each batch has 30 environment decompositions. The total number of environment decompositions used in the experiments is $36 \times 30 = 1080$ environment decompositions. The result of the experiments will be shown using 9 charts, where each chart has 120 environment decompositions indexed based on the number of agents as shown in Table 4.2.

<table>
<thead>
<tr>
<th>Index</th>
<th>Number of agents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-30</td>
<td>10-50</td>
</tr>
<tr>
<td>31-60</td>
<td>51-100</td>
</tr>
<tr>
<td>61-90</td>
<td>101-150</td>
</tr>
<tr>
<td>91-120</td>
<td>151-200</td>
</tr>
</tbody>
</table>

Table 4.2: Indexes of the experimental environment decompositions

### 4.2 Experimental Results

Using the 1080 environment decompositions, DAER is evaluated in comparison with the Action-GDL and DCTE algorithms. In all of the 1080 environment decompositions, DAER was able to revise the environment decomposition and construct the JT agent organization, which shows the soundness of DAER. The results are shown in three main sections. The first section shows the privacy loss when using DAER, Action-GDL, and DCTE to construct the JT agent organization. The runtime in milliseconds is shown in the second section. The last section shows the number of
messages exchanged between agents during the construction of the JT agent organization.

**Privacy Loss** The figures from Fig. 4.1 to Fig. 4.9 show the privacy loss occurred, as calculated by the privacy loss quantification scheme in Section 3.2. Each figure contains 120 environment decompositions indexed based on the number of agents as shown in Table 4.2. For simplicity, privacy loss will be referred to as “PL”, the maximum border size will be referred to as “MBS”, and private variables to shared variables ratio will be referred to as “ratio”. The mean ($\mu$) and the standard deviation ($\sigma$) is shown in the caption of each figure for the three algorithms.

Fig. 4.1 to Fig. 4.9 show that DAER constantly incurs less privacy loss than the other two algorithms throughout all batches. All of the three algorithms tend to incur more privacy loss as the number of agents increase. For example, in Fig. 4.1, it can be seen that the environment decompositions in the first batch (indexes 1-30) is incurring the least privacy loss, and the last batch (indexes 90-120) incurs the most privacy loss in all of three algorithms.

The biggest difference in privacy loss is between DAER and DCTE in Fig. 4.9, where DAER averages at 1.86 and DCTE at 4.99. Hence, in average, DAER incurs more than 1,300 times less privacy loss than DCTE ($10^{4.99}/10^{1.86}$) when the number of agents is between 10-200, the MBS is between 11-15, and the ratio of private to shared variables is 3:1. In the same figure (Fig. 4.9), DAER average to incur more than 23 times less privacy loss than Action-GDL ($10^{3.24}/10^{1.86}$) when the number of agents is between 10-200, the MBS is between 11-15, and the ratio of private to shared variables is 3:1.

In the figures that have MBS between 1-5 (Fig. 4.1, 4.2, and 4.3), there is a drop in privacy loss in the environments indexed 1, 11, and 22. This drop is happening
because the environment decomposition indexed 1, 11, and 22 have the lowest num-
ber of agents among their batch (see Table 4.2). The same case is present in the
figures that have MBS between 6-10 (Fig. 4.4, 4.5, and 4.6), where the environment
decompositions indexed 26 to 30 have the lowest number of agents among their batch.

DCTE tends to incur more privacy loss as the MBS increase. For instance, in
Fig. 4.1, where the MBS is between 1-5, DCTE averages at 4.19, but in Fig. 4.7,
where the MBS is between 11-15, DCTE averages at 4.55. On the other hand, DAER
tends to incur less privacy loss as the MBS increase with an average of 2.64 in Fig. 4.1
and an average of 1.88 in Fig. 4.7.

Finally, since DAER execute over the communication graph, the ratio of private
to shared variables does not affect the privacy loss of DAER. For example, Fig. 4.1, 4.2,
and 4.3 are only different in the ratio of private to shared variables, and DAER
averages at 2.64, 2.65, and 2.66 in the mentioned figures respectively. However, in
the same figures (Fig. 4.1, 4.2, and 4.3), DCTE averages at 4.19, 4.44, and 4.60
respectively.

Figure 4.1: PL of constructing a JT, MBS is between 1-5, and the ratio is 1:1
(µ, σ): DAER(2.64, 0.46), Action-GDL(3.17, 0.52) DCTE(4.19, 0.60)
Figure 4.2: PL of constructing a JT, MBS is between 1-5, and the ratio is 2:1 \((\mu, \sigma)\): DAER(2.65, 0.46), Action-GDL(3.17, 0.51) DCTE(4.44, 0.61)

Figure 4.3: PL of constructing a JT, MBS is between 1-5, and the ratio is 3:1 \((\mu, \sigma)\): DAER(2.66, 0.46), Action-GDL(3.17, 0.51) DCTE(4.60, 0.62)

Figure 4.4: PL of constructing a JT, MBS is between 6-10, and the ratio is 1:1 \((\mu, \sigma)\): DAER(2.35, 0.57), Action-GDL(3.29, 0.4) DCTE(4.41, 0.45)
Figure 4.5: PL of constructing a JT, MBS is between 6-10, and the ratio is 2:1 \((\mu, \sigma): \text{DAER}(2.34, 0.56), \text{Action-GDL}(3.29, 0.39) \text{ DCTE}(4.68, 0.45)\)

Figure 4.6: PL of constructing a JT, MBS is between 6-10, and the ratio is 3:1 \((\mu, \sigma): \text{DAER}(2.35, 0.57), \text{Action-GDL}(3.29, 0.39) \text{ DCTE}(4.84, 0.45)\)

Figure 4.7: PL of constructing a JT, MBS is between 11-15, and the ratio is 1:1 \((\mu, \sigma): \text{DAER}(1.88, 0.59), \text{Action-GDL}(3.25, 0.77) \text{ DCTE}(4.55, 0.87)\)
Figure 4.8: PL of constructing a JT, MBS is between 11-15, and the ratio is 2:1
($\mu, \sigma$): DAER(1.88, 0.58), Action-GDL(3.25, 0.77) DCTE(4.82, 0.88)

Figure 4.9: PL of constructing a JT, MBS is between 11-15, and the ratio is 3:1
($\mu, \sigma$): DAER(1.86, 0.6), Action-GDL(3.24, 0.77) DCTE(4.99, 0.88)
Runtime The figures from Fig. 4.10 to Fig. 4.18 show the runtime in milliseconds. The computer used for the experiments has a quad-core processor with a clock speed of 2.4 GHz. For simplicity, runtime will be referred to as “RN”, the maximum border size will be referred to as “MBS”, and private variables to shared variables ratio will be referred to as “ratio”. The mean (µ) and the standard deviation (σ) is shown in the caption of each figure for the three algorithms.

Fig. 4.10 to Fig. 4.18 show that all three algorithms tend to take more time as the number of agents increase. Action-GDL has the shortest runtime throughout all the batches. However, as the private to shared variables ratio increase, Action-GDL takes more time and get closer to the other two algorithms. For example, in Fig. 4.16, 4.17, and 4.18, Action-GDL averages at 2.72, 3.30, and 3.70 respectively. DAER and DCTE have a close runtime in comparison. They have a similar averages in most of the figures. Furthermore, in Fig. 4.14 and Fig. 4.18, DAER and DCTE both have the same average at 3.53 and 3.76 respectively.

In the figures that have MBS between 1-5 (Fig. 4.10, 4.11, and 4.12), there is a drop in runtime in the environments indexed 1, 11, and 22. This drop is happening because these environment decompositions have the lowest number of agents among their batch. The same case is present in the figures that have MBS between 6-10 (Fig. 4.13, 4.14, and 4.15), where the environment decompositions indexed 26 to 30 have the lowest number of agents among their batch.

DAER has the shortest runtime compared to Action-GDL and DCTE when the number of agents is low and the MBS is high. For instance, in Fig. 4.16, 4.17, and 4.18, where the MBS is between 11-15, DAER performs the best in the first batch (indexes 1-30), where the number of agents is between 10-50.
Finally, since DAER execute over the communication graph, the ratio of private to shared variables does not affect the runtime of DAER. For example, Fig. 4.10, 4.11, and 4.12 are only different in the ratio of private to shared variables, and DAER averages at 3.36, 3.32, and 3.32 respectively.

Figure 4.10: RN of constructing a JT, MBS is between 1-5, and the ratio is 1:1 ($\mu, \sigma$): DAER(3.36, 0.82), Action-GDL(1.94, 0.19) DCTE(3.17, 0.70)

Figure 4.11: RN of constructing a JT, MBS is between 1-5, and the ratio is 2:1 ($\mu, \sigma$): DAER(3.32, 0.86), Action-GDL(2.24, 0.33) DCTE(3.17, 0.72)
Figure 4.12: RN of constructing a JT, MBS is between 1-5, and the ratio is 3:1 $(\mu, \sigma)$: DAER(3.32, 0.86), Action-GDL(2.51, 0.43) DCTE(3.18, 0.71)

Figure 4.13: RN of constructing a JT, MBS is between 6-10, and the ratio is 1:1 $(\mu, \sigma)$: DAER(3.57, 0.75), Action-GDL(2.21 0.27) DCTE(3.44, 0.67)

Figure 4.14: RN of constructing a JT, MBS is between 6-10, and the ratio is 2:1 $(\mu, \sigma)$: DAER(3.53, 0.78), Action-GDL(2.64, 0.35) DCTE(3.53, 0.70)
Figure 4.15: RN of constructing a JT, MBS is between 6-10, and the ratio is 3:1
$(\mu, \sigma)$: DAER(3.53, 0.79), Action-GDL(2.99, 0.41) DCTE(3.51, 0.57)

Figure 4.16: RN of constructing a JT, MBS is between 11-15, and the ratio is 1:1
$(\mu, \sigma)$: DAER(3.78, 1.23), Action-GDL(2.72, 0.50) DCTE(3.66, 1.15)

Figure 4.17: RN of constructing a JT, MBS is between 11-15, and the ratio is 2:1
$(\mu, \sigma)$: DAER(3.77, 1.24), Action-GDL(3.30, 0.68) DCTE(3.75, 1.19)
Figure 4.18: RN of constructing a JT, MBS is between 11-15, and the ratio is 3:1 $(\mu, \sigma)$: DAER(3.76, 1.25), Action-GDL(3.70, 0.76) DCTE(3.76, 1.14)

**Messages Passed** The figures from Fig. 4.19 to Fig. 4.27 show the number of messages passed between the agents during the construction of the JT. For simplicity, the number of messages passed between the agents will be referred to as “MP”, the maximum border size will be referred to as “MBS”, and private variables to shared variables ratio will be referred to as “ratio”. The mean $(\mu)$ and the standard deviation $(\sigma)$ is shown in the caption of each figure for the three algorithms.

Fig. 4.19 to Fig. 4.27 show that DAER has the largest number of messages passed compared to Action-GDL and DCTE. The number of messages passed increase as the number of agents increase in all of the three algorithms. This increase is showing in DAER since all the agents contribute to evaluate the best expansion plan in the elimination-expansion stage of DAER.

In the figures that have MBS between 1-5 (Fig. 4.19, 4.20, and 4.21), there is a drop in the number of messages in the environments indexed 1, 11, and 22. This drop is happening because these environment decompositions have the lowest number of agents among their batch. The same case is present in the figures that have MBS between 6-10 (Fig. 4.13, 4.14, and 4.15), where the environment decompositions
indexed 26 to 30 have the lowest number of agents among their batch.

The number of messages passed between the agents in Action-GDL increase as the MBS and the ratio increase. For example, in Fig. 4.19, Fig. 4.22, and Fig. 4.25, where the MBS is between 1-5, 6-10, and 11-15, Action-GDL averages at 3.65, 4.06, and 4.54 respectively. Furthermore, in Fig. 4.25, Fig. 4.26, and Fig. 4.27, where the ratio is 1:1, 2:1, and 3:1, Action-GDL averages at 4.54, 4.95, 5.22 respectively.

On the other hand, DAER and DCTE are not affected by the number of shared variables in term of messages passed. DAER is not affected by the number of private variables because DAER execute over the communication graph. For instance, in Fig. 4.25, Fig. 4.26, and Fig. 4.27, DAER averages at 5.46 in all of the three charts. DCTE is not affected by the number of shared variables because the agents announce all the variables to their neighboring agents. Hence, the number of private variables affect the size of the messages passed, but not the number of messages. For instance, in Fig. 4.25, Fig. 4.26, and Fig. 4.27, DCTE averages at 3.99 in all of the three charts.

Figure 4.19: MP in constructing a JT, MBS is between 1-5, and the ratio is 1:1 $$(\mu, \sigma): \text{DAER}(5.28, 0.82), \text{Action-GDL}(3.65, 0.36) \text{ DCTE}(3.82, 0.55)$$
Figure 4.20: MP in constructing a JT, MBS is between 1-5, and the ratio is 2:1 $(\mu, \sigma)$: DAER(5.28, 0.82), Action-GDL(4.04, 0.37) DCTE(3.82, 0.55)

Figure 4.21: MP in constructing a JT, MBS is between 1-5, and the ratio is 3:1 $(\mu, \sigma)$: DAER(5.28, 0.82), Action-GDL(4.31, 0.37) DCTE(3.82, 0.55)

Figure 4.22: MP in constructing a JT, MBS is between 6-10, and the ratio is 1:1 $(\mu, \sigma)$: DAER(5.51, 0.63), Action-GDL(4.06, 0.25) DCTE(4.02, 0.43)
Figure 4.23: MP in constructing a JT, MBS is between 6-10, and the ratio is 2:1
\(\mu, \sigma\): DAER(5.50, 0.63), Action-GDL(4.46, 0.25) DCTE(4.02, 0.43)

Figure 4.24: MP of constructing a JT, MBS is between 6-10, and the ratio is 3:1
\(\mu, \sigma\): DAER(5.51, 0.63), Action-GDL(4.73, 0.25) DCTE(4.02, 0.43)

Figure 4.25: MP of constructing a JT, MBS is between 11-15, and the ratio is 1:1
\(\mu, \sigma\): DAER(5.46, 1.24), Action-GDL(4.54, 0.44) DCTE(3.99, 0.88)
Figure 4.26: MP of constructing a JT, MBS is between 11-15, and the ratio is 2:1
$(\mu, \sigma)$: DAER(5.46, 1.24), Action-GDL(4.95, 0.44) DCTE(3.99, 0.88)

Figure 4.27: MP of constructing a JT, MBS is between 11-15, and the ratio is 3:1
$(\mu, \sigma)$: DAER(5.46, 1.24), Action-GDL(5.22, 0.44) DCTE(3.99, 0.88)
Chapter 5
Conclusion

5.1 Summary of Contributions

This thesis introduces a new algorithm suite called DAER. This algorithm suite is an extension to the HTBS algorithm suite presented in [22]. HTBS is the only known algorithm to recognize and construct the JT agent organization without leaking any privacy. However, when the environment decomposition does not admit to the JT agent organization, HTBS recognizes the non-existence of the JT without being able to construct it. Action-GDL and DCTE, which were designed to solve constraint optimization problems (DCOPs), revise the environment decomposition to construct the JT even if the environment decomposition does not admit to the JT agent organization. However, Action-GDL and DCTE leak agent privacy whether the environment decomposition admits to the JT agent organization or not.

DAER algorithm suite revises the environment decomposition and constructs the JT agent organization while trying to keep the privacy loss to the minimum. When the environment decomposition admits the JT agent organization, DAER recognizes and constructs the JT without leaking any privacy. On the other hand, if the environment decomposition does not admit a JT agent organization, DAER revises the environment decomposition and constructs the JT agent organization while incurring significantly less privacy loss than Action-GDL and DCTE. In some experimental
cases, DAER incurred 23 times less privacy loss than Action-GDL, and 1300 times less against DCTE. Furthermore, DAER does not leak any privacy of the private variables. During the execution of DAER, agents are able to quantify the possible privacy loss and compare the significance of each piece of private information internally through the quantification scheme presented in Section 3.2.

Finally, the work in this thesis has been submitted to the 9th international conference on probabilistic graphical models as a joint paper with my supervisor Dr. Yang Xiang, and it has been accepted [19].

5.2 Future Work

DAER achieved its purpose of recognizing and constructing the JT agent organization even if the environment decomposition does not admit a JT. DAER was proven to be significantly better than Action-GDL and DCTE in terms of preserving agents’ privacy. Since DAER follows a greedy approach, it does not always guarantee an optimal solution, but it might be very close to the optimal solution in most cases. No known algorithm can find the optimal solution of constructing the JT agent organization with the minimum privacy loss possible. The next step is to show how close DAER is to the optimal solution and determine whether or not it is possible to find an algorithm that presents a better solution than DEAR.

When agents estimate the best expansion plan during the elimination-expansion stage in DAER, the privacy loss on agent identities is assumed in some cases. When an agent $A_i$ does not know if his two neighboring agents $A_j$ and $A_k$ are connected, the agent $A_i$ does not know if the privacy of their identities will be leaked or not. Therefore, $A_i$ assumes that their identities will be leaked in the process, and estimate
its expansion plan with this assumption. This assumption was necessary to prevent
any more privacy loss during the estimation process. If DAER can get around this
assumption and be able to find the actual privacy loss without leaking any private
information in the process, DAER will be even closer to the optimal solution.

Another task for this work is to allow DAER to elect the leader agent inter-
nally. As HTBS, the leading agent of the first stage in DAER is assumed to be
chosen externally as shown in Section 3.1.3. There are many algorithms for electing
leader agents in distributed environments such as the approach used in Action-GDL.
However, existing algorithms of electing the leader agent incur privacy loss on agent
identities. Therefore, finding an algorithm that elects the leader agent without leaking
any privacy is yet to be explored.
Bibliography


Appendix A

Appendix

The abbreviations used in the thesis are combined in the following table and sorted alphabetically for the reader’s convenience:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BG</td>
<td>Boundary Graph</td>
</tr>
<tr>
<td>CG</td>
<td>Communication Graph</td>
</tr>
<tr>
<td>DAER</td>
<td>Distributed Agent Environment Re-decomposition</td>
</tr>
<tr>
<td>DCTE</td>
<td>Distributed Cluster Tree Elimination</td>
</tr>
<tr>
<td>DPMST</td>
<td>Distributed Privacy preserving Maximum Spanning Tree</td>
</tr>
<tr>
<td>EE</td>
<td>Elimination-Expansion</td>
</tr>
<tr>
<td>GDL</td>
<td>Generalized Distributive Law</td>
</tr>
<tr>
<td>HTBS</td>
<td>HyperTree construction based on Boundary Set</td>
</tr>
<tr>
<td>JT</td>
<td>Junction Tree</td>
</tr>
<tr>
<td>MAS</td>
<td>Multi-Agent System</td>
</tr>
<tr>
<td>MBS</td>
<td>Maximum Border Size</td>
</tr>
<tr>
<td>MP</td>
<td>number of Messages Passed</td>
</tr>
<tr>
<td>MSPL</td>
<td>Maximum System Privacy Loss</td>
</tr>
<tr>
<td>MST</td>
<td>Maximum Spanning Tree</td>
</tr>
<tr>
<td>NSPL</td>
<td>Normalized System Privacy Loss</td>
</tr>
<tr>
<td>PL</td>
<td>Privacy Loss</td>
</tr>
<tr>
<td>RN</td>
<td>RuNtime</td>
</tr>
<tr>
<td>SPL</td>
<td>System Privacy Loss</td>
</tr>
</tbody>
</table>
Understanding how HTBS works is essential to understanding how DAER works. Section 2.5 explained how HTBS works, and the original procedures of HTBS can be found in the original work in [22]. However, for reader’s convenience, the procedures are included here in a similar format to the DAER algorithm presented in Chapter 3. The parameters used in the HTBS procedures are as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Initial value</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>state ∈ IN, OUT</td>
<td>IN</td>
<td>Indicates whether this agent has been eliminated (OUT) or not (IN).</td>
</tr>
<tr>
<td>nbsta(A_k) ∈ IN, OUT</td>
<td>IN</td>
<td>Indicates whether the bordering agent A_k has been eliminated or not.</td>
</tr>
<tr>
<td>curtok</td>
<td>nil</td>
<td>The token value keeping track of the round number that indicates if the agent has been visited in this DFT round or not.</td>
</tr>
<tr>
<td>visited(A_k) ∈ true, false</td>
<td>false</td>
<td>Indicates whether the neighboring agent A_k has been visited in this DFT round or not.</td>
</tr>
<tr>
<td>parent</td>
<td>nil</td>
<td>Keeps track of the sender A_c because the agent might send a report message back to A_c.</td>
</tr>
<tr>
<td>Y_i</td>
<td>W_i</td>
<td>Keeps track of the active boundary of the agent since whenever a neighboring agent perform self-elimination relative to this agent, the active boundary will be updated.</td>
</tr>
</tbody>
</table>
The procedures of the HTBS algorithm suite are as follows:

**Algorithm 10** StartNewDFT(tok)

1. if $A_c$ is a bordering agent then
2.   $\text{nbsta}(A_c) = \text{OUT};$
3.   if there exists no $A_j$ with $\text{nbsta}(A_j) = \text{IN}$ then
4.     declare "a hypertree exists" and start halting;
5.     else
6.     $Y_i = \phi;$
7.     for each bordering $A_k$ where $\text{nbsta}(A_k) = \text{IN}$ do
8.       $Y_i = Y_i \cup I_{ik};$
9.     end for
10.    end if
11.   end if
12.   $\text{curtok} = \text{tok}; \ \text{parent} = \text{nil};$
13.   run DoDFT
Algorithm 11 DoDFT

1: if there exists $A_j$ with $nsta(A_j) = IN$ and $Y_i = I_{ij}$ then $\triangleright$ self eliminate
2: \hspace{0.5em} $state = OUT;$
3: \hspace{0.5em} for each bordering $A_k \neq A_j$ where $nbsta(A_k) = IN$ do
4: \hspace{1.5em} send $Eliminated$ to $A_k;$
5: \hspace{0.5em} end for
6: \hspace{0.5em} send $StartNewDFT(curtok + 1)$ to $A_k;$
7: else $\triangleright$ no $IN$ agent satisfies $Y_i = I_{ij}$
8: \hspace{0.5em} $parent = A_c;$
9: \hspace{0.5em} for each bordering $A_k \neq parent$ where $nbsta(A_k) = IN$ do
10: \hspace{1.5em} set $visited(A_k) = false;$
11: \hspace{0.5em} end for
12: \hspace{0.5em} if there exists $A_k \neq parent$ where $nbsta(A_k) = IN$ and $visited(A_k) = false$
13: \hspace{1.5em} send message $DFT(curtok)$ to $A_k;$
14: \hspace{0.5em} else
15: \hspace{1.5em} send report to $parent;$
16: \hspace{0.5em} end if
17: end if
Algorithm 12 Response to DFT(tok)

1: if curtok = tok then ▷ tok is not fresh
2:     send Report to Ac;
3: else ▷ tok is fresh to Ai
4:     curtok = tok;
5:     parent = Ac; visited(Ac) = true;
6:     run DoDFT;
7: end if

Algorithm 13 Response to Report

1: visited(Ac) = true;
2: if there exists Ak ≠ parent such that nbsta(Ak) = IN and visited(Ak) = false then
3:     send message DFT(curtok) to Ak;
4: else ▷ no unvisited bordering agent
5:     if parent = nil then
6:         declare "no hypertrees" and start halting;
7:     else
8:     send Report to parent
9: end if
10: end if