Master's Thesis

Identifying Sets of Believed Arguments in a Network of Cooperating Agents

Soutaro Shimizu

Supervisor Teeradaj Racharak

Graduate School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology
(Information Science)

March, 2024

Abstract

In communication, we each hold various ideologies and assertions, contemplating whether we can ultimately convince others of our own claims. At times, we may ignore or conceal arguments that are disadvantageous to our own claims in order to persuade others to accept them. We analyze a formalization of which arguments are hidden or ignored in order to get someone to accept a certain claim.

This behavior is particularly prevalent in interactive multiplayer games, where actions are often taken to achieve one's victory conditions. One well-known example of such a game is the Werewolf game [1,2], where players are divided into Werewolf and Villager factions. All players engage in discussions, with the Villager faction aiming to identify and eliminate Werewolves from the village, while the Werewolf faction seeks to deceive the Villagers and eliminate them. Among these two factions, the werewolf faction, in particular, often ignores or conceals arguments unfavorable to their claims in order to make them more acceptable. This is because, fundamentally, the werewolf faction finds themselves at a disadvantage in debates. While the villager faction moves closer to victory by honestly sharing the information they possess, the werewolf faction must progress discussions in a distorted manner, such as by telling lies, in order to navigate the situation. Hence, the werewolf faction frequently engages in such behavior.

To address this issue, we formalize the act of ignoring or concealing unfavorable arguments to persuade acceptance using Abstract Argumentation Framework [3]. Then, we propose an implementation according to our proposed formalization and verify through algorithms how this affects multi-agent systems. Our algorithmic procedure can be briefly explained as follows: First, each agent provides arguments through debate, creating a set of arguments. Then, we examine from the debates which arguments are critical of each argument and form a set of relationships based on this. Through an algorithm, arguments are removed based on the set of arguments and their relationships. The sets of arguments and relationships are then reduced. And then, the process involves identifying the set of arguments that have been reduced by eliminating unfavorable arguments to ensure the acceptance of the desired claim.

By identifying the set of arguments being reduced, it becomes possible to understand which arguments and relationships of arguments within the arguments need to be removed in order to gain acceptance of the desired claim by others.

Keywords: Abstract Argumentation, Multi-agent System, Deception Detection, Dynamic Argument.

Acknowledgment

I would like to express my deep gratitude to Professor Teeradaj Racharak, who has provided guidance in the laboratory where I am currently affiliated, and Professor Satoshi Tojo, who provided guidance in the laboratory I was previously affiliated. Thanks to their advice and guidance on research direction during times when I was struggling, I could successfully conduct my research. Without them, my research would not have been possible.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Motivation

In communication, we each hold various ideologies and assertions, contemplating whether we can ultimately convince others of our own claims. At times, we may ignore or conceal arguments that are disadvantageous to our own claims in order to persuade others to accept them. We analyze which arguments are hidden or ignored in order to get someone to accept a certain claim.

These actions range from serious issues to games, such as serious instances occurring in trials or company meetings. In trials, for instance, prosecutors and defense attorneys engage in debates to provide the desired outcome for their respective clients. They present various arguments to convince the judge or presiding officer, sometimes ignoring unfavorable arguments to support their claims. Acts such as tampering with evidence or giving false testimony are prohibited by Article 104 of the Penal Code and Article 169 of the Penal Code, so concealing arguments would not occur [6]. Thus, there might be instances where arguments are disregarded, even in serious cases like trials. As for gaming examples, well-known ones include the Werewolf game and Among Us [1, 2, 4]. These multiplayer games are interactive and remain popular. In the Werewolf game, players are divided into the factions of werewolves and villagers. All players engage in discussions, with the Villager faction aiming to identify and eliminate Werewolves from the village, while the Werewolf faction seeks to deceive the Villagers and eliminate them.

We formalize the act of ignoring or concealing unfavorable arguments to persuade acceptance using Abstract Argumentation Framework [3]. Then, we verify through our proposed algorithms how this affects multi-agent systems [7]. In various ways, we engage in the suppression or concealment of disadvantageous arguments in order to persuade others to accept our own assertions. Moreover, this process can be logically articulated.

The mechanism behind this was elucidated by Dung in 1995, with the framework of Abstract Argumentation (AA) [3] and its associated semantics.

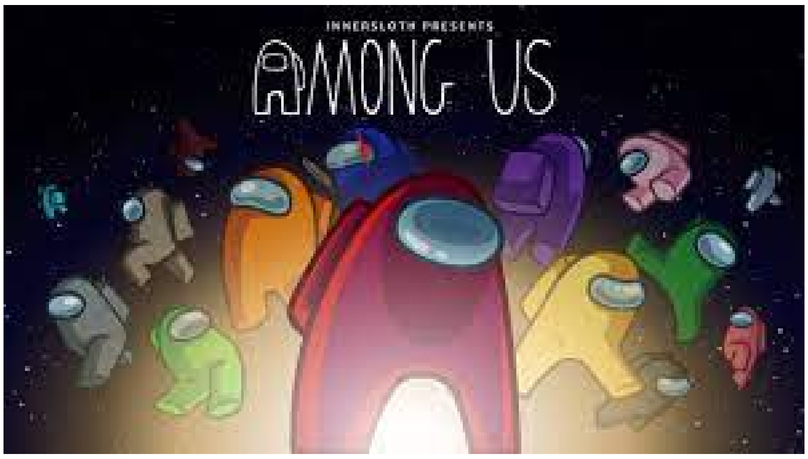


Figure 1.1: Among Us [4]

Through these, it became possible to derive conclusions in arguments using sets of formalized arguments and attack relations [8].

We formalize acts such as "ignoring or concealing" disadvantageous arguments to persuade others to accept our assertions using AA. Furthermore, we aim to verify through algorithms how this behavior affects multi-agent systems. By dynamically manipulating the arguments held by each agent and their attack relations in the AA framework (a.k.a. an AA graph) (cf. an example of such a dynamic manipulation in [7]), it is possible to influence the interpretation of multi-agent systems and their discourse. This enables the alteration of arguments deemed acceptable in discourse.

In order to gain acceptance for desirable arguments from others, what arguments would need to be ignored or concealed? Our proposed methodology is outlined as follows: First, each agent contributes arguments through debate, forming a set of arguments. Next, attack relations are determined from the debate, forming a set of attack relations. Through our algorithm, arguments are then eliminated based on the set of arguments and attack relations. Subsequently, the set of arguments and attack relations related to the eliminated arguments is reduced. And then, the process involves identifying the set of arguments that have been reduced by eliminating unfavorable arguments to ensure the acceptance of the desired claim. By identifying this, it becomes possible to understand which arguments and attack relations of within the arguments need to be removed in order to gain acceptance of the desired claim by others.

Using this methodology, we analyze which arguments need to be ignored or concealed in order for others to accept a certain assertion. In this study, we utilize the Werewolf game as an example to validate these concepts.

Table 1.1: Victory conditions [1, 2]

Faction	Victory conditions
Villager	If all the Werewolves can be executed, it results in victory.
Werewolf	Victory is achieved if the number of living members in the Werewolf faction is equal to or fewer than the number of living members in the Villager faction.

1.2 The Werewolf game

The game consists of alternating day and night turns. During the night turn, characters with special abilities take action, and the Werewolves target and eliminate one member of the Villager faction. During the day turn, based on the information from the night turn, all surviving players engage in a maximum 5-minute discussion. They then vote to execute one player whom they suspect to be a Werewolf. The player with the most votes is eliminated. This cycle repeats, and the game concludes if either the Villager faction or the Werewolf faction satisfies their respective victory conditions. Each faction's victory conditions are as outlined in Table 1.1 [1, 2].

Next, let's explain the abilities of the role we will use in the experiment this time. The abilities of each position is represented in Table 1.2.

The Werewolf faction and the Villager faction will engage in discussions aiming for the victory conditions outlined in Table 1.2 by effectively utilizing the roles in Table 1.1.

1.3 Objectives

We propose an algorithm to analyze which arguments are hidden or ignored when a certain arguments is accepted. We aim to demonstrate that our algorithm is adaptable to examples such as the Werewolf game. The objectives of our system are as follows:

1. Provide explanations regarding Abstract Argumentation.
2. Represent which arguments are ignored or concealed.
3. Visualize all AA graphs when a certain assertion is accepted.
4. Analyze which arguments are ignored or concealed when a certain assertion was accepted.
5. Measure the execution time of the algorithm to determine its ability to handle graphs of varying complexity.

Table 1.2: Each role [1, 2]

Faction	**Role**	**Ability**
Villager	**Villager**	Nothing.
	Fortune teller	At night, during his/her turn, he/she can designate one person other than him/herself and determine whether they are werewolf or not.
	Knight	One nights other than the first, he/she choose one person to protect from werewolf attack during the night. He/She cannot protect him/herself.
Werewolf	**Werewolf**	He/She identify who his/her fellow werewolves are. During night turns, excluding the first night, he/she can choose one non-werewolf player to devour.
	Maniacs	The ability is nothing. Their victory condition aligns with the werewolf faction. However, he/she is unaware of who the werewolves are. The maniacs supports the werewolves by trying to identify them during discussions and employing tactics such as pretending to have a different role.

Our system demonstrates the analysis and practicality of hidden or ignored arguments in this study.

1.4 Contribution

In recent years, research has been conducted on inference in debates among multi-agents using Argumentation Frameworks and AA Frameworks [9–13]. However, there has been little research on dynamically [7] modifying AA to identify hidden or ignored arguments. We summarize the contributions of our study as follows:

1. We are the first who provides an algorithmic procedure for identifying hidden or ignored arguments when AA is changed.

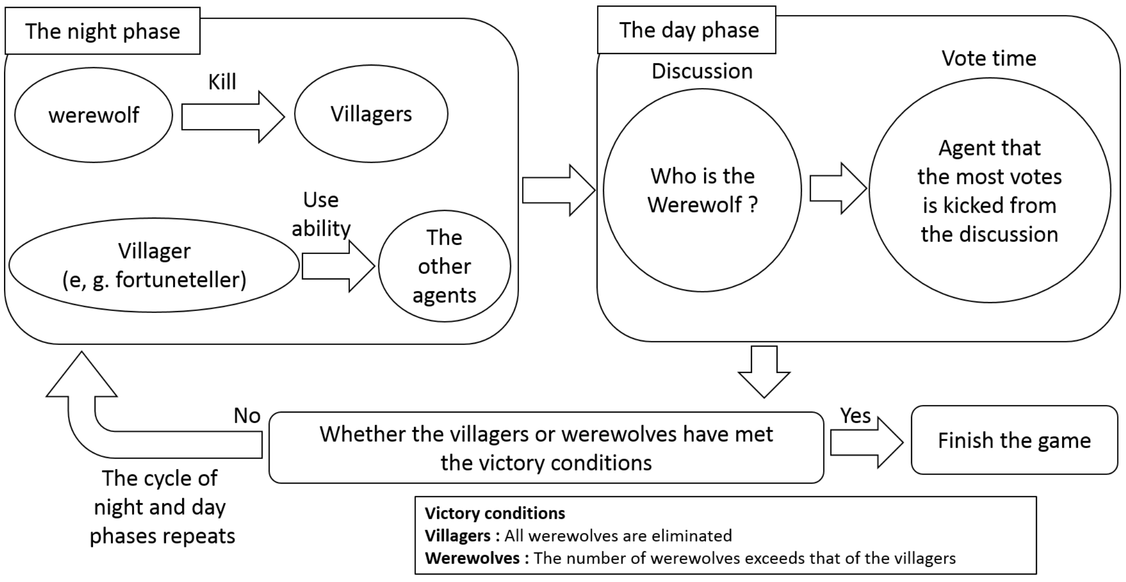

The night phase
werewolf
Kill
Villagers
Use ability
Villager (e, g. fortuneteller)
The other agents
The day phase
Discussion
Who is the Werewolf ?
Vote time
Agent that the most votes is kicked from the discussion
No
Whether the villagers or werewolves have met the victory conditions
Yes
Finish the game
The cycle of night and day phases repeats
Victory conditions
Villagers : All werewolves are eliminated
Werewolves : The number of werewolves exceeds that of the villagers

Figure 1.2: The flow of the Werewolf game [1, 2]

2. We show application of our algorithm in the Werewolf game, e.g.,
    - Assume a set of arguments wanted to be believed, the werewolf needs to lie to all the other agents (villagers, etc).

Figure 1.3: Discussions of the werewolf game [5]

Chapter 2

Preliminary

2.1 Overview

In this chapter, we will provide an explanation of the Abstract Argumentation (AA) [3] used in this study.

Traditionally, artificial intelligence has focused on single human thoughts, such as formalization of non-monotonic reasoning [3, 14]. However, in everyday human society, multiple people gather together, present their own opinions based on their own ideas on problematic issues, and argue and refute each other to solve various problems and make decisions, leading to consensus. [8]

Since 1995, intelligent agents (software) on distributed networks have continued to develop, and there has been a growing trend to have multiple intelligent agents solve problems, such as discussions and communication, as multiple humans do [8]. Indeed, in 1995, Dung elucidated the basic mechanism of human argumentation in a gathering [8]. Then, he defined an AA framework to realize this mechanism on a computer [8].

2.2 Abstract Argumentation

This section explains a series of definitions in the abstract argumentation framework and the semantics proposed for determining acceptable arguments regarding Dung's sense.

Definition 2.1: Abstract Argumentation framework [3]
An argumentation framework (AF) [3] is a pair $\mathcal{F} = (\mathcal{A}, \mathcal{R})$ where

- $\mathcal{A}$ is set of arguments and
- $\mathcal{R} \supseteq \mathcal{A} \times \mathcal{A}$ is a relation representing "attacks" of arguments.

Different semantics in AA Framework are proposed in [3].

- We say that an argument $X \in \mathcal{A}$ *attacks* an argument $Y \in \mathcal{A}$ if $(X, Y) \in \mathcal{R}$

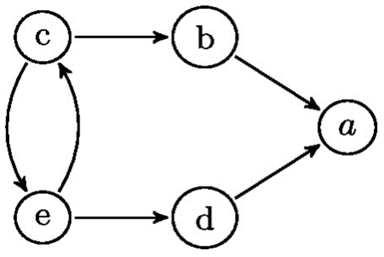


Figure 2.1: AA graph

- A set S of argument *attacks* an argument Y if there exists $X \in S$ such that X *attacks* Y
- A set S of arguments *defends* an argument X if S *attacks* every arguments attacking X
- S is *conflict-free* if for any $X, Y \in S$, there is no $(X, Y) \in \mathcal{R}$.

Definition 2.2: Abstract Argumentation semantics [3]
Let $S \subseteq \mathcal{A}$. Several semantics are defined as follows:

- S is **admissible** if S is **conflict-free** and S defends all arguments attacking S.
- S is **preferred** if S is the **maximal** (w.r.t set inclusion) admissible.
- S is **grounded** if S is the **minimal** (w.r.t set inclusion) complete.
- S is **complete** if S is **admissible** and S **contains all arguments** it defends.
- S is **stable** if S is **conflict-free** and S attacks each argument $X \in A \backslash S$.
- S is **ideal** if S is **admissible** and S is the (**maximal set**) intersection of all preferred semantics.

The results of executing the semantics are arguments' set called *extensions*. We now show an example of AA Framework. In Figure 2.1, according to Definition 2.2, the extensions of each semantics would be as follows:

- **admissible** $= \{\emptyset\}, \{c\}, \{e\}, \{a, c\}, \{c, d\}$
- **preferred** $= \{a, c\}, \{c, d\}$
- **grounded** $= \{\emptyset\}$
- **complete** $= \{\emptyset\}, \{a, c\}, \{c, d\}$
- **stable** $= \{a, c\}, \{c, d\}$
- **ideal** $= \{c\}$

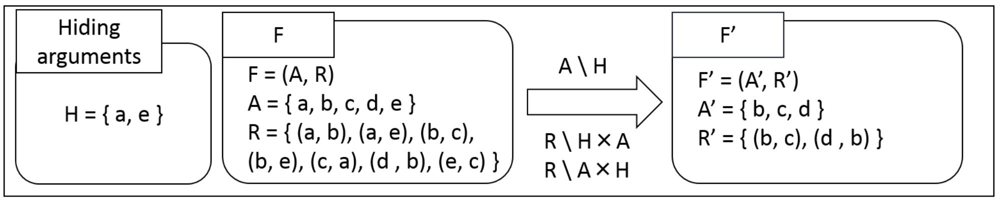


Figure 2.2: The process of the Dynamic Abstract Argumentation

2.2.1 Dynamic Abstract Argumentation

Here, we provide a brief definition of Dynamic Abstract Argumentation , as lightly discussed in Section 1.1.

Definition 2.3: Dynamic Abstract Argumentation
Sets $\mathcal{A}$ and $\mathcal{R}$ can be changed through time. Let AF be denoted as $F' = (\mathcal{A}', \mathcal{R}')$. Given that H $\supset \mathcal{A}$ be a set of the hiding arguments, then

- $\mathcal{A}' \coloneqq \mathcal{A} \setminus H$ if H is not empty.
- $\mathcal{R}' \coloneqq \mathcal{R}\setminus$ (H $\times\mathcal{A} \cup \mathcal{A}\times$ H) if H is not empty.

According to Definition 2.3, we perform the process as illustrated in Figure 2.2. For example, when considering Figure 2.1, executing this process would result as follows. Assuming H is $\{c, b\}$ and argument is $\{a, b, c, d, e\}$, according to the definition, argument $\setminus$ H is conducted. Thus, the argument becomes $\{a, d, e\}$. Additionally, attack relations associated with $\{c, b\}$ are removed, so the attack relations become $\{(e, d), (d, a)\}$. In this case, according to Definition 2.3, the new extensions would be as follows:

- **admissible** $= \{\emptyset\}, \{a\}, \{e\}, \{a, e\}$
- **preferred** $= \{a, e\}$
- **grounded** $= \{a\}, \{e\}$
- **complete** $= \{a\}, \{e\}, \{a, e\}$
- **stable** $= \{a, e\}$
- **ideal** $= \{\emptyset\}$

Executing Dynamic Abstract Argumentation in this manner results in changes to the content of arguments, attack relations, and extensions. Then, the above process is repeated, and the set of combinations of arguments and attack relations that emerge from it is computed.

Chapter 3

Methodology

3.1 Overview

People may ignore or withhold arguments that are disadvantageous to them in order to assert their accepted arguments as true. This practice is called "*hiding argument*" or "*hidden argument*".

Definition 3.1: Hidden Arguments
Given $\mathcal{F} = (\mathcal{A}, \mathcal{R})$ and $\mathcal{F}' = (\mathcal{A}', \mathcal{R}')$ be argumentation frameworks where $\mathcal{A}' \subset \mathcal{A}$ and $\mathcal{R}' \subset \mathcal{R}$, we say arguments $\mathcal{A} \setminus \mathcal{A}'$ is *hidden* from $\mathcal{F}$.

Example
Let $\{1, 2, 3, 4\}$ be a set of arguments, and $\{(1, 2), (3, 1), (3, 2), (4, 3)\}$ be an attack relations among these arguments. Suppose that we would like to know what arguments would need to be hidden so that arguments $\{1, 4\}$ become accepted. From the original AA graph, by removing such arguments that result in $\{1, 4\}$ being accepted in the AA graph, we can identify Hidden Arguments. Following this viewpoint, in Figure 3.1, it can be observed that $\{3\}$ is Hidden Argument.

Any communication of agent's dialogue is represented by an abstract argumentation framework $\mathcal{F}$ where each *node* is a full sentence put forward by an agent and each *edge* represents an attack relation between them.

In this chapter, we describe the methodology and mathematically algorithmic completeness of hiding arguments in discussions. It is an approach that shrinks arguments and attack relations in an a discussion using the Abstract Argumentation Framework and the uniquely defined Dynamic Abstract Argumentation to assert the accepted arguments of one's agents as true. In addition, by displaying these Abstract Argumentation graphs, it is possible to compare which arguments are hidden and which accepted arguments are true. Furthermore, we prove the completeness of the algorithm to aid the correctness of this methodology.

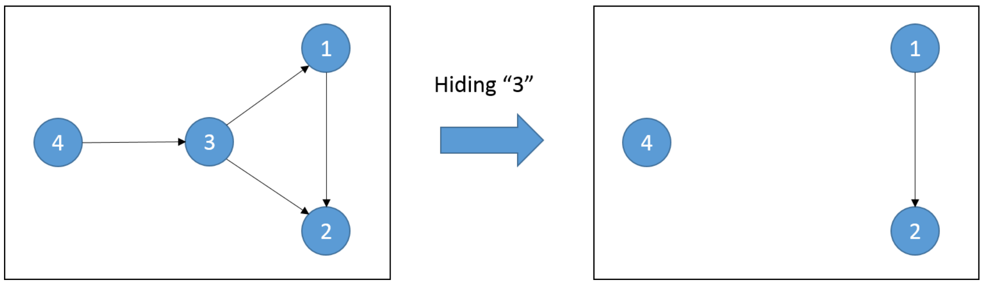


Figure 3.1: The AA graph after hiding "3"

To solve this problem, we divide it into three sub-steps:

1. **Calculate the ground extension:** This is used to identify sets of Hidden Arguments. By calculating the grounded extension, it is possible to determine which set constitutes the Hidden Arguments. Furthermore, this facilitates graph comparison by referencing it.
2. **Shrinking AA (i.e. the dynamics):** By dynamically altering the AA, we create sets of Hidden Arguments, enabling the creation of all sets of Hidden Arguments.
3. **Prove mathematically the correctness of the algorithm:** By proving this, we confirm that the identification of sets of Hidden Arguments is being done correctly.

The purpose of this chapter is to explain these sub-steps.

3.2 Algorithmic Procedure

In Figure 3.2, the whole algorithm's flowchart is depicted.

At first, we input the original AF. This AF will be dynamically modified to conduct the research.

Next, we seek to find all combinations of elements from the arguments. "All combinations" refers to selecting combinations of one element from the arguments to combinations of n-1 elements. Further details will be explained in Subsection 3.2.2.

And then, we iterate through the elements obtained from combinations, comparing each with the arguments of the AFs. We remove any matching elements from the arguments and also delete the attack relations related to the removed arguments. For example, if the first element of the combination is selected and the same element exists in the arguments, we delete it. Additionally, we delete the attack relations associated with the removed arguments. These combinations of arguments and attack relations represent the AF subjected to Hidden Arguments. The combinations of deleted arguments and attack relations are stored in a list. This process is repeated for each combination. Further details will be explained in detail in Subsection 3.2.2 and Section 3.3.

Afterward, we compute the grounded extension. Calculate the grounded of all the AFs subjected to Hidden Arguments that were stored earlier, and store them in a list. This is done by executing ASP from Python to determine the grounded. Further details will be explained in detail in Subsections 3.2.1.

Next, we analyze the Hidden Argument of each AF. According to the Definition 3.1, this can be determined by subtracting the nodes of the original AF from the nodes of the shrunken AF. Repeat this process for each shrunken AF to find each Hidden Argument.

Using the stored list of grounded or Hidden Arguments, we can select AFs that match specific conditions. The list of grounded stored is used to display its AF when the desired argument is accepted, or to analyze its Hidden Arguments. For example, in Figure 3.3, to display only AFs or Hidden Arguments that include $\{3\}$ in the grounded, it is possible to achieve this by saving the index 6, which corresponds to the list element containing it. This is because the indices of the list containing all grounded correspond to the indices of the list containing all AFs and the list containing all Hidden Arguments. On the other hand, the list of Hidden Arguments is used to display its AF or the arguments accepted due to its influence when the arguments is hidden. This is made possible by identifying the indices of the list of Hidden Arguments where the desired Hidden Arguments is stored, similar to when the list of grounded is used. This is because, as shown in Figure 3.3, the indices of each list correspond to each other. In other words, if we can grasp the index of the list element containing the grounded or Hidden Arguments of the desired AFs, it is possible to display the desired AFs.

Finally, we use the stored index to display AFs that match specific conditions.

The algorithm is operating in the manner as described in the above.

3.2.1 Computing the grounded extension

Here, we deal with the Werewolf game.

Example 1
The role of each agent is as shown in Table 3.1. Villagers consist of agents A and E, and the Fortune teller is agent C; these agents belong to the villager's faction. The Werewolf faction consists of agents B and D.

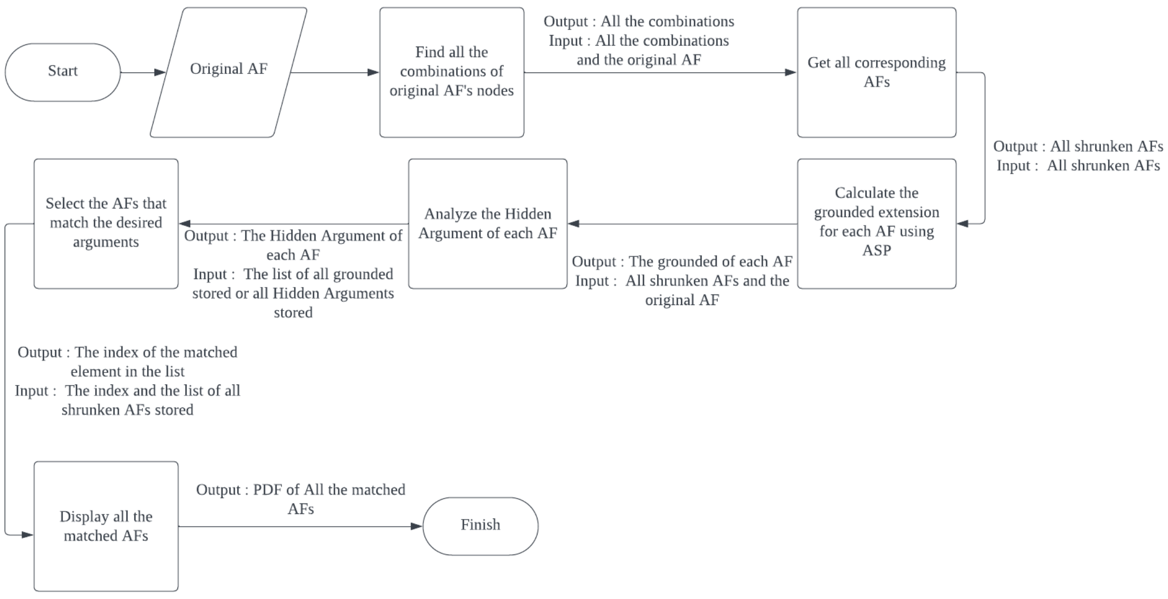

Start
Original AF
Find all the combinations of original AF's nodes
Output : All the combinations
Input : All the combinations and the original AF
Get all corresponding AFs
Output : All shrunken AFs
Input : All shrunken AFs
Calculate the grounded extension for each AF using ASP
Output : The grounded of each AF
Input : All shrunken AFs and the original AF
Analyze the Hidden Argument of each AF
Output : The Hidden Argument of each AF
Input : The list of all grounded stored or all Hidden Arguments stored
Select the AFs that match the desired arguments
Output : The index of the matched element in the list
Input : The index and the list of all shrunken AFs stored
Display all the matched AFs
Output : PDF of All the matched AFs
Finish

Figure 3.2: The flowchart of the algorithm

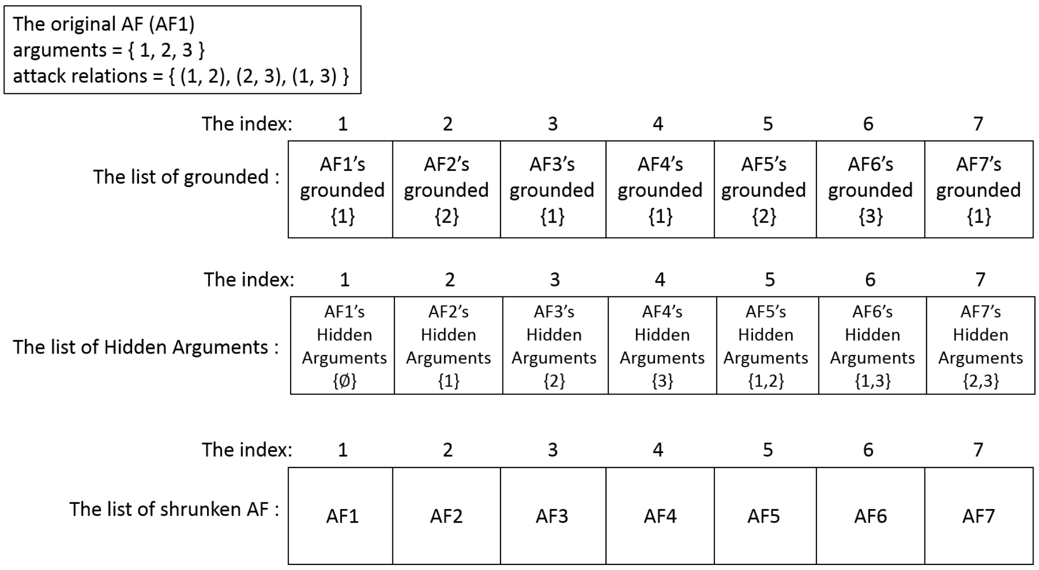
In the Example1, The discussion is being conducted as shown in 3.2.

Each agent has an argument in his/her knowledge base and presents it in the discussion. Then, the arguments raised there are combined into one. From Figure 3.4, the relationship between arguments and attacks is as follows.

Based on the definitions explained in Background and Figure 3.4, admissible, complete, and grounded extensions are as shown in Table 3.3.

From the above information, arguments 1, 2, 4, and 6 are acceptable for grounded extensions. Arguments 1 and 2 are given by C, argument 4 by A, and argument 6 by E. Notationally,
$A = \{4\}, B = \{3\}, C = \{1, 2\}, D = \{5\}, E = \{6\}$.
Arguments $= \{1, 2, 3, 4, 5, 6\}$.
Attack relations $= \{(3, 1), (3, 2), (4, 3), (5, 4), (6, 3), (6, 5)\}$.

Finally, we compute the grounded extension by executing Answer Set Programming (ASP) [15] from Python. In this case, the villager wins (i.e., his argument is accepted). These calculations are handled as shown in Figure


The original AF (AF1)
arguments = { 1, 2, 3 }
attack relations = { (1, 2), (2, 3), (1, 3) }

The index:	1	2	3	4	5	6	7
The list of grounded :	AF1's grounded {1}	AF2's grounded {2}	AF3's grounded {1}	AF4's grounded {1}	AF5's grounded {2}	AF6's grounded {3}	AF7's grounded {1}

The index:	1	2	3	4	5	6	7
The list of Hidden Arguments :	AF1's Hidden Arguments {∅}	AF2's Hidden Arguments {1}	AF3's Hidden Arguments {2}	AF4's Hidden Arguments {3}	AF5's Hidden Arguments {1,2}	AF6's Hidden Arguments {1,3}	AF7's Hidden Arguments {2,3}

The index:	1	2	3	4	5	6	7
The list of shrunken AF :	AF1	AF2	AF3	AF4	AF5	AF6	AF7

Figure 3.3: The correspondence of each list's index

Table 3.1: Agent's role

Agent ID	Agent's role
A	Villager
B	Werewolf
C	Fortuneteller
D	Werewolf
E	Villager

3.5 .

3.2.2 Shrinking AA (i.e. the dynamics)

We implement a naive method for shrinking the AA graph.

We visualize the AA graph by inputting arguments as *nodes* and attack relations as *edges* into the graph.

Next, we will store combinations of arguments with potential hidden elements obtained from agents into a list called "patterns". At this point, the number of elements in the set of set families within the "patterns" list ranges from 0 to 5. This is because, as evident from Figure 3.6, if the number of elements were 6, result in the deletion of all arguments, rendering the AA graph non-existent. If there is no AA graph, it implies that no agent

Table 3.2: Discussion's flow

Argument No.	Agent ID	Argument
1	C	A is not a werewolf by the result of divination.
2	C	B is a werewolf by the result of divination.
3	B	The result of divination are meaningless when fortuneteller C is false.
4	A	C is not confirmed to be a fake and should be used as a reference.
5	D	B is not a werewolf, so C is lying.
6	E	The one who doubts the result of divination is suspicious.

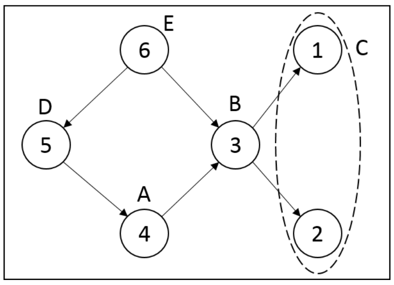


Figure 3.4: AA graph of example1

has submitted any arguments. Therefore, if the total number of arguments were n (n> 0, n∈ ℕ), the number of elements in the set families within the "patterns" list must be between 0 and n-1, inclusively. However, there is one exception. They are the presence of "None". Although this applies not only to the list "patterns", in the list we will use from now on, "None" is placed at index 0. Placing "None" at index 0 prevents confusion when extracting the first graph from the list, as it would be less clear if the first graph were located at index 1. Therefore, index 0 is intentionally excluded.

Next, the list "pattern" is compared to the *nodes* and *edges* of the graph and the matches are reduced. In "Graphset", "None" is stored at index 0 and "∅" is stored at index 1. At index 1, the original graph is stored because the first element of "patterns" is "∅". Then, the graph obtained by the *nodes* the remove the elements matching the second element of "patterns" and its associated *edges* from the original graph is stored at index 2 of "Graphset". Next, the graph obtained by the *nodes* remove the elements matching the third element of "patterns" and its associated *edges* from the original graph is stored at index 3 of "Graphset". This process is repeated

Table 3.3: Each extension of the AAF in Example 1

Abstract Argumentation framework	**Set**
Admissible	$\emptyset, \{6\}, \{1,6\}, \{2,6\}, \{4,6\}, \{1,4,6\}, \ldots$ $\{2,4,6\}$
Complete	$\{1,2,4,6\}$
Grounded	$\{1,2,4,6\}$

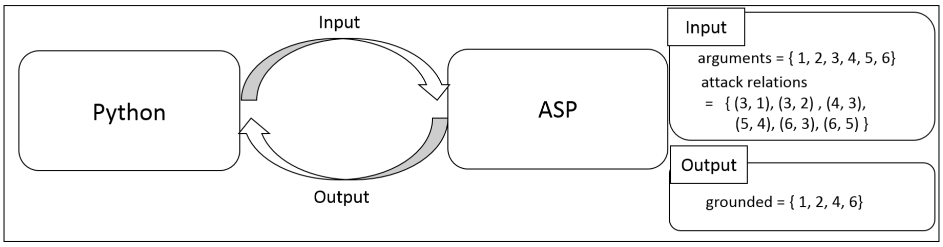


Figure 3.5: Executing ASP from python

for the length of "patterns". As a result, all the shrunk graphs are stored in the list "GraphSet".

Finally, the *nodes* and *edges* stored in the list "GraphSet" are sent from Python to ASP. The extensions computed in ASP are returned to Python and stored in the list "grounded".

This allows you to identify AA graphs that match specific conditions. For example, in Example 1 , when you want to identify only AA graphs that include $\{3\}$ in the grounded, you can extract those AA graphs by the indices of the list "grounded". This is because the indices of the list "grounded" correspond to the indices of the list "GraphSet". In other words, the element in the list "grounded" at index 1 corresponds to the grounded of the AA graph of the list "GraphSet" at index 1.

From Definition 3.1, Hidden Arguments of specific AA graphs can be analyzed by removing *nodes* from the original AA graph that correspond to the *nodes* of specific AA graphs.

By dynamically modifying the multi-agent knowledge base in this way, it is possible to represent "Hiding Argument".

3.3 Correctness of the Shrinking Algorithm

In this section, we will prove mathematically the correctness of the algorithm. We denote arguments gathered from agents as Equation 3.2 and define attack

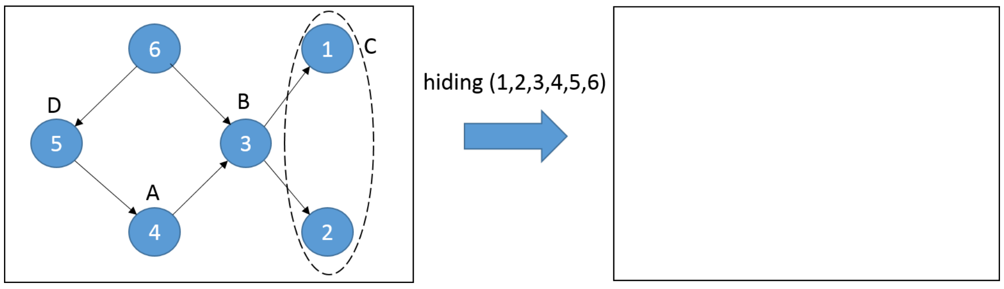


Figure 3.6: Hiding all arguments from the AA graph

relations as Equation 3.3.

$$n \in \mathbb{N}, \quad n > 0 \tag{3.1}$$

$$\text{argument} = \{a_1, a_2, a_3 \cdots, a_n\} \tag{3.2}$$

$$\begin{aligned} \text{attack relation} = \{&(a_1, a_1), (a_1, a_2), \cdots, (a_1, a_n), \\ &(a_2, a_1), (a_2, a_2), \cdots, (a_2, a_n), \\ &(a_3, a_1), \cdots, (a_3, a_n), \cdots, (a_n, a_1), \cdots, (a_n, a_n)\} \end{aligned} \tag{3.3}$$

First, we calculate all possible combinations from arguments in Equation 3.2. All combinations refer to obtaining all possible combinations from selecting zero item to selecting $n-1$ items ($n > 0$, $n \in \mathbb{N}$). The meaning of selecting 0 elements is to choose nothing. In this case, the set stored in the list "pattern" is $\emptyset$. When referenced, according to Definition 2.3, argument $\backslash\emptyset$ is conducted. At this point, argument and attack relation remain unchanged. This indicates the original AA graph. The reason why $n-1$ is the maximum is to prevent the AA graph from becoming $\emptyset$. For example, when there are arguments for Equation 3.2, the elements of combinations selecting n items will be the same as the arguments. Let's call this element H. Then, according to Definition 2.3, conducting arguments $\backslash$ H results in $\emptyset$. This means the AA graph disappears. When there's no AA graph, it implies that no agent has submitted any information and thus no discussion is taking place. Therefore, $n-1$ becomes the maximum. Based on these reasons, all combinations refer to obtaining all possible combinations from selecting zero item to selecting $n-1$ items ($n > 0$, $n \in \mathbb{N}$). In this case, the total number of combinations is represented by Equation 3.4.

$$\text{combination} = \sum_{r=0}^{n-1} {}_nC_r \tag{3.4}$$

Then, based on this combination, remove one argument at a time and delete the edges associated with the removed argument. Repeat this process for the number of combinations. Then, record all sets of arguments and edges after their removal. The number of sets at this point will be equal to the number of combinations.

For example, suppose there is an argument for Equation 3.5 and an attack relations against Equation 3.6. In that case, the combination would result in Equation 3.7.
The number of combinations calculated based on Equation 3.4 matches Equation 3.8, and this count is equivalent to enumerating the attack relations of Equation 3.6 one by one. Therefore, Equation 3.4 is proven to be correct.

$$\text{argument} = \{1, 2, 3, 4, 5\} \tag{3.5}$$

$$\text{attack relation} = \{(1, 2), (2, 1), (4, 1), (4, 3), (3, 4), (5, 2)\} \tag{3.6}$$

$$\begin{aligned}\text{combination} = \{&\emptyset, 1, 2, 3, 4, 5, \{1, 2\}, \{1, 3\}, \{1, 4\}, \{1, 5\}, \{2, 3\}, \{2, 4\},\\ &\{2, 5\}, \{3, 4\}, \{3, 5\}, \{4, 5\}, \{1, 2, 3\}, \{1, 2, 4\}, \{1, 2, 5\}, \{1, 3, 4\},\\ &\{1, 3, 5\}, \{1, 4, 5\}, \{2, 3, 4\}, \{2, 3, 5\}, \{2, 4, 5\}, \{3, 4, 5\}, \{1, 2, 3, 4\},\\ &\{1, 2, 3, 5\}, \{1, 2, 4, 5\}, \{1, 3, 4, 5\}, \{2, 3, 4, 5\}\}\end{aligned} \tag{3.7}$$

$$\begin{aligned}\text{combination} &= \sum_{r=0}^{5-1} {}_5C_r\\ &= 31\end{aligned} \tag{3.8}$$

Next, using Equation 3.7 , we will proceed to remove one argument at a time and delete attack relations associated with the removed argument. Then, we will retrieve all sets obtained from this process. At that point, the calculations are performed as follows: When the first element of the combination, $\{\emptyset\}$, is chosen, according to Definition 2.3, Equation 3.9 becomes like this.

$$\begin{aligned}\mathit{node} &= \text{argument} \setminus \emptyset\\ \mathit{edge} &= \text{attack relation} \setminus \emptyset\end{aligned} \tag{3.9}$$

When the second element of the combination, $\{1\}$, is chosen, Equation 3.10 becomes like this.

$$\begin{aligned}\mathit{node} &= \text{argument} \setminus \{1\}\\ \mathit{edge} &= \text{attack relation} \setminus \{\{1, 2\}, \{2, 1\}, \{4, 1\}\}\end{aligned} \tag{3.10}$$

When the seventh element of the combination,$\{1,2\}$, is chosen, Equation 3.11 becomes like this.

$$\begin{aligned} node &= \text{argument}\backslash\{\{1,2\}\} \\ edge &= \text{attack relation} \setminus \{\{1,2\},\{2,1\},\{4,1\},\{5,2\}\} \end{aligned} \tag{3.11}$$

For each combination, we reduce the set in a manner similar to Equation 3.9 to Equation 3.11. Then, we save all combinations of nodes and edges obtained through this process.

Next, based on Definition 2.2, we use Python to invoke Answer Set Programming (ASP) [15] to determine the grounded of each shrunken set obtained from the previous process. And the grounded result is as equation 3.12.

$$\begin{aligned} \text{grounded} = \{&\{5\},\{5\},\{5\},\{4,5\},\{1,3,5\},\{5\},\{5,4\},\{5,3\},\{2\},\{5,4\},\{5,3,1\}, \\ &\{5,1\},\{4,2\},\{3\},\{4,5\},\{3,5\},\{5\},\{2,4\},\{2,3\},\{1,5\},\{4\},\{1,3\}, \\ &\{5\},\{4\},\{3\},\{2\},\{1\}\} \end{aligned} \tag{3.12}$$

The sets of shrunken sets and grounded sets correspond in position, where the first element of the grounded set, $\{5\}$, corresponds to the grounded set of the first shrunken set. Similarly, the second element of the grounded set, $\{5\}$, corresponds to the grounded set of the second shrunken set. In this way, the sets of shrunken sets and grounded sets correspond to each other. Using this correspondence, we can display the desired AA graph. For example, if we want to display the AA graph where the shrunken set containing $\{1\}$ is included in the grounded, we can identify the position of $\{1\}$ within the grounded set and select all sets at that position to display. In this case, the elements containing 1 in the grounded sets are the ones in the fifth, eleventh, twelfth, twentieth, twenty-second and twenty-seventh positions. Therefore, we can select the fifth, eleventh, twelfth, twentieth, twenty-second and twenty-seventh shrunken sets. Similarly, we can compare different AA graphs based on different grounded using a similar method.

Chapter 4

Experiments

4.1 Overview

In this chapter, we measure how the algorithm's execution time varies for different numbers of arguments, allowing us to assess the algorithm's practicality for discussions of varying complexity. For this study, we prepared discussions with 6, 10, 14, 18, and 20 arguments. The discussions chosen for this example all revolve around the Werewolf game. In this game, players are divided into the Werewolf faction and the Villager faction, engaging in discussions.

We use these to examine the differences in the algorithm's execution time.

4.2 Time Complexity

Table 4.1 shows the time complexity of each process for analyzing Hidden Arguments.

In "Make original graph", the execution time is proportional to the number of arguments and attack relations. When the number of arguments is n, and n *nodes* are added, the time complexity becomes $O(n)$. When the number of attack relations is m, and m *edges* are added, the time complexity becomes $O(m)$. Therefore, the overall time complexity is $O(n + m)$.

In "Get combinations", we need to find all combinations from 0 items to n-1 items. In this case, since we need to loop n times, the time complexity becomes $O(n)$. Moreover, the time complexity of finding combinations is $O({}_nC_r)$. Furthermore, in this algorithm, the overall time complexity is dominated by the two inner loops. Therefore, the final time complexity becomes $O(n \times (2^n))$.

In "Store all shrunken graph", when the number of combinations is r, the time complexity becomes $O(r)$ since all these combinations are used to shrink the AA graph. Next, to maintain the original graph, a copy of the graph needs to be created. In this case, the time complexity depends on the number of *nodes* and *edges* in the graph. If the number of nodes is n and

Table 4.1: Time Complexity

The algorithm	**Time complexity**
Make the original graph	$O(n+m)$
Get combinations and store them to list "pattern"	$O(n \times 2^n)$
Store all shrunken graph to list "Graphset"	$O(r \times (n+m+n \times k))$
Identify all graph's grounded and store the them to list "grounded"	$O(r \times (n+m+\alpha))$
Show "Hidden Arguments" for all graphs	$O(n)$
Display all the matched graphs	$O(m \times n^2)$

the number of edges is m, the time complexity of this operation is $O(n+m)$. And then, a nested loop is performed. First, it iterates over the *nodes* in the graph. If the number of *nodes* is n, the time complexity of this loop is $O(n)$. Next, an iteration is performed based on the number of elements in the set of combination families. If the number of elements is k, the time complexity of this loop is $O(k)$. Finally, within that loop, the deletion of nodes from the graph is performed. At this time, the average time complexity is $O(1)$, but in the worst case scenario, it becomes $O(n)$. Therefore, the overall time complexity becomes $O(r \times (n+m+n \times k))$.

In "Identify all graph's grounded and store them", the objective is to determine all grounded in each graph. If the number of graphs is r, we need to perform a loop r times to examine all graphs' grounded. At this point, the time complexity becomes $O(r)$. Next, as a preliminary step to examining the grounded, we write the *nodes* and *edges* of each graph to a file. If the number of *nodes* is n and the number of *edges* is m, the time complexity becomes $O(n+m)$. Finally, to examine the grounded, we send the file created earlier from Python to ASP for execution. Since the execution time of Clingo is independent, let's denote it as α. Thus, the time complexity becomes $O(\alpha)$. Both $O(n+m)$ and $O(\alpha)$ occur within the same loop, so the overall time complexity becomes $O(r \times (n+m+\alpha))$.

In "Show Hidden Arguments for all graphs", the time complexity depends on the number of graphs. If the number of graphs is n, then the overall time complexity becomes $O(n)$.

In "Display all the matched graphs", the goal is to draw all graphs that match specific conditions. If the number of graphs is m, we need to perform a

Table 4.2: The agent's roles when there are 6 arguments

Agent ID	**Agent's role**
A	Villager
B	Werewolf
C	Fortuneteller
D	Werewolf
E	Villager

Table 4.3: The number of arguments are 6.

Argument No.	**Agent ID**	**Argument**
1	C	A is not a werewolf by the result of divination.
2	C	B is a werewolf by the result of divination.
3	B	The result of divination are meaningless when fortuneteller C is false.
4	A	C is not confirmed to be a fake and should be used as a reference.
5	D	B is not a werewolf, so C is lying.
6	E	The one who doubts the result of divination is suspicious.

loop m times to draw each matched graph. At this point, the time complexity becomes $O(m)$. Next, network layout calculations are performed within the loop. The time complexity of these calculations depends on the number of the *nodes*. If the number of the *nodes* is n, the time complexity becomes $O(n)$ [16]. And then, the time complexity of drawing depends on the number of nodes and edges. Therefore, the time complexity becomes $O(n^2)$ [17, 18]. Since these occur within the loop, the overall time complexity becomes $O(m \times n^2)$.

4.3 The algorithm's Execution Time

4.3.1 Case 1: When There are 6 Arguments

In this subsection, we verify the execution time when there are 6 arguments and 5 agents. At this time, the roles of each agent are as follows: they become like Table 4.2, and the discussions become like Table 4.3.

Table 4.4: In the case of having 6 arguments, the algorithm's execution time

Algorithm	**Execution time**
Making AA graph	0.0002s
Getting combinations and store them to list "pattern"	0.0002s
Storing all shrunk graph to list "Graphset"	0.0027s
Identifying all graph's grounded and store them to list "grounded"	7.7537s
Specifying rows and columns for displaying graph	0.0004s
Displaying all graph	38.0151s
Total time	45.7696s

The time series of the discussions in Table 4.3 corresponds to the ascending order of argument number.
The discussion begins with Agent C, who is the fortune teller, declaring their role as in Argument No.1. Additionally, since Table 4.3 assumes discussions immediately after the first night, the only agent with information to advance the discussion is Agent C, the fortune teller who gains information during the first night. Therefore, to progress the discussion, the fortune teller, Agent C, reveals the result of their divination as in Argument No.2.

In Argument No.3, Agent B, who has been revealed as the werewolf through the divination, is making a defense to avoid being executed during the voting phase. In Argument No.4, Agent A is refuting Agent B's claim made in Argument No.3 that Agent C is the fake fortune teller. In Argument No.5, Agent D, an ally of Agent B who is a werewolf, is providing support to ensure that Agent B is not executed during the voting phase. Finally, in Argument No.6, Agent E expresses suspicion that Agents B and D, who are accusing Agent C of being a fake despite no one else declaring themselves as a fortune teller besides Agent C, might be werewolves.

In this flow, the villagers' faction and the werewolves' faction are engaging in discussions to fulfill their respective victory conditions.

The argument in Table 4.3 consists of $\{1,2,3,4,5,6\}$ and the attack relations is $\{(3,1),(3,2),(4,3),(5,4),(6,3),(6,5)\}$. In this case, the number of AA graphs, including the original AA graph, is 63. And the execution time for Table 4.3 is as shown in Table 4.4.

Table 4.5: The agent's roles when there are 10 arguments

Agent ID	**Agent's role**
A	Villager
B	Werewolf
C	Fortuneteller
D	Werewolf
E	Villager
F	Villager

Table 4.6: The number of arguments are 10.

Argument No.	**Agent ID**	**Argument**
1	C	I am the fortune teller.
2	C	A is not a werewolf by the result of divination.
3	B	I am the fortune teller.
4	B	E is the werewolf by the result of divination.
5	A	Since I am a villager, C is the true fortune teller.
6	D	Since B is the true fortune teller, C is the fake.
7	D	E is a werewolf.
8	E	I am a villager.
9	E	B and D is suspicious.
10	F	Since we don't know who is the true fortune teller, we should not use the result of divination.

4.3.2 Case 2: When There are 10 Arguments

In this subsection, we verify the execution time when there are 10 arguments and 6 agents. At this time, the roles of each agent are as follows: they become like Table 4.5, and the discussions become like Table 4.6.

In Table 4.6, just like in Table 4.3, the discussion begins with the declaration of roles by the fortune teller and the presentation of the divination's results. However, this time, in order to prevent Agent C from becoming the true fortune teller as in Argument No.3 and No.4, Agent B is falsely declaring themselves as the fortune teller and providing false the result of divination. In Argument No.5, Agent A is stating that he/she is a villager

Table 4.7: In the case of having 10 arguments, the algorithm's execution time

Algorithm	**Execution time**
Making AA graph	0.0003s
Getting combinations and store them to list "pattern"	0.0019s
Storing all shrunk graph to list "Graphset"	0.4137s
Identifying all graph's grounded and store them to list "grounded"	1.7978m
Specifying rows and columns for displaying graph	0.0041s
Displaying all graph	11.2837m
Total time	13.0886m

and affirming that the divination's result of Agent C are correct. Therefore, Agent A expresses the belief that Agent C is the true fortune teller. Then, in Argument No.6, Agent D, who is a werewolf, is making statements to support Agent B, who is also a werewolf and pretending to be the fortune teller, in order to deceive others and protect Agent B. Furthermore, in Argument No.7, Agent D is supporting Agent B by referencing Argument No.4 and stating that Agent E is a werewolf in order to direct the hatred of the other agents towards Agent E. Next, in Argument No.8, Agent E declares his/her role, and in Argument No.9, Agents B and D claims that Agent E is a werewolf, which makes Agent E suspicious of both them. Finally, in Argument No.10, Agent F points out that based solely on the arguments presented in the current discussion, it is impossible to determine which of Agent B or C is the true fortune teller. Therefore, Agent F suggests disregarding the results of divination and continuing the discussion.

The argument in Table 4.6 consists of $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ and the attack relations is $\{(3, 1), (1, 3), (4, 8), (8, 4), (5, 3), (6, 1), (7, 8), (8, 7), (9, 3), (9, 4), (9, 6), (9, 7), (10, 2), (10, 4), (10, 5), (10, 7)\}$. In this case, the number of AA graphs, including the original AA graph, is 1023. And the execution time for Table 4.6is as shown in Table 4.7.

4.3.3 Case 3: When There are 14 Arguments

In this subsection, we verify the execution time when there are 14 arguments and 6 agents. At this time, the roles of each agent are as follows: they become like Table 4.8, and the discussions become like Table 4.9.

Argument No.1 through No.4 follows the same pattern as the previous

Table 4.8: The agent's roles when there are 14 arguments

Agent ID	**Agent's role**
A	Villager
B	Werewolf
C	Fortuneteller
D	Werewolf
E	Villager
F	Knight

subsection, with both true and false fortune tellers declaring their roles and presenting their results of divination. Next, in Argument No.5, Agent A is expressing the same sentiment as Argument No.5 in Table 4.6. In Argument No.6, leveraging the statements made in Argument No.5, Agent D is echoing similar sentiments, supporting Agent B from the werewolf faction. Furthermore, utilizing Arguments No.5 and No.6, Agent D is diverting attention away from him/herself and Agent B by making Argument No.7 and No.8. Argument No.9 follows the proposal made in Argument No.8. In this argument, Agent E declares his/her role and advocates for the execution of someone other than him/herself. In Argument No.10, Agent F asserts that as a member of the village faction with special abilities, he/she should not be executed first. Then, in Argument No.11, Agent F states that if following Argument 8, then Agent E should be the one to be executed. However, Agent F holds a different opinion and proposes Arguments No.13 and No.14.
The argument in Table 4.9 consists of $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14\}$ and the attack relations is $\{(3, 1), (1, 3), (5, 3), (6, 1), (10, 9), (9, 10), (11, 9),$ $(12, 2), (12, 4), (12, 7), (12, 8), (13, 2), (13, 4), (13, 5), (13, 6), (14, 1), (14, 3),$ $(14, 8)\}$. In this case, the number of AA graphs, including the original AA graph, is 16385. And the execution time for Table 4.9 is as shown in Table 4.10.

The phrase "12h or more" means that the algorithm was running for at least 12 hours but did not complete processing. In this instance, the algorithm ran for approximately 16 hours without completing processing. To ensure consistency in future occurrences, it has been decided to use the notation "12h or more" in all cases where the algorithm runs for at least 12 hours without completion.

Table 4.9: The number of arguments are 14.

Argument No.	Agent ID	Argument
1	C	I am the fortune teller.
2	C	A is not a werewolf by the result of divination.
3	B	I am the fortune teller.
4	B	D is not the werewolf by the result of divination.
5	A	Since I am a villager, C is the true fortune teller.
6	D	Since B is the true fortune teller, B is the true.
7	D	We should refer to the result of divination.
8	D	Those who don't have the result of divination and have no role should be executed.
9	E	Since I am a villager, F should be executed first.
10	F	You should not execute me because I am a knight.
11	F	If we are going to execute them, we should do it from E.
12	F	However, the result of divination should not be used as a guide.
13	F	It is possible that either A or D is a werewolf.
14	F	Definitely one of them is a werewolf, B or C should be executed first.

4.3.4 Case 4: When There are 18 Arguments

In this subsection, we verify the execution time when there are 18 arguments and 7 agents. At this time, the roles of each agent are as follows: they become like Table 4.11, and the discussions become like Table 4.12.

Argument No.1 through No.4 follow the same pattern as the previous subsection, with both true and false fortune tellers declaring their roles and presenting their results of divination. However, in Table 4.12, unlike the previous three subsections, there is the presence of Agent G, who is a maniac. Agent G, despite being a maniac aligned with the werewolf faction, is not identified as a werewolf in the result of divination. Utilizing this fact, Agent G deceives others by falsely claiming to be the fortune teller. This is a strategy employed by Agents B and D to avoid taking risks and prevent the confirmation of the true identity of the fortune teller. In Argument No.5 and No.6, Agent A is attempting to advance the discussion by using the results of divination. Then, Agent B realizes from Argument No.4 that Agent G is a maniac, and in Argument No.7 and No.8, Agent B is

Table 4.10: In the case of having 14 arguments, the algorithm's execution time

Algorithm	Execution time
Making AA graph	0.0003s
Getting combinations and store them to list "pattern"	0.0275s
Storing all shrunk graph to list "Graphset"	3.1950s
Identifying all graph's grounded and store them to list "grounded"	28.7036m
Specifying rows and columns for displaying graph	0.0101s
Displaying all graph	12h or more
Total time	12.48h or more

Table 4.11: The agent's roles when there are 18 arguments

Agent ID	Agent's role
A	Villager
B	Werewolf
C	Fortuneteller
D	Werewolf
E	Villager
F	Knight
G	Maniac

supporting Agent G. Agent E is certain that either Agent C or G is a member of the werewolf faction. Therefore, in Argument No.9, Agent E expresses a critical opinion regarding Argument No.5. Furthermore, based on the aforementioned reasons, Agent E suggests that either Agent C or G should be executed first. Next, Agent F criticizes Argument No.10 in Arguments No.11 and No.12, arguing that sequentially executing both a powerless maniac and a powerful true fortune teller does not benefit the villagers' faction. Agent D agrees with Argument No.6 and expresses their opinion in Argument No.13. Agent F also makes a statement in Argument No.14 that is similar to Argument No.13. In Argument No.15, Agent A is suspicious of Agent B, who claims that G is the true fortune teller. Following this, Agent A expresses their opinion in Argument No.16. Argument No.17 is also just the opinion of Agent E. In Argument No.18, Agent G is suspicious of discussing something when they don't know who the true fortune teller is. The argument in Table 4.12 consists of $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,$

Table 4.12: The number of arguments are 18.

Argument No.	Agent ID	Argument
1	C	I am the fortune teller.
2	C	A is not a werewolf by the result of divination.
3	G	I am the fortune teller.
4	G	B is not the werewolf by the result of divination.
5	A	We should refer to the result of divination.
6	A	Those who don't have the result of divination and have no role should be executed.
7	B	Since I am a villager, G is the true fortune teller.
8	B	C should be executed first.
9	E	The result of divination is not helpful.
10	E	C or G should be executed first.
11	F	Either C or G could be the maniac.
12	F	C or G should not be executed.
13	D	Since I am a villager, I think that E or F should be executed first. - not me.
14	F	Since I am the knight, D or E should be executed first.
15	A	B, who assume that G is the true fortune teller, is suspicious.
16	A	G is the fake fortune teller.
17	E	Since D has a little to say, he/she should be executed first.
18	G	It is strange that people are now talking about the truth or falsehood or fortune teller.

$15, 16, 17, 18\}$ and the attack relations is $\{(3,1), (1,3), (7,1), (9,2), (9,4), (9,6), (5,9), (9,5), (11,1), (11,3), (12,8), (12,10), (14,13), (15,7), (16,3), (17,13), (18,7), (18,16)\}$. In this case, the number of AA graphs, including the original AA graph, is 262145. And the execution time for Table 4.12 is as shown in Table 4.13.

4.3.5 Case 5: When There are 20 Arguments

In this subsection, we verify the execution time when there are 20 arguments and 7 agents. At this time, the roles of each agent are as follows: they become like Table 4.14, and the discussions become like Table 4.15.

Table 4.13: In the case of having 18 arguments, the algorithm's execution time

Algorithm	**Execution time**
Making AA graph	0.0003s
Getting combinations and store them to list "pattern"	0.1127s
Storing all shrunk graph to list "Graphset"	38.9800s
Identifying all graph's grounded and store them to list "grounded"	7.6682h
Specifying rows and columns for displaying graph	0.0773s
Displaying all graph	12h or more
Total time	19.68h or more

Table 4.14: The agent's roles when there are 20 arguments

Agent ID	**Agent's role**
A	Villager
B	Werewolf
C	Fortuneteller
D	Werewolf
E	Villager
F	Knight
G	Maniac

Argument No.1 through No.6 follow the same pattern as the previous subsection, with both true and false fortune tellers declaring their roles and presenting their results of divination. Agent B believes that either Agent C or Agent G is the Maniac, and if the true fortune teller can also execute, then it's acceptable to execute the Maniac. Therefore, in Arguments No.7 and No.8, he/she express opinions on executing the two individuals sequentially. Agent E agrees with Arguments No.4 and No.5, and then expresses his/her own thoughts in Arguments No.9 and No.10. Next, in Argument No.11, Agent F declares his/her role to bolster his/her argument. Utilizing Argument No. 11, Agent F expresses his/her opinions in Arguments No.12 and No.13. Agent D, to avoid conceding the opinions from Arguments No.11 to No.13, similarly declares the role of the knight in Argument No.14, following the example set by Agent F. Furthermore, Agent D criticizes Agent F in Argument No.15. Agent E, on the other hand, presents opinions in Arguments No.16 and

No.17 based on Arguments No.11 to No.15. Agent B criticizes Agent E for presenting Argument No. 9 in Argument No. 18. Finally, Agent C expresses the opinion in Argument No. 19 that regardless of which one is the true knight, both should be executed over the course of two turns, rendering their identities irrelevant. In Argument No. 20, Agent C sets the direction for the discussion.

The argument in Table 4.15 consists of $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20\}$ and the attack relations is $\{(3, 1), (1, 3), (7, 2), (7, 4), (5, 7), (7, 5), (7, 6), (8, 6), (9, 1), (9, 3), (10, 8), (11, 14), (14, 11), (12, 10), (13, 7), (15, 12), (12, 15), (16, 14), (17, 15), (18, 16), (19, 15), (19, 17), (20, 15)\}$. In this case, the number of AA graphs, including the original AA graph, is 1048577. And the execution time for Table 4.15 is as shown in Table 4.16.

Table 4.15: The number of arguments are 20.

Argument No.	**Agent ID**	**Argument**
1	C	I am the fortune teller.
2	C	A is not a werewolf by the result of divination.
3	G	I am the fortune teller.
4	G	E is not the werewolf by the result of divination.
5	A	We should refer to the result of divination.
6	A	Those who don't have the result of divination and have no role should be executed.
7	B	The result of divination is not helpful.
8	B	C or G should be executed first.
9	E	C or G may be the maniac.
10	E	B, D or F should be executed first.
11	F	I am the knight.
12	F	You should not execute me.
13	F	It is strange to make a statement that ignores the results of divination.
14	D	I am the knight.
15	D	F should be executed first.
16	E	Since F claimed to be the knight first, so F is the true knight.
17	E	D should be executed first.
18	B	E, who assume that D is the fake fortune teller, is strange.
19	C	We can execute D and F consecutively over two turns, so it doesn't matter which one start with for the execution.
20	C	We execute D first.- then F.

Table 4.16: In the case of having 20 arguments, the algorithm's execution time

Algorithm	**Execution time**
Making AA graph	0.0007s
Getting combinations and store them to list ”pattern”	0.5555s
Storing all shrunk graph to list “Graphset”	3.4551m
Identifying all graph's grounded and store them to list “grounded”	12h or more
Specifying rows and columns for displaying graph	0.1520s
Displaying all graph	12h or more
Total time	24.05h or more

Chapter 5

Conclusion

5.1 Concluding Remark

We analyzed a formalization of which arguments are hidden or ignored in order to get someone to accept a certain claim in this thesis.

To achieve this, both AA and the custom-defined Dynamic Abstract Argumentation were necessary. Dynamic Abstract Argumentation dynamically alters the arguments accepted in discussions by manipulating the *nodes* and *edges* of the AA framework or graph. This enables the identification of all potentially changing shrunken AA graphs. Then, using these shrunken graphs and the original graph, it is possible to analyze the Hidden Arguments of each shrunken graph. Furthermore, by storing the grounded of all these shrunken AA graphs, it becomes possible to call and display these graphs. While the explanation was demonstrated using the example of the Werewolf game, these attempts can be considered successful.

However, there are several challenges. As mentioned in Section 4.3, the execution time of the algorithm is not a concern for simple argument instances, but it significantly increases when the number of shrunken AA graphs, such as 16385, becomes large. Furthermore, the increase in the number of shrunken AA graphs begins to have a significant impact on the calculation of bases. Due to this impact, if the grounded of all shrunken graphs cannot be obtained, it becomes impossible to analyze which arguments are hidden to make desirable arguments accepted by others. This is considered a serious issue.

The second issue is that it cannot yet be applied to realistic, serious cases like court cases. In addition to the aforementioned problem with execution time, there is an issue with how arguments and attack relations are inputted. The current algorithm requires manual input of information, so it becomes impractical for complex cases with a large number of arguments, such as those found in court cases, due to the enormous amount of time required and thus it is not practical.

From these challenges, it can be inferred that theories like Dynamic

Abstract Argumentation are not flawed, but there might be issues with the development approach of the algorithm.

In this thesis we analyzed which arguments are hidden or ignored in order to get someone to accept a certain claim, and this experiment itself could be considered successful. However, there are still several issues remaining, and these need to be resolved to make it practical.

5.2 Future Directions

My future design is as follows:

1. Extend the current framework using Argumentation-based Agent (ABA) [14] approach. Unlike AA which lacks structure for abstract concepts, ABA arguments have a structure [8]. This enables expressing ambiguous parts like hypotheses.
2. In this study, it is necessary to extend the current framework to represent not only attack relationships, such as "a attacks b", but also trust relationships, such as "a trusts b". This is because in reality, there are often allies who support each other's claims rather than criticizing each other. If we were to represent trust relationships, the AA graph would be expected to resemble Figure 5.1. To express this, new methods like Trust-Based Argumentation Framework (TBAF) [13] and ABA would need to be utilized to expand the current framework. In simple terms, TBAF is a framework where all arguments within it are mapped to trust values for comparison and ranking [13]. Additionally, each argument is mapped to an attack level, indicating the extent to which it is being attacked by other arguments [13].
3. Address the problems outlined in the sections. Without resolving these, it cannot be considered practical. Therefore, exploring new approaches by revisiting algorithms is necessary. Additionally, combining methods such as ABA and TBAF might open up new avenues.

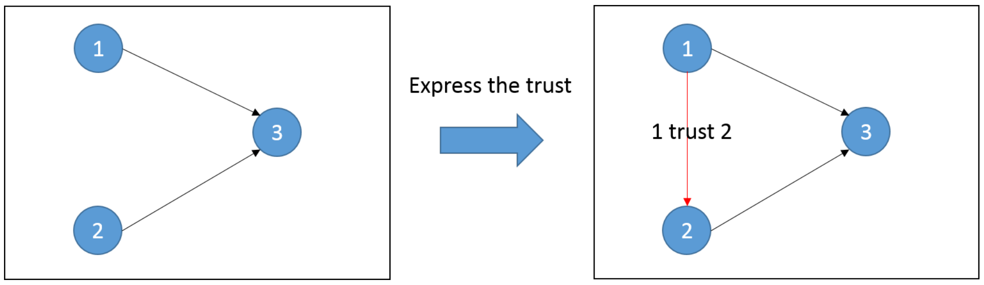


Figure 5.1: Express the trust relationship

Appendix A

Example

We consider the input for the algorithm as “arguments = $\{1, 2, 3, 4\}$, attacks = $\{(1, 2), (2, 3), (1, 3), (3, 4)\}$”.

An AA graph is created in the Figure B.3 based on the aforementioned arguments and attacks. In this case, the execution result of Figure B.4 would be as depicted in the figure A.1.

Next, the algorithm is executed from Figure B.5 to FigureB.6. Subsequently, the grounded obtained from these operations are verified in Figure B.7. The result is as depicted in Figure A.2.

And then, the Hidden Arguments were analyzed by executing the *nodes* of the original graph and the *nodes* of each graph stored in the list “Graphset” to produce Figure B.8. The result is as depicted in Figure A.3.

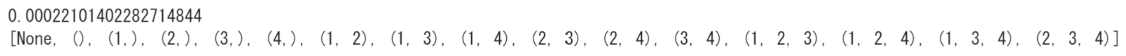
When executing Figure B.9, the input would be as depicted in Figure A.4. As a result, the list “check” would contain $1, 2$.

When executing Figure B.10, the list “check” would contain the indices of graphs that include the grounded 2. When the list ”grounded” corresponds to the one from Figure A.2, then the contents of “check” would be $\{2, 7, 8, 14\}$. Next, we calculate the rows and columns for displaying the graphs that include the grounded 2. The result of this execution would be as depicted in Figure A.5.

In Figure B.11, the rows and columns are being calculated for displaying all graphs. The result of this execution would be as depicted in Figure A.6.

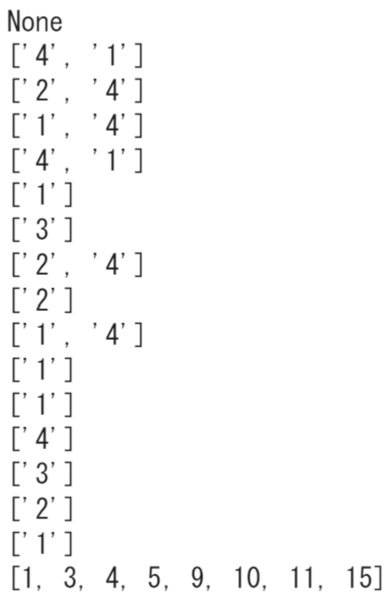
In Figure B.12, the display of specific graphs is executed using the list “check”, rows, and columns obtained from Figure B.10. We are using the execution result from Figure A.5 this time. In this case, the result of the execution in Figure B.12 would be as depicted in Figure A.7.

In Figure B.13, all graphs are displayed and then converted into PDF format. Using the list “grounded” from Figure A.2 and the rows and columns obtained from Figure A.6, the execution result would resemble Figures A.8 to A.11. In this case, the title font size in Figure B.13 is set to 10, but it has been changed to 20.



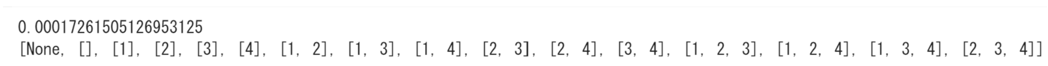
```
0.00022101402282714844
[None, (), (1,), (2,), (3,), (4,), (1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4), (1, 2, 3), (1, 2, 4), (1, 3, 4), (2, 3, 4)]
```

Figure A.1: The execution result of Figure B.4



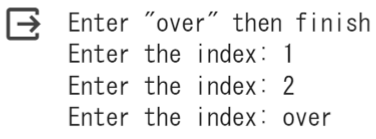
```
None
['4', '1']
['2', '4']
['1', '4']
['4', '1']
['1']
['3']
['2', '4']
['2']
['1', '4']
['1']
['1']
['4']
['3']
['2']
['1']
[1, 3, 4, 5, 9, 10, 11, 15]
```

Figure A.2: The execution result of Figure B.7



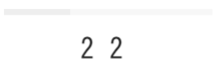
```
0.00017261505126953125
[None, [], [1], [2], [3], [4], [1, 2], [1, 3], [1, 4], [2, 3], [2, 4], [3, 4], [1, 2, 3], [1, 2, 4], [1, 3, 4], [2, 3, 4]]
```

Figure A.3: The execution result of Figure B.8



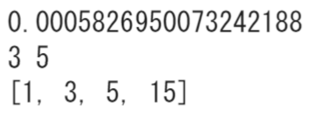
```
Enter "over" then finish
Enter the index: 1
Enter the index: 2
Enter the index: over
```

Figure A.4: The execution result of Figure B.9



```
2 2
```

Figure A.5: The execution result of Figure B.10



```
0.0005826950073242188
3 5
[1, 3, 5, 15]
```

Figure A.6: The execution result of Figure B.10

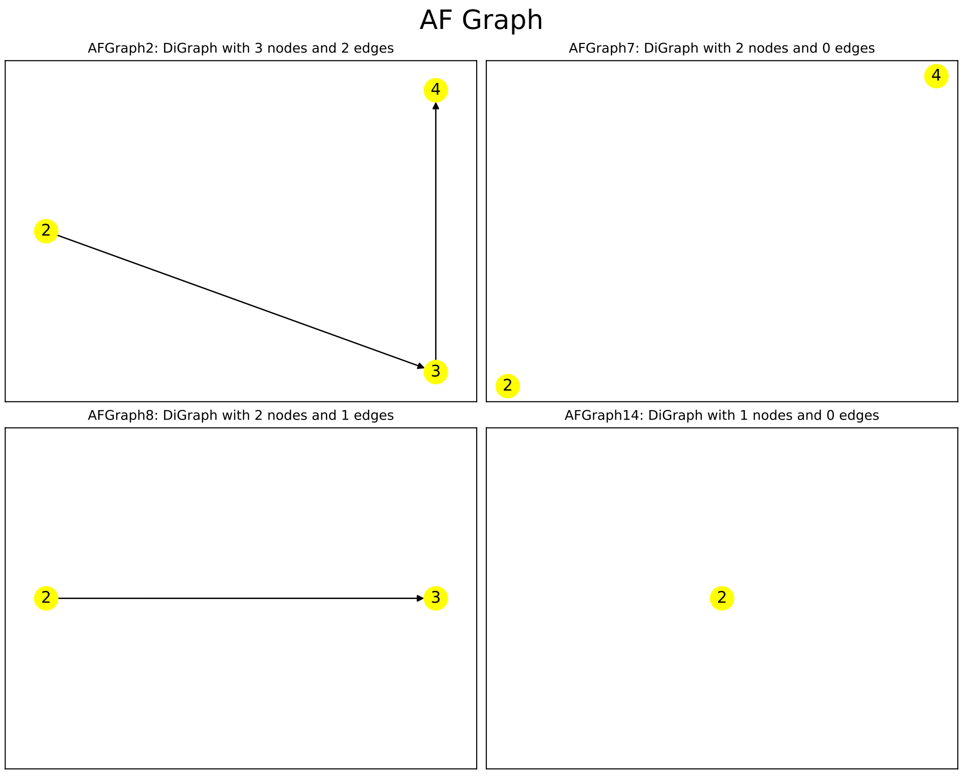

AF Graph
AFGraph2: DiGraph with 3 nodes and 2 edges
4
2
3
AFGraph7: DiGraph with 2 nodes and 0 edges
4
2
AFGraph8: DiGraph with 2 nodes and 1 edges
2
3
AFGraph14: DiGraph with 1 nodes and 0 edges
2

Figure A.7: The execution result of Figure B.12

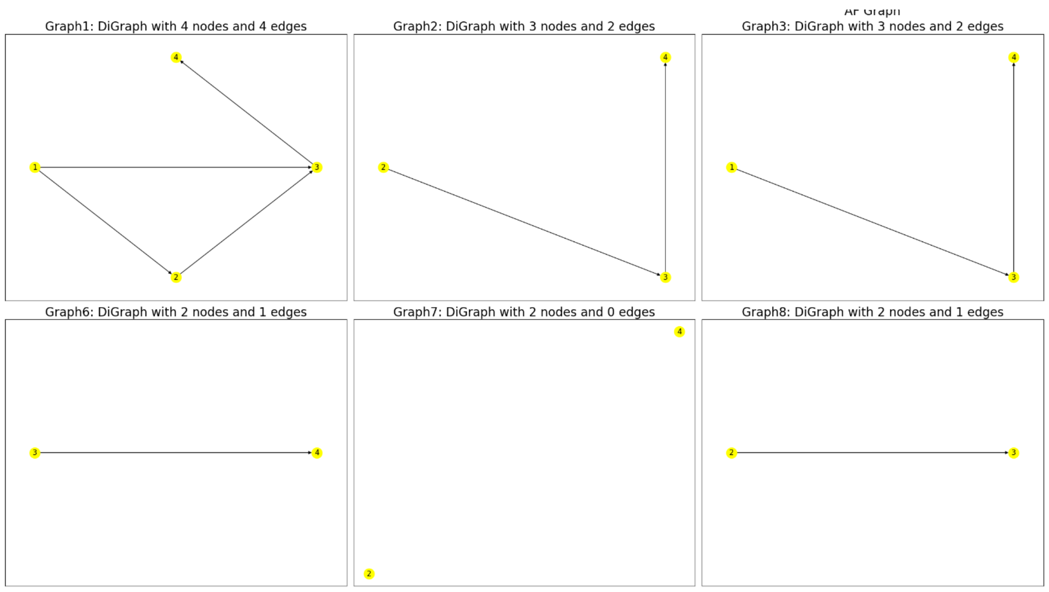

AF Graph
Graph1: DiGraph with 4 nodes and 4 edges
Graph2: DiGraph with 3 nodes and 2 edges
Graph3: DiGraph with 3 nodes and 2 edges
Graph6: DiGraph with 2 nodes and 1 edges
Graph7: DiGraph with 2 nodes and 0 edges
Graph8: DiGraph with 2 nodes and 1 edges
1
2
3
4

Figure A.8: The execution result of Figure B.13

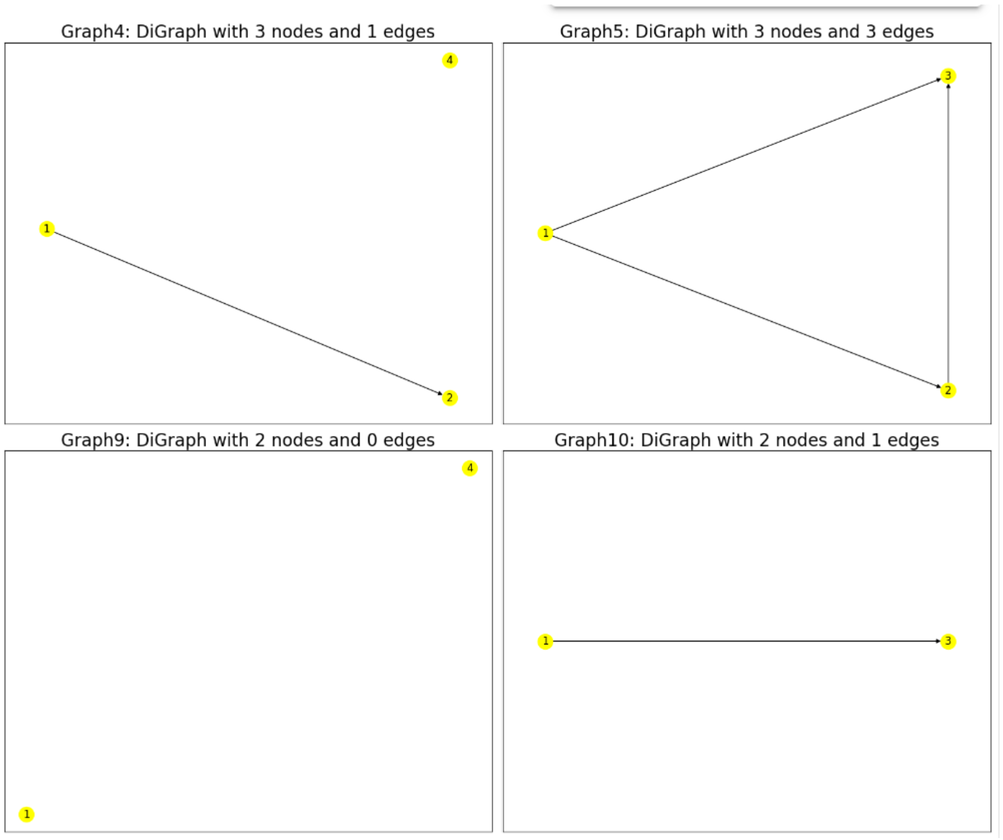
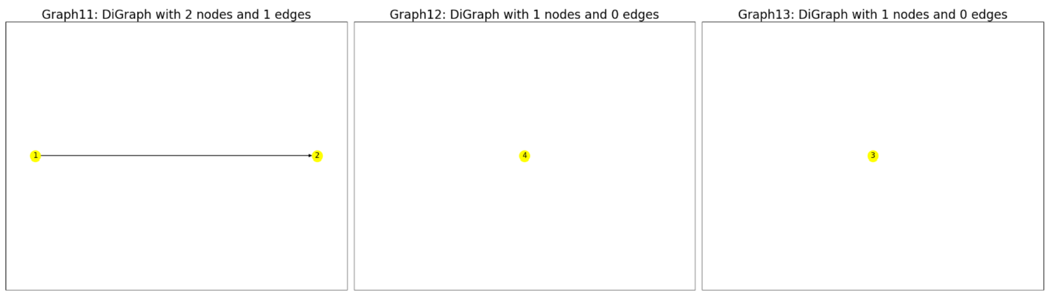

Graph4: DiGraph with 3 nodes and 1 edges
4
1
2
Graph5: DiGraph with 3 nodes and 3 edges
3
1
2
Graph9: DiGraph with 2 nodes and 0 edges
4
1
Graph10: DiGraph with 2 nodes and 1 edges
1
3

Figure A.9: The execution result of Figure B.13


Graph11: DiGraph with 2 nodes and 1 edges
1
2
Graph12: DiGraph with 1 nodes and 0 edges
4
Graph13: DiGraph with 1 nodes and 0 edges
3

```
9.001205444335938
```

Figure A.10: The execution result of Figure B.13

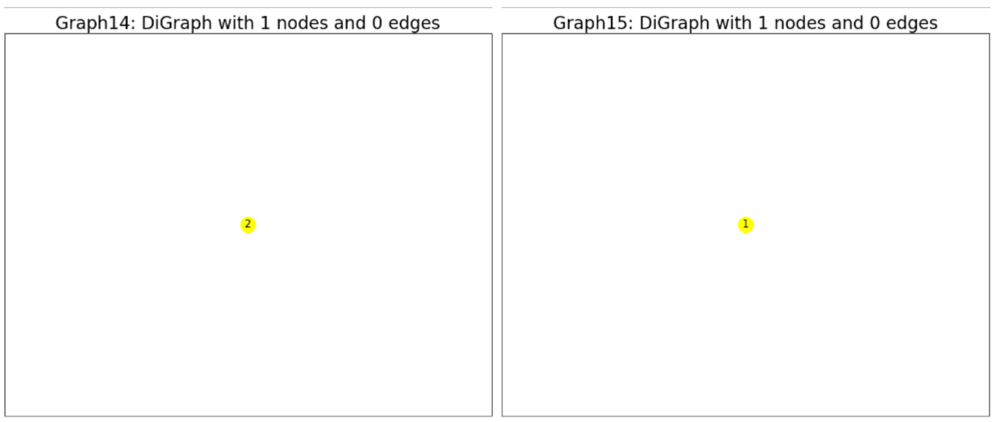

Graph14: DiGraph with 1 nodes and 0 edges
2
Graph15: DiGraph with 1 nodes and 0 edges
1

Figure A.11: The execution result of Figure B.13

Appendix B

Code

In Figure B.1, Matplotlib and NetworkX are introduced to enable graph plotting, and the time tool is introduced for measuring execution time. Additionally, three lists for arguments, attack relation, and index of graph are declared, along with a directed graph G for storing them.

In Figure B.2, values to be included in arguments and attack relations are predetermined before executing the algorithm.

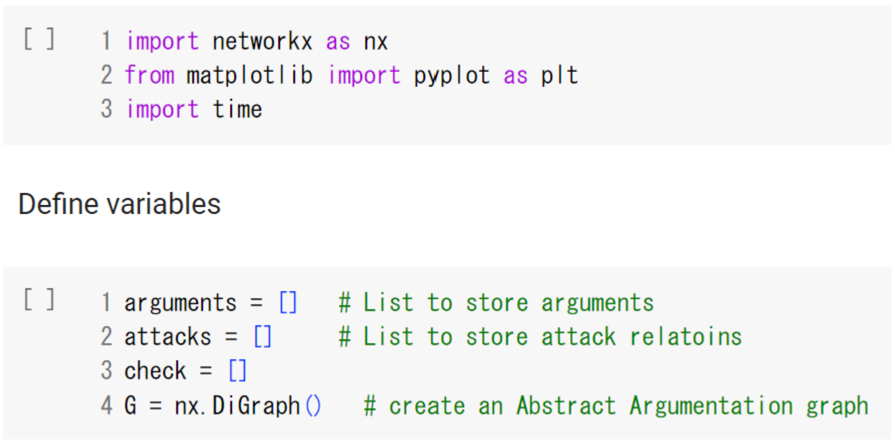
In Figure B.3, arguments are assigned to the *nodes* of the directed graph G, while attack relations are assigned to the *edges*. Subsequently, the execution time is measured.

In Figure B.4, the modules re and itertools are first imported. Then, three lists are declared, with "None" inputted into the 0th index of these lists to make it unusable. Subsequently, we store $\emptyset$ in the list "pattern". Next, when the number of arguments is n ($n > 0$, $n \in \mathbb{N}$), store all combinations of selecting 0 to n-1 elements in the list "pattern". Finally, we measure the execution time to store it in a list "pattern". Here, the *nodes* and *edges* of the original the AA graphs are associated with the arguments and attack relations from Figure B.2.

In Figure B.5, all Hidden Argument annotated AA graphs are stored in the list "Graphset". To do this, the AA graphs are shrunk using the list "pattern" obtained in Figure B.4. Subsequently, the execution times for this process are measured. Here, the *nodes* and *edges* of the original the AA graphs are associated with the arguments and attack relations from Figure B.2.

In Figure B.6, the grounded of all AA graphs stored in the list "Graphset" are stored in the list "grounded". During this process, the grounded are obtained by executing ASP from Python. Subsequently, the execution times for these process are measured. Here, the *nodes* and *edges* of the original the AA graphs are associated with the arguments and attack relations from Figure B.2.

In Figure B.7, the correctness of the grounded is being verified to ensure they are properly instantiated. Here, the *nodes* and *edges* of the original the AA graphs are associated with the arguments and attack relations from


```
[ ]  import networkx as nx
     from matplotlib import pyplot as plt
     import time
```

Define variables

```
[ ]  arguments = []   # List to store arguments
     attacks = []     # List to store attack relatoins
     check = []
     G = nx.DiGraph()    # create an Abstract Argumentation graph
```

Figure B.1: Install tools and define variables

Figure B.2.

In Figure B.8, The Hidden Arguments of each shrunken graphs are analyzed and stored in the list “Hidden”. According to Definition 3.1, Hidden Arguments are identified by executing “the *nodes* of the original graph \ the *nodes* of the shrunken graph”.

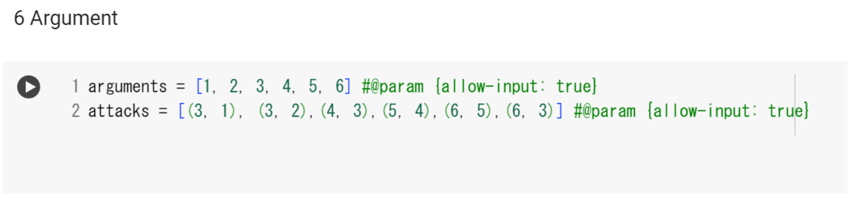
In Figure B.9, the index of AA graphs that are to be compared are manually inputted during the execution of the algorithm. Here,he *nodes* and *edges* of the original the AA graphs are associated with the arguments and attack relations from Figure B.2.

In Figure B.10, the indices of AA graphs containing specific grounded are being stored in the list “check”. Here, the *nodes* and *edges* of the original the AA graphs are associated with the arguments and attack relations from Figure B.2. In Figure B.10, the index of AA graphs containing the grounding 2 is being stored. Next, we calculate the rows and columns for displaying the graphs to be compared.

In Figure B.11, the rows and columns are being calculated for displaying all AA graphs. Subsequently, the execution time is measured. Here, the *nodes* and *edges* of the original the AA graphs are associated with the arguments and attack relations from Figure B.2.

In Figure B.12, the list “check” is used to display the specific AA graphs. And then, those displayed AA graphs are converted into PDF format.

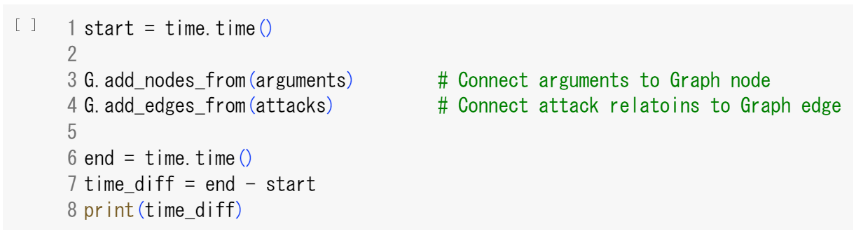
In Figure B.13, all AA graphs are displayed. And then, those displayed AA graphs are converted into PDF format. Subsequently, the execution time is measured.



6 Argument

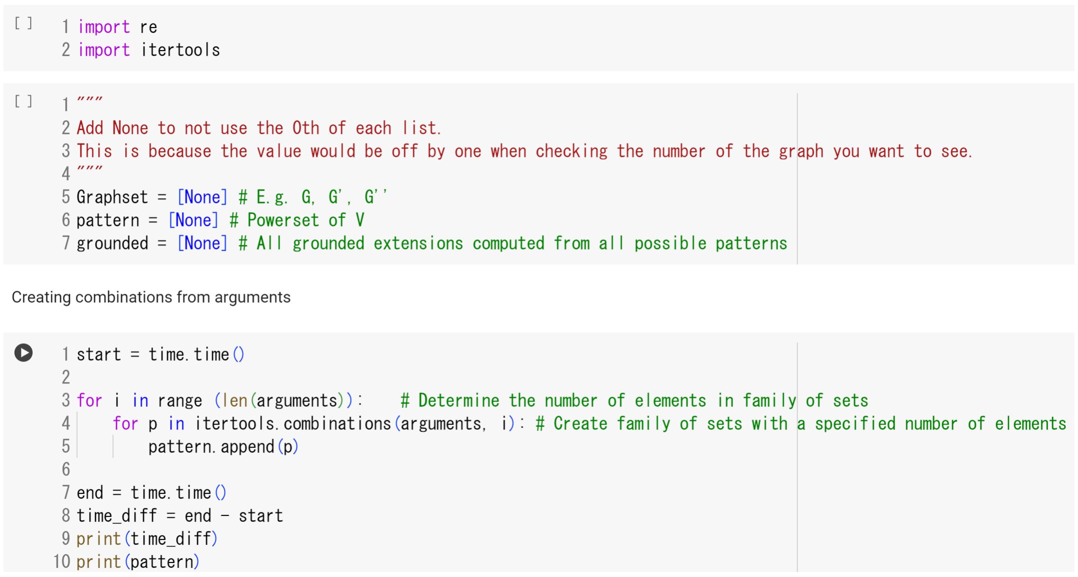
```python
arguments = [1, 2, 3, 4, 5, 6] #@param {allow-input: true}
attacks = [(3, 1), (3, 2), (4, 3), (5, 4), (6, 5), (6, 3)] #@param {allow-input: true}
```

Figure B.2: Predetermine values before executing the algorithm.



```python
start = time.time()

G.add_nodes_from(arguments)         # Connect arguments to Graph node
G.add_edges_from(attacks)           # Connect attack relatoins to Graph edge

end = time.time()
time_diff = end - start
print(time_diff)
```

Figure B.3: argument and attack relations are assigned to the directed graph G



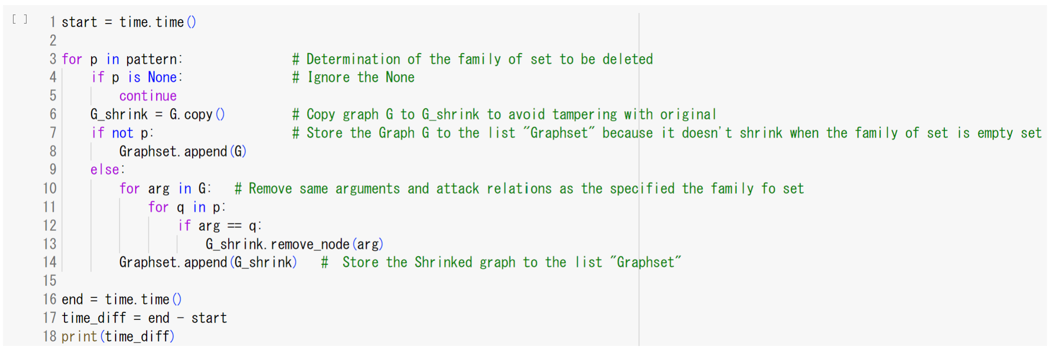
```python
import re
import itertools
```

```python
"""
Add None to not use the 0th of each list.
This is because the value would be off by one when checking the number of the graph you want to see.
"""
Graphset = [None] # E.g. G, G', G''
pattern = [None] # Powerset of V
grounded = [None] # All grounded extensions computed from all possible patterns
```

Creating combinations from arguments

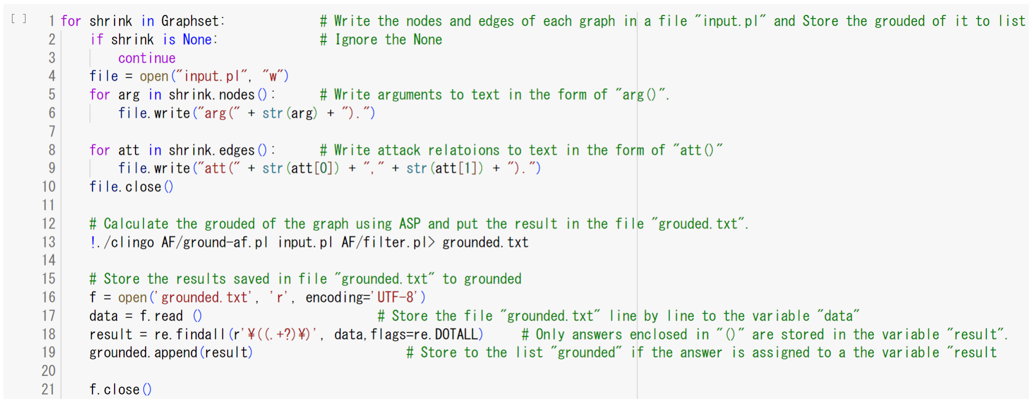
```python
start = time.time()

for i in range (len(arguments)):    # Determine the number of elements in family of sets
    for p in itertools.combinations(arguments, i): # Create family of sets with a specified number of elements
        pattern.append(p)

end = time.time()
time_diff = end - start
print(time_diff)
print(pattern)
```

Figure B.4: store combinations in the list



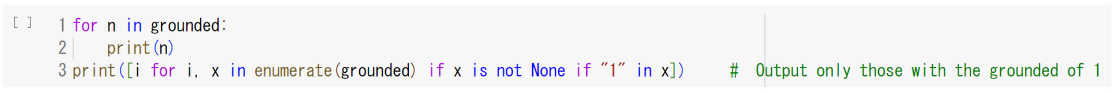
```
[ ]
start = time.time()

for p in pattern:                # Determination of the family of set to be deleted
    if p is None:                # Ignore the None
        continue
    G_shrink = G.copy()          # Copy graph G to G_shrink to avoid tampering with original
    if not p:                    # Store the Graph G to the list "Graphset" because it doesn't shrink when the family of set is empty set
        Graphset.append(G)
    else:
        for arg in G:   # Remove same arguments and attack relations as the specified the family fo set
            for q in p:
                if arg == q:
                    G_shrink.remove_node(arg)
        Graphset.append(G_shrink)    #  Store the Shrinked graph to the list "Graphset"

end = time.time()
time_diff = end - start
print(time_diff)
```

Figure B.5: Store all Hidden Argument of AA graph in the list



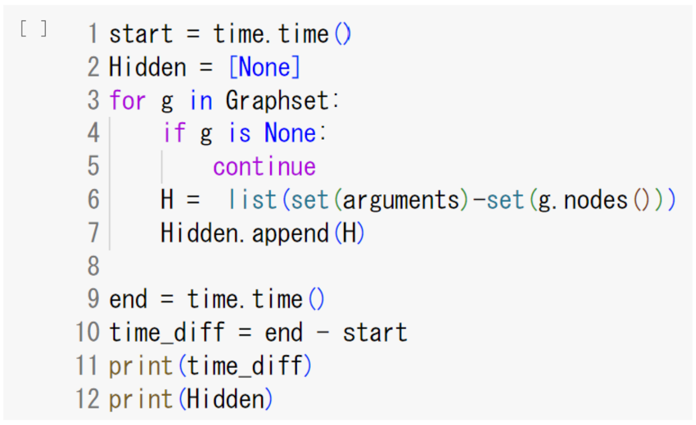
```
[ ]
for shrink in Graphset:            # Write the nodes and edges of each graph in a file "input.pl" and Store the grouded of it to list
    if shrink is None:             # Ignore the None
        continue
    file = open("input.pl", "w")
    for arg in shrink.nodes():     # Write arguments to text in the form of "arg()".
        file.write("arg(" + str(arg) + ").")

    for att in shrink.edges():     # Write attack relatoions to text in the form of "att()"
        file.write("att(" + str(att[0]) + "," + str(att[1]) + ").")
    file.close()

    # Calculate the grouded of the graph using ASP and put the result in the file "grouded.txt".
    !./clingo AF/ground-af.pl input.pl AF/filter.pl> grounded.txt

    # Store the results saved in file "grounded.txt" to grounded
    f = open('grounded.txt', 'r', encoding='UTF-8')
    data = f.read ()                        # Store the file "grounded.txt" line by line to the variable "data"
    result = re.findall(r'¥((.+?)¥)', data,flags=re.DOTALL)     # Only answers enclosed in "()" are stored in the variable "result".
    grounded.append(result)                     # Store to the list "grounded" if the answer is assigned to a the variable "result

    f.close()
```

Figure B.6: Calculate the grounded extensions



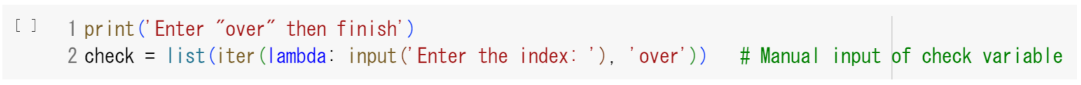
```
[ ]
for n in grounded:
    print(n)
print([i for i, x in enumerate(grounded) if x is not None if "1" in x])     #  Output only those with the grounded of 1
```

Figure B.7: verify the correctness of the grounded



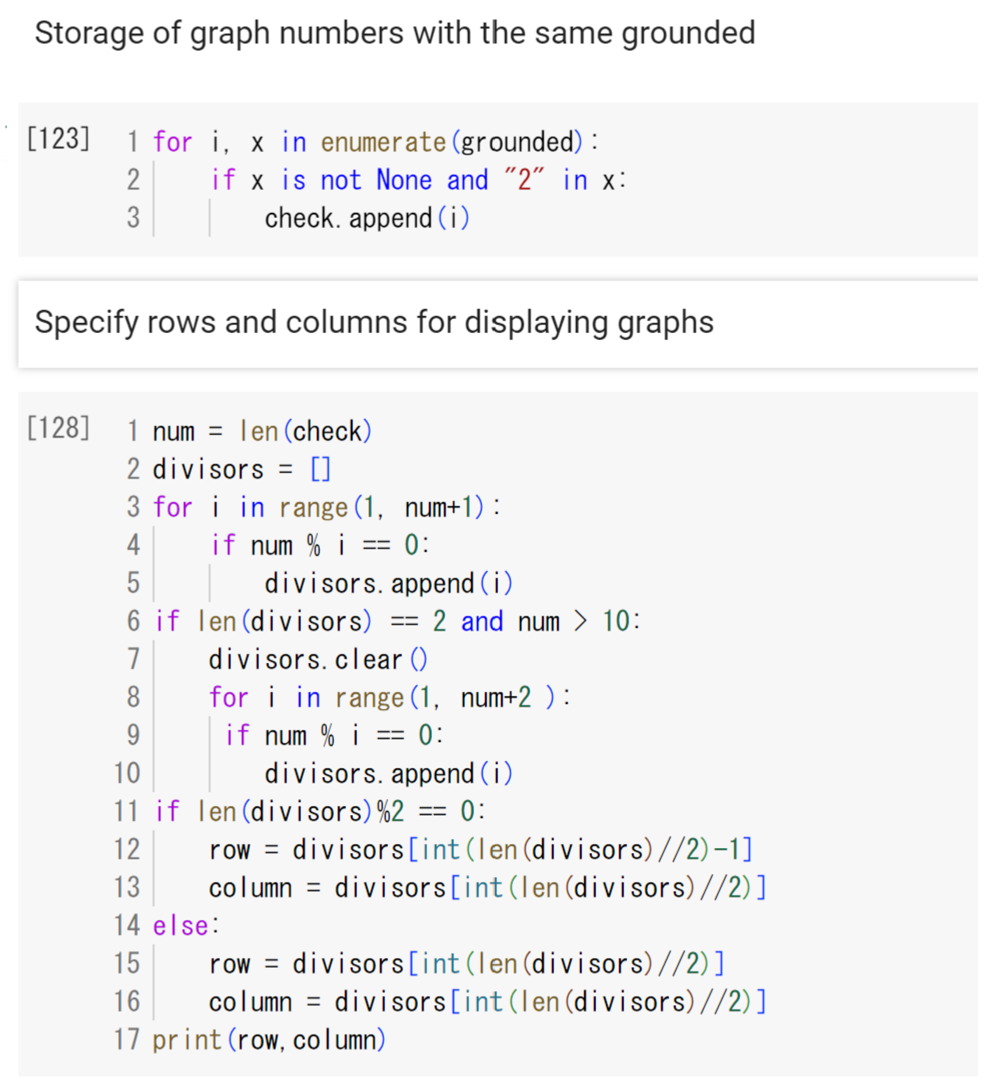
```python
start = time.time()
Hidden = [None]
for g in Graphset:
    if g is None:
        continue
    H =  list(set(arguments)-set(g.nodes()))
    Hidden.append(H)

end = time.time()
time_diff = end - start
print(time_diff)
print(Hidden)
```

Figure B.8: Analyze Hidden Arguments



```python
print('Enter "over" then finish')
check = list(iter(lambda: input('Enter the index: '), 'over'))   # Manual input of check variable
```

Figure B.9: The index of AA graphs are manually inputted during the execution of the algorithm



Storage of graph numbers with the same grounded

[123]

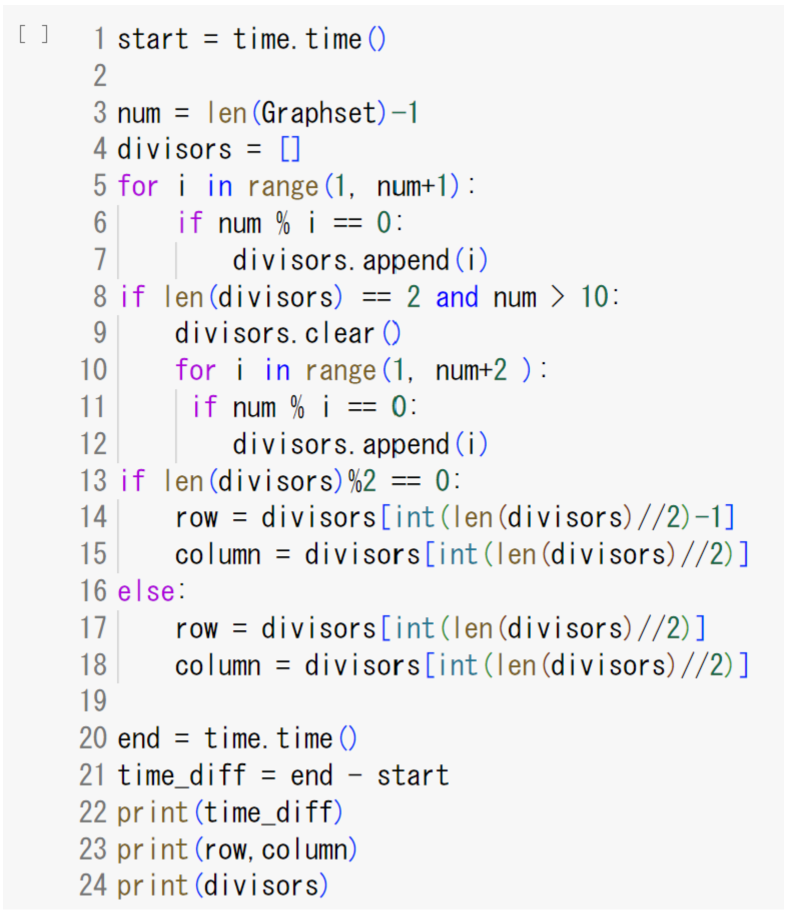
```python
for i, x in enumerate(grounded):
    if x is not None and "2" in x:
        check.append(i)
```

Specify rows and columns for displaying graphs

[128]

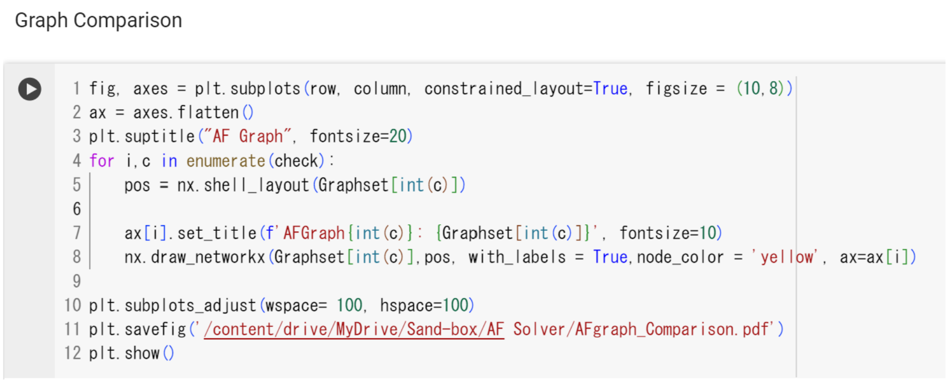
```python
num = len(check)
divisors = []
for i in range(1, num+1):
    if num % i == 0:
        divisors.append(i)
if len(divisors) == 2 and num > 10:
    divisors.clear()
    for i in range(1, num+2 ):
     if num % i == 0:
        divisors.append(i)
if len(divisors)%2 == 0:
    row = divisors[int(len(divisors)//2)-1]
    column = divisors[int(len(divisors)//2)]
else:
    row = divisors[int(len(divisors)//2)]
    column = divisors[int(len(divisors)//2)]
print(row,column)
```

Figure B.10: Storage of AA graph number and Specify rows and columns



```
start = time.time()

num = len(Graphset)-1
divisors = []
for i in range(1, num+1):
    if num % i == 0:
        divisors.append(i)
if len(divisors) == 2 and num > 10:
    divisors.clear()
    for i in range(1, num+2 ):
     if num % i == 0:
        divisors.append(i)
if len(divisors)%2 == 0:
    row = divisors[int(len(divisors)//2)-1]
    column = divisors[int(len(divisors)//2)]
else:
    row = divisors[int(len(divisors)//2)]
    column = divisors[int(len(divisors)//2)]

end = time.time()
time_diff = end - start
print(time_diff)
print(row,column)
print(divisors)
```

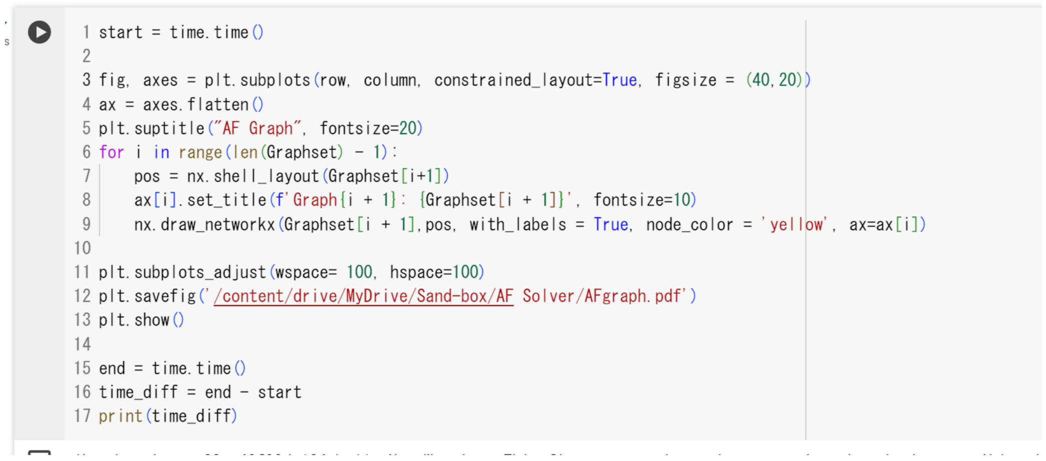
Figure B.11: calculate the rows and column for displaying all AA graphs



Graph Comparison

```
fig, axes = plt.subplots(row, column, constrained_layout=True, figsize = (10,8))
ax = axes.flatten()
plt.suptitle("AF Graph", fontsize=20)
for i,c in enumerate(check):
    pos = nx.shell_layout(Graphset[int(c)])

    ax[i].set_title(f'AFGraph{int(c)}: {Graphset[int(c)]}', fontsize=10)
    nx.draw_networkx(Graphset[int(c)],pos, with_labels = True,node_color = 'yellow', ax=ax[i])

plt.subplots_adjust(wspace= 100, hspace=100)
plt.savefig('/content/drive/MyDrive/Sand-box/AF Solver/AFgraph_Comparison.pdf')
plt.show()
```

Figure B.12: Display the specific AA graphs



```python
start = time.time()

fig, axes = plt.subplots(row, column, constrained_layout=True, figsize = (40,20))
ax = axes.flatten()
plt.suptitle("AF Graph", fontsize=20)
for i in range(len(Graphset) - 1):
    pos = nx.shell_layout(Graphset[i+1])
    ax[i].set_title(f'Graph{i + 1}: {Graphset[i + 1]}', fontsize=10)
    nx.draw_networkx(Graphset[i + 1],pos, with_labels = True, node_color = 'yellow', ax=ax[i])

plt.subplots_adjust(wspace= 100, hspace=100)
plt.savefig('/content/drive/MyDrive/Sand-box/AF Solver/AFgraph.pdf')
plt.show()

end = time.time()
time_diff = end - start
print(time_diff)
```

Figure B.13: Displaying all AA graphs

References

[1] " — ," https://www.spacemarket.com/magazine/know-how/jinro/, accessed: 2024-01-31.

[2] "gmbot-," https://sites.google.com/view/jinro-gm/%E3%83%AB%E3%83%BC%E3%83%AB%E8%AA%AC%E6%98%8E, accessed: 2024-01-31.

[3] P. M. Dung, "On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games," *Artificial intelligence*, vol. 77, no. 2, pp. 321 – 357, 1995.

[4] "Among us," http://www.h2int.com/games/among-us/, accessed: 2024-01-31.

[5] "How to play the werewolf(party game) : Setup and rules," https://www.wikihow.com/Play-Werewolf-(Party-Game), accessed: 2024-01-31.

[6] "—e-gov," https://elaws.e-gov.go.jp/document?lawid=140AC0000000045, accessed: 2024-01-31.

[7] T. Racharak, "Doing analogical reasoning in dynamic assumption-based argumentation frameworks," in *the 34th IEEE international Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2022, pp. 383 – 390.

[8] K. N. Wakatsuki, Toshiko., . Tokyo, Japan: Tokyo Denki University Press, 2017.

[9] A. Y.-G. Andreas Herzig, "Abstract argumentation with qualitative uncertainty: An analysis in dynamic logic," p. 190–208, Oct. 2021.

[10] N. S. K. I. Gauvain Bourgne, Yutaro Totsuka, "Identifying belief sequences in a network of communicating agents," p. 370–386, Oct. 2019.

[11] V. G. Rodica Condurache, Riccardo De Masellis, "Dynamic multi-agent systems : Conceptual framework, automata-based modelling and verification," p. 106–122, Oct. 2019.

[12] S. T. Mikimon, Kobayashi., "Agent communication for dynamic belief update," *Transactions of the Japanese Society for Artificial Intelligence*, vol. 24, no. 3B, pp. 314–321, 2009.

[13] D. F. B. A. N. J. C. Vossers, J., "Towards deception detection in multiplayer dialogue games using trust-based argumentation," in *Ethics of Game Artificial Intelligence (EGAI).* Kraków, Poland: Cezara, Păstrăv.,Leila, Methnani.,Rui, Prada., Sahar, Asadi., Georgios, Yannakakis., 2023, pp. 383 – 390.

[14] D. R. A. . K. F. T. Phan, Minh, *Argumentation in Artificial Intelligence.* Springer New York, NY, 2009.

[15] J. P. W. S. W. Wolfgang Dvořák, Anna Rapberger, "An answer-set programming based system for abstract argumentation," in *Foundations of Information and Knowledge Systems*, Cezara, Păstrăv.,Leila, Methnani.,Rui, Prada., Sahar, Asadi., Georgios, Yannakakis. Springer, Cham, Jan. 2020, p. 79–89.

[16] "shell_layout - networkx 3.2.1 documentaion," https://networkx.org/documentation/stable/reference/generated/networkx.drawing.layout.shell_layout.html Accessed: 2024-03-17.

[17] "draw_networkx - networkx 3.2.1 documentation," https://networkx.org/documentation/stable/reference/generated/networkx.drawing.nx_pylab.draw_networkx.html, accessed: 2024-03-17.

[18] "The math behind match making," https://medium.com/@paultay/the-math-behind-match-making-69a845503f91, accessed: 2024-03-17.