Mechanism to Detect Mismatches and Provide Recommendations about the Users' Preference in Negotiation Support Systems

A Case Study about Issue Weight Mismatches with the Pocket Negotiator

Yuhao. Xuan

TU Delft

Mechanism to Detect Mismatches and Provide Recommendations about the Users' Preference in Negotiation Support Systems

A Case Study about Issue Weight Mismatches with the Pocket Negotiator

by

Yuhao. Xuan

to obtain the degree of Master of Science
in Embedded System
at the Delft University of Technology,
to be defended publicly on Tuesday June 20, 2023 at 15:00 PM.

Student number: 4696514
Thesis committee:
Prof. Dr. C. M. Jonker, TU Delft, Thesis advisor
Dr. Luciano. C. Siebert, TU Delft, Daily supervisor
Sietze. Kuilman, TU Delft, Daily co-supervisor
Dr. Arjan. van. Genderen, TU Delft, Lecturer and MSc. coordinator

TU Delft

Preface

This is the thesis report for obtaining the Master Degree at TU Delft. I was interested in the Pocket Negotiator, which was the reason I chose this topic. After digging more into the Pocket Negotiator, I was able to find out the current limitations of the Negotiation Support Systems.

The thesis project is done under the supervision of Catholijn, Luciano and Sietze. I would like to thank Catholijn for bringing up the topic and being willing to guide me. I am also grateful to Luciano and Sietze for their guidance throughout the project to keep me in the correct direction and their inspiration when I got stuck, also for their understanding during my difficult time.

I also want to show my appreciation to my family. Thank you for cheering me up when I hesitated.

With their help, I was able to continue to design the mechanism that could improve the interaction between the users and the Negotiation Support Systems. I also concluded the future thoughts about how this mechanism could be extended. I hope that it can be helpful for others.

Yuhao. Xuan
Delft, June 2023

Abstract

Negotiation Support Systems (NSSs) can provide help based on the preference setting (domain, issue weights, issue ranking, strategies, etc.) of the users of the systems. However, sometimes the users of the systems might make mistakes in the preference setting. With wrong preferences, the NSSs might provide suggestions that conflict with the users' desires. This thesis focuses on designing a mechanism that could detect the mismatches in issue weights of the users of NSSs and provide recommendations on updating the issue weights with a scoring method. This mechanism also has the ability to provide different recommendations based on the users' self-confidence level. A case study is done with six simulation experiments to test the mechanism. The results show that during the negotiation, the users of the NSS (PN) can have a chance to change the issue weights when mismatches occurred and the NSS (PN) can provide better suggestions after the issue weights are updated. The limitations of the mechanism are concluded at the end together with the future thoughts.

Contents

1 Introduction

Negotiation is a kind of process in which several parties with different preferences take part to reach an acceptable agreement. This kind of process plays an essential role in both personal life and professional life [33]. On the one hand, sometimes negotiation takes place without people noticing, e.g. when going out to grab a meal with friends or a group of people decide which movie they would like to go to together. On the other hand, negotiation can also be very noticeable in a job interview, for example. Negotiations are very important in our daily life. Everyone has their own thought, and when making decisions, without negotiations, arguments and chaos will occur. Since negotiations are not avoidable, these activities require paying a lot of attention. During the negotiation, the values of the bids are evaluated and chosen. The evaluation process requires human negotiators to be patient and careful. This results in a lot of computation and can be too time-consuming for human negotiators to deal with. Negotiation Support Systems(NSSs) with integrated automated negotiating agents could solve this problem [31]. Existing automated negotiating agents could significantly improve the outcome if the negotiation space is precisely developed and studied since computers can better deal with the problems caused by the complexity of the computations [2]. However, to develop the negotiation space properly, human parties need to jointly explore their interests with the NSSs [41]. It is impossible for agents in the NSSs to handle negotiation alone due to inherent semantic problems and emotional issues, and a human-machine collaborative system is required [41]. With proper interaction, the NSSs can have enough information to process the bids and provide acceptable suggestions. However, in some cases, the human negotiator can make mistakes and put a wrong preference which will cause a mismatch in information. In that case, the NSSs with wrong information have conflicts in dealing with the bids and can cause a loss of performance.

Current NSSs([22] [43] [37] [10]and [11]) have no function to pause the system when mismatches in the preference of the users occur. This leads to problems such as benefit loss, time-consuming, confusion for the user, etc. Wrong preference profiles might cause the NSSs to keep suggesting bids which do not meet the users' needs. Not only does this influence the user, but also it has a bad influence on the NSSs. The NSSs with a wrong preference setting could have a poor result even though there is no logical mistake. This will cause the developer to dig more into the NSSs if not aware of what causes the poor performance, which is not good for the research.

To avoid these issues, most NSSs require an additional function to detect the mismatch, then stop the negotiation and ask for human negotiators' interaction to update any possible wrong settings in their preference setting. This thesis focuses on providing a mechanism to increase the interaction between humans and NSSs. With this mechanism, the NSSs can be able to detect the mismatches in the issue weights setting and provide recommendations about updating it. A case study is done to test the mechanism, and future perspectives are given at the end.

1.1. Research Questions

Most of the NSSs ([22] [43] [37] [10]and [11]) use the preference profile to translate the bids into manageable variables. All of the above NSSs have similar functionalities and phases. Most of them use utility functions to represent the value for each bid, and all of them have promising results. The users

of these NSSs put the preference profile in the first two phases. In the bidding phases, the NSSs use these preferences to provide suggestions. The users of these systems can modify the suggestions and enter the offer according to their wishes. This will cause problems if the users always have conflicts with the suggestions. The NSSs do not know if the suggestions they provide are good or not, and the users do not have any idea whether what they provide is correct or not. The above limitations suggest that current NSSs need modifications to improve the connections to the users, and based on these limitations, the main research question is raised:

- **Main RQ**: How to find a feasible way to improve the interactions between the NSSs and the users so the users can have fewer wrong preferences for the NSSs to provide better (suitable) suggestions?

To answer this question, knowing what could be done is essential. The interactions usually take place in the preference-setting part. In this part, the most common mistakes are the users' misunderstandings. The users sometimes believe they prefer one issue over another but only notice this is not the case when bidding. This will cause mismatches in the preference profile. However, none of the existing NSSs can detect if the user makes mistakes during the preference setting. With a wrong preference setting, the NSSs will use the wrong data to provide suggestions which will lose benefits and cause the users of the systems to be confused. A wrong preference setting with the wrong priority of issue and issue weight can lead the NSSs to fail to provide proper suggestions. In conclusion, the limitations of current NSSs are stated below:

- Lack of a mismatch detection process in the Bidding phase.
- The bidding phase in the NSSs needs more interaction with the user.

The interactions between the NSSs and the users can be significantly improved by avoiding mismatches. In this case, the **Main RQ** can be further extended as follows:

- **RQ1**: What mechanism could determine when to pause the NSSs for the human to interact?
- **RQ2**: What methods could detect the mismatch in the preference setting?
- **RQ3**: How to update the preference and provide recommendations on the preference settings to the users after the NSSs are paused in the bidding phase?

1.2. Research Goals

The **Main Goal** of this thesis is to develop a mechanism to help improve the interaction between the users and the NSSs. This goal could be further extended as follows:

- **G1:** Give the users a chance to fix the mistakes they made in the preference-setting during the bidding phases.
- **G2:** Provide a preference setting recommendation so that the users can decide if they should change the setting.

To achieve these goals, this thesis focuses on the Bidding section to design a mechanism to provide a proper mismatch detection which uses the users' input preference, such as issue weight, issue ranking, issue utility and issue values, to detect the mismatches and pause the system, and the mechanism should use the issue weight and issue utility to update the preference profile and provide a recommendation on how the user should correct the mismatches, so the users can correct the error they made.

The ways to approach these goals are concluded below:

- **A1:** Develop a mechanism which could pause the bidding phase when necessary
- **A2:** Define a suitable mechanism to provide recommendations on the user's preference profile.
- **A3:** Simulate the mechanism and perform case studies so that the mechanism can be easily modified and used in most NSSs.

1.3. Contributions

In NSSs, much information is taken from the users, such as interests, issues to negotiate, issues weights, etc. In the bidding strategy section, all the information is used to calculate the total utility of each bid for each issue. With the total utility, the NSSs can provide suggestions.

The case study and the simulation show that the mismatch detection mechanism could pause the bidding phase when necessary and provide recommendations for correction. The mechanism could provide different recommendations to different users. There are limitations of the mechanism, and future thoughts about possible ways to overcome the limitations and what could be done to improve the interactions further are given, such as combining the issue weights mismatch detection with strategies mismatch detection and ideas about guessing the actual issue weights, etc.

In summary, the contributions of this thesis are:

- Introduce a mismatch detection mechanism to determine when the NSSs should interrupt and ask for modification or more information.
- A conclusion about the advantages and disadvantages of the mismatch detection and preference update mechanism is given.
- Future thoughts on improving the interaction between the users and the NSSs.

1.4. Outline

The rest of the thesis report is organised as below:

1. In Chapter 2, the background information is given, in which the negotiation process is introduced in detail, current NSSs are presented and compared, as well as the limitation of the current NSSs.
2. In Chapter 3, the Related Work of this thesis is introduced, in which different detection mechanisms are compared, and update methods are given and discussed.
3. In Chapter 4, the methodology of the project is stated, in which detailed information on how the mismatch detection and the preference update mechanism work are given, including a discussion of the expected outcome.
4. In Chapter 5, the experiment results are presented, in which the case study is introduced, and data collected from the experiments are analysed and concluded.
5. In Chapter 6, the conclusion of the thesis, together with some future thoughts on the project, are given.

2

Negotiation

In this chapter, the background knowledge of negotiation is introduced, and then existing negotiation support systems are compared.

Negotiation is a collaborative decision-making process[42]. During the process, multiple parties make offers and hope to reach an agreement that meets their needs. Negotiation can happen every day with or without our notice. However, some people want to avoid doing this as much as possible because they believe they lack the necessary skills. This leads to wealth disparity, political paralysis, and social injustice[13]. NSSs can assist in avoiding these issues, as well as provide a variety of additional benefits such as improved outcomes, reduced stress, and reduced time commitment[2]. Because human negotiators are excessively sluggish and expensive in comparison to NSSs, they will become increasingly widespread in our daily lives[2]. To understand the NSSs better, it is necessary to know how the NSSs could represent us. This requires the background information of negotiation.

2.1. Negotiation Process

2.1.1. Basic Elements

To understand the negotiation process better, it is important to know some definitions first. This paper [29] introduced the negotiation scenario that could help us understand the content of negotiations and the terminology used for most NSSs. In their work [29], they suggested that the negotiation scenario is essential for later picking up a suitable strategy. They demonstrated that it included these elements:

- Domain.
- The number of players in the negotiations and their style.
- The context of other negotiations.

The element **Domain** has information about the issues to be negotiated over and the values associated with the issue [29]. For example, if you are an applicant to a job interview, you will have a negotiation with the employer, then the Domain will be the job, and the issue can be salary, fte, work from home, etc. And the issue value within the issue salary can be 2000 Euros, 3000 Euros, etc.

The element **The number of players in the negotiations and their style** can be further extended as this [29]:

- **Preference**: The Preference reflects how the parties think about the issue. How they favour one issue over another or how they like one specific value set of issues.
- **Decision-making style**: The Decision-making style shows the risk's attitude of the parties of the negotiation, cognitive style, cognitive capability, truthfulness and characteristic strategies [29].

The element **the context of other negotiations** indicates whether the parties could be influenced by previous negotiations' experience.

The above elements can form a large space with multiple dimensions, which still require to be understood [29]. This thesis mainly focuses on element **preference**, how an untruthful preference could influence the outcome of the NSSs and ways to detect mismatches in the preferences. This requires the preference to be well-presented. In Section 2.2, the terminology and definitions of this element is discussed and extended.

After having a basic understanding of the elements of the negotiation, it is time to move on to the negotiation process. Negotiation can normally be divided into four phases [18]. In the following sections, each phase will be introduced.

2.1.2. Preparation Phase

The users should decide what they want to obtain out of the negotiation during the preparation phase. The users consider the issues they want to negotiate over (also known as negotiation variables) and the potential worth of these issues (issue weights, issue rankings) [19].

While considering the issues and their value ranges, the users should also consider their preference profiles, which are the rankings of the possible negotiation outcomes. A utility function that translates bids to numbers or a CP net that compares bids using a preference relation can be used to describe a preference profile [20]. When considering this, it is better to also consider establishing a reservation value, which is linked to a BATNA (Best Alternative to a Negotiated Agreement).

2.1.3. Exploration Phase

The exploration phase is intended to get to know the other parties involved in the negotiations. The goal of this phase is to assess and maybe change the negotiation domain, as well as to create an environment in which an agreement has the best possibility of success, try to make the negotiation space as big as possible [21]. The users may discover that they need to add issues to the negotiation during their interaction with the other parties, whether because one of the other negotiation parties insists on discussing it or because they realize the users omitted some details. The users may also discover that their preference profile, BATNA, and/or reservation value require updating [20].

By conversing with the other parties, The users can have a deeper understanding of their preference profiles, BATNAs, and reservation values.

At this step, the users establish what issues will be under negotiation, as well as what methods and protocols will be followed during the bidding phase [21] [20].

2.1.4. Bidding Phase

The critical processes usually take place during the bidding phase, in which the users exchange offers with an opponent or a group of other negotiation parties. The exchange of offers may concern only one or a subset of the issues (partial bids) or may concern all issues concurrently (complete bids) [20]. Additionally, such an offer may include many forms of motivation or arguments.

The bidding phase concludes when the negotiating parties reach an agreement (all other parties accept an offer made by one of them), when the discussions run into some form of the deadline, or when a sufficient number of parties withdraw from the negotiations [21].

2.1.5. Closing Phase

The closing phase of a negotiation's content is defined by how the bidding phase concludes [20]. If the parties walk away, the closing phase may be ended without agreements. Still, it may also convey under what circumstances the parties are willing to return to the negotiating table. If, however, the bidding phase resulted in an agreement, the closing phase is the stage during which the negotiation parties confirm the achieved agreement [20]. This phase of human negotiations finishes with signing a contract, thanking all parties for reaching an agreement, and, if suitable, looking forward to a future in which the contract will be mutually fulfilled.

2.2. Utility Function

In the negotiation process, a bid includes lots of information. It is complex and difficult to process for the NSSs to use directly. Therefore the NSSs normally transform bids into total utilities and use the total utilities to decide what to recommend.

This section discusses the terminology and definitions of the utility function. Note that the utility function can be nonlinear [20] [28] [29] and complex for different NSSs. In this section, a basic linear utility function is introduced, This type of utility function is most frequently used in NSSs [29] [7] [32]. Most of the equations are from TU Delft Lecture paper [20].

First, the basic information of the negotiation is translated. As for the parties of the negotiation. Assume that P represents a collection of at least two bidding parties. The negotiating parties' names are:

$$p_1 \dots p_k \tag{2.1}$$

with $k = |P|$. $p \in P$.

Then define the negotiation space that contains all potential bids. Let S represent the space of all potential bids and

$$b_p^r \in S \tag{2.2}$$

represent the bid made by negotiator p in round $r \in \mathbb{N}$. Given that any negotiation is believed to be finite, rm represents the negotiation's last round.

Now, it is ready to define the utility.

$$u_n : S \mapsto [0,1] \tag{2.3}$$

u_n denotes the negotiation party's utility function.

Assume that U is the function that translates bids $b \in S$ to the P-dimensional utility space $[0,1]^P$ spanned by the negotiation parties' utility functions:

$$U : S \longmapsto \Pi_{p \in P}[0,1] \tag{2.4}$$

Which is defined by:

$$\forall b \in S : U(b) = \langle u_1(b), \dots, u_k(b) \rangle \tag{2.5}$$

Multi-issue negotiations aim to obtain an agreement on a finite number of problems β of $\mathrm{I} = \{i_1, \dots, i_\beta\}$. Each issue $i \in \mathrm{I}$ has its probable value range. Si. Therefore, in general.$S = \Pi_{i \leq \beta} S_i$ Thus

$$S = \langle S_1, \dots, S_\beta \rangle \tag{2.6}$$

The utility functions of the negotiating parties are also functions that accept an input of a multi-issue bid and map it to the range $[0,1]$; more precisely, $\langle S1, \dots S\beta \rangle \mapsto [0,1]$ [20].

These functions can be complex nonlinear functions or very basic additive linear functions[28].

Here is an example of a linear additive utility function constructed from simpler utility functions per issue, which are frequently referred to as issue evaluation functions[1] or occasionally valuation functions [20], such that for all $p \in P$, define [20]:

$$u_p(\langle i_1, \dots, i_\beta \rangle) = \sum_{j=1}^{\beta} w_{p,j} u_{p,j}(i_j) \tag{2.7}$$

where $w_{p,j}$ is the weight that p assigns to issue j, and $u_{p,j} : S_j \mapsto [0,1]$ is the p's utility function for issue j, such that $\sum_{j=1}^{\beta} w_{p,j} = 1$.

2.2.1. Normalization

Please note that while negotiating in random domains, the evaluation ranges of the issues under negotiation may not always be in the range $[0,1]$ [20]. In such a situation, it is necessary to normalize the evaluation and utility functions to the range $[0,1]$ [20]. The normalization function is given below:

$$u_j(x) = \frac{v_j(x)}{max_{y \in S_j} v_j(y)} \tag{2.8}$$

where $j \in \mathrm{I}$ with a range Sj and $v_j : S_j \mapsto \mathbb{R}$ is the unnormalized evaluation function, $u_j : S_j \mapsto [0,1]$ is the normalized evaluation function.

[1]These utilities have connections to the users' preference profile such as issue weights and issue rankings

2.3. Discussion

From Equation 2.7, we can see that the issue weights and the issue utilities determine the total utility, which later is used by the agent in the NSSs to provide suggestions, which makes it important for the users of the NSSs to put in the correct issue weights. If the users put in the wrong values of the issue weights, the total utility could be different. For example, some person plans to negotiate about jobs. He thinks he prefers career development opportunities over salary but puts the preference oppositely. This leads to the fact that when calculating the total utility, the agent of the NSSs might give up the low issue weight issue first, in this case, salary. The suggestion given to the person will be low salary with high career development opportunities chances. The user is unaware of why the NSSs keep suggesting like this, and continues to change the suggestions indicating that there is a mismatch in the issue weights for issue salary and issue career development opportunities. This mismatch influences the interaction between the person and the NSSs since he has to change the offer made by the NSSs constantly.

The goal of this thesis is to define a mechanism that could solve this problem by detecting the mismatches in the preference setting and providing recommendations on changing it during the bidding phase and after the update, the NSSs should be able to provide acceptable suggestions.

2.4. Examples of Negotiation Support Systems

In this section, existing NSSs are introduced and compared, a discussion about the limitation of the NSSs and the motivation of the thesis is given at the end.

2.4.1. The Pocket Negotiator(PN)

The Pocket Negotiator (PN) is an example of a NSS [22]. It is an interest-based negotiation product [22] During the four phases in the negotiation(Prepare, Explore, Bidding, Closing, see Figure 2.1), the PN first collect the chosen domain, the duration of the negotiation and the strategy to represent the user in the Prepare and Exploration phases. Then in the Bidding phase, a suggestion for a bid is provided in each round. Whenever a party decide to accept the other's offer, the negotiation will go to the Closing phase in which an acceptance strategy will be used. If no one accepts the bid, the negotiation could also end without an offer. The PN can give suggestions to the user. However, the system does not have the ability to determine if the data it collected during the first two phases is correct or not. This leads to the fact that if the user made a mistake in their preference profile, the system would provide suggestions that do not fit the user's need, this conflict could decrease the performance of the agent since the PN uses the user's preference such as issue weight to calculate the total utility and the total utility is essential when providing suggestions.

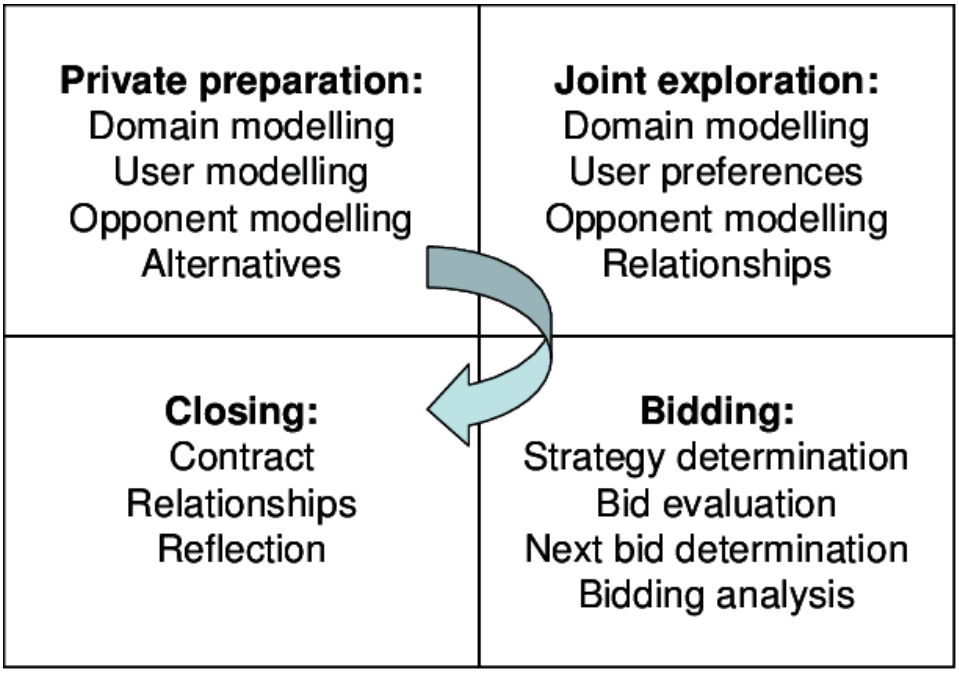

Figure 2.1: The four stages of the Pocket Negotiator [16].

An example image of the PN is given below:

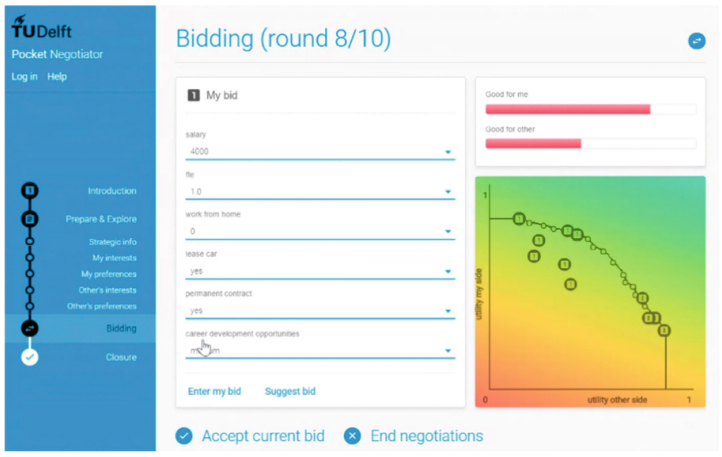

Figure 2.2: Example of Pocket Negotiator(Bidding phase) [1].

2.4.2. The EmoNeg System

The EmoNeg provides user suggestions based on the utility function. It contains a utility estimating function [43]. The paper [43]illustrates the technique using the board game ”Monopoly.” but makes no mention of whether it could provide assistance with domain elicitation. The EmoNeg system focuses on emotion support based on Newcomb’s ABX model [30] in order to assist with next-move negotiations. The system’s mechanism is rule-based and is based on [40], in which the user is asked to perform Emotion Recognition of the facial expressions of the other negotiator.

2.4.3. The Negoisst System

Negoisst is a Java-based web-based NSS that utilizes a client–server architecture [37]. On the client side, user interfaces are shown that allow for system interaction. The program is hosted on the server, where a three-tier architecture consisting of a display layer, an application layer, and a data layer is implemented [37].

Different from other NSSs, the Negoisst system has a conversation panel where each party can negotiate issues, discuss problems and reach an agreement. The user is able to set up the domain and be prepared in the panel. Similar to the PN, the Negoisst System has a function that could show the negotiation history. But not as clear compared to the PN. It does not show the result with Pareto Optimal Frontier.

2.4.4. The eAgora system

The eAgora system is another type of NSSs; it has functionality similar to PN. It has a preference profiling function, domain selection, exploring phase and end phase[10]. Unlike the others mentioned in this section, the eAgora explains during the suggestion offering process[10]. Suppose the value of the bid is not acceptable. In that case, the system will notify the user with a text like ”You should not accept this offer because it is valued at -20 (any value below o is outside your preference level)”, which makes the user effortless and save more time[10].

2.5. Conclusion

From the literature study, one thing in common is that issue utility is frequently mentioned in most of the equations mentioned in this chapter. The negotiation agents use the utilities to translate bids into something which could be used to compare and provide suggestions. As for the NSSs, all of the above NSSs have similar functionalities and phases. Most of them use utility functions to represent the value for each bid, and all of them have promising results. There is a limitation in the above NSSs. The users of the NSSs might not understand themselves as they should. They may value one issue over another but put the wrong issue weights in the preference profiles. The agents of the NSSs which use the wrong profiles calculate the utility of these profiles. This will cause problems. With the wrong utilities, the agents cannot provide suggestions that fit the users' requirements, which will lose benefits and cause the users of the systems to be confused.

With a well-designed mechanism to detect the mismatch in the preference setting, the NSSs should be able to provide suggestions that fit the needs of the users. The users who receive the recommendations should be able to correct the preference profiles when needed. This mechanism should be easy to modify so that it can be used for most NSSs. Therefore, in this thesis, information directly available to the users is used, which is the issue value differences between the suggestions made by the NSS and the input issue values(modified after the suggestions from the NSSs). The PN has the aspect mentioned here and is available in the lab, so this thesis chose the PN as the NSS to test the mechanism.

To design a mechanism that could detect the mismatches in the issue weights, it is vital to assess the users' behaviour during the bidding phase of the negotiation. To make recommendations about issue weights, finding a proper way to update the values according to the users' choices is necessary. This requires further research on related work, which will be discussed in the next chapter.

3

Related Work

After analysing the limitations of current NSSs, it is time to solve the research questions.

RQ1 and **RQ2** are about detecting the mismatches in the users' preference and pausing the NSSs, this leads to a question, when do the NSSs need to stop? From my perspective, it is necessary for the NSSs to stop when the performance of the NSSs is influenced by the activities either within the system or the activities of the users. To be more specific, the users' activities might cause mismatches, which will further influence the activities within the NSSs. In that case, the mismatches can be considered as threats or risks. The mechanism should be able to assess the risks and, when it is critical, notify the risks to the users. What could be a suitable risk assessment method for this thesis needs to be researched.

RQ3 is about what to recommend after mismatches have been detected. This request ideas on developing a suitable method for updating values.

To be well-prepared before designing the mechanism, literature research has been done on False Analysis, Risk Assessment and Update mechanism. In this chapter, related work of the thesis is given. A literature study is done to provide ideas on how and when to pause the NSSs, and ways to update values are presented and compared at the end.

3.1. Mismatch Detection

The mismatch occurs when the users make mistakes in the preference setting, which can cause errors when calculating the total utility. The errors caused by the mismatch cannot be easily observed since the actual value of the preference is not directly available. To detect the mismatch, it is necessary to detect the errors and analyse the risks. This requires methods of error tracing and risk assessment.

In this section, several papers are presented. In the first part of this section, the Fault Tree Mechanism is introduced. In the second part of this section, papers about scoring methods for risk assessment are compared, and in the end, a discussion about the papers and their relations to the thesis project is concluded.

3.1.1. Fault Tree Mechanism

Fault tree analysis(FTA) is an important and well-known method to analyse the risks, trace back errors, etc [36]. It has already been implemented in many fields, such as data centres, power plants, etc [36]. The fault tree mechanism is a graphical tool. The leaves of the tree represent all the possible outcomes, and the roots represent the activity which leads to the leaves. An example of a Fault tree is given in Figure 3.1

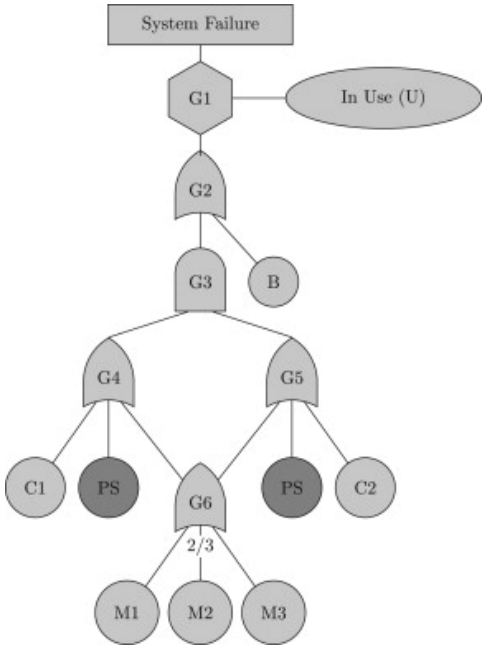

Figure 3.1: An example of Fault tree [35] modified from [27] [3]

In Figure 3.1, B, C1, C2, M1, M2, M3, and PS represent a bus, two CPUs, three memory units and a power supply. $Gi(i \in [1,6] \ \ \& \ \ i \in interger)$, represent the gates, which shows how the failure is generated. Gates always have logic behind them. For example, $G3$ means that the failure is caused by both $G4$ and $G5$, and is caused by some of the leaves listed below them.

Since the Standard Fault Tree can only detect the system failure with a certain sequence of component failures [36], it is not suitable to detect if certain components can lead to failures with a different sequence, extensions of the Fault Tree are necessary. Different types of extended FT are shown below:

- Dynamic Fault Tree [5]
- Repairable Fault Tree [34]
- Fuzzy Fault Tree [39] [12] [25]
- State-event Fault Tree [24] [23]

A comparison of these FT is given below.

Discussion
Though Dynamic Fault Tree(DFT) is the most commonly used extension [12], in some cases, the performance of DFT is not good enough. For example, Fuzzy Fault Tree(FFT) can have better performance when the probability of failure for some components and their activities of them are unknown. In a case when one component's failure might increase the chance of failure of another component, the extended FT allow the FT to observe the system and model it. State-event Fault tree can update the status of the components with states, which provides real-time modelling. All of the above extended FT show the promising results which can be used for risk assessment and error detection.

In a word, the FT is able to assess the systems. Provide the root of the failure, which could be used to detect where the errors occur. In the next section, examples of some FTs are given and analysed. A conclusion is given after the next section.

3.1.2. Examples of Fault Tree mechanism

Boiler Explosion Risk Assessment
Song[38] presented their work on an explanation support system for risk assessment system for the boiler.

In their work, they traced back the system in detail, flagged all the possible outcomes of the system, and prepared for each circumstance; explanation text was established beforehand. Whenever a flag had been triggered, the system would automatically trace back to see what happened and ready the explanation.

Since the goal of their work is to provide an explanation, they use SFT to model the boiler system. The FT architecture is shown in Figure 3.2

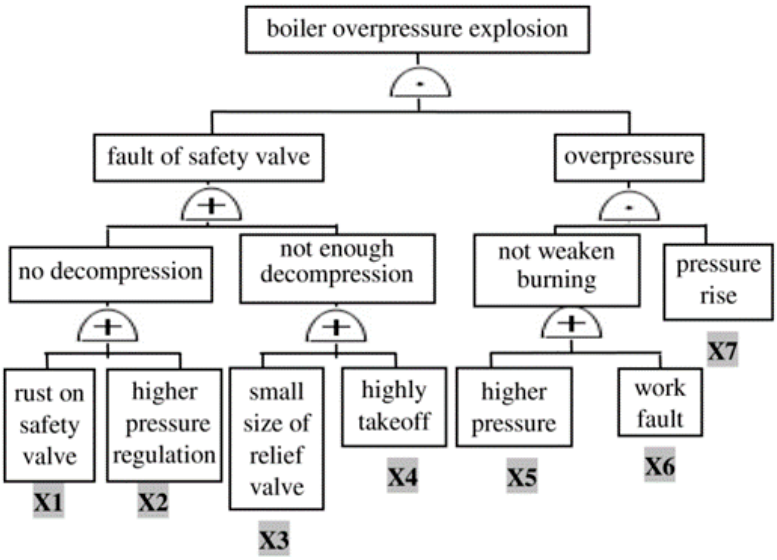

Figure 3.2: Fault Tree of the Accident of Boiler Over pressure Explosion (extracted from [38])

In Figure 3.2, the failure occurs as the root. The leaves represent the reasons that lead to the explosion. **AND** and **OR** gates are used to combine the reasons. at the end of the leaves, they label them with flags, and whenever a flag is detected, the alarm will be triggered, and explanations will be given.

Risk Assessment of Work Accident in Container Terminals

Budiyanto and Fernanda [8] did a study on risk assessment in container terminals. In their work, they studied recent years' data and concluded possible reasons that led to the accidents. They used FT to analyse the risk. An example is given in Figure 3.3

Figure 3.3 shows existing reasons that can cause the fire in the container. Different types of causes are labelled with different colours. It can be directly observed why the fire occurs from this figure. The FT, together with risk metric [8] conclude which type of accident is the most severe one and whenever a certain leaf activity is observed, there is a possibility that the accident happens, which provides ways to detect the accident before it happens.

Conclusion

The FT mechanism is able to detect the error but requires a detailed analysis of the system. Possible causes need to be figured out before establishing the FT. FT is a graphical method. It can present all the possible reasons in one figure, which is easy for the user of this mechanism to understand and accept. However, to successfully establish the FT, all the scenarios that cause the failure should be included. In some cases, one activity might influence the other, which can make the FT to be too complicated. The advantages of the FT are concluded as follows:

- Visual Result: The FT is a graphical method, the FT can easily demonstrate how the errors or failures are generated.
- Detailed information provided: The FT can provide what leads to the errors or failures
- Uncertainty causes analysis: Extended FT can be able to use fuzzy set and probability theory to assess the risk and alarm earlier.

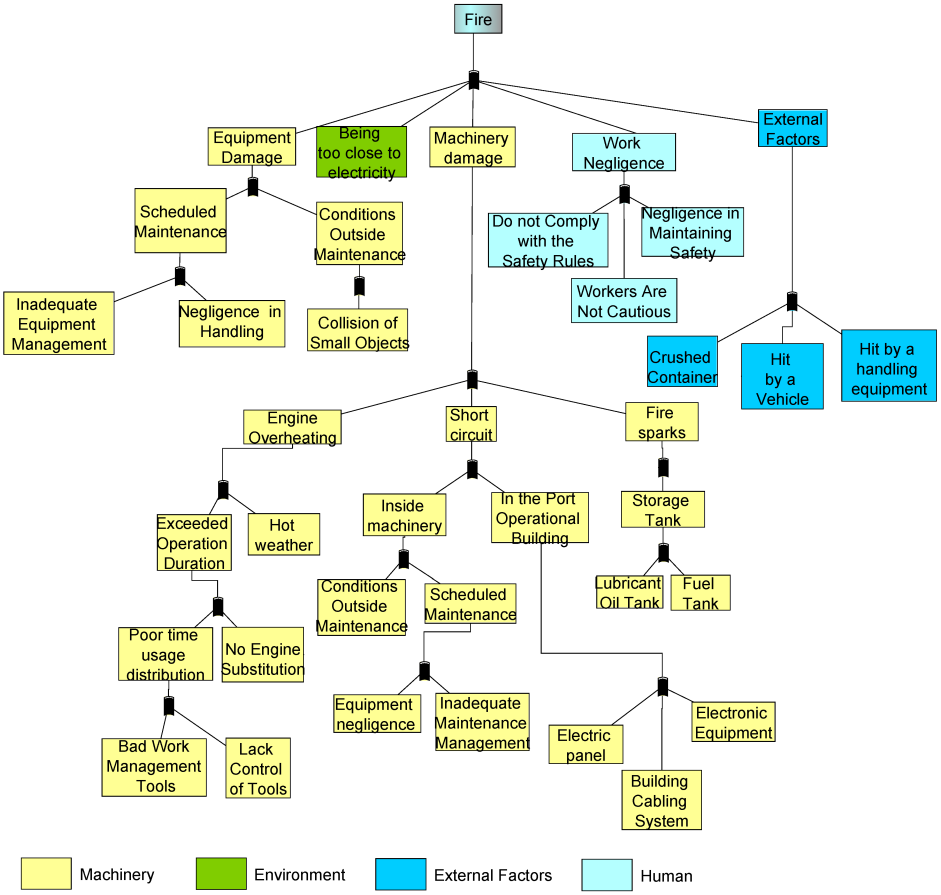

Figure 3.3: Fault Tree Analysis (FTA) of Fire [8]

The disadvantages of FT are concluded as:

- The FT requests a full analysis of all the scenarios. In some cases, this cannot be achieved.
- If the systems are too complicated, the root of the tree can be pretty deep, and the search for its children is time-consuming.

3.1.3. Scoring Method

The scoring method has been widely used in multiple fields such as medical [17] [4], group division [26] etc. It is not one specific approach, but ideas about keeping score about the current status. The risk is determined and assessed according to the score. In this section, examples of scoring method are given and discussed.

Scoring Method in Medical Field

The idea behind this method is to grade each being evaluated issue with a different grade and, according to the grade, assess the current situation and proceed to the next step. In the medical field, for example, sometimes it can be very busy in a hospital. Patients who need surgery might need to wait in line for the operations. The scoring mechanism can grade each patient with their health conditions and, according to the grade, see if the patient requires the operation immediately [9]. Current study [17] [4] [9] on the medical field shows that the scoring method can provide risk level of current situations. In [9], the authors mention that if an individual has a score beyond a limit, this individual is not suitable for a particular therapy treatment.

Scoring Method for Group Division

Zhang Bofeng and Liu Yue introduced an explanation support mechanism for earthquake prediction system[6]. This explanation mechanism divides users into different levels according to their knowledge base. In this mechanism, four steps are implemented.

- **Step 1 Self Testing:** In this part, the users need to choose their user levels according to their knowledge base(For the users who have used the system before, this step is to assess the difference and a different questionnaire could be prepared in step 2).
- **Step 2 Q&A Section:** A proper questionnaire is provided, and different levels of questions are given.
- **Step 3 Match the knowledge level:** According to the user level, different explanations will be given according to the matching algorithm.
- **Step 4 Update:** Update the user level for the users.

In their work [6] tested and provided an algorithm that mapped the users' levels accurately according to **Step 1** and **Step 2**. In **Step 1**, the user will provide the user level L_0 based on the acknowledgement with confidence degree W_0, where W_0 is usually between 0 and 1. With the equation below, the user level could be obtained.

$$L_R = W_0 \times L_0 + (1 - W_0)\frac{1}{Q_R}\sum_{i=1}^{Q_R} L_i \tag{3.1}$$

Where L_0 denotes the level given by the user. L_i denotes the level according to the answer in Q&A. If the question is correct $L_i = D(Q_i)$, otherwise it is set to $D(Q_i) - 1$. In **Step 3**, a matching algorithm is introduced

$$|L_R - Diff| < d \tag{3.2}$$

if $|L_R - Diff| > d$, then the knowledge is too difficult, meaning the system needs to explain the knowledge more simply. The author using this mechanism is testing it with Expert System for Earthquake Prediction, in which the user is divided into five levels. Detail information can be found in the table 3.1

In their work [6], they introduced confidence degree W_0, which represents self-confidence. People who have higher self-confidence receive less influence from the questionnaire and tend to stay on the level they choose. People with lower self-confidence receive more influence from the questionnaire, and the level they choose will have a partial decrease.

User level	The level of explanation
novices	Predict the time, space, and magnitude of the earthquake. The evidence used in the inferring procedure.
common user	Conclusions drawn by the inference engine. The knowledge used in the inferring procedure.
advanced user	How is the conclusion drawn? The evidence, rules and the corresponding conclusion during the time, space, and magnitude of the earthquake prediction. The conclusion drawn by using single precursory items.
domain expert	Not matched evidences. Used or not E-R information
knowledge engineer	Correlative evidences and rules. Revised evidences.

Table 3.1: The customized levels in ESEP [6]

Conclusion
The Scoring method is a very basic and simple method. It collects the information that is available and analysis the risk to avoid failure. Apart from that, the Scoring method can also collect current status and be useful in many aspects other than risk analysis.

The advantages of the Scoring method are concluded as follows:

- The Scoring method is a very basic and simple method. It can be easily implemented in most systems.
- The Scoring method uses relevant data to keep score and assess the risk. For different topics, the scoring function can be different. The Scoring method can be easily adapted to systems.

The disadvantages of the Scoring method are concluded as follows:

- The reasons that lead to the mismatches require detection beforehand. Only after the reasons are known can the Scoring functions be established.
- It is a mathematical process. When the alarm of mismatch is triggered, it is not possible to directly demonstrate to the users what are the reasons that cause the mismatch.

3.1.4. Discussion

The FT is suitable for detecting errors and risk assessment. However, the FT is not a good choice for this thesis. The mismatch in the NSSs is considered an error or a failure. The reason causing the mismatch is that the users of the NSSs put the wrong values. Issue weights, for example, the users themselves do not know the actual values. The actual values are in their mind. The differences between the suggested issue values and the input issue values can determine if there is are mismatches. However, if one issue weight has a mismatch, it means all the others have the wrong values. How one issue weight can influence the others is impossible to predict since one issue weight might need to be increased while the others can either increase or decrease. It is impossible to figure out all the possible outcomes. For strategy mismatch, it is possible to figure out all the possible outcomes since there are limited agents to choose from. Since this thesis focus on issue weight recommendation, FT can be removed from the table but can be further studied for strategy mismatch detection.

For the Scoring method, it can collect whatever information is required and evaluate each round's score. With the addition of each round's score as a risk, when the risk is above a threshold, the mismatch is detected. In this way, the mechanism can have enough time to update the issue weight. This thesis focuses on issue weights mismatch detection. To this end, the scoring method is sufficient enough to detect the mismatch and pause the NSSs.

3.2. Update Mechanism

In this section, two examples of update mechanisms are provided and discussed, and the comparison between these mechanisms is given together with the conclusion.

3.2.1. Scoring method for Group Update

The thesis is aimed to provide an update on issue weights after the mismatches are detected. As discussed before, the actual issue weight of the users is not available to the user and the system. This

indicates that the destination of the update is not clear. There are limited papers mentioning similar situations like this. However, one paper [6] has a similar issue.

As described previously, they used a questionnaire to define a score to divide the users into different groups. The users are asked to choose an initial level they believe they should belong to. This is like the input issue weight. And the update function in Equation 3.1 could tell the users the new levels. The Confidence Degree was asked, and the Q & A section started. They received the averaged scores based on the Q & A, which is how much the initial level can be updated, and the Confidence Degrees, which represent the self-confidence of the users, decides how much the level should be updated.

With some modifications on Equation 3.1, if there is an issue value difference between suggested value and input value, a function consists of the initial input issue weight and how much the issue weight should change to update the issue weight, the issue weight can be updated iteratively.

3.2.2. Automatic updating mechanism

Guan [15] proposes the idea of an automatic updating mechanism. This mechanism can sufficiently optimize the goal, using all the constraints that are available. They first introduce an improved ant colony optimization, and then the update method is implemented to achieve the goal.

The paper focused on the Constraint satisfaction problem (CSP). It is a problem that consists of several value pairs that fit some constraints. A CSP can be defined as:

$$CSP : (X, D, C) \tag{3.3}$$

In this triple, X is the value set consisting of variables, D is the function that maps the Variables in X to the domain D, and C is the constraint set that decides the value of the variables in X.

In their work, they indicated that there were a lot of algorithms which were suitable for this problem [15]. Among them, ant colony optimization (ACO) had been proven to be very useful in solving this task [14].

The algorithm of ACO is described below (**Algorithm 1**) [15]:

Algorithm 1 Algorithm of ACO

Require: a CSP (X, D, C); the number of iterations N; population size M
Ensure: $bestA$
  $Initialization$
  for $j = 1$ to N **do**
    for $k = 1$ to M **do**
      Construct a complete assignment A_k
      if $Cost(A_k) < Cost(bestA)$ **then**
        $bestA \leftarrow Ak$
      end if
    end for
    Update pheromone on each vertex
  end for
  return $bestA$

According to the **Algorithm 1**. ACO had a problem that would cause the stability of the result to decrease. To this end, an update mechanism was introduced (**Algorithm 2**) [15]:

Algorithm 2 Algorithm of Automatic Update

Require: a selected assignment A_k; domain size L
Ensure: an updated assignment A_k
$Initialization$
for each selected variable x_i **do**
  for p = 1 to L **do**
    if v_p does not assigned to x_i in the domain $D(x_i)$ **then**
      v_p is assigned to x_i
      $A_k^* \leftarrow A_k U < x_i, v_p >$
      if $Cost(A_k^*) < Cost(A_k)$ **then**
        $A_k \leftarrow A_k^*$
      end if
    end if
  end for
end for
return A_k;

This method is aimed to decrease $Cost(A_k)$, it is done by testing different values in V to compare the $Cost(A_k)$, if the new value is smaller than the original one, then the A_k will be updated. And with the update mechanism implemented, the ACO is re-designed, and a new AU-ACO is introduced. The new AU-ACO is able to automated test the variables that fit the constraints best, which is more stable and reliable.

The main idea of the update method in this paper [15] is to test all the possible values. In this thesis, the update target is the issue weight, which is a continuous value. It is not possible to test all the possible values since the target set cannot be confirmed. The actual issue weight is unknown, and cannot be tested to get.

In a word, the updating method in this paper [15] has provided a simple way to achieve the goal of updating. However, it is not suitable for this thesis.

3.2.3. Discussion

These two methods provide ideas on how to update values in systems. The first paper [6] introduced a mechanism which used the change to update the user level. The second paper [15] described a way to test all the values and compared the outcome of each tested value and updated according to the comparison results. The first method is able to update according to the changes, while the second one provides an update according to the results. In this thesis, the mismatch occurs when there is a difference between the suggested value and the input value. The difference can be considered as change, which indicates the first method can be suitable. The PN uses the total utility to provide suggestions. It is computed with the issue weights and the issue utilities, which are defined by the preference settings. However, this thesis assumes that there can be a mismatch in issue weights, and the actual issue weights are not available. Though the second method is more direct and maybe more accurate compared to the first method, in this thesis, it is not possible to predict the actual issue weight before the implementation of the update mechanism. If there exists a method that can have the actual issue weight, the second method can be a suitable way to confirm the prediction. In conclusion, the first method could be an approach, while the second one is more likely to be an evaluation method, which can be researched in the future.

3.3. Conclusion

In this chapter, several papers about risk assessment are discussed and two methods to achieve the error detection are compared. FT is not suitable for issue weights mismatch detection but can be useful in further studies about strategies mismatch detection. The scoring mechanism is suitable for issue weights mismatch detection by grading the users activities during the bidding process of the negotiations. As for the update methods. One paper with the same challenge as the goal of the thesis is discussed. Another updating mechanism provides thought on evaluating the issue weights. If in the future, a database of all possible issue weights is established, the second paper with a proper training method, could be useful to update and provide recommendations.

4 Methodology

4.1. System Overview

As stated in Chapter 1, to approach the goal which is to improve the interactions between the users and the NSSs, this thesis focuses on designing a mismatch mechanism to detect the mismatch that occurs in the users' preference profile, in this case, issue weights. Due to availability, the Pocket Negotiator (PN) from TU Delft and WeGain is used as the NSS to test the mechanism. The system overview can be found in Figure 4.1.

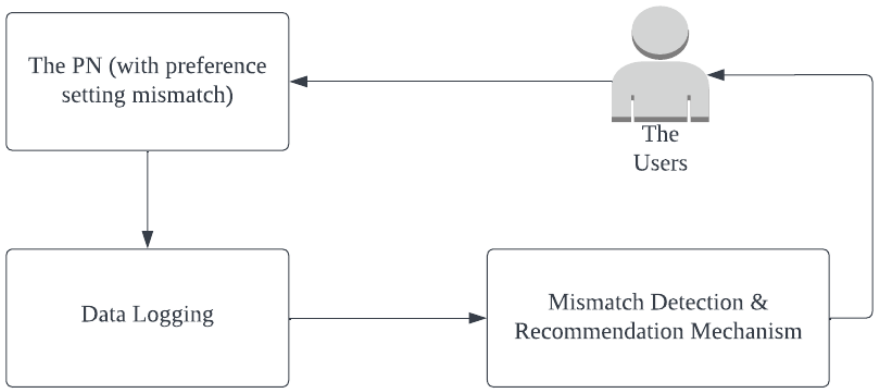

Figure 4.1: System Overview.

The basic concepts of the three blocks in Figure 4.1 are given below:

- The PN (with preference setting mismatch): In this block, the users put in their preferences, and negotiate with other parties.
- The Data Logging: This block represents the data logging process. During the process, all the necessary data is logged out from the PN, later sent to the Mismatch Detection & Recommendation Mechanism.
- Mismatch Detection & Recommendation Mechanism: This block represents the process of mismatch detection and recommendation, in which the designed mechanism is applied to detect the mismatch and provide update recommendations about issue weights to the users.

In the following sections, detailed information about the system is introduced, and each block in Figure 4.1 will be expanded.

4.2. System design preparation

It is essential to understand what data is required to design the mechanism. To answer this question, another question is raised. What could be wrong with the user's preference profile and what data is relevant to determine the mismatches?

4.2.1. Types of mismatch

According to the structure of an NSS, PN, for example. In the first two phases, the PN collects the user's preferences, such as domain, role, agent (strategy), interest weight, issue weight, and issue rank [22].

For the domain, there are three domains in the current available PN (**jobsiV2**, **energyV2** and **watermanagement**). The PN asks the users later in the second phase for the users to put in the interest weights and the issue weights, which will present all the issues to the users. The issue names are quite different for different domains. It is not likely for the users to have a mismatch in the domain.

As for the role, take the **jobsiV2** domain as an example. There are two roles in this domain, **applicant** and **employer**. Similar to what was mentioned before, the PN will ask the users to change the interest weights and issue weights. However, careless users might accidentally choose the wrong side and put an opposite preference for these two topics. After the users choose the roles, the users will continue to change the issue rankings. The different roles obviously differ in the issue rankings in the PN. In this case, the users will not likely have a mismatch in the role selection.

Now we continue to the agent (strategy) selection. Not like human negotiators who decide and change strategy while they bid, the NSSs such as PN provide the agent (strategy) and ask the users to choose beforehand, and this cannot be changed while bidding. This might cause some problems. Imagine a case like this, a user indeed is a hardliner but thought themselves could accept the conceding through the Pareto Optimal Frontier and choose a strategy like this. When the user starts the bidding phase, the PN will always concede as the user decides to fix on one bid (just like hardliners do). Therefore, the user keeps changing the bid to the set value, and the PN keeps conceding on the frontier and providing suggestions. In the end, the user might end without a contract. Therefore, a mismatch could occur in the agent(strategy) selection process.

One interesting thing is that the PN is an interest-based NSSs which links the interest weight to the issue weight [22]. This is a unique design and unlike other NSSs only use issue weights to translate the bids. This thesis aimed to design a mismatch detection mechanism that could be easily modified and applied to most NSSs. Therefore, when considering what types of mismatch could occur, interest and issue weights should be considered separately. Fortunately, the PN has a function that disconnects the link between the interest weights and the issue weights and only considers the issue weights. For the question of this section, a mismatch could happen in issue weights.For example, in **jobsiV2** domain, a user who owns a car might still want another one so other family members could use the car they hold. Therefore the user puts the issue weight of **lease car** higher than **work from home**. However, when the bidding phase starts, the PN begins to sacrifice the value of **work from home** to fix the value of **lease car** to **yes**, which later seems not feasible to the user. The user decides to give up on the **lease car** to try to get more flexibility to **work from home**, but the PN continue to decrease the value and stay very positive on the **lease car**. Since the agent uses the issue weights and issue utilities to translate the bid to the total utility and provide suggestions, the mismatch of issue weight will lead to total utility loss and influence the performance of the PN.

In conclusion: Mismatches occur mostly in agents(strategies) selection and issue weights modification process. These types of mismatches have a significant influence both on the performance of the NSSs and the users' acts. Avoiding these mismatches is difficult since it is unlikely for the system and the user to notice these before strange activities start to show up. However, observing these strange activities provides a chance to interrupt and pause the system. If there is a mechanism that can monitor the actions and detect the mismatch, the users can have an opportunity to change their preferences and correct the mismatch. And with a proper recommendation on the modification, the chance of the mismatch re-appear can be decreased.

This thesis focuses on the mismatch of the issue weights. Hence, for the rest of this chapter, the strategy-selecting process is assumed to be correct. Future thoughts on how to detect the mismatch of the agent(strategy) are given in the last chapter.

4.2.2. Data required to detect the Mismatch

This section introduces and discusses the data required for the mechanism to observe the strange activities of both the mechanism and the user.

In the Bidding phase of the PN, the agent automatically provides a suggestion at the beginning of each round. The users can either accept the suggestion by entering the bid shown on the panel or do some modifications and then enter the bid. If the issue weights made by the users are accurate, with no mismatch in the agent(strategy), the PN should be able to provide a proper suggestion which needs minor or no modification. However, suppose the input issue weights have mismatches compared to the actual issue weights, which means the users wrongly evaluate themselves. In that case, the PN could provide suggestions that way from what the user desire. A situation like this will cause the users to constantly change the bids, which takes time and can be confusing and annoying.

An example of this situation is given with the domain **jobsiV2** in the PN. The actual issue weight has a mismatch compared to the user's issue weight input. The mismatch is stated in Table 4.1.

Issue	Actual Issue Weight	Input Issue Weight
lease car	**0.04**	**0.14**
permanent contract	0.16	0.16
career development opportunity	0.2	0.2
fte	0.3	0.3
salary	**0.2**	**0.1**
work from home	0.1	0.1

Table 4.1: A mismatch in issue weight in **jobsiV2** in the PN

According to the input issue weight of the users, the users think the issue **lease car** is more important than the issue **salary**. The agent in the PN uses this information to translate bids into manageable values and combine these values to calculate the total utility, which is to be used by the agents' algorithms to optimize the total utility to provide suggestions (see Table 4.2). However, the users actually thinks the issue **salary** matters more than the issue **lease car**. The user changes the value and enters the modified bid. This act will occur multiple times until the users notice that they made a mistake in the issue weight.

Issues	Issue Value suggested by the PN	Issue Value made by the user
lease car	**yes**	**no**
permanent contract	yes	yes
career development opportunity	high	high
fte	1	1
salary	**3000**	**4000**
work from home	0	0

Table 4.2: The improper suggestion offered by the PN

To decrease the chance of mismatch as stated in Table 4.1. How the suggestion is generated requires to be figured out. The PN translated the bis to total utility with the utility function [22]. According to the discussion in Section Two. The actual total utility is determined by both the issue weights and the issue utilities which are defined by the users and is not available before the mismatch happened. After the PN receives the information from the users, the PN calculates and evaluate all the possible issue value with the data it obtained. In this way, the total utility calculated by the PN might have a difference compared to the actual total utility of the users. With the agent using the wrong total utility, the PN provides improper suggestions.

The mechanism needs to use data that is available, in this case, the issue values and wrong issue weights, to detect and provide recommendations for updates. In the PN, all the issue values are evaluated after the preference profile is collected. The string-type issue values can be transformed to the evaluated issue values for later comparison.

In a word, the data we could use is stated below:

- Issue weight.
- Evaluated Issue Values.
- Issue Values recommended by the PN.
- Issue Values received from the input of the user.

4.3. Data Logging

To successfully test the mechanism, the data extracted from the PN is used. The System Overview in Figure 4.1 can be expanded, see Figure 4.2.

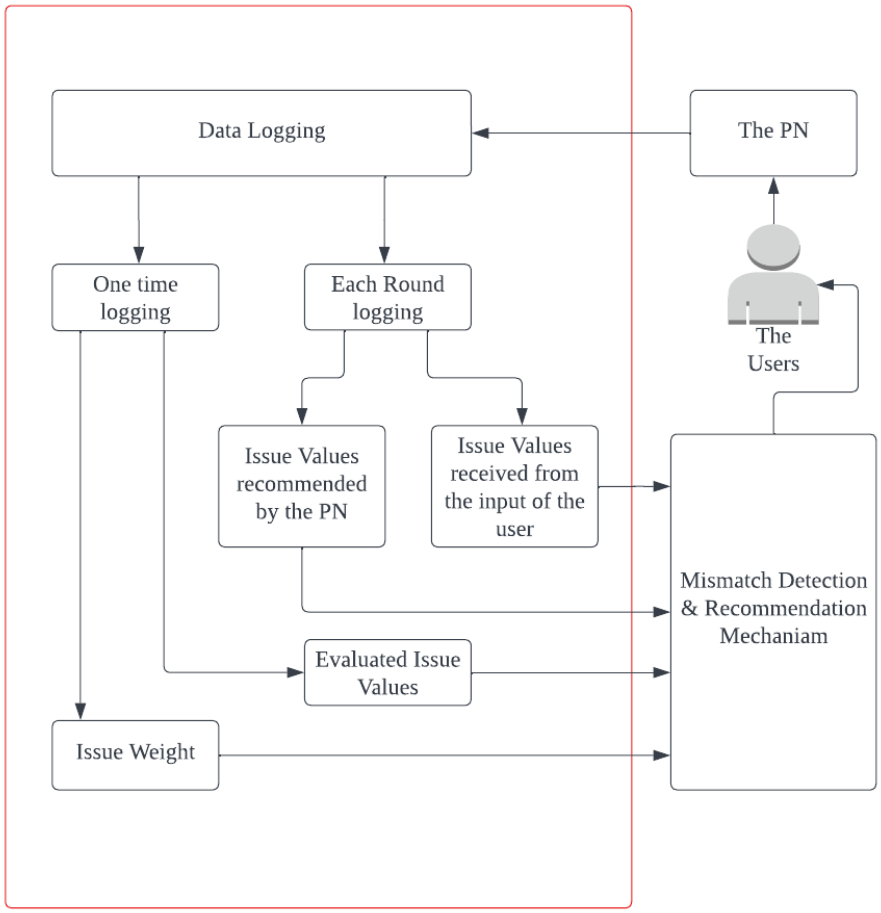

Figure 4.2: Data Logging Process.

In this figure (Figure 4.2), the Data Logging process of the project is presented(in the red block). The PN has its data logging. However, the data logging in the PN only provides the information at the closing phase, which concludes the duration of the bidding phase, the total utility at the end, the percentage of agents' moves etc. The data logging in the PN does not have the data the mechanism requires. Hence, the data logging of this thesis required re-designed.

At the beginning of the negotiation, the PN receives the preference of the users, and a **json** file is generated which contains the information of **Issue weight** and the **Evaluated Issue Value**. Later in the bidding phase, at each round, the **Issue Value** of the PN's recommendation and the user's input is logged out. After this process, the mechanism could have enough information to proceed to detect the mismatches and provide recommendations.

To conclude, the information that can be obtained from the PN is stated below:

- Domain.
- Role.
- R(Duration of the bidding phase).
- W_i: Issue weight.
- $U(i)$: Evaluated Issue Values.
- V_{NSS}: Issue Values recommended by the PN.
- V_{User}: Issue Values received from the input of the user.

The users receive the recommendations and update the preference in the PN and re-do the negotiation and compare the updated issue weights(recommendations) to the actual issue weights logged out from another PN. With the result analysis, the performance of the mechanism can be concluded.

4.4. Confidence Degree

After understanding what is necessary to design the system, it is better to know more about the receiving side of the system, which is the users. The mechanism's goal is to detect mismatches in issue weights and provide updates. On the users' side, when they receive an update, they will always compare the new one to the old one. Different people might react differently. For example, one might believe in the old setting and refuse to put the new setting into the NSSs, while others might completely believe in the new setting and decide to put that into the NSSs. This could be because different people have different self-confidence. People with high self-confidence believe in themselves and the decisions they make. People with low self-confidence tend to question themselves a lot. With the same suggestion, people with low self-confidence might be willing to change, while people with high self-confidence might not. The idea is to persuade the users to try a new setting. To this end, self-confidence should be considered when designing the system.

For the rest of the report, the Confidence Degree is used to represent self-confidence. It is presented as C. When the update mechanism is being designed, take the Confidence Degree into consideration so that people with higher Confidence Degree(self-confidence) will receive fewer changes in the issue weights update.

4.5. Mechanism Design

In this section, the block in Figure 4.1 is expanded. Figure 4.3 represents how the simulation system works. In this Figure, the red block represents the simulation system. **Red** and **Yellow** blocks are mismatch detection and recommendation mechanisms. From Figure 4.3, detailed information about the simulation system is given. When the system starts, it first collects the files from the data-logging process and begins the initialization process in which the system sets the values to prepare the running. In this process, the Confidence Degree is collected, which will be used in the threshold calculation and Issue Weight update process. Within the runnable rounds, the system should detect the mismatch and provide a recommendation if necessary. The system will terminate if the current round Ri is beyond the limit R, or if a recommendation is offered.

In the following sections, how the system is able to detect the mismatch and update the issue weight is given.

4.5.1. Mismatch Detection

In this section, when and how to detect the mismatch and pause the PN is presented. For the first topic, issue value differences are considered. For this thesis, two types of mismatch are discussed. One is the case when a large difference could be observed in the issue values between the input and the suggestions of the PN. Another one is the case when little difference in issue values between the input and the suggestions of the PN is noticed but can be observed multiple times.

As described in Figure 4.3, two scoring methods are implemented in the system. These two methods compare the issue value differences between the PN recommendation and the input by the user, which could be directly observed and contains information about the possible mismatch.

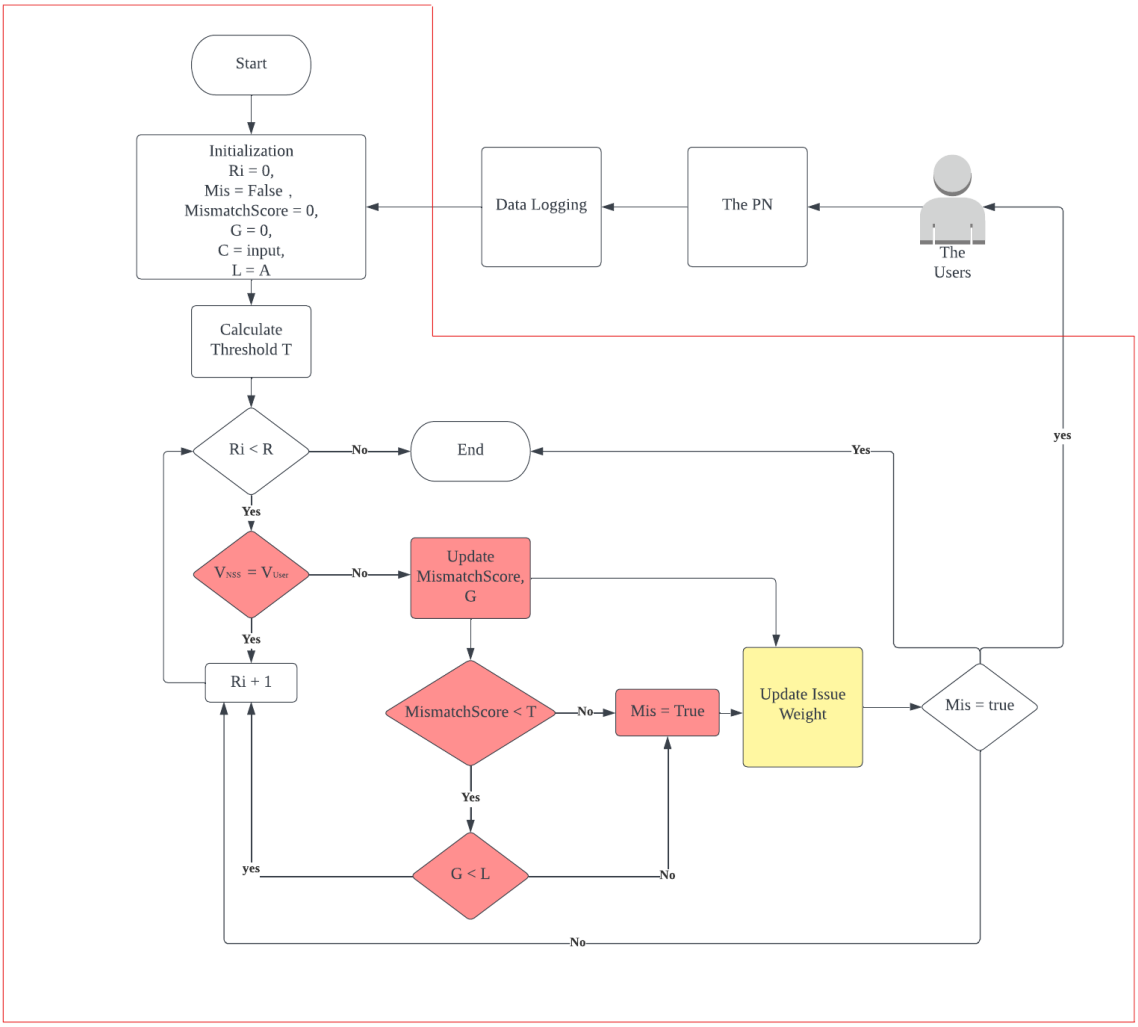

Figure 4.3: Mismatch Detection and Recommendation System Architecture.

In each round of the bidding phase, the issue values of the suggested bid offered by the PN and the modified bid by the users are compared. If there exists a difference, the two method starts to calculate the score for this round. First, the mechanism detects if the difference is large enough to trigger the alarm, and then the mechanism checks if the total times of the differences are beyond a certain limit. In the rest of this section, the two methods are presented and discussed separately.

Large Differences Detection

Assume the user is negotiating over the **jobsiV2** domain. The suggestion offered by the PN shows a difference compared to the users' input, as stated in Table 4.3.

Issues	Issue Value suggested by the PN	Issue Value made by the user
lease car	**yes**	**no**
permanent contract	yes	yes
career development opportunity	high	high
fte	1	1
salary	**2500**	**4000**
work from home	0	0

Table 4.3: The large difference in issue value observed in the PN

The issue value of **salary** offered by the PN and the users has a large difference. The PN suggests a value of **2500** while the user decides it should be **4000**. This difference is significant and can already show there exists a mismatch in the issue weight of salary. The users might have wrongly input this value to be too small. It makes more sense to stop and provide an update on the issue weight rather than keep counting the difference and wait for the mismatch number hit the threshold. Hence, a large difference detection method is required in the mechanism.

The large difference detection method uses a scoring function, and it is designed in three steps. First, a threshold is defined. As discussed before, the Confidence Degree is important. It can be used to determine when to pause the negotiation. For the large difference detection method, the threshold determines when there is a large mismatch and stop. It is defined as:

$$T = M \times C \tag{4.1}$$

T is the threshold for the scoring method, C denotes the Confidence Degree and M is determined later in the case study. The parameter M should be small enough since the large difference in one issue requires immediate action and M will always stay positive ($M > 0$).

Users with high Confidence Degrees tend to believe in themselves, while users with low Confidence Degrees might not. In the mechanism, the thresholds for low Confidence Degree users are smaller than those with higher Confidence Degree, as described in Equation 4.1. In that case, users with low self-confidence can receive recommendations sooner than those with high self-confidence.

Then, whenever a large difference is detected, a score is calculated and updated as follows:

$$MismatchScore(j) = MismatchScore(j-1) + Gain \times \sum_{i}^{I} (W_i \times |U_{User_i}(j) - U_{NSS_i}(j)|) \tag{4.2}$$

j represents the current round number.

$Gain$ is set to 10 to ease the computation.

W_i denotes the issue weight of issue i in issue space I.

$U_{User_i}(j)$ and $U_{NSS_i}(j)$ are the evaluated values of issue i made by the user and the PN in current round j.

$U_{User_i}(j)$ and $U_{NSS_i}(j)$ are set before the negotiation according to the users' preference(issue ranking).

Since all the variables in Equation 4.2 are very small, the $Gain$ is set to 10 so that the computation and the result of the score are more clear. It is not necessary to prefer a particular value of the gain, the users can change this $Gain$ according to their wish.

Whenever large differences are detected, the $MismatchScore$ will be updated. It contains how the differences are evaluated in the PN, and by comparing these evaluated values, how much the differences could be obtained.

At last, the $MismatchScore$ is compared with the threshold T:

$$MismatchScore \geq T \tag{4.3}$$

If the $MismatchScore \geq T$, the mechanism will pause and recommend changing the issue weight.

Multiple Small Differences Detection

Another type of mismatch could be observed like this. During the bidding phase, multiple small differences occur in one issue for multiple rounds. Take **jobsiV2** in the PN as an example(see Table 4.4)

Issues	Issue Value suggested by the PN	Issue Value made by the user
lease car	**yes**	**no**
permanent contract	yes	yes
career development opportunity	high	high
fte	1	1
salary	**3500**	**4000**
work from home	0	0

Table 4.4: The small difference in issue value observed in the PN

The users have a small conflict in the issue value of the issue **salary**, which is normal in some cases. However, this act continues to show up for multiple rounds. This should be an alarm to the user since this indicates that the users have a higher weight in the issue **salary** than the input issue weight, and a mismatch occurs.

To detect this type of mismatch, a score keeps counting the total difference times is introduced:

$$G_i(j) = G_i(j-1) + 1 \tag{4.4}$$

Where $i \in I$ is the issue in issue space I, j denotes the current round number. $G_i(j)$ denotes the current count of the difference in issue i.

At last, the counting score is compared to a limit $L = A$.

$$G_i(j) \geq L \tag{4.5}$$

If the $G_i(j) \geq L$, the subsystem will pause and recommend changing the issue weight.

4.5.2. Update and Recommendation of the Issue Weight

After when to provide the recommendation on changing the issue weights is determined, the system will begin the recommending process(the yellow block in Figure 4.3). The recommendation aims to ask the users to change the issue weights. The difficult part of this process is to get closer to the actual issue weights of the users, which is not available to the system and is in the users' minds.

This thesis designed a Scoring method to update the issue weights each round when necessary, detail of the method could be found below:

Update the issue weight according to the change

This method takes longer for the updated issue weights to reach the actual issue weights of the users. The update function is presented below:

$$W_i(j) = W_i(j-1) + (1-C) \times (W_i(j) \times (U_{User_i}(j) - U_{NSS_i}(j))) \tag{4.6}$$

The $U_{User_i}(j) - U_{NSS_i}(j)$ is the difference between the evaluated issue value of issue i from the NSS and the users' input value of the current round j. Together with the issue weight W_i, determine how much the issue weight can be updated. $(1-C)$ decides how much the issue weight should be updated.

The user with lower self-confidence(C is small) tends to update faster compared to the ones with higher self-confidence.

When the mechanism decides it is time to provide a recommendation, the update process will conclude all the issue weights:

$$W_i = W_i \times \frac{1}{\sum_i^I W_i} \tag{4.7}$$

In this equation, i denotes to issue i, where $i \in I$, I is the issue space, and W_i represents the issue weight. This process is necessary since the sum of the issue weight should be equal to 1.

4.6. Conclusion

The designed mechanism should be able to detect the mismatch and provide recommendations. For the mismatch detection mechanism, it should have the ability to detect whenever there is a large difference in one issue value between the input of the users and the suggestions of the PN or small differences in issue values between the input of the users and the suggestions of the PN occur multiple times, and then the recommendation mechanism should be able to provide an update of issue weights to the users after the mismatch is confirmed. For the recommendation process, this thesis project used updating mechanism to provide recommendations on issue weights, which might take more time to get close to the desired value but should be more reliable and stable. It is noticeable that the Confidence Degree is called several times. With the Confidence Degree being used, the mechanism should be able to provide different recommendations to different users. Higher self-confidence users should receive smaller updates in issue weight recommendations.

The expected outcome of the mechanism is listed below:

- Increase the response time: Since the response time is the waiting rounds until the mismatch alarm is triggered, with updated issue weights being used next, the waiting rounds are expected to extend.
- The difference between the updated issue weights and the actual issue weights (error) should decrease.
- The average error (average difference between the actual issue weights and input issue weights of the users) for higher self-confidence users should expect to decrease slower than the rest.

5

Experiment Setup

In this part, how the experiments of the thesis project were built up is discussed. The simulation experiments indicate that no user tests are done to conclude the performance of the mechanism. The simulation experiments have the advantage of being less time-consuming but increase the rate of subjectivity. To decrease subjectivity, a case study is implemented to cover up situations as much as possible. In the meantime, the person who performed the simulations used another PN to represent the actual issue weights (desired value) to log out the necessary information. A Simulation Overview can be found in Figure 5.1

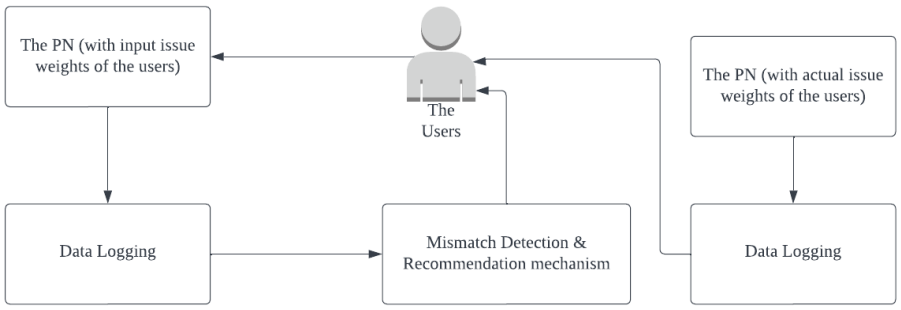

Figure 5.1: Simulation Process.

To start with, the actual issue weights were logged out, and all the data was provided to the users, then the users began the negotiating process with the PN, the data of the negotiation was supplied to the subsystem that used this mechanism, and finally, after the detecting and recommending process, the users can have the results of the simulation.

5.1. Case Study

To decide on what is valuable in a case study, variables that might influence the performance must be figured out first.

The most called parameters and variables in the Equation 4.1, 4.2,and 4.6 are the Confidence Degree C and the issue weights W_i. Hence for the case study, these two topics are considered.

5.1.1. Confidence Degree

The Confidence Degree is the value that shows how much people believe in themselves and the acts they have performed. For this simulation, the Confidence Degree was divided into three levels: Low, Medium and High, with values of 0.2, 0.5 and 0.8, each representing the users with low self-confidence, medium self-confidence and high self-confidence.

5.1.2. Mismatch in Issue Weight

The influences caused by the issue weights mismatches difference depend on the users. Different users have different desires. The most common cases of mismatch are two types:

- Large differences in issue weight (between the actual issue weights and the input issue weights of the users).
- Small differences in issue weight (between the actual issue weights and the input issue weights of the users).

These two cases should cover most real-life situations. Other types, such as completely opposite issues weight in real life, could happen but are very unlikely. Hence this case lacks the value to simulate.

To conclude, with these two variables to establish a case study, it is sufficient to test if the subsystem is reliable.

5.1.3. Conclusion

In conclusion. For the case study, two scenarios are considered, and in each scenario, three Confidence Degrees are tested, in total six simulations are done to test the mechanism, as described in Table 5.1.

Scenario	Confidence Degree		
	Low	Medium	High
Large Differences in Issue Weight	0.2	0.5	0.8
Small Differences in Issue Weight	0.2	0.5	0.8

Table 5.1: Different Confidence Degree settings for the two scenarios

5.2. Simulations Procedures

In this section, the procedures of the experiments are presented. First, the assumptions of the case study are given, some tests were done before the case study[1] to have some ideas about what to focus on, then the results of the simulation are provided, and in the end, the different scenarios are compared and concluded.

5.2.1. Assumptions

As presented in Figure 5.1, another PN with a correct preference setting should be used to extract the actual issue weights of the users for analysis and comparison. The simulations are processed assuming the mismatches only occur in the issue weights, and this indicates that the agent (strategy) chosen is correct. The update mechanism requires some data to get close to the correct value. This requires the negotiation to be a long process. To this end, these hypothesises are made:

1. The PN used to provide correct values of issue weight can represent the actual issue weight of the users.
2. There is no mismatch in the agent (strategy) selecting process.
3. The negotiations are always above 25 rounds.
4. The user will only make conceded moves according to the issue weights' ranking.

5.2.2. Simulation setup

From the test run, we can conclude that the mechanism can detect mismatches in the issue weights setting. The response time suggests that for multiple small differences in issue values, it requested the $L = 14$ in Equation 4.5. This setting is also used in the case study. In the case study, it is necessary to extend the update times to see the influence of the **Over Correct**.In this situation, the simulation for each case has 7 updates.

[1] See Appendix A for detail information.

For the simulation setting, **jobsiV2** was chosen to be the domain, the strategy for the user was the **Pareto optimal conceder**, the strategy for the opponent was the **hardliner**, and the negotiation duration was set to 25. The total update times were 7 and L, which in Equation 4.5 was set to 14.

The agent on the users' side ensured that the suggestions offered would always be on the Pareto optimal frontier. The agents on the opponents' side ensured that the opponents cannot concede, which guaranteed the length of the bidding.

In these simulations, as stated in Section 5.1.1 different Confidence Degrees were tested to see the influence of the **Over Correct**. The setting of the case study is stated below 5.2:

	Large Differences in Issue Weight	**Small Differences in Issue Weight**
Domain	jobsiV2	jobsiV2
Update Times	7	7
Agents on the users' side	Pareto Optimal Conceder	Pareto Optimal Conceder
Agents on the opponents' side	Hardliner	Hardliner
Confidence Degree	0.2, 0.5 and 0.8	0.2, 0.5 and 0.8

Table 5.2: The settings of the simulations

5.3. Determine the threshold T

The parameter M in T as described in Equation 4.1, requires to be defined before the experiment. To do that, a total of five experiments were done. In these experiments, the users manually changed one value of the issue so that it had a very large difference compared to the suggested value, then the mechanism used Equation 4.2 to calculate the score of this mismatch and finally took an average of these five scores. The final result is:

$$MismatchScore = 0.63$$

. The Confidence Degree is used in Equation 4.1. The user with a higher Confidence Degree can have a larger T so that the system can wait longer in comparison to one with a lower Confidence Degree. Therefore, T is defined as:

$$T = 0.63 \times C$$

5.3.1. Analysis Procedure

After the simulations ended, the update of the issue weights was provided to the users. The results can be evaluated in three aspects: Differences between the updated issue weights and the actual issue weights as stated in another PN. For the rest of this chapter, it will be denoted as **error**, total utility at the end of the negotiation and the response time of the system.

For the error, it can directly show the distance to the target issue weight is increased or decreased, which shows the reliability of the mismatch detection mechanism.

For the total utility difference obtained before and after the update of the issue weights, it is difficult to see the performance of the mechanism. Hence, the total utility is moved out from the table.

For the response time, the desired response time is expected to increase after the update of the issue weights. For example, a test run shows that it takes a total of 7 rounds for the mechanism to respond and update the issue weights. The issue weights are close to the actual values of the users but not good enough. Hence a second test is performed with the updated issue weights of the last test as the input of this test. This time, it takes 12 rounds for the mechanism to respond, and a third test uses the output of the second test as input is performed. After this test, the update of the issue weights is very similar to the actual issue weight, and it takes 18 rounds for the system to respond. The extension of the response time indicates that there are fewer and fewer mismatches observed, meaning that the suggestions from the PN are more acceptable to the users, causing the response time of the mechanism to be longer, which shows the improvement of the PN with the mechanism.

In conclusion, the results of the simulations are assessed based on the two aspects below:

- Differences between the input issue weights and the actual issue weights of the users, which are the errors.
- Response time of the system.

5.4. Results of the Case Study

5.4.1. Low Confidence Degree with Large differences in issue weight

In this test, the user chose the confidence degree to be 0.2, which indicated that the user did not have confidence in the issue weights had put in before. There were 7 updates in total for this case. The result is shown in Figure 5.2.

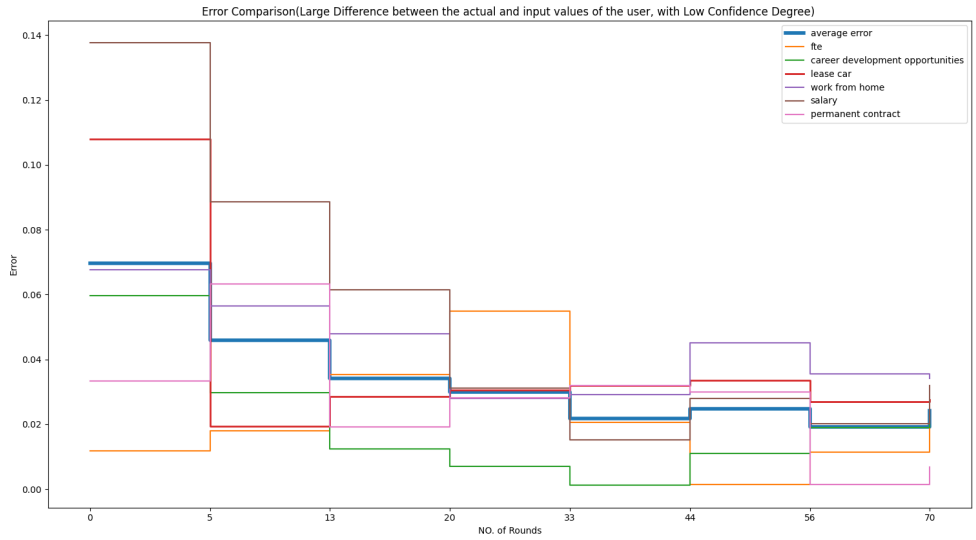

Figure 5.2: Error Comparison (Large difference in issue weight with confidence degree $C = 0.2$).

The detail information about the average error and the response time is stated below:

	No. of Updates							
	0(Initial)	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Average Error	0.0697	0.0459	0.0341	0.0300	0.0217	0.0249	0.0191	0.0243
Response Time (rounds)	0	5	8	7	13	11	12	14

Table 5.3: Average Error and Response Time for C = 0.2 in Large Difference in Issue Weight

In Figure 5.2, the **blue line** is the average error of all updated issue weights; it can be observed that the average error had a trend of decrease at the first 4 updates, which indicates that this detection-update mechanism can do its job. From the 5th update, the average error started to increase. This is because some issues that had already hit the target started to move away, which caused the error for these issues to increase (**lease car**, **work from home**,**fte**, etc.). The average error decreased from 0.0697 to 0.0243, with a minimum of 0.0191 at the 6th update, which means the mechanism self-corrected this problem in the 6th update, but the same problem occurred again.

The response time of each update, as stated in Table 5.3, started with 5 rounds at the beginning and increased to 14 rounds. This indicates that the update mechanism could provide proper suggestions since the users tended to make fewer modifications to the suggestions offered by the PN after receiving recommendations. However, the response time reached 13 rounds and stayed within a range, and this shows the influence of the **Over Correct**[2]. Without a proper method to preserve the issue weights that meet the requirement, **Over Correct** will make the response time stay within a range, preventing the mechanism from helping improve the interaction between the users and the PN.

[2]This is because the mechanism was not aware of the fact that some of the values might already hit the target and continued updating with the negotiation process proceeding. The discussion of **Over Correct** can be found in Section 5.5.5

Since the users had less self-confidence, the mechanism reacted aggressively, which caused the updates to reach the actual issue weights very early, and **Over Correct** occurred. The self-correct of the mechanism can decrease the error but cannot entirely avoid this problem.

5.4.2. Medium Confidence Degree with Large differences in issue weight

In this test, the user chose the confidence degree to be 0.5, which indicates that the user had doubts about the issue weights put in before. Same as the test before, in total 7 updates were considered. The result is shown in Figure 5.3

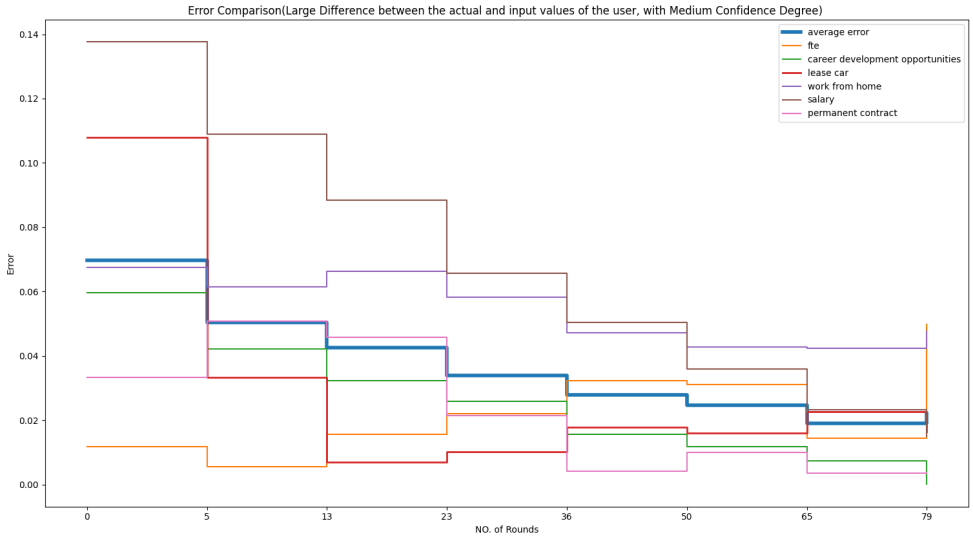

Figure 5.3: Error Comparison (Large difference in issue weight with confidence degree $C = 0.5$).

The average error and the response time for each update can be found below:

	No. of Updates							
	0(Initial)	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Average Error	0.0697	0.0504	0.0426	0.0340	0.0280	0.0246	0.0190	0.0221
Response Time (rounds)	0	5	8	10	13	14	15	14

Table 5.4: Average Error and Response Time for C = 0.5 in Large Differences in Issue Weight

In Figure 5.3, the average error decreased at the first update. It had a trend of decrease for 6 updates and increased at the 7th update due to the **Over Correct** of issue weights. The average error decreased from 0.0697 to 0.0221 and was minimal at the 6th update(0.0190). The **Over Correct** problem occurred at the 7th update but could be observed in the previous update. The error decrease of the first update was more significant than the second and the third. This is because the number of issues which suffer from **Over Correct** increased with the simulation process.

As for the response time of the update (see Table 5.4), it started from 5 rounds and increased to a maximum of 15 rounds, indicating that the suggestions provided to the users were suitable and could help the PN to provide more proper advice. Compared to the response time with the low Confidence Degree case, the **Over Correct** kicked in later, making the response time in the 5th update a bit longer than the low Confidence Degree Case. After the average error decreased to the minimum, it stayed around 15 rounds, the same as the low Confidence Degree case. Also, the **Over Correct** made the response time remain 14 - 15 rounds for some updates, which indicates that the mechanism could be extended and have the potential to further increase the performance of NSSs in the future.

It is a medium Confidence Degree Case, the users had doubts about the choice they made, but not much as the low confidence cases, the mechanism reacts smoothly, and the error continued decreasing for 6 updates, which indicates the idea behind the confidence degree, changes less if the user has more self-confidence.

5.4.3. Large Confidence Degree with Large differences in issue weight

In this test, the user chose the Confidence Degree to be 0.8, which indicates that the user had a higher self-confidence. The mechanism should update less. The result is shown in Figure 5.2.

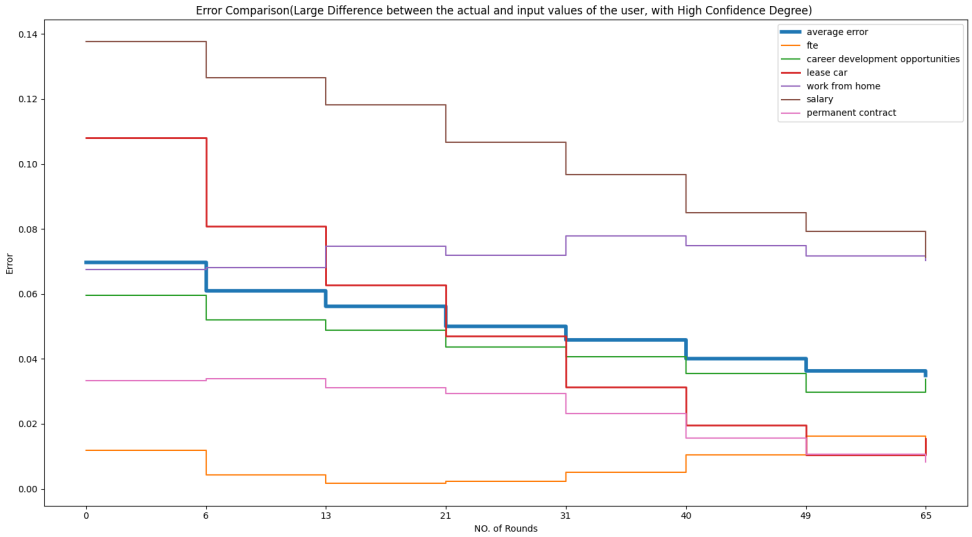

Figure 5.4: Error Comparison (Large difference in issue weight with confidence degree $C = 0.8$).

Detailed information about the average error and the response time are listed below:

	No. of Updates							
	0(Initial)	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Average Error	0.0697	0.0609	0.0562	0.0501	0.0458	0.0401	0.0363	0.0350
Response Time (rounds)	0	6	8	8	10	9	9	16

Table 5.5: Average Error and Response Time for C = 0.8 in Large Differences in Issue Weight

In Figure 5.2, the average error decreased through the process, from 0.0697 to 0.0350. There seemed to be no **Over Correct**, but the error decrease seemed to be lower and lower for the last three updates. This is because the **Over Correct** problems occurred in issue **fte**, **work from home** and **career development opportunities**. Since C was pretty large, according to Equation 4.6, the change for each issue weight was little. These little changes could not trigger the average error to increase from a big view.

As for the response time in Table 5.5, it started with 6 rounds, increased during the process and ended with 16 rounds, which shows the effectiveness of the designed mechanism. Compared to the low and medium Confidence Degree cases, the response time differences between each update were pretty small, around 9 rounds. This shows that with high Confidence Degree, the users received the least changes in updates, which caused the detection process to trigger the alarm sooner in later updates than in the low and medium cases. The response time indicates that the designed mechanism could help the users interact better with the PN and increase the performance of the PN.

The user had higher self-confidence in this case, and the step is smoother than the previous two. This fits the purpose of the mechanism, fewer changes in the recommended values for higher self-confidence users.

5.4.4. Low Confidence Degree with Small differences in issue weight

In this test, the Confidence Degree was set to be 0.2, with small differences between the input issue weights and the actual issue weights (two of the issue had an opposite ranking while the others were very close to the actual values). The result is shown in Figure 5.5.

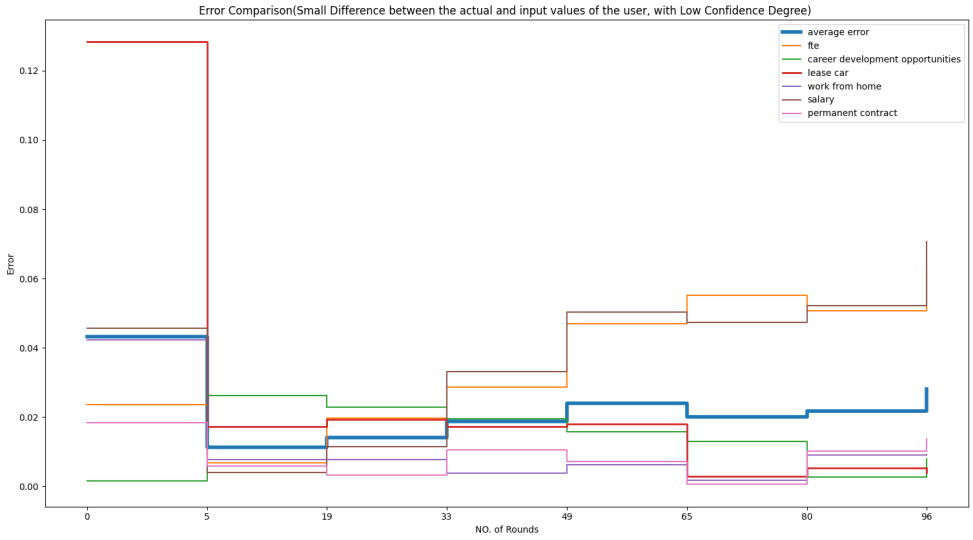

Figure 5.5: Error Comparison (Small difference in issue weight with confidence degree $C = 0.2$).

The average error and the response time of this case can be found below:

	No. of Updates							
	0(Initial)	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Average Error	0.0433	0.0113	0.0140	0.0188	0.0240	0.0201	0.0217	0.0281
Response Time (rounds)	0	5	14	14	16	15	16	16

Table 5.6: Average Error and Response Time for C = 0.2 in Small Differences in Issue Weight

In Figure 5.5, the average error first decreased from 0.433 to 0.0140 and increased from 0.0140 to 0.0281. With small differences and low Confidence Degree, the mechanism suffered from **Over Correct** in the 2nd update. This leads to the fact that the mechanism could not provide enough help for the users in this case, even though the mechanism tried to self-correct at the 5th update. It can be seen that all the issue weights would continue to move away from the actual values. If this continued, at some point, it would not be the case that there were small differences in the issue weights setting. It would be a brand new case (most likely a large difference scenario like the previous sections). For the mismatch issues (in this case, **lease car** and **work from home**), not much decrease can be found. For **lease car,** only the first update had a very significant drop, which shows the ability of the mechanism. For the others, most of the updates showed increases. This is because the input issue weights were very similar to the actual issue weights. This resulted in the lower-ranking issue's issue weights kept decreasing. In comparison, the higher ranking issue's issue weights continued to increase, which would cause both of these issues' errors to grow until they hit a limit.

For the response time in this result, it started with 5 rounds, 14 rounds for the next two updates, reached the highest of 16 rounds, then decreased to 15 rounds and stayed at 16 rounds for the rest of the updates. From the result, it can be concluded that the mechanism could help the users in some way, even though the average error continued to grow after the first update. This is because the error of other issues that **Over Correct**, these **Over Correct** can cause the average error to grow, which influences the issue weights, but not enough to influence the ranking of the issues. Since the mismatch issues had been detected and updated, the response time grew, but the **Over Correct** limited the growth of the response time.

Since this is the case that the users had lower self-confidence but put in a very reasonable setting, the result indicates that the mechanism did its job as it should. The mechanism does not have the ability to avoid **Over Correct**. If it does, the average error should continue decreasing and stay at a very low value.

5.4.5. Medium Confidence Degree with Small differences in issue weight

In this case, the Confidence degree was set to 0.5. The result could be found in Figure 5.6.

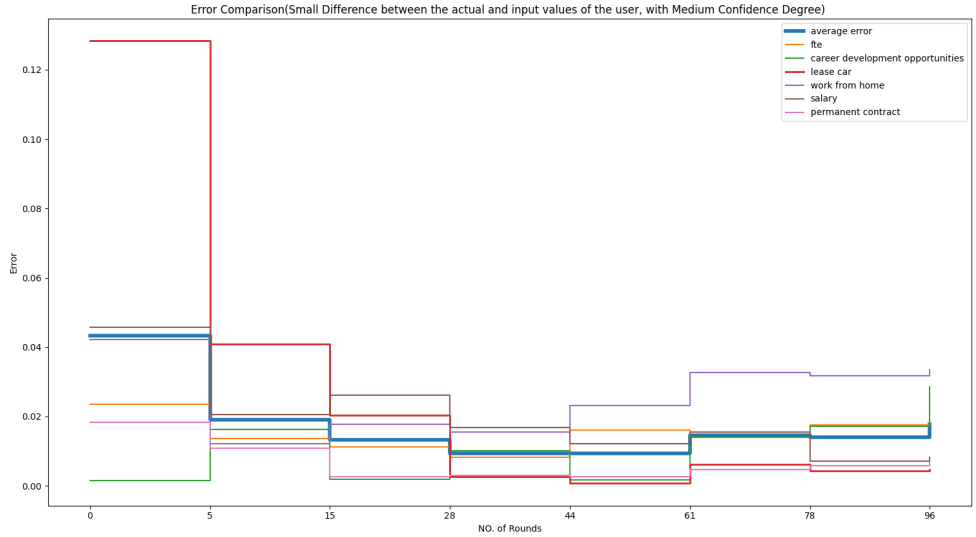

Figure 5.6: Error Comparison (Small difference in issue weight with confidence degree $C = 0.5$).

The average error and the response time are stated below:

	No. of Updates							
	0(Initial)	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Average Error	0.0433	0.0191	0.0133	0.00938	0.00940	0.0145	0.0140	0.0177
Response Time (rounds)	0	5	10	13	16	17	16	18

Table 5.7: Average Error and Response Time for C = 0.5 in Small Differences in Issue Weight

In Figure 5.6, the average error decreased from 0.433 to 0.00938 and then increased. By the last update, it hit 0.0177. This is because the medium confidence degree determined that the issue weights should be updated at a medium speed (According to Equation 4.6). The average error reached its minimum at the 3rd update and started to increase until the last update due to **Over Correct**. Compared with the low Confidence Degree case, the changes in the average error were smaller between each update. Note that in the 4th update, the updated error was almost the same as the second update in a

big view. From the data, it was 0.00002 larger than the second update, and this reflects the influence of **Over Correct**. Though some issues were still reaching the target (**lease car**), others were not, which caused the updated error started to increase. The medium Confidence Degree suggests that the users had higher confidence in the input preference setting. The mismatch issue (**lease car**) is presented in **red line**. According to the change of **lease car**, it is clear that the change in each update was less compared with the low Confidence Degree case.

As for the response time(see Table 5.7), 5 rounds as started and ended with 18 rounds. Compared with the low Confidence Degree case, the increase in the response time was slower. The medium Confidence Degree decided the mechanism should change less in each update, leading the issue weights slowly get to the target location, which fits the purpose of this mechanism.

Since the users have medium self-confidence, the mechanism should be able to provide less change in updates. According to the update of **lease car** and the average error, the mechanism proves to be useful. The mechanism without a way to pause the update might cause the same problem as the low self-confidence case. Causing the lower issue weights to be lower and the higher to be higher.

5.4.6. Large Confidence Degree with Small differences in issue weight

In this case, the Confidence Degree was set to 0.8. The result is shown in Figure 5.7

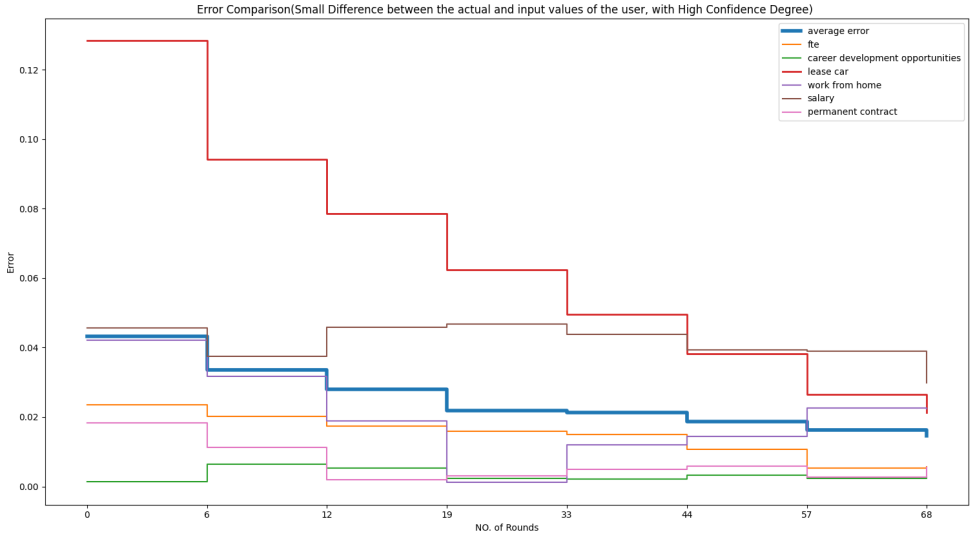

Figure 5.7: Error Comparison (Small difference in issue weight with confidence degree $C = 0.8$).

The information about the average error and the response time is listed below:

	No. of Updates							
	0(Initial)	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Average Error	0.0433	0.0335	0.0280	0.0219	0.0212	0.0187	0.0164	0.0146
Response Time (rounds)	0	6	6	7	12	11	13	15

Table 5.8: Average Error and Response Time for C = 0.8 in Small Differences in Issue Weight

In Figure 5.7, both the **Over Correct** problems and the decrease in the error can be found. The average error decreased from 0.433 to 0.0146. At the 3rd and 4th updates, the **Over Correct** can be observed directly since the average error seems to be the same. This is because the issue **work from home** hit its actual value and started to **Over Correct**. In the fifth update, the mechanism began to

self-correct and further decreased the error. The users of this case were believed to have high self-confidence. This caused the mechanism to think it was better to change less in each update. This can be directly observed in the issue **lease car** in comparison to the previous two cases. The error of **lease car** decreased at a low speed which means the issue weight of **lease car** was getting close to the actual value at a low speed, and by the end of the test, the error of **lease car** reached 0.0317. Compared to the previous, it can still be decreased further, but others might start to **Over Correct**, which will influence the performance of the mechanism

The response time of the mechanism, as stated in Table 5.8, increased steadily from 6 rounds to 15 rounds, indicating the mechanism is effective. The updated issue weights can help the NSSs work better. The response time was not as long as the previous cases because the Confidence Degree was high enough for the mechanism to believe that there was less that needed to change, which resulted in fewer updates in each response, which caused the mismatch issue weight (**lease car**) moved little by little to the actual value. Until it hit the correct value, the response time would increase accordingly. This also made the influence of the **Over Correct** not as much as in previous cases.

In this case, the users had a higher self-confidence. The mechanism should recognize this as fewer changes should be given to the user. The result proves that the mechanism worked as it should.

5.5. Discussion

5.5.1. Low Confidence Degree

A low confidence degree means that the update should change more. In the case study, two scenarios were considered. In the first case, large differences (multiple issue weights mismatch) existed between the input issue weights and the actual issue weights (the result is in Figure 5.2). In the second case, small differences could be found between the input and actual values (result is in Figure 5.5). Both results indicate that the mechanism could detect the mismatch and update the issue weights due to the fact that the response time continued to grow during the process for both cases).

However, there was no function to determine if the updated values hit the target, which caused the **Over Correct** to show up. For the first case, since there were lots of wrong inputs, the **Over Correct** occurred much later than in the other cases. The **Over Correct** can have influences. In the first case, it occurred at the 5th and 7th updates (44, 70 rounds). As for the second case, the **Over Correct** happened at the beginning, without knowing when it should stop and preserve the value that was close to the actual one; the mechanism continued to update, and at some point (49 rounds), this caused the suggestion to have a significant difference, similar to the first case. The mechanism can self-correct to a certain level with the negotiating process.

5.5.2. Medium Confidence Degree

A medium Confidence Degree suggests the mechanism should update the issue weights at a medium rate. The same scenarios described before are used here, with large differences and small differences between input and actual issue weights (results can be found in Figure 5.3 and Figure 5.6). The **Over Correct** could also be observed for both cases (79 rounds in Figure 5.3 and 44 rounds in Figure 5.6). Both results show the functionality of detecting mismatch, but in the second case, the **Over Correct** caused the average error to grow much earlier than in the first case. The response time for the first case indicates that the mechanism required multiple attempts to reach the actual value. This means that in real life, if multiple mismatches occur simultaneously, the users with medium Confidence Degrees tend to believe in the previous setting, multiple attempts can give them hints that they made mistakes indeed, which fulfil the purpose of the Confidence Degree in the mechanism.

5.5.3. High Confidence Degree

A high Confidence degree indicates that the users believe in the preference setting they made before. This means the mechanism will update the issue weights slowly to persuade the users to change the input. Same scenarios as previous sections are used for this part (results can be found in Figure 5.4 and Figure 5.5). The average errors of these two cases had a similar pattern. They had trends of decreasing. The average error decreased throughout the process. In the second case, it had some apparent signs of **Over Correct**. However, the **Over Correct** did happen in the first case, which caused the final two steps to be shallower compared to the previous steps. As for the response time, both cases show growth in time, indicating that the mechanism can provide a proper update. The response time for

both cases was shorter than the rest, which means there might still be mismatches in the issue weights that have not been detected, which means that the mechanism can help people and the NSSs to some extent, under the sacrifice of losing some potential performance of NSSs. Since these were the cases where the users had high self-confidence, the slow updates fit the purpose of the mechanism.

5.5.4. Large Differences VS Small Differences

The comparison of the results between the two scenarios can be found in Table 5.9

Scenario	Initial Average Error	Average Error at the end for Different C		
		0.2	0.5	0.8
Large Differences in Issue Weight	0.0697	0.0243	0.0221	0.0350
Small Differences in Issue Weight	0.0433	0.0281	0.0177	0.0146

Table 5.9: The comparison of the results between the two scenarios.

In both scenarios, the average error decreased at first. For **Large Differences in Issue Weight**, and for different Confidence Degree cases, the average errors decreased from 0.0697 to 0.0243 ($C = 0.2$), 0.0221 ($C = 0.5$) and 0.0350 ($C = 0.8$). For **Small Difference in Issue Weight**, and for different Confidence Degree cases, the average error showed the same trend as the previous scenario. It decreased from 0.433 to 0.0281 ($C = 0.2$), 0.0177 ($C = 0.5$) and 0.0146 ($C = 0.8$). This means that although **Over Correct** existed, the mechanism could successfully detect the mismatches that occurred for both scenarios. One of the differences is how much the errors decreased (See Table 5.10).

Scenario	Decrease of Average Error at the end for Different C		
	0.2	0.5	0.8
Large Differences in Issue Weight	0.454	0.476	0.347
Small Differences in Issue Weight	0.152	0.256	0.4184

Table 5.10: The decrease of average errors between the two scenarios.

For **Large Differences in Issue Weight**, the change of average error for the three Confidence Degree cases were 0.454, 0.476 and 0.347, while in the other scenario, the change of average errors were 0.152, 0.256, 0.4184. It is clear that the average error of low and medium Confidence Degree cases in the first scenario decreased more compared to the second scenario. This is because, for **Large Differences in Issue Weight** scenario, the differences between the actual issue weights and the input issue weights were larger than **Small Differences in Issue Weight** scenario. The change in average error for high Confidence Degree cases in the second scenario was larger than the first one. This is because the second scenario had small issue weight mismatches in multiple issues, and the mechanism decided to provide minor changes in each update, which made the error change little by little, making **Over Correct** occurred less. With little **Over Correct**, the errors were able to further decrease. This did not happen the same for the first scenario since the errors for all issues were too large, and the update was too slow for the average error to decrease like low and medium cases. It can be foreseen that if the high Confidence Degree case could be extended for the first scenario, the change of the average would have shown a similar result to the second one.

As for the response time. All the cases in both scenarios had an increased response time, meaning that the mechanism could improve the performance of the PN. The response time had a difference compared to the Test Run, they were shorter at the beginning (5 rounds for the first update for low and medium Confidence Degree cases, 6 rounds for the high Confidence Degree cases) than the response time of the Test Run (19 rounds) [3], this is because the large difference detection as stated in Equation 4.2 was on and there existed very large differences in between the suggested issue values and input issue values, which fulfil the idea of the mechanism (pause earlier if necessary).

[3]The detail information about test run before the simulations can be found in Appendix A

5.5.5. Over Correct

In the Case Study, **Over Correct** is a topic that has been mentioned frequently. To understand this better, it is critical to understand how the **Over Correct** is generated.

The mechanism designed for this thesis can detect the mismatches and provide recommendations about the issue weights but not include the topic of how to guess the actual issue weight of the users, which is why these experiments were done based on the assumption that the actual issue weights of the users were known beforehand. The mechanism does not have the function to detect if the recommendations are close to the actual issue weights. If the mechanism users continue listening to the recommendations, the mechanism will keep updating the issue weights, and the issue weights close to the actual issue weights will be updated and move away from the actual value, leading to the **Over Correct**.

The influence of the **Over Correct** cannot be neglected since not only does it influence the response time of the mechanism (reflecting the improvement of the interaction between the users and the PN), but also it makes the PN give up some issue values too soon. The **Over Correct** problems can make the lower issue weights to be lower, and in the PN, the total utility is determined by the issue weights, the lower issue weights always gets abandoned by the agents in the PN to achieve a higher total utility, which will cause the suggestions made by the PN to give up on these issues at the very beginning, and sometimes it is not what the users of the PN want.

In a word, **Over Correct** is the main issue of the mechanism, and it requests further study to solve this problem.

5.5.6. Limitations of the Case Study

One major limitation is that the case study cannot cover all the real-life scenarios. In real life, when people negotiate, they think of things from lots of aspects. For example, there might be a case that when a person negotiates, he indeed puts in his right preference, and for the first half of the negotiation, he is happy with the suggestions offered by the NSSs, but later influenced by something and then there is a mismatch, he starts to change the bids quite often. The case study did not cover this scenario. The case study was done under many assumptions. One of them is that the users of the system will always concede. In real life, the bid can go up and down. Therefore, the update of the issue weights might request more attempts until they can reach the actual values. Another limitation is that in the case study, the actual issue weights were assumed to be known beforehand. In real life, they are not.

The purpose of the case study is to know the potential of the designed mechanism. It is essential to perform user tests in the future.

5.5.7. Conclusion

The results of the Case Study show that the mechanism can detect mismatches and provide recommendations for different users. The Confidence Degree, together with the update function, proved to be helpful. With the help of the Confidence Degree, the mechanism can provide different recommendations according to the users' self-confidence levels. The users who believe in themselves can receive recommendations similar to the previous input. In contrast, the users who do not believe in themselves can receive recommendations that change a lot compared to the input they made before.

The response time, as shown in the results, suggests that the mechanism can help the PN to cope better with the users. The users of this mechanism can pause the negotiation and update the issue weights, and with updated issue weights, the PN can provide better suggestions that fit the users' requirements.

The main problem with this mechanism is that it lacks the ability to preserve the updated values that are close to the actual values of the users, which will cause the mechanism to continue updating the issue weights and lead to **Over Correct** and further influence the performance of the PN, making the response time stay still, losing the potential of improving the performance of the PN further.

Another problem is that the mechanism cannot make the issue weights go to the target directly. It requests multiple iterations for the error to be down to an acceptable level. It does not influence the accuracy of the mechanism, but it is too time-consuming, and in real life, the negotiation might not last that long. Therefore to solve this problem, it requests further research.

6

Conclusion and Future Work

6.1. Conclusion

The advantages and disadvantages of the mismatch detection and preference profile recommendation mechanism can be concluded by analysing the case study.

For both scenarios, after the users changed their issue weights according to the recommendations provided by the mechanism, the response time increased, which means the PN could generate more proper suggestions with the help of the mechanism, indicating that the PN could perform better with this mechanism.

In each scenario with different settings of the Confidence Degree, the average error [1] decreased at the beginning, As for the error of each issue for both scenarios, the errors of most issues had signs of decreasing. Therefore the update mechanism can make the input issue weights closer to the actual issue weights.

The Confidence Degree decides how much the mechanism should update the input issue weights. From the results, it can be concluded that the mechanism has the ability to provide different recommendations to different users.

The results prove that the designed mechanism is able to cope with both the users and the NSSs. The detecting method can inform the users about the possible mistakes they made, the recommending process can provide issue weights update recommendations according to the Confidence Degree and with the users correcting the mistakes, the NSSs can provide better suggestions.

The advantages of this mechanism can be concluded as follows:

- This mechanism can improve the NSSs' performance.
- This mechanism can detect mismatches in issue weight.
- This mechanism is able to provide different update recommendations of input issue weights to different users.
- This mechanism can provide updates on issue weights and improve the interaction between the users and the NSSs to some extent.

The main disadvantage of this mechanism is that it cannot avoid the **Over Correct**. This thesis focused on developing a mechanism to detect mismatches in the input issue weights of the users and provide recommendations for updating the issue weights, it did not cover the topic of predicting the actual issue weights of the users. This results in the mechanism do not have the ability to stop the update when the update issue weights are close to the actual issue weights, and after some time, the mechanism would start to over-correct the issue weights, causing the highest value to be higher and the lowest to be lower, as described in Chapter 5. The **Over Correct** always occurred after one or more issue weights reached their actual values. It can influence the mechanism's performance, causing the response time to pause and further affecting the performance of the NSSs.

[1] Differences between the update issue weights and the actual issue weights of the users.

The case study suggests that the update mechanism requires multiple attempts for the input issue weights to reach the actual issue weights of the users. For all the cases, the average errors showed decreased but cannot reach the minimal with one recommendation. One reason is that the Confidence Degree can cause the update for each attempt to be less. The other reason is that the mechanism does not have the ability to decrease the error in one attempt. From Low Confidence Cases (The change of average error between each update should be the most) for both scenarios, the average error decreased significantly at the first update, and when the users applied the recommendations and used the updated issue weights to negotiate, the average errors decreased again. This suggested that the mechanism cannot correct the mismatches in one attempt in some cases. It is an iterative process.

The disadvantages of this mechanism can be concluded as follows:

- This mechanism cannot preserve the issue weights close to the actual issue weights (actual issue weight is impossible to determine in real life), causing the mechanism to over-correct.
- The mechanism cannot provide recommendations that is close to the actual issue weights of the users in one attempt.

There could be improvements to the thesis as well. Due to many reasons, this mechanism is tested with a case study. It makes the results subjective and cannot cover all possible cases. Also, due to time limits, the mechanism does not cover all types of mismatches that could happen, such as strategy mismatches in the users' preferences. If a focus group could be formed to discuss all possible cases, better simulations of the case study might have been able to help further analyse possible disadvantages. Also, if time allowed, adding other methods to detect other types of mismatches could have reduced the assumptions of the simulations and maybe the performance of the PN could have been further improved.

In conclusion, the designed mechanism can help improve the interaction between the users and the NSSs, it can provide proper updates for different users, but limitations still exist and require further research. In the next section, some ideas about how to reduce the limitations and possible directions to extend the mechanism are given and discussed.

6.2. Future Work

This thesis provides ideas on detecting the mismatches of issue weights and provides thoughts on how to update the issue weights according to the input of the users. Also, this thesis introduces an approach that uses self-confidence as a standard to provide different recommendations to different users.

This thesis performed a case study but not in real life. It means that some settings of the experiments are too ideal. In real life, some of these settings can not be fixed, and with more flexibility, it brings challenges to extend the mechanism. Some of the challenges are listed below:

- Find out the actual issue weights of the users: The case study assumes that we already know the actual issue weights of the users. However, in real life, it is impossible to know the actual issue weights before the negotiation. Without knowing the ideal values, it is challenging to design a mechanism that can preserve the update issue weights that is close to the actual issue weights of the users. Getting into a person's mind to know what they want is impossible. One way to approach this is to further increase the connection between the human and the NSSs. It might be possible to set up a question base and sort the questions based on the domains of the negotiation and before negotiation, the user first needs to answer some questions and, based on that, the NSSs can guess the actual issue weight of the users. And with the guessing issue weights, finding a way to preserve the issue weight is what needs to be done next.
- Reduce the update times: For this thesis, reaching the actual issue weight in one attempt is not always possible. Decreasing the number of attempts could increase the performance of the mechanism and NSSs. Future studies could be done on this to solve this problem.
- Extend the mechanism so it can detect other types of mismatch: Mismatches can occur in other parts of the NSSs, such as strategy settings. It is possible to flag the NSSs' suggestions and the input bids of the users and design another grading method to achieve detecting the mismatches. Detecting mismatches and then providing proper suggestions for both issue weights and strategies could also be an interesting topic.

- Introducing Trust in the mechanism: The Confidence Degree used in this thesis represents the self-confidence of the users. There exists another type of confidence. This confidence reflects the trust between the users and the NSSs. This confidence can determine the frequency of the differences between the input bids and the suggestions. It might be an interesting research direction to modify the mechanism so that the recommendations could have more level and more precise.

The above challenges are very attractive and interesting, but due to limited time, this thesis was not able to try to solve the above challenges. However, this thesis can still be a good starting point. Ideas behind the mechanism might be useful when someone decides to research the above topics.

A

Test run before simulations

In this test run, **jobsiV2** was the chosen domain, the Confidence Degree was set to 0.2, with a total negotiation round of 25, and The number of updates was 3. Also, for the Test Run, the large differences detection, which is used to detect significant differences in one issue values between the suggestions of the PN and the input values, was off to check if the mechanism is able to alarm earlier later in the case study.

The results of the three tests are shown below:

First Run
The difference in comparison to the actual issue weight is shown in Figure A.1

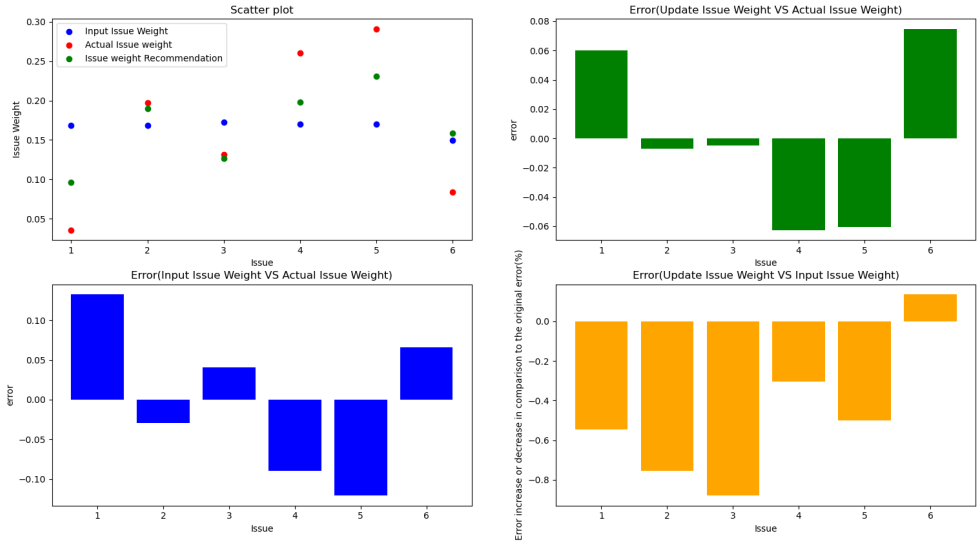

Figure A.1: Error comparison with Low Confidence Degree (First Run).

In Figure A.1, the comparisons are made. There are four images, each of which is discussed below.

The up-left image in Figure A.1 shows the input, update and actual Issue Weights, in which blue dots indicate the input of the user, the red dots represent the actual Issue Weights (Target Issue Weight) and the green dots show the update Issue Weights, from this image, it can be observed that with Confidence Degree C = 0.2, the update mechanism can already let the input get close to the target values, especially for Issue 2 and 3 (**permanent contract** and **career development opportunities**). For Issues 1, 4 and 5, the update values (green) showed great movement toward the target.

For the up-right and down-left images, the sign of the values indicates the orientation of the green and blue dots to the red dots, while the values of the green and blue dots represent the differences.

The up-right image in Figure A.1 indicates the distance between the target and the update values. From this image, it can be seen that Issues 1, 4 and 5 have more distance (error) in comparison to Issues 2 and 3.

The down-left image in Figure A.1 is the result that shows the difference between the input and the target in each issue of the issue space. From this image, how much each value still requires updating is demonstrated.

The down-right image in Figure A.1 is the performance of the update mechanism. In this image, the error after the update is compared with the original error, in which how much the errors were increased or decreased is demonstrated. It can be concluded that in the first run, the error in each issue had already reduced a lot except for Issue 6.

Second Run

The update values of the **First Run** were used as input for this run. In Figure A.2, the comparisons are made.

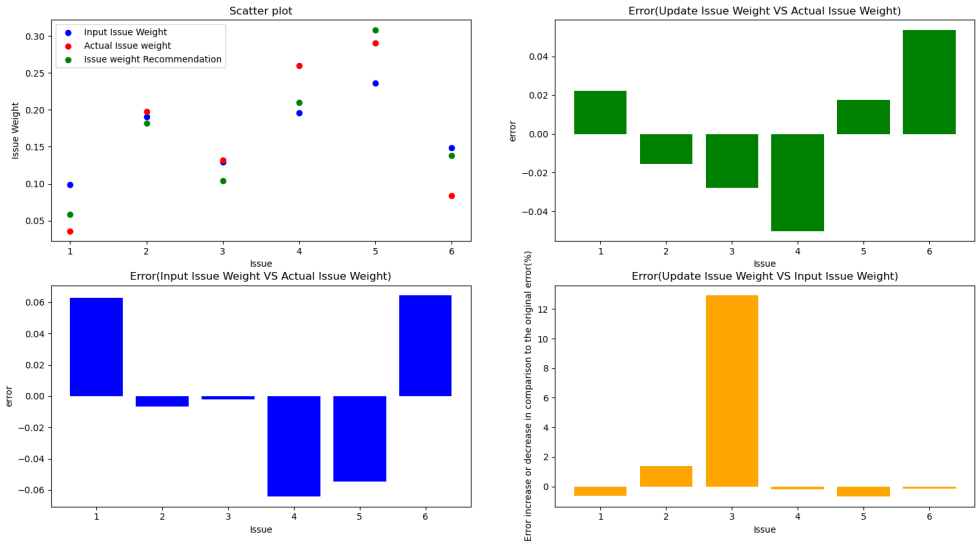

Figure A.2: Error comparison with Low Confidence Degree (Second Run)

The up-left image in Figure A.2 shows the input, update and actual Issue Weights. In this image, almost all the update values (green) show movements toward the target values (red). As can be seen in this image, Issue 3 in the First Run already hit the target in this Run went further which caused the **Over Correct** issue.

The up-right image in Figure A.2 as described before, shows the distance of the update values to the target. Though it seems that the error increased a bit compared to the image in Figure A.1, it did not since the y-axis value is about $\frac{1}{2}$ compared to the First Run.

The down-right image in Figure A.2 indicates that the distance decreased. Issue 6 shows a bit increase, this is because the suggested values tended to go up and down which averages out the update. However, for Issue 3, the error increases due to **Over Correct**.

Third Run

The result of the **Third Run** can be found in Figure A.3

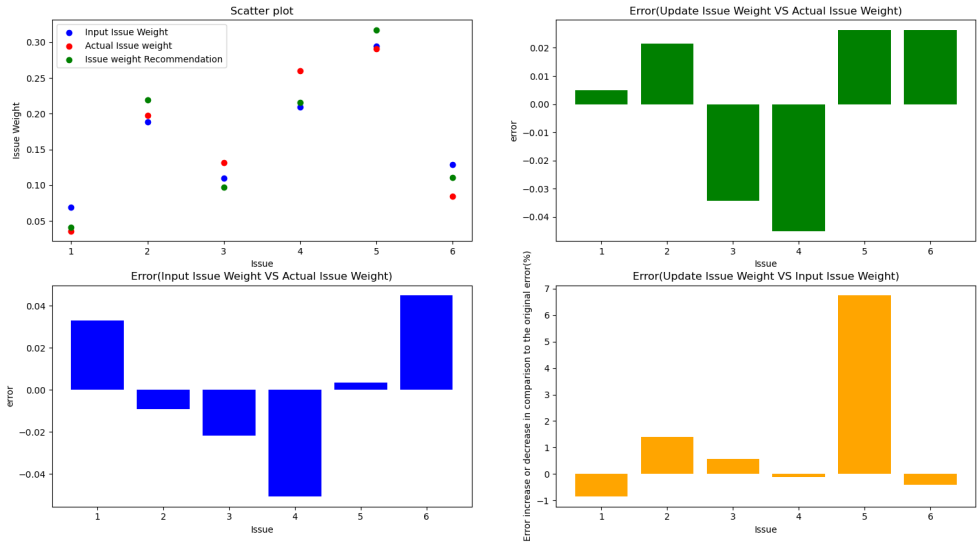

Figure A.3: Error comparison with Low Confidence Degree (Third Run)

In Figure A.3, the comparisons are made.

The up-left image in Figure A.3 shows the input, update and actual Issue Weight. Issue 1 finally hit the target. And there existed the problems of **Over Correct**. Issue 6 was getting toward the target, and the issues that already hit the target in Second Run were moving a bit far from the target.

The up-right image in Figure A.3, is the same as described in the second run, the distance, in fact, decrease in comparison to the First and Second Run.

The down-right image in Figure A.3 shows the change of the error. It is clear that the most severe **Over Correct** occurred in the process of update of Issue 3. Other Issues' error increased except for Issue 1 and Issue 6, which shows the same result as the up-left image.

Conclusion

In this part, the result of the average error of all issues updated for each Run is presented in Table A.1

	Original	First Run	Second Run	Third Run
Average Error	0.0797	0.045	0.031	0.026
Response Speed		19	20	24

Table A.1: Error and Response Speed (Low Confidence Degree)

From Table A.1, the change in the average error(updated issue weights compared to the actual issue weights) can be observed. The error has a significant decrease, as described in Table A.1, from 0.797 to 0.026, which indicates that the update mechanism could get close to the actual values. In Table A.1, the response time increased during the three experiments, which indicates that with a low Confidence Degree of the user, the mismatch detection can successfully pause the system. The increase in the response time indicates that when the update was getting close to the correct value, the time the mismatch score hit the threshold would be longer. Note that from this Table, the **Over Correct** cannot be directly observed. It can be said that there is **Over Correct** since the differences in the average error decreased. In the case study, it is necessary to find another way to present the result.

Bibliography

[1] Reyhan Aydoğan and Catholijn M Jonker. Bidding support by the pocket negotiator improves negotiation outcomes. In *Recent Advances in Agent-Based Negotiation: Applications and Competition Challenges*, pages 52–83. Springer, 2023.

[2] Tim Baarslag, Michael Kaisers, Enrico H. Gerding, Catholijn M. Jonker, and Jonathan Gratch. When will negotiation agents be able to represent us? the challenges and opportunities for autonomous negotiators. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 4684–4690, 2017. doi: 10.24963/ijcai.2017/653. URL https://doi.org/10.24963/ijcai.2017/653.

[3] K Bansch, Axel Hein, Manish Malhotra, and K Trivedi. Comment/correction: Dependability modeling using petri nets. *IEEE Transactions on Reliability*, 45(2):272–273, 1996.

[4] Andrew Beswick and Peter Brindle. Risk scoring in the assessment of cardiovascular risk. *Current opinion in lipidology*, 17(4):375–386, 2006.

[5] Andrea Bobbio and D Codetta Raiteri. Parametric fault trees with dynamic gates and repair boxes. In *Annual Symposium Reliability and Maintainability, 2004-RAMS*, pages 459–465. IEEE, 2004.

[6] Zhang Bofeng and Liu Yue. Customized explanation in expert system for earthquake prediction. In *17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'05)*, pages 5 pp.–371, 2005. doi: 10.1109/ICTAI.2005.54.

[7] Ronen Brafman and Carmel Domshlak. Preference handling-an introductory tutorial. *AI magazine*, 30(1):58–58, 2009.

[8] Muhammad Arif Budiyanto and Haris Fernanda. Risk assessment of work accident in container terminals using the fault tree analysis method. *Journal of Marine Science and Engineering*, 8(6): 466, 2020.

[9] P Chamnan, RK Simmons, SJ Sharp, SJ Griffin, and NJ Wareham. Cardiovascular risk assessment scores for people with diabetes: a systematic review. *Diabetologia*, 52(10):2001–2014, 2009.

[10] Eva Chen, Rustam M. Vahidov, and Gregory E. Kersten. Agent-supported negotiations in the e-marketplace. *Int. J. Electron. Bus.*, 3:28–49, 2005.

[11] Daniel Druckman, James Druckman, and Tatsushi Arai. e-mediation: Evaluating the impacts of an electronic mediator on negotiating behavior. *SSRN Electronic Journal*, 01 2004. doi: 10.2139/ssrn.573561.

[12] Joanne Bechta Dugan, Salvatore J Bavuso, and Mark A Boyd. Fault trees and sequence dependencies. In *Annual Proceedings on Reliability and Maintainability Symposium*, pages 286–293. IEEE, 1990.

[13] Theodore Eisenberg and Charlotte Lanvers. What is the settlement rate and why should we care? *Cornell Law Faculty Publications*, 2009.

[14] Hrishikesh J Goradia. Ants with limited memory for solving constraint satisfaction problems. In *2013 IEEE Congress on Evolutionary Computation*, pages 1884–1891. IEEE, 2013.

[15] Boxin Guan, Yuhai Zhao, and Yuan Li. An improved ant colony optimization with an automatic updating mechanism for constraint satisfaction problems. *Expert Systems with Applications*, 164: 114021, 2021.

[16] Koen V Hindriks and Catholijn M Jonker. Creating human-machine synergy in negotiation support systems: Towards the pocket negotiator. In *Proceedings of the 1st International Working Conference on Human Factors and Computational Models in Negotiation*, pages 47–54, 2008.

[17] Christopher G Isles, Lewis D Ritchie, Peter Murchie, and John Norrie. Risk assessment in primary prevention of coronary heart disease: randomised comparison of three scoring methods. *Bmj*, 320 (7236):690–691, 2000.

[18] Catholijn M Jonker. The pocket negotiator, synergy between man and machine. *NWO Grant proposal*, 2007.

[19] Catholijn M Jonker and Reyhan Aydoğan. Deniz: A robust bidding strategy for negotiation support systems. In *International Workshop on Agent-Based Complex Automated Negotiation*, pages 29–44. Springer, 2018.

[20] Catholijn M Jonker and Tim Baarslag. Negotiating agents(lecture reading from tu delft), 2022.

[21] Catholijn M Jonker, Koen V Hindriks, Pascal Wiggers, and Joost Broekens. Negotiating agents. *AI Magazine*, 33(3):79–79, 2012.

[22] Catholijn M. Jonker, Reyhan Aydogan, Tim Baarslag, Joost Broekens, Christian A. Detweiler, Koen V. Hindriks, Alina Huldtgren, and Wouter Pasman. An introduction to the pocket negotiator: A general purpose negotiation support system. In Natalia Criado Pacheco, Carlos Carrascosa, Nardine Osman, and Vicente Julián Inglada, editors, *Multi-Agent Systems and Agreement Technologies*, Lecture Notes in Computer Science, pages 13–27. Springer, 2017. ISBN 978-3-319-59293-0. doi: 10.1007/978-3-319-59294-7_2.

[23] Bernhard Kaiser. Extending the expressive power of fault trees. In *Annual Reliability and Maintainability Symposium, 2005. Proceedings.*, pages 468–474. IEEE, 2005.

[24] Bernhard Kaiser, Catharina Gramlich, and Marc Förster. State/event fault trees—a safety analysis model for software-controlled systems. *Reliability Engineering & System Safety*, 92(11):1521–1537, 2007.

[25] Ching-Torng Lin and Mao-Jiun J Wang. Hybrid fault tree analysis using fuzzy sets. *Reliability Engineering & System Safety*, 58(3):205–213, 1997.

[26] Xia Liu, Yejun Xu, Yao Ge, Weike Zhang, and Francisco Herrera. A group decision making approach considering self-confidence behaviors and its application in environmental pollution emergency management. *International Journal of Environmental Research and Public Health*, 16(3): 385, 2019.

[27] Manish Malhotra and Kishor S Trivedi. Dependability modeling using petri-nets. *IEEE Transactions on reliability*, 44(3):428–440, 1995.

[28] Ivan Marsa-Maestre, Mark Klein, Catholijn M. Jonker, and Reyhan Aydoğan. From problems to protocols: Towards a negotiation handbook. *Decision Support Systems*, 60(0):39–54, 2014. ISSN 0167-9236. doi: http://dx.doi.org/10.1016/j.dss.2013.05.019. URL http://www.sciencedirect.com/science/article/pii/S016792361300167X. Automated Negotiation Technologies and their Applications.

[29] Ivan Marsa-Maestre, Mark Klein, Catholijn M Jonker, and Reyhan Aydoğan. From problems to protocols: Towards a negotiation handbook. *Decision Support Systems*, 60:39–54, 2014.

[30] Theodore M Newcomb. An approach to the study of communicative acts. *Psychological review*, 60(6):393, 1953.

[31] Filip Đoković et al. E-negotiation: Can artificial intelligence negotiate better deals? In *Sinteza 2020-International Scientific Conference on Information Technology and Data Related Research*, pages 289–294. Singidunum University, 2020.

[32] Howard Raiffa. *The art and science of negotiation: How to resolve conflicts and get the best out of bargaining*. Harvard University Press, Cambridge, MA, 1982.

[33] Howard Raiffa. *Negotiation analysis: The science and art of collaborative decision making*. Harvard University Press, 2007.

[34] Daniele Codetta Raiteri, Giuliana Franceschinis, Mauro Iacono, and Valeria Vittorini. Repairable fault tree for the automatic evaluation of repair policies. In *International Conference on Dependable Systems and Networks, 2004*, pages 659–668. IEEE, 2004.

[35] Ariel Rubinstein. Perfect equilibrium in a bargaining model. *Econometrica*, 50(1):97, 1982.

[36] Enno Ruijters and Mariëlle Stoelinga. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer science review*, 15:29–62, 2015.

[37] Mareike Schoop, Aida Jertila, and Thomas List. Negoisst: a negotiation support system for electronic business-to-business negotiations in e-commerce. *Data & Knowledge Engineering*, 47(3):371–401, 2003. ISSN 0169-023X. doi: https://doi.org/10.1016/S0169-023X(03)00065-X. URL https://www.sciencedirect.com/science/article/pii/S0169023X0300065X. The language/action perspective.

[38] Wenhua Song, Huifang Shi, and Qinggong Li. Study of an explanation mechanism in expert system based on fault tree for safety risk assessment. In *2010 2nd International Conference on Future Computer and Communication*, volume 2, pages V2–479–V2–483, 2010. doi: 10.1109/ICFCC.2010.5497492.

[39] Hideo Tanaka, LT Fan, FS Lai, and K Toguchi. Fault-tree analysis by fuzzy probability. *IEEE Transactions on reliability*, 32(5):453–457, 1983.

[40] Leigh Thompson, Victoria Medvec, Vanessa Seiden, and Shirli Kopelman. *Poker Face, Smiley Face, and Rant 'n' Rave: Myths and Realities about Emotion in Negotiation*, pages 139 – 163. 01 2008. ISBN 9780470998458. doi: 10.1002/9780470998458.ch6.

[41] Rebekka Wohlrab and David Garlan. A negotiation support system for defining utility functions for multi-stakeholder self-adaptive systems. *Requirements Engineering*, pages 1–20, 2022.

[42] H. Peyton Young. Negotiation analysis. *University of Michigan*, 1992.

[43] M. Yuasa, Y. Yasumura, and K. Nitta. A negotiation support tool using emotional factors. In *Proceedings Joint 9th IFSA World Congress and 20th NAFIPS International Conference (Cat. No. 01TH8569)*, pages 2906–2911 vol.5, 2001. doi: 10.1109/NAFIPS.2001.943688.